Efficient Deep Learning: From Theory to Practice

by

Lucas Liebenwein

B.Sc., Swiss Federal Institute of Technology Zurich (2015)
S.M., Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Department of Electrical Engineering and Computer Science
August 27, 2021

Certified by .....
Daniela Rus
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Efficient Deep Learning: From Theory to Practice

by

Lucas Liebenwein

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Modern machine learning often relies on deep neural networks that are prohibitively expensive in terms of the memory and computational footprint. This in turn significantly inhibits the potential range of applications where we are faced with non-negligible resource constraints, e.g., real-time data processing, embedded devices, and robotics. In this thesis, we develop theoretically-grounded algorithms to reduce the size and inference cost of modern, large-scale neural networks. By taking a theoretical approach from first principles, we intend to understand and analytically describe the performance-size trade-offs of deep networks, i.e., the generalization properties. We then leverage such insights to devise practical algorithms for obtaining more efficient neural networks via pruning or compression. Beyond theoretical aspects and the inference time efficiency of neural networks, we study how compression can yield novel insights into the design and training of neural networks. We investigate the practical aspects of the generalization properties of pruned neural networks beyond simple metrics such as test accuracy. Finally, we show how in certain applications pruning neural networks can improve the training and hence the generalization performance.

Thesis Supervisor: Daniela Rus
Title: Professor of Electrical Engineering and Computer Science

This doctoral thesis has been examined by a Committee of the Department of Electrical Engineering and Computer Science:

Daniela Rus .....
Chairperson, Thesis Supervisor
Professor of Electrical Engineering and Computer Science

Michael Carbin .....
Member, Thesis Committee
Associate Professor of Electrical Engineering and Computer Science

Song Han .....
Member, Thesis Committee
Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

This thesis would have not been possible without the support from many people throughout my years as graduate student at MIT.

I would like to start by expressing my deep gratitude towards my advisor Daniela Rus. When I met Daniela five years ago, I just started out in my research career and did not know where this path would take me. Her invaluable guidance over the years were pivotal in my development and helped me grow both as a researcher and person. Daniela's enthusiasm for science and passion for solving some of the most impactful problems are inspiring and have consistently encouraged me to think hard about how our research can have a positive influence in the world. Her drive towards reaching a goal not only helped me to stay focused while navigating the infinite depth of research but has also taught me to be intentional and mindful in my time management. I would also like to thank Daniela for fostering such an interdisciplinary, open-minded, collaborative, and social community within our lab. I am already starting to miss my time as graduate student in your lab.

Together with Daniela, Song Han and Michael Carbin formed my thesis committee that offered critical advice and guidance towards the final steps in my PhD, including my defense and the creation of this thesis. For that, I would like to thank them.

Even before working towards this thesis, I had the opportunity to collaborate with and learn from many amazing people around the world including my undergraduate advisors Max Kriegleder and Raffaello D'Andrea at ETH Zurich; Emilio Frazzoli; as well as Javier Alonso-Mora, Jonathan DeCastro, Sertac Karaman, Russ Tedrake, and Cristian-Ioan Vasile during my years as SM student at MIT.

All of the research presented in this thesis is a result of many fruitful collaborations with inspiring, exceptional individuals with whom I had the pleasure to interact with and to learn from during my time as PhD student. Specifically, I would like to thank Alexander Amini, Zahra Babaiee, Cenk Baykal, Brandon Carter, Dan Feldman, Oren Gal, David Gifford, Igor Gilitschenski, Ramin Hasani, Mathias Lechner, Björn Lütjens, Alaa Maalouf, Ryan Sander, Wilko Schwarting, and Tim Seyde for exploring

so many exciting and interesting avenues of research with me. I also want to especially emphasize my gratitude towards Cenk Baykal who has been an integral part of many – if not most – of the research projects during my time at MIT. I truly enjoyed working so closely with you over the years.

The days on campus would have not been the same without the exceptional group of people at the Distributed Robotics Lab, with many of whom I have become close friends. The countless memories, including all of our lunches, coffee breaks, deeply passionate discussions about research and politics or both, late-night work sessions, lab outings, and so much more, will always make me smile when I will be looking back upon my time at MIT. For that, I am thankful to all of my current and former lab mates: Murad Abu-Khalaf, Javier Alonso-Mora, Alexander Amini, Brandon Araki, Thomas Balch, Yutong Ban, Rohan Banerjee, Cenk Baykal, James Bern, Thomas Buchner, Noam Buckman, Veevee Cai, Lillian Chin, Changhyun Choi, Jeana Choi, Sebastian Claici, Joseph DelPreto, David Dorhout, Stephanie Gil, Hunter Hansen, Ramin Hasani, Josie Hughes, Robert Katzschmann, Shuguang Li, Xiao Li, Jeffrey Lipton, Robert MacCurdy, Mieke Moran, Felix Naser, Teddy Ort, Liam Paull, Alyssa Pierson, Aaron Ray, John Romanishin, Guy Rosman, Andrés Salazar-Gómez, Ryan Sander, Wilko Schwarting, Tim Seyde, Hayim Shaul, Andy Spielberg, Ryan Truby, Paul Tylkin, Alex Wallar, Johnson Wang, and Annan Zhang.

I also want to thank Radu Grosu, Ramin Hasani, and all the other members of the CPS group at TU Vienna who gave me a second research home away from my research home at home while spending a year in Vienna during the Covid-19 pandemic. It has been wonderful interacting with everyone and getting to know many of you during that time.

Outside of the realms of research, I am incredibly thankful for all my friends, including my friends in Vienna, in Zurich, and in Boston. Thank you for your support, friendship, and fun times, especially in those moments where it is just helpful to forget about research for a while.

To all my family back home in Austria, thank you for always encouraging me to pursue my dreams and for giving me the strength to follow through with them. I want

to thank my mother, Jutta, for all the conversations and the advice over the years; my father, Karl, for always supporting me and having my back; my sister, Leonie, for being everything you could wish for in a sister; Stefanie, Constantin, and Nicolaus for all the happy moments together; Karin and Wolfgang for never being more than a phone call away; and my grandmother, Berta, for constantly inspiring me with an endless array of stories from a different time.

Finally, I cannot express the amount of love, joy, and gratitude I feel towards you, Pia. You make me feel alive in the morning when I wake up next to you, cheerful in the afternoon when I look at you, jovial in the evening when I talk to you, and serene at night when I fall asleep in your arms. Through hardship and struggle, you are there for me every step of the way and I could not imagine life without you. Your compassion, dedication, joyfulness, and kindness never fail to amaze me.

To you, my love.

Contents

List of Figures **19**

List of Tables **29**

1 Introduction **37**

1.1 Motivation 37

1.2 Vision 39

1.3 Challenges 40

1.3.1 Provable Coresets for Layers 40

1.3.2 Composable Neural Network Pruning Guarantees 41

1.3.3 Practical Pruning from Theoretical Guarantees 42

1.3.4 Modular Compression Techniques 42

1.3.5 Scalable Pruning Solutions 43

1.3.6 Generalization and Robustness in Pruning 44

1.3.7 Improved Generalization via Pruning 44

1.4 Contributions 45

1.4.1 Overview of Contributions 45

1.4.2 Open-source Implementation 47

1.4.3 Detailed Contributions 47

1.5 Outline 53

2 Related Work **55**

2.1 Coresets and Theoretical Foundations 55

2.2 Neural Network Compression and Pruning . . . 56
2.2.1 Unstructured Pruning . . . 57
2.2.2 Structured Pruning via Filter Pruning . . . 58
2.2.3 Low-rank Compression . . . 59
2.2.4 Network-aware Compression . . . 60
2.2.5 Retraining of Pruned Networks . . . 60
2.3 Generalization of Neural Networks . . . 61
2.3.1 Generalization . . . 61
2.3.2 Robustness . . . 62
2.3.3 Robust Training and Pruning . . . 63
2.3.4 Implicit Regularization via Overparameterization . . . 63
2.4 Architecture Design and Search . . . 64
2.5 Continuous-depth Models . . . 65

I Theoretical Foundations 67

3 Sensitivity-informed Compression Bounds for Neural Networks 69
3.1 Overview . . . 69
3.1.1 Contributions . . . 70
3.1.2 Relevant Papers . . . 70
3.1.3 Outline . . . 70
3.2 Problem Definition . . . 71
3.2.1 Fully-connected Neural Networks . . . 71
3.2.2 Neural Network Coreset Problem . . . 71
3.3 Method . . . 72
3.3.1 Sparsifying Weights in Neurons . . . 72
3.3.2 Neural Network Sparsification . . . 73
3.4 Analysis . . . 73
3.4.1 Preliminaries . . . 74
3.4.2 Empirical Sensitivity for Positive Weights . . . 75

3.4.3 Importance Sampling Bounds for Positive Weights 78
3.4.4 Importance Sampling Bounds for all Weights 84
3.5 Network Compression Bounds 91
3.5.1 Layer-wise Approximation 91
3.5.2 Network Compression 93
3.5.3 Generalization Bounds 95
3.6 Results 95
3.6.1 Experimental Setup 95
3.6.2 Results 97
3.7 Discussion 98

4 Generalized Compression Bounds **99**
4.1 Overview 99
4.1.1 Contributions 100
4.1.2 Relevant Papers 100
4.1.3 Outline 100
4.2 Problem Definition 100
4.3 Method 101
4.4 Analysis 103
4.4.1 Channel Sparsification 103
4.4.2 Main Compression Theorem 108
4.4.3 Extension to Filters 108
4.4.4 Boosting Sampling via Deterministic Choices 109
4.5 Discussion 116

II Efficient Neural Networks **117**

5 Provable Filter Pruning **119**
5.1 Overview 119
5.1.1 Contributions 120
5.1.2 Relevant Papers 121

5.1.3 Outline . . . 121
5.2 Filter Pruning . . . 121
5.2.1 Preliminaries . . . 122
5.2.2 Sampling-based Filter Pruning . . . 122
5.2.3 A Tightly-concentrated Estimator . . . 123
5.2.4 Empirical Sensitivity . . . 125
5.2.5 Derandomized Filter Pruning . . . 127
5.3 Optimal Budget Allocation . . . 128
5.4 Results . . . 130
5.4.1 Experimental Setup . . . 131
5.4.2 Comparison Methods . . . 131
5.4.3 LeNet Architectures on MNIST . . . 133
5.4.4 Convolutional Neural Networks on CIFAR-10 . . . 135
5.4.5 Convolutional Neural Networks on ImageNet . . . 137
5.4.6 Application to Real-time Regression Tasks . . . 140
5.5 Discussion . . . 142

6 Automatic Layer-wise Decomposition 145
6.1 Overview . . . 145
6.1.1 Contributions . . . 147
6.1.2 Relevant Papers . . . 147
6.1.3 Outline . . . 147
6.2 Method . . . 148
6.2.1 Preliminaries . . . 148
6.2.2 Local Layer Compression . . . 150
6.2.3 Global Network Compression . . . 153
6.2.4 Automatic Layer-wise Decomposition Selector (ALDS) . . . 157
6.3 Results . . . 161
6.3.1 Experimental Setup . . . 161
6.3.2 One-shot Compression with Baselines . . . 166

6.3.3 ImageNet Benchmarks . . . 168
6.3.4 Ablation Study . . . 170
6.3.5 Extensions of ALDS . . . 172
6.4 Discussion . . . 173

III Applications 174

7 Pruning Beyond Test Accuracy 175
7.1 Overview . . . 175
7.1.1 Contributions . . . 177
7.1.2 Relevant Papers . . . 178
7.1.3 Outline . . . 178
7.2 Methodology . . . 178
7.2.1 Pruning Setup . . . 178
7.2.2 Experiments Roadmap . . . 181
7.3 Function Distance . . . 182
7.3.1 Methodology . . . 182
7.3.2 Results . . . 184
7.4 Pruning under Distribution Changes . . . 186
7.4.1 Methodology . . . 188
7.4.2 Results . . . 189
7.5 Towards Robust Pruning . . . 194
7.5.1 Methodology . . . 195
7.5.2 Results . . . 195
7.6 Discussion . . . 197

8 Pruning Continuous-depth Models 201
8.1 Overview . . . 201
8.1.1 Contributions . . . 204
8.1.2 Relevant Papers . . . 204
8.1.3 Outline . . . 204

8.2 Background . . . 205
8.3 Pruning Neural ODEs . . . 206
8.3.1 A General Framework for Training Sparse Flows . . . 207
8.3.2 From Dense to Sparse Flows . . . 207
8.4 Experimental Setup . . . 209
8.5 Experiments . . . 209
8.5.1 Density Estimation on 2D Data . . . 211
8.5.2 Density Estimation on Real Data – Tabular . . . 212
8.5.3 Density Estimation on Real-Data – Vision . . . 213
8.5.4 Pruning Flattens the Loss Surface . . . 216
8.5.5 On the Robustness of Decision Boundaries . . . 217
8.6 Discussion . . . 219

9 Conclusion **221**
9.1 Summary . . . 221
9.2 Lessons Learned . . . 222
9.3 Closing Remarks . . . 225

A Appendix: Automatic Layer-wise Decomposition **227**

B Appendix: Pruning Beyond Test Accuracy **231**
B.1 Detailed Methodology and Prune Results . . . 231
B.1.1 Experimental Setup for CIFAR-10 . . . 232
B.1.2 Pruning Performance on CIFAR-10 . . . 233
B.1.3 Experimental Setup on ImageNet . . . 233
B.1.4 Pruning Performance on ImageNet . . . 235
B.1.5 Experimental Setup for Pascal VOC . . . 236
B.1.6 Pruning Performance on VOC . . . 236
B.2 Additional Results for Function Distance . . . 237
B.2.1 Comparison of Informative Features . . . 238
B.2.2 Noise Similarities . . . 241

B.3 Additional Results for Prune Potential 248
B.3.1 Detailed Results for Prune Potential with Corruptions 248
B.3.2 Choice of Commensurate Accuracy 255
B.3.3 Detailed Results for Excess Error with Corruptions 255
B.3.4 Results for Overparameterization 259
B.4 Detailed Results for Robust Pruning 260
B.4.1 Experimental Setup and Prune Results 260
B.4.2 Results for Prune Potential 261
B.4.3 Results for Excess Error 266
B.4.4 Results for Overparameterization 269

List of Figures

1-1 An overview of the thesis research organized around the three parts. . 54

3-1 Evaluation of drop in classification accuracy after compression against the *MNIST*, *CIFAR-10*, and *FashionMNIST* datasets with varying number of hidden layers (L) and number of neurons per hidden layer (η^*). Shaded region corresponds to values within one standard deviation of the mean. 97

3-2 Evaluation of relative error after compression against the *MNIST*, *CIFAR-10*, and *FashionMNIST* datasets with varying number of hidden layers (L) and number of neurons per hidden layer (η^*). 97

5-1 Overview of our pruning method. We use a small batch of data points to quantify the relative importance s_j^ℓ of each filter W_j^ℓ in layer ℓ by considering the importance of the corresponding feature map $a_j^\ell = \phi(z_j^\ell)$ in computing the output $z^{\ell+1}$ of layer $\ell + 1$, where $\phi(\cdot)$ is the non-linear activation function. We then prune filters by sampling each filter j with probability proportional to s_j^ℓ and removing the filters that were not sampled. We invoke the filter pruning procedure each layer to obtain the pruned network (the *prune* step); we then retrain the pruned network (*retrain* step), and repeat the *prune-retrain* cycle iteratively. 120

5-2 The performance of our approach on a LeNet300-100 architecture trained on MNIST with no derandomization (denoted by "rand"), with partial derandomization (denoted by "partial"), and with complete derandomization (denoted by "derand"). The plot in (a) and (b) show the resulting test accuracy for various percentage of retained parameters $1 -$ (pruneratio) before and after retraining, respectively. The additional error of the derandomized algorithm can be neglected in practical settings, especially after retraining. 128

5-3 Early layers of VGG are relatively harder to approximate due to their large spatial dimensions as shown in (a). Our error bounds naturally bridge layer compressibility and importance and enable us to automatically allocate relatively more samples to early layers and less to latter layers as shown in (b). The final layer – due to its immediate influence on the output – is also automatically assigned a large portion of the sampling budget. 129

5-4 The accuracy of the generated pruned models for the evaluated pruning schemes for various target prune ratios. Note that the x axis is the percentage of **parameters retained**, i.e., $(1 -$ pruneratio). ThiNet was omitted from the plots for better readability. Our results show that our approach generates pruned networks with minimal loss in accuracy even for high prune ratios. Shaded regions correspond to values within one standard deviation of the mean. 138

5-5 The results of our evaluations of the algorithms in the *prune-only* scenario, where the network is iteratively pruned down to a specified target prune ratio and the fine-tuning step is omitted. Note that the x axis is the percentage of parameters retained, i.e., $(1 -$ pruneratio). 140

5-6 The performance of our approach on a regression task used to infer the steering angle for an autonomous driving task (Amini et al., 2018). (a) An exemplary image taken from the data set. (b) The performance of our pruning procedure before retraining evaluated on the test loss and compared to competing filter pruning methods. Note that the x axis is percentage of parameters retained, i.e., $1 -$ (pruneratio). 141

6-1 **ALDS Overview**. The framework consists of a global and local step to obtain an optimal per-layer low-rank compression. We first randomly initialize the number of subspaces for each layer. We then optimize for the optimal per-layer, per-subspace rank by minimizing the maximum relative compression error (global step). Given a per-layer budget, we then optimize for the number of low-rank subspaces in each layer (local step). Both steps are repeated iteratively until convergence. 146

6-2 **Convolution to matrix multiplication.** A convolutional layer of $f = 20$ filters, $c = 6$ channels, and 2×2 kernel ($\kappa_1 = \kappa_2 = 2$). The input tensor shape is $6 \times 3 \times 3$. The corresponding weight matrix has $f = 20$ rows (one row per filter) and 24 columns ($c \times \kappa_1 \times \kappa_2$), as for the corresponding feature matrix, it has 24 rows and 4 columns, the 4 here is the number of convolution windows (i.e., number of pixels/entries in each of the output feature maps). After multiplying those matrices, we reshape them to the desired shape to obtain the desired output feature maps. 149

6-3 **Low-rank decomposition for convolutional layers via singular value decomposition (SVD).** The given convolution, c.f. Figure 6-2, has 20 filters, each of shape $6 \times 2 \times 2$, resulting in a total of 480 parameters. After extracting the corresponding weight matrix $W \in \mathbb{R}^{20\times 24}$ ($f \times c\kappa_1\kappa_2$), we compute its ($j = 7$)-rank decomposition to obtain the pair of matrices $U \in \mathbb{R}^{20\times 7}$ ($f \times j$) and $V \in \mathbb{R}^{7\times 24}$ ($j \times c\kappa_1\kappa_2$). Those matrices are encoded back as a pair of convolutional layers, the first (corresponding to V) has $j = 7$ filters, $c = 6$ channels and a 2×2 ($\kappa_1 \times \kappa_2$) kernel, whereas the second (corresponding to U) is a 1×1 convolution of $f = 20$ filters, and $j = 7$ channels. The resulting layers have 308 parameters. 150

6-4 **Left: 2D convolution. right: decomposition used for Automatic Layer-wise Decomposition Selector (ALDS).** For a $f \times c \times \kappa_1 \times \kappa_2$ convolution with f filters, c channels, and $\kappa_1 \times \kappa_2$ kernel, our per-layer decomposition consists: (1) k parallel $j \times {}^{c}/_{k} \times \kappa_1 \times \kappa_2$ convolutions; (2) a single $f \times kj \times 1 \times 1$ convolution applied on the first layer's (stacked) output. 153

6-5 One-shot compress+retrain experiments on CIFAR-10 with baseline comparisons. 166

6-6 The size-accuracy trade-off for various compression ratios, methods, and networks. Compression was performed after training and networks were re-trained once for the indicated amount (**one-shot**). (a, b, d, e): the difference in test accuracy for fixed amounts of retraining. (c, f): the maximal compression ratio with less-than-1% accuracy drop for variable amounts of retraining. 167

6-7 One-shot compress+retrain for DeeplabV3-ResNet50 on VOC. 167

6-8 The difference in test accuracy ("Delta Top1 Test Accuracy") for various target compression ratios, ALDS-based/ALDS-related methods, and networks on CIFAR-10. 172

6-9 The difference in test accuracy ("Delta Top1 Test Accuracy") for various target compression ratios, ALDS-based/ALDS-related methods, and networks on CIFAR-10. The networks were compressed once and not retrained afterwards. 172

7-1 The accuracy of the generated pruned models for the evaluated pruning schemes for various target prune ratios using iterative fine-tuning. . . 181

7-2 Heatmap of confidences on informative pixels from pruned ResNet20 models. Y-axis is the model used to generate 10% pixel subsets of 2000 sampled CIFAR-10 test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a)Weight Thresholding (WT), (b) Filter Thresholding (FT), (c) Sensitivity-informed Provable Pruning (SiPP), (d) Provable Filter Pruning (PFP). 184

7-3 The functional similarities between pruned ResNet20 models and their unpruned parent. We consider the difference in the output after injecting various amounts of noise into the input, see (a), (b) and (c), (d) for networks weight-pruned with WT and filter-pruned with FT, respectively. The differences between a separately trained network and the unpruned parent is also shown. The plots depict the difference measured as the percentage of matching predictions and as norm-based difference in the output after applying softmax, see (a), (c) and (b), (d), respectively. 185

7-4 The functional similarities between pruned ResNet20 models and their unpruned parent. 186

7-5 Example images from the CIFAR-10 test dataset that were used in this study with various levels of noise injected. A human test subject can classify the images equally well despite the noise present. 189

7-6 The prune potential (%) achievable over various levels of noise injected into the input on different network architectures trained and pruned on CIFAR-10. 190

7-7 The prune potential for a ResNet20 on CIFAR-10-C test datasets. We observe that depending on the type of corruption the network has significantly less prune potential than when measured w.r.t. the nominal CIFAR-10 test accuracy. 192

7-8 Prune potential of a ResNet18 (ImageNet). 193

7-9 The prune potential (b) and excess error (c) of a ResNet20 shown for corruptions that were included (train distribution) and excluded (test distribution) during training. The prune-accuracy curves in (a) are shown for corruptions from the test distribution. 195

8-1 Pruning neural ordinary differential equations (neural ODEs) improves their generalization with at least 1 order of magnitude less parameters. CIFAR-10 density estimation. Values and methods are described in Table 8.6. 202

8-2 Improved generalization through pruning. 8-2a: pruning enhances generalization of continuous-depth models. Structured pruning (green), unstructured pruning (blue). More details in Section 8.5. 8-2b: flat minima result in better generalization compared to sharp minima. Pruning neural ODEs flattens the loss around local minima. Figure 8-2b is reproduced from Keskar et al. (2017). 203

8-3 Negative log likelihood of Sparse Flow as function of prune ratio. . . . 212

8-4 Unstructured pruning of FFJORD (PR= Prune ratio). 213

8-5 Multi-modal Gaussian flow and pruning. We observe that Sparse Flows attract the vector-field directions uniformly towards the mean of each Gaussian distribution, while an unpruned flow does not exploit this feature and contains converging vectors in between Gaussians. 214

8-6 Loss vs. prune ratio for MNIST and CIFAR10. Unstructured Pruning is applied. 214

8-7 Negative log-likelihood versus prune ratio on tabular datasets with unstructured pruning. 215

8-8 Robustness of decision boundaries for pruned networks. Column 1 is the decision boundary. Column 2 = state-space, and column 3 = the flow of data points. 218

B-1 The difference in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR-10 for the evaluated pruning schemes for various target prune ratios. 233

B-2 The accuracy of the generated pruned models trained on ImageNet for the evaluated pruning schemes for various target prune ratios. 235

B-3 The accuracy of the generated pruned models trained on VOC for the evaluated pruning schemes and various target prune ratios for a DeeplabV3-ResNet50 architecture. 237

B-4 Heatmap of confidences on informative pixels from pruned VGG16 models. Y-axis is the model used to generate 10% pixel subsets of 2000 sampled CIFAR-10 test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a) Weight Thresholding (WT), (b) Filter Thresholding (FT), (c) SiPP, (d) Provable Filter Pruning (PFP). 239

B-5 Heatmap of confidences on informative pixels from pruned ResNet20 models. Y-axis is the model used to generate 10% pixel subsets of 2000 randomly sampled CIFAR-10-C corrupted test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a) Weight Thresholding (WT), (b) Filter Thresholding (FT). 240

B-6 Heatmap of confidences on informative pixels from pruned VGG16 models. Y-axis is the model used to generate 10% pixel subsets of 2000 randomly sampled CIFAR-10-C corrupted test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a) Weight Thresholding (WT), (b) Filter Thresholding (FT). 240

B-7 The functional similarities between pruned ResNet56 models and their unpruned parent. 242

B-8 The functional similarities between pruned ResNet110 models and their unpruned parent. 242

B-9 The functional similarities between pruned VGG16 models and their unpruned parent. 243

B-10 The functional similarities between pruned DenseNet22 models and their unpruned parent. 243

B-11 The functional similarities between pruned WRN16-8 models and their unpruned parent. 244

B-12 The functional similarities between pruned ResNet56 models and their unpruned parent. 245

B-13 The functional similarities between pruned ResNet110 models and their unpruned parent. 246

B-14 The functional similarities between pruned VGG16 models and their unpruned parent. 246

B-15 The functional similarities between pruned DenseNet22 models and their unpruned parent. 247

B-16 The functional similarities between pruned WRN16-8 models and their unpruned parent. 247

B-17 The prune potential of a ResNet20 achievable for CIFAR-10 out-of-distribution data sets. 249

B-18 The prune potential of a ResNet56 achievable for CIFAR-10 out-of-distribution data sets. 250
B-19 The prune potential of a ResNet110 achievable for CIFAR-10 out-of-distribution data sets. 251
B-20 The prune potential of a VGG16 achievable for CIFAR-10 out-of-distribution data sets. 251
B-21 The prune potential of a DenseNet22 achievable for CIFAR-10 out-of-distribution data sets. 252
B-22 The prune potential of a WRN16-8 achievable for CIFAR-10 out-of-distribution data sets. 252
B-23 The prune potential of a ResNet18 achievable for ImageNet out-of-distribution data sets. 253
B-24 The prune potential of a ResNet101 achievable for ImageNet out-of-distribution data sets. 253
B-25 The prune potential of a DeeplabV3 achievable for Pascal VOC out-of-distribution data sets. 254
B-26 The prune potential of a ResNet20 trained on CIFAR-10 for WT and FT, respectively. In each figure the same experiment is repeated with different values of δ ranging from 0% to 5%. 254
B-27 The difference in excess error for a ResNet20 trained on CIFAR-10. . 256
B-28 The difference in excess error for a ResNet56 trained on CIFAR-10. . 256
B-29 The difference in excess error for a ResNet110 trained on CIFAR-10. . 257
B-30 The difference in excess error for a VGG16 trained on CIFAR-10. . . 257
B-31 The difference in excess error for a DenseNet22 trained on CIFAR-10. 257
B-32 The difference in excess error for a WRN16-8 trained on CIFAR-10. . 257
B-33 The difference in excess error for a ResNet18 trained on ImageNet. . . 258
B-34 The difference in excess error for a ResNet101 trained on ImageNet. . 258
B-35 The difference in excess error for a DeeplabV3 trained on Pascal VOC. 258
B-36 The difference in test accuracy (nominal CIFAR-10) to the uncompressed network. 261

B-37 The prune potential of a robustly pruned ResNet20 achievable for CIFAR-10 out-of-distribution data sets. 262
B-38 The prune potential of a robustly pruned ResNet56 achievable for CIFAR-10 out-of-distribution data sets. 263
B-39 The prune potential of a robustly pruned ResNet110 achievable for CIFAR-10 out-of-distribution data sets. 263
B-40 The prune potential of a robustly pruned VGG16 achievable for CIFAR-10 out-of-distribution data sets. 264
B-41 The prune potential of a robustly pruned DenseNet22 achievable for CIFAR-10 out-of-distribution data sets. 264
B-42 The prune potential of a robustly pruned WRN16-8 achievable for CIFAR-10 out-of-distribution data sets. 265
B-43 The difference in excess error for a robustly pruned ResNet20 trained on CIFAR-10. 266
B-44 The difference in excess error for a robustly pruned ResNet56 trained on CIFAR-10. 266
B-45 The difference in excess error for a robustly pruned ResNet110 trained on CIFAR-10. 267
B-46 The difference in excess error for a robustly pruned VGG16 trained on CIFAR-10. 267
B-47 The difference in excess error for a robustly pruned DenseNet22 trained on CIFAR-10. 267
B-48 The difference in excess error for a robustly pruned WRN16-8 trained on CIFAR-10. 268

List of Tables

5.1 We report the hyperparameters used during MNIST training, pruning, and fine-tuning for the LeNet architectures. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During fine-tuning we used the same hyperparameters except for the ones indicated in the lower part of the table. 133

5.2 The prune ratio and the corresponding test error of the sparsest network – with commensurate accuracy – generated by each algorithm. . 134

5.3 We report the hyperparameters used during training, pruning, and fine-tuning for various convolutional architectures on CIFAR-10. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During fine-tuning we used the same hyperparameters except for the ones indicated in the lower part of the table. $\{30, \ldots\}$ denotes that the learning rate is decayed every 30 epochs. 135

5.4 Overview of the pruning performance of each algorithm for various CNN architectures. For each algorithm and network architecture, the table reports the prune ratio (PR, %) and pruned FLOPs ratio (FR, %) of pruned models when achieving test accuracy within 0.5% of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). Our results indicate that our pruning algorithm generates smaller and more efficient networks with minimal loss in accuracy, when compared to competing approaches. 136

5.5 The performance of our algorithm and that of state-of-the-art filter pruning algorithms on modern CNN architectures trained on CIFAR-10. The reported results for the competing algorithms were taken directly from the corresponding papers. For each network architecture, the best performing algorithm for each evaluation metric, i.e., Pruned Err., Err. Diff, PR, and FR, is shown in **bold**. The results show that our algorithm consistently outperforms state-of-the-art pruning approaches in nearly all of the relevant pruning metrics. 137

5.6 The hyper-parameters used for training and pruning residual networks trained on the ImageNet data set. 139

5.7 Comparisons of the performance of various pruning algorithms on ResNets trained on ImageNet (Russakovsky et al., 2015). The reported results for the competing algorithms were taken directly from the corresponding papers. For each network architecture, the best performing algorithm for each evaluation metric, i.e., Pruned Err., Err. Diff, PR, and FR, is shown in bold. 139

5.8 We report the hyperparameters used for training and pruning the driving network of Amini et al. (2018) together with the provided data set. No fine-tuning was conducted for this architecture. 141

6.1 The experimental hyperparameters for training, compression, and retraining for the tested CIFAR-10 network architectures. "LR" and "LR decay" hereby denote the learning and the (multiplicative) learning rate decay, respectively, that is deployed at the epochs as specified. "$\{x, \ldots\}$" indicates that the learning rate is decayed every x epochs. . 164

6.2 The experimental hyperparameters for training, compression, and retraining for the tested ImageNet network architectures. "LR" and "LR decay" hereby denote the learning and the (multiplicative) learning rate decay, respectively, that is deployed at the epochs as specified. "$\{x, \ldots\}$" indicates that the learning rate is decayed every x epochs. . 165

6.3 The experimental hyperparameters for training, compression, and retraining for the tested VOC network architecture. "LR" and "LR decay" hereby denote the learning and the learning rate decay, respectively. Note that the learning rate is polynomially decayed after each step. 166

6.4 Baseline results for Δ-Top1$\geq -0.5\%$ for one-shot. Results coincide with Figures 6-5, 6-6, 6-7. 168

6.5 AlexNet and ResNet18 Benchmarks on ImageNet. We report Top-1, Top-5 accuracy and percentage reduction in terms of parameters and FLOPs denoted by CR-P and CR-F, respectively. Best results with less than 0.5% accuracy drop are bolded. 169

7.1 Overview of the pruning methods evaluated. Here, $a(x)$ denotes the activation of the corresponding layer with respect to a sample input x to the network. 179

7.2 The prune potential of various networks trained on CIFAR-10 (upper part) and ImageNet (lower part) evaluated on the train and test distribution, which consist of nominal data and the average over all corruptions, respectively. 194

8.1 Pruning Methods. 208

8.2 **Toy Dataset** Hyperparameters. 210

8.3 **Tabular Datasets** Hyperparameters. 211

8.4 **Image Datasets** Hyperparameters. 211

8.5 Negative test log-likelihood (NLL) in nats of tabular datasets from (Papamakarios et al., 2017) and corresponding architecture size in number of parameters (#params). Sparse Flow (based on Free-form Jacobian of Reversible Dynamics (FFJORD)) with lowest NLL and competing baseline with lowest NLL are bolded. 215

8.6 Negative test log-likelihood (NLL) in bits/dim for image datasets and corresponding architecture size in number of parameters (#params). Sparse Flow (based on FFJORD) with lowest NLL and competing baseline with lowest NLL are bolded. 216

8.7 Eigenanalysis of the Hessian H in terms of the largest eigenvalue (λ_{max}), trace (tr), and condition number (κ) of pruned and unpruned continuous normalizing flows on the mixture of Gaussian task. Numbers are normalized with respect to the unpruned flow. 217

A.1 The maximal compression ratio for which the drop in test accuracy is at most some pre-specified δ on CIFAR-10. The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded. 228

A.2 The maximal compression ratio for which the drop in test accuracy is at most $\delta = 1.0\%$ for ResNet20 (CIFAR-10) for various amounts of retraining (as indicated). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded. 228

A.3 The maximal compression ratio for which the drop in test accuracy is at most some pre-specified δ on ResNet18 (ImageNet). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded. 229

A.4 The maximal compression ratio for which the drop in test accuracy is at most $\delta = 1.0\%$ for ResNet18 (ImageNet) for various amounts of retraining (as indicated). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded. 229

A.5 The maximal compression ratio for which the drop in test accuracy is at most some pre-specified δ on DeeplabV3-ResNet50 (Pascal VOC2012). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded. 229

B.1 We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on CIFAR-10. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During retraining we used the same hyperparameters. $\{30, \ldots\}$ denotes that the learning rate is decayed every 30 epochs. 232

B.2 Overview of the pruning performance of each algorithm for various CNN architectures evaluated on the CIFAR data set. For each algorithm and network architecture, the table reports the prune ratio (PR, %) and the ratio of flop reduction (FR, %) of pruned models when achieving test accuracy within $\delta = 0.5\%$ of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). The top values for the error and either PR (for weight-based) or FR (for filter-based algorithms) are bolded, respectively. 234

B.3 We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on ImageNet. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. 234

B.4 Overview of the pruning performance of each algorithm for various CNN architectures trained and evaluated on the ImageNet data set. For each algorithm and network architecture, the table reports the prune ratio (PR, %) and the ratio of flop reduction (FR, %) of pruned models when achieving test accuracy within $\delta = 0.5\%$ of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). The top values for the error and either PR (for weight-based) or FR (for filter-based algorithms) are bolded, respectively. 236

B.5 We report the hyperparameters used during training, pruning, and retraining for various architectures on Pascal VOC 2011. LR hereby denotes the learning rate and LR decay denotes the learning rate decay. Note that the learning rate is polynomially decayed after each step. . 237

B.6 Overview of the pruning performance of each algorithm for DeeplabV3 trained and evaluated on Pascal VOC segmentation data. For each algorithm, the table reports the prune ratio (PR, %) and the ratio of flop reduction (FR, %) of pruned models when achieving IoU test accuracy within $\delta = 0.5\%$ of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). The top values for the error and either PR (for weight-based) or FR (for filter-based algorithms) are bolded, respectively. 238

B.7 The average and minimum prune potential computed on the train and test distribution, respectively, for weight prune methods (WT, SiPP). The train distribution hereby consists of nomimal data, while the test distribution consists of the CIFAR-10-C, ImageNet-C, VOC-C corruptions. 259

B.8 The average and minimum prune potential computed on the train and test distribution, respectively, for filter prune methods (FT, PFP). The train distribution hereby consists of nomimal data, while the test distribution consists of the CIFAR-10-C, ImageNet-C, VOC-C corruptions. 260

B.9 The list of corruptions used for the train and test distribution, respectively, categorized according to type. 261

B.10 The average and minimum prune potential computed on the train and test distribution, respectively, for weight prune methods (WT, SiPP). The train and test distribution hereby each consist of a mutually exclusive subset of corruptions as listed in Table B.9. 270

B.11 The average and minimum prune potential computed on the train and test distribution, respectively, for filter prune methods (FT, PFP). The train and test distribution hereby each consist of a mutually exclusive subset of corruptions as listed in Table B.9. 270

Chapter 1

Introduction

1.1 Motivation

Within the past decade, large-scale deep neural networks (DNNs) have demonstrated unprecedented empirical success in high-impact applications such as object classification (Chen et al., 2017), speech recognition (Graves et al., 2013), computer vision (Krizhevsky et al., 2012), and natural language processing (Vaswani et al., 2017). In addition, neural networks (NNs) have become the backbone of many other learning problems such as self-supervised learning (Brown et al., 2020), reinforcement learning (Mnih et al., 2013), and applications thereof such as robotics (Levine et al., 2016), autonomous driving (DeCastro et al., 2018; Schwarting et al., 2020), games (Silver et al., 2018), and sustainable artificial intelligence (Lütjens et al., 2019).

However, with the ever-increasing size of state-of-the-art (SOTA) neural networks, the resulting storage and computation requirements of these models are becoming increasingly prohibitive in terms of both time and space. For example, GPT-3 (Brown et al., 2020), one of the current state-of-the-art models for natural language processing (NLP), contains 175 Billion Parameters. As noted by Brown et al. (2020), the sheer size of GPT-3 results in a training cost of "*several thousand petaflop/s-days of compute*" or equivalently *several hundred single-GPU-years* assuming 20 teraflop/s for a single graphics processing unit (GPU). Training and inference with such models are thus only possible through massive parallelization requiring thousands of GPU units.

This has significant financial and environmental implications as well. For example, the training procedure of GPT-3 roughly costs about $12 million to train a single instance of the model (Wiggers, 2021). Even earlier neural network architectures, such as those proposed by Badrinarayanan et al. (2015); He et al. (2016); Krizhevsky et al. (2012); Long et al. (2015), contain millions of parameters, rendering them prohibitive to deploy on platforms that are resource-constrained, e.g., embedded devices, mobile phones, or small-scale robotic platforms.

To this end, prior work has considered the possibility of reducing the size, inference cost, and training cost of large-scale DNNs without degradation in performance. Among others, techniques include quantization (Rastegari et al., 2016; Wu et al., 2016), where weights are represented using less bits, knowledge distillation (Hinton et al., 2015; Park et al., 2019), where a small network is trained to imitate a large one, and neural architecture search (NAS) (Liu et al., 2019a), where the architecture and the weights of the neural network are learned simultaneously. Additional methods are low-rank compression, where the underlying weight tensors are represented in a lower dimensional subspace, and pruning (Han et al., 2015a), where individual weights or structures of the network are removed.

Pruning and compressing existing architectures particularly stand out as succinct ways to study and describe the phenomenon of overparameterization in deep neural networks (Arora et al., 2018; Liebenwein et al., 2021a). By starting out from large architectures we can theoretically and empirically study to what amount the network architecture can be reduced without degradation in performance. Consequently, we can also investigate algorithms that yield minimal representations for a given learning task. Moreover, such techniques enable us to consider individually compressing different parts of the network, including weights or neurons in layers, or representing layers in lower dimensional subspaces (low-rank compression). We can thus gain novel insights into efficient representations for single layers as well as the overall width-depth trade-off in networks.

Typical pruning algorithms either proceed by gradually pruning the network during training (Gale et al., 2019; He et al., 2018; Peste et al., 2021; Zhu and Gupta, 2017)

or by pruning the network after training followed by a retraining period (Baykal et al., 2021b; Han et al., 2015a; Liebenwein et al., 2020; Renda et al., 2020). By pruning individual parameters or structures, the network subsequently requires less storage and has reduced inference cost thus promising to alleviate some of the concerns that come with large-scale DNNs. An overview of recent approaches is, e.g., given by Blalock et al. (2020); Gale et al. (2019); Hoefler et al. (2021).

However, current pruning approaches fall short in several key areas. They are generally based on heuristics (Han et al., 2015a; He et al., 2018; Lee et al., 2019; Li et al., 2016; Luo et al., 2017; Ullrich et al., 2017; Yu et al., 2018b) that lack guarantees on the size and performance of the pruned network. Most prior approaches also require cumbersome ablation studies (He et al., 2018; Li et al., 2016) or manual hyper-parameter tuning (Luo et al., 2017) in order to obtain practical results. Others heavily rely on assumptions such as that parameters and structures of the network with large weight magnitudes are more important, which does not hold in general (Li et al., 2016; Ye et al., 2018; Yu et al., 2018b).

In short, we currently lack a principled understanding of what constitutes an efficient or optimally-parameterized architecture, what are the trade-offs between different architectures, and what are algorithms that enable us to obtain optimally-parameterized neural networks. Much of modern machine learning research is driven by empirical performance, which is justified as many of the application are ultimately driven by the empirical success of such models. Despite the predominantly empirical nature of deep learning, we are motivated by the vision that theoretical insights and principled approaches to designing and training neural networks inherently yield more efficient, more scalable, and more performant models.

1.2 Vision

The vision of this thesis is thus to enable the wide-spread adaptation of deep learning-based systems and applications beyond current, mostly cloud-based applications. We do so by advancing our theoretical understanding and practical toolkit for efficiently

designing and deploying such systems. In today's machine learning systems, large parts of the system itself, specifically the architecture and the associated hyperparameters, are predominantly hand-engineered. Consequently, these parts are neither well-understood nor optimized for efficiency. This makes the design process of intelligent machine learning systems prohibitively expensive for small-scale systems, such as Internet of Things (IoT) devices, mobile device, or robots. For example, to design a deep neural network for a robot, we currently need to design, test, and tune the architecture of the network to ensure adequate performance on the target application, e.g., robot control. Simultaneously, we must ensure that we meet the computational resource constraints of the system, e.g., a small microcontroller.

By contrast, the techniques proposed as part of this thesis enable a more principled understanding of deep learning. Thereafter, this thesis introduces methods building upon our improved theoretical understanding to automatically design and implement more efficient neural networks. As a result, the design process, e.g., for a learning-based robotic system, becomes less cumbersome and more flexible. At the same time, we enable faster prototyping times and the ability to conveniently pick the optimal trade-off between performance and computation time suited for the task at hand. We envision that the techniques presented in this thesis will therefore be a driving force in broadening the range of possible applications for deep learning previously inhibited by the resource- and labor-intensive nature of current state-of-the-art systems.

1.3 Challenges

In order to realize our vision we have to overcome fundamental challenges of conceptual as well as theoretical and practical nature. Below, we discuss key technical challenges that this thesis aims to make progress towards solving.

1.3.1 Provable Coresets for Layers

A prototypical NN usually consists of a number of linear and non-linear layers, i.e., operations, that are either stacked together sequentially, in parallel, or a combination

thereof. Having a precise understanding and tight (analytical) description of individual layers can thus serve as crucial starting point for analyzing neural network architectures. Specifically, linear layers usually contain the vast majority of parameters (and thus computational complexity of the network). We note that provable approximation techniques for large-scale linear algebra in itself is a well-studied problem, e.g., see the work of Achlioptas et al. (2013). However, the challenge hereby lies in developing techniques that maximize the parameter-efficiency trade-offs using insights from the actual input data distribution of the network. Unlike generic matrices, individual layers of neural networks are trained for specific input data – an insight that we can leverage to develop more efficient approximation techniques.

In this thesis, we analyze individual (linear) layers of neural networks to provably quantify the trade-off between approximation quality and size of the layer. We build upon the rich body of literature in coresets (Feldman, 2020), i.e., a weighted subset of the original set of points (or parameters in this case). Coreset techniques can be used to analytically describe the original set up to some desired approximation error and have found numerous applications in classical machine learning problems such as k-means (Feldman et al., 2007). Moreover, we consider designing and analyzing our coreset techniques in a data-informed manner, thus leveraging the network- and data-dependent distribution of inputs to the various layers.

1.3.2 Composable Neural Network Pruning Guarantees

Neural network architectures can be arbitrarily composed out of individual layers giving rise to a diverse set of architectures. Thus, in addition to deciphering individual layers, our network analysis has to be composable and adaptable according to the given architecture. The challenge hereby lies in accurately describing the approximation error across multiple layers and capturing how errors propagate through the network in order to leverage our layer-wise coresets analysis.

To this end, we develop a composable error analysis for simultaneously compressing multiple layers of a neural network. We also incorporate concepts of generalization theory from the neural network literature (Arora et al., 2018) to bridge the fields of

neural network pruning and generalization theory. Both fields share the common goal of understanding the trade-offs between the size and performance of the network.

1.3.3 Practical Pruning from Theoretical Guarantees

While theoretical frameworks and bounds can provide comprehensive insights into the foundation of deep learning, instantiating them for practical, large-scale architectures often leads to loose bounds. Consequently, there exists a large gap between theoretical predictions and empirical observations in the neural network literature. Thus, considering the transition of theoretical guarantees into practical algorithms requires a careful trade-off between relevant parts of the theoretical analysis and observation-driven algorithms that are usually heuristic in nature. In the context of pruning, it is particularly challenging to capture the global effects of pruning individual layers. In order words, we need to be able to handle the trade-off between how much each individual layer should be pruned in order to obtain an approximately optimal network for the desired size.

Building upon our theoretical coresets tools, we develop practical pruning algorithms advancing the Pareto-optimal frontier of performance-size trade-offs for neural networks. We consider mainly vision networks that are composed of numerous individual layers of different types, including linear, convolutional, batch normalization, and non-linear activation layers. Moreover, we frame the per-layer pruning problem as a constrained optimization problem. The goal is to achieve minimal overall difference in loss compared to the original, unpruned network subject to some desired overall reduction in network parameters.

1.3.4 Modular Compression Techniques

Beyond parameter efficiency in terms of weights, we can consider other structures of the network as well and investigate their redundancy. This question is related to dimensionality reduction (Cohen et al., 2015) and can provide useful insights into the macro parameters of the network architecture, such as the minimal required width

of individual layers. From a implementation perspective we note that the former requires specialized software and hardware to leverage the resulting sparsity, while the latter directly emits a smaller architecture leading to faster runtimes in practice.

In addition to unstructured pruning, i.e., removing individual weights, we provide a principled approach to structured pruning, i.e., removing filters and neurons, and low-rank compression. We generalize our notion of parameter pruning to pruning substructures and provide the accompanying theoretical compression guarantees. In addition, we consider low-rank compression as an alternative subprocedure to dimensionality reduction and highlight how we can incorporate low-rank decomposition into our pruning framework.

1.3.5 Scalable Pruning Solutions

The usual design process for neural network architectures requires a human expert to parameterize each layer individually in a labor-intensive, iterative process. Alternatively, prior work considered NAS, e.g., see the recent approaches of Cai et al. (2018); Liu et al. (2019a). The search essentially proceeds by optimizing over both the weights and connections of a generic architecture thus significantly decreasing the human labor effort at the cost of a significant increase in computational complexity. A fundamental challenge towards more scalable and automated training of efficient neural networks is thus to overcome the trade-offs between automated design and the resulting computational cost of the interleaved training and design process.

In this thesis, we leverage our insights on pruning via composable coresets to devise algorithm that can significantly reduce the human effort required to design novel architectures. Specifically, we leverage our network analysis to understand how a given base architecture can be optimally pruned and re-factored to suit the task at hand while simultaneously generating an architecture with minimal inference cost. We incorporate both structured pruning and low-rank decomposition into our pruning framework enabling us to significantly reduce the dimensionality of layers leading to novel architectures on the fly.

1.3.6 Generalization and Robustness in Pruning

Complementary to our theoretical understanding of pruning and generalization, studying practical aspects of generalization can provide crucial insights into the performance capabilities of neural networks. Since pruned networks are highly optimized for efficiency, we have to vigilantly examine their potential side effects, brittleness, or bias to an even greater extent than for regular networks. The challenge is thus to develop measurable and practical benchmarks that enable us to reliably quantify the performance of a pruned network across a wide range of tasks.

Specifically, in this thesis we establish empirical benchmarks to more accurately assess the performance-size trade-off of pruned networks that go beyond simple metrics such as test accuracy. We study networks from a practical lens when faced with out-of-distribution input data points in order to understand how pruned networks differ. Our insights shed light on the need to consider task-specific evaluation metrics during pruning, prior to the deployment of a pruned network to, e.g., safety-critical systems such as robots. In that sense, we aim to provide a robust, empirical framework to measure the amount of genuine overparameterization in networks.

1.3.7 Improved Generalization via Pruning

Pruning provides a principled approach to optimize the computational efficiency of deep learning practice. More broadly speaking, we can study the training and the design of neural networks through the lens of pruning. Here we are interested in understanding whether pruning may improve the stability of training or potentially lead to more performant architectures in the first place. To examine this question further, we consider the model of neural ordinary differential equations (neural ODEs) (Chen et al., 2018b) and other time-continuous neural networks (Grathwohl et al., 2019). Unlike regular, fixed-depth networks, neural ODEs can naturally model depth-varying (time-varying) tasks. Applications include robotic control tasks, where we want to control the robot at arbitrary moments in time (Lechner et al., 2020a), and generative models, where the transformation from noise to data is modelled in a time-continuous

fashion (Grathwohl et al., 2019). Due to the depth-varying nature of such models, designing an appropriate architecture becomes an even more challenging task. In addition, the training is more prone to suffer from numerical issues such as exploding or vanishing gradients (Lechner and Hasani, 2020).

In this thesis, we investigate continuous-depth models via pruning to understand some of their inherent generalization properties. Based on these insights, we devise a framework to train sparse, continuous-depth models. We improve upon the existing performance-size trade-off in typical generative modeling tasks such as image generation (Grathwohl et al., 2019). Overall, sparse models lead to better practical performance while requiring less parameters and sometimes less training as well.

1.4 Contributions

Towards overcoming the challenges outlined in the previous section, we present novel techniques that are based on theoretically-grounded algorithms to reduce the size and inference cost of modern large-scale neural networks. We present practical modifications to our algorithms and study their effectiveness in a variety of experimental settings. Finally, we study the effects of pruning neural networks in the context of robustness and generalization beyond typical benchmarks.

1.4.1 Overview of Contributions

List of Contributions

This thesis contributes the following:

- A provable pruning approach via coresets for any linear neural network layer followed by an error analysis for composed networks and its application to generalization bounds.
- A generalization of our coreset tools and analysis to structured pruning in conjunction with a theoretically-grounded, yet practical approach to filter pruning for arbitrary network architectures.

- A unified pruning framework extending our pruning results to decomposition-based, low-rank compression with empirical pruning benchmarks highlighting the efficacy of the proposed method.

- A large-scale empirical study on the limitations of pruning and on the generalization properties of pruned neural networks including out-of-distribution robustness properties.

- An investigation of continuous-depth neural network models with applications to vision-based generative modeling tasks leading to enhanced generalization capabilities of such architectures via pruning.

New Capabilities

By taking a theoretical approach from first principles, we analytically describe the performance-size trade-offs, i.e., the generalization properties, of modern neural networks. To the best of our knowledge, we provide the first coreset-based analysis and accompanying theoretical error guarantees of compressed neural networks. Our approach is flexible with regards to the architecture, accounts for various types of pruning, and provides novel insights into the theoretical study of neural networks.

We then leverage our coreset tools to devise practical algorithms for obtaining efficient neural network architectures via pruning. Building upon our novel approach to network pruning, we conceive a modular, fully-automated framework to neural network compression. We highlight its efficacy in the context of structured pruning and low-rank compression. Compared to prior work, we achieve higher prune ratios for the same accuracy levels while simultaneously providing a set of more flexible and automated pruning procedures. Hence, our pruning techniques are not only highly effective in practice but also broadly applicable beyond common benchmarks due to minimal required effort for hyperparameter tuning.

Finally, we study practical aspects of the generalization properties of pruned neural networks beyond simple metrics such as test accuracy and consider how pruning may negatively affect their robustness properties. In a separate study, we highlight

that pruning can also have a significant positive impact on the generalization performance of generative, continuous-depth models including generative modeling of images. Unlike prior work, which is predominantly focused on obtaining the smallest network possible for a desired level of accuracy, our work highlights that pruning may alter the generalization abilities of networks beyond the aforementioned performance-size trade-offs. In that sense, we open up an array of potential future work to study pruning as a novel means to regularize the training of neural networks.

1.4.2 Open-source Implementation

Apart from the results presented in this thesis, we contribute an open-source library for pytorch-based neural network pruning and compression:

- Lucas Liebenwein. Torchprune: A research library for pytorch-based neural network pruning, compression, and more. https://github.com/lucaslie/torchprune, 2021.

The aforementioned library contains the necessary code and hyperparameters to reproduce the experimental results presented throughout this thesis. In addition, our code also contains the pruning and compression methods contributed by this thesis as well as commonly-used benchmark pruning methods. Finally, we provide a modular application programming interface (API) to build on top of our research code.

1.4.3 Detailed Contributions

We provide a chapter-by-chapter overview of our contributions below including references to the papers on which the respective contributions are based.

Sensitivity-informed Compression Bounds

We introduce a provable approach to weight pruning via layer-wise sparsification of trained, fully-connected models in a way that approximately preserves the model's predictive accuracy. Based on the insight that we can leverage a small batch of input

data points to approximately estimate the relative importance of individual parameters, we derive a provable sparsification scheme for individual layers. We then derive compositional network compression bounds that analytically capture and quantify the parameter-error trade-off when pruning a network that is sequentially composed of individual layers. Finally, we present weight pruning results for fully-connected networks. Our empirical results suggest that our framework reliably generates compressed networks and outperforms existing provable approaches to matrix sparsification in the context of neural networks. In short, our contributions are as follows:

- A provable coreset approach to sparsifying neural network parameters via importance sampling based on a novel, empirical notion of sensitivity.

- Analytical results establishing guarantees on the approximation accuracy, size, and generalization of the compressed neural network.

- Evaluations on real-world data sets that demonstrate the practical effectiveness of our algorithm in compressing neural network parameters and validate our theoretical results.

These results are based on the following joint papers with Cenk Baykal:

- Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

- Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *SIAM Journal on Mathematics of Data Science (Under Review; arXiv preprint arXiv:1910.05422)*, 2021b.

We present our results in Chapter 3. For further context and additional results, we may also refer the interested reader to Cenk Baykal's doctoral thesis (Baykal, 2021).

Provable Filter Pruning

We present a provable approach for generating compact convolutional neural networks (CNNs) by identifying and removing redundant filters from an over-parameterized network. Our algorithm uses a small batch of input data points to assign a saliency score to each filter and constructs an importance sampling distribution where filters that highly affect the output are sampled with correspondingly high probability. In contrast to existing filter pruning approaches, our method is simultaneously data-informed, exhibits provable guarantees on the size and performance of the pruned network, and is widely applicable to varying network architectures and data sets. Our analytical bounds bridge the notions of compressibility and importance of network structures, which gives rise to a fully-automated procedure for identifying and preserving filters in layers that are essential to the network's performance. Our experimental evaluations on popular architectures and data sets show that our algorithm consistently generates sparser and more efficient models than those constructed by existing filter pruning approaches. Our contributions here are as follows:

- An extension of our provable coreset approach in the context of filter and neuron pruning based on a generalized notion of empirical sensitivity.
- Analytical results establishing guarantees on the trade-off between approximation accuracy and size of the pruned neural network.
- An improved pruning framework via a mixture approach of sampling-based and deterministic pruning and accompanying analysis highlighting the improved sample efficiency.
- A layer-wise allocation procedure to optimally allocate a per-layer prune budget based on our layer-wise theoretical error guarantees.
- Evaluations on large-scale data sets and networks that demonstrate the practical effectiveness of our algorithm in compressing neural networks via filter pruning.

These results are based on the following paper:

- Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

We present our results in Chapters 4 and 5.

Automatic Layer-wise Decomposition Selector (ALDS)

We present a novel global compression framework for neural networks that automatically analyzes each layer to identify the optimal per-layer compression ratio, while simultaneously achieving the desired overall compression. Our algorithm hinges on the idea of compressing each convolutional (or fully-connected) layer by "slicing" its channels into multiple groups and decomposing each group via low-rank decomposition. At the core of our algorithm is the derivation of layer-wise error bounds from the Eckart–Young–Mirsky theorem. We then leverage these bounds to frame the compression problem as an optimization problem, where we wish to minimize the maximum compression error across layers and propose an efficient algorithm towards a solution. Our experiments indicate that our method outperforms existing low-rank compression approaches across a wide range of networks and data sets. In short, our contributions are as follows:

- An efficient layer-wise decomposition framework relying on a straightforward decomposition of each layer that is based on the singular value decomposition (SVD).
- A generalization of our layer-wise decomposition via an automatically-determined splitting of the underlying weight tensor into multiple subsets.
- A global framework that optimally determines the type of decomposition (number of subsets) and the optimal per-layer low-rank compression based on minimizing the maximum relative error incurred.
- Extensive experimental evaluations on multiple benchmarks, models, and datasets, including large-scale datasets, establishing the competitive performance of our

algorithm relative to existing approaches.

These results are based on the following paper:

- Lucas Liebenwein, Alaa Maalouf, Oren Gal, Dan Feldman, and Daniela Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. In *Advances in Neural Information Processing Systems (Under Review; arXiv preprint arXiv:2107.11442)*, 2021c.

We present our results in Chapter 6.

Pruning Beyond Test Accuracy

We evaluate whether the use of test accuracy alone is sufficient to assess that pruned models perform well across a wide spectrum of "harder" metrics such as generalization to out-of-distribution data and resilience to noise. Across evaluations on varying architectures and data sets, we find that pruned networks effectively approximate the unpruned model, however, the prune ratio at which pruned networks achieve commensurate performance varies significantly across tasks. These results call into question the extent of *genuine* overparameterization in deep learning and raise concerns about the practicability of deploying pruned networks, specifically in the context of safety-critical systems, unless they are widely evaluated beyond test accuracy to reliably predict their performance. In short, our contributions are as follows:

- A class of novel functional distance metrics for classification-based neural networks to investigate the functional similarities between pruned networks and their unpruned counterpart.

- A principled approach to quantify the amount of overparameterization in a network across multiple inference tasks via the notion of *prune potential*, i.e., the maximal prune ratio with commensurate performance.

- A unified framework to establish task-specific guidelines that help practitioners assess the effects of pruning during the design and deployment of neural networks in practice.

- A broad range of experiments for multiple data sets, architectures, and pruning methods showing that our observations hold across common pruning benchmarks and real-world scenarios.

These contributions are based on the following paper:

- Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 93–138, 2021a.

We present our results in Chapter 7.

Pruning Continuous-depth Models

Continuous deep learning architectures enable learning of flexible probabilistic models for predictive modeling as neural ODE, and for generative modeling as continuous normalizing flows (CNFs). We assess whether pruning can improve the generalization performance of neural ODEs by extracting the essential weights while removing redundant ones in order to improve the convergence properties of the training procedure. Specifically, we design a framework to decipher the internal dynamics of such continuous-depth models by pruning their network architectures. Our empirical results suggest that pruning improves generalization for neural ODEs in generative modeling. Moreover, pruning finds minimal and efficient neural ODE representations with up to 98% less parameters compared to the original network, without loss of accuracy. Finally, we show that by applying pruning we can obtain insightful information about the design of better neural ODEs.

Our contributions can be summarized as follows:

- A generic pruning framework for unstructured and structured pruning of continuous-depth neural networks, including neural ODEs and CNFs.
- An extensive experimental analysis that highlights the improved generalization performance of sparse neural ODEs in classification and generative modeling tasks on low-dimensional toy data and high-dimensional tabular data.

- A generalization of our sparsity-inducing pruning framework to generative modeling of images highlighting a loss-size trade-off with over an order-of-magnitude improvement over prior work.

- A Hessian-based analysis of the training of sparse flows that indicates an improved, i.e., flattened, loss landscape of pruned neural ODEs further corroborating our findings.

The results, which are presented in Chapter 8, are based on the following paper:

- Lucas Liebenwein, Ramin Hasani, Alexander Amini, and Daniela Rus. Sparse flows: Pruning continuous-depth models. In *Advances in Neural Information Processing Systems (Under Review; arXiv preprint arXiv:2106.12718)*, 2021b.

1.5 Outline

The premise of the thesis is to develop practical algorithms for efficient and scalable deep neural networks rooted in the theoretical foundations of coresets. The thesis is structured according to the outline shown in Figure 1-1.

In Part I, we introduce the theoretical foundations for our pruning algorithms. We study the problem of sparsifying the weights of a fully-connected network in a way that the output of the network is provably approximated in Chapter 3. We generalize our theoretical analysis and resulting compression bounds to pruning filters in convolutional neural networks in Chapter 4. We also conduct preliminary pruning experiments based on our theoretical analysis and derive novel generalization bounds from our compression bounds.

In Part II, we focus on devising practical and efficient pruning algorithms derived from our theoretical compression bounds. We introduce a novel approach to structured pruning, i.e., filter pruning, in Chapter 5 and provide experimental evidence to highlight the effectiveness of our approach. In Chapter 6, we build upon our insights into pruning to devise a global compression framework for low-rank compression that

simultaneously considers the overall compression ratio while optimizing for the per-layer compression ratio.

In Part III, we consider various applications of pruning to study its effects in greater depth. We investigate the effects of pruning on the generalization capabilities of the network to out-of-distribution data in Chapter 7. In Chapter 8, we analyze continuous-depth models and explore how pruning can help designing better-generalizing and more efficient architectures for generative modeling tasks.

In Chapter 9, we conclude the thesis by discussing our findings, providing insights into some of the crucial aspects of our work, and proposing directions for future work.

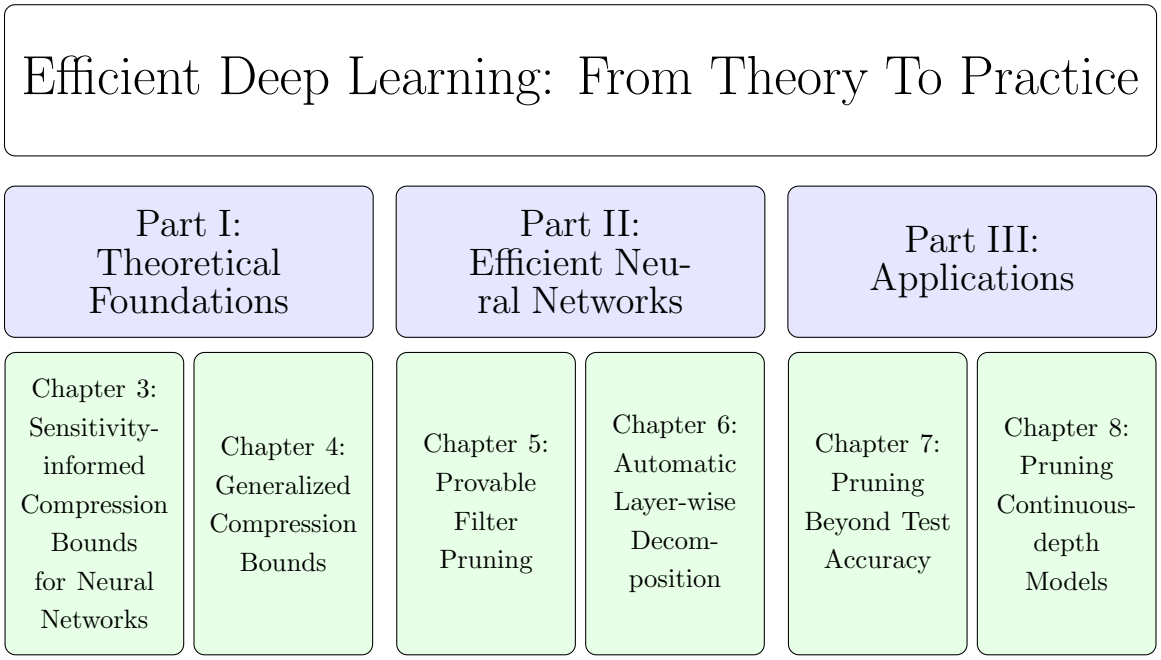


Figure 1-1: An overview of the thesis research organized around the three parts.

Chapter 2

Related Work

This thesis is inspired by the diverse and rich literature on modern machine learning and beyond. Overall, the process of designing and understanding neural networks, their training, and their architectural properties are fundamental cornerstones in modern machine learning and serve as foundation to enable the empirical success of deep learning across many machine learning disciplines and applications. Specifically, this thesis builds upon prior work in the areas of coresets, deep learning, neural network compression, pruning, and architecture design as well as applications thereof such as computer vision, natural language processing, and robotics. We also draw inspirations from the literature on specialized neural network architectures, such as continuous-time and continuous-depth models that are frequently used to model time-series data.

2.1 Coresets and Theoretical Foundations

Coreset constructions were originally introduced in the context of computational geometry (Agarwal et al., 2005) and subsequently generalized for applications to other problems via an importance sampling-based *sensitivity* framework (Braverman et al., 2016; Langberg and Schulman, 2010). Coresets have been used successfully to accelerate various machine learning algorithms such as k-means clustering (Braverman et al., 2016; Feldman and Langberg, 2011), graphical model training (Molina et al., 2018), support vector machines (Baykal et al., 2017; Tukan et al., 2020a),

and logistic regression (Huggins et al., 2016). The surveys of Bachem et al. (2017); Munteanu and Schwiegelshohn (2018) contain additional references as well. Other coreset applications include dimensionality reduction (Laparra et al., 2015), computational speed-ups (Maalouf et al., 2019), spectral clustering (Peng et al., 2015), feature selection (Gallagher et al., 2017), graph theory (Zhang and Rohe, 2018), randomized linear algebra (Cohen et al., 2015, 2017; Drineas et al., 2008; Maalouf et al., 2020), active learning (Baykal et al., 2021a), and sampling-based reachability analysis (Liebenwein et al., 2018).

Our work extends coreset constructions beyond classical machine learning application to the realm of modern, deep-learning based systems. In contrast to most prior work, we generate coresets for reducing the number of parameters rather than data points. Via a novel construction scheme based on an efficiently-computable notion of data-driven sensitivity our approach is amenable to deep learning pipelines and stands in contrast to worst-case bounds often seen in other coresets literature.

2.2 Neural Network Compression and Pruning

The need to tame the excessive storage requirements and costly inference associated with large, over-parameterized networks has led to a rich body of work in network pruning and compression dating back to some of the early work by LeCun et al. (1990). These approaches include those inspired by classical tensor decompositions (Denton et al., 2014) applicable both during and after training (Alvarez and Salzmann, 2017; Ioannou et al., 2015; Jaderberg et al., 2014; Kim et al., 2015b; Tai et al., 2015; Yu et al., 2017). In a similar vein, there are approaches based on random projections and hashing (Arora et al., 2018; Chen et al., 2015a,b; Shi et al., 2009; Ullrich et al., 2017; Weinberger et al., 2009) that compress a pre-trained network. Other methods enable inference speed-ups by embedding sparsity as an objective directly in the training process (Alvarez and Salzmann, 2017; Ioannou et al., 2015), by using quantization (Gong et al., 2014; Wu et al., 2016; Zhou et al., 2017), or by exploiting the tensor structure to induce sparsity (Cheng et al., 2015; Choromanska et al., 2016; Sindhwani et al.,

2015; Wen et al., 2016; Zhao et al., 2017).

One of the most prevalent type of neural network compression entails pruning (Han et al., 2015a), where individual weights or structures of a – usually pre-trained – network are removed, which is then often times followed by a retraining period to regain the original accuracy. In general, pruning can be categorized into two main types of pruning: (i) unstructured pruning (Frankle and Carbin, 2019; Han et al., 2015a) aims to reduce the number of non-zero parameters by inducing sparsity into weight tensors, which can achieve high compression rates but requires specialized software and/or hardware in order to achieve faster inference times; (ii) structured pruning (He et al., 2018; Li et al., 2019b) aims to modify the structure of the underlying weight tensors, which usually results in smaller compression rates but directly achievable faster inference times without requiring specialized software (Luo and Wu, 2018). A thorough overview of recent pruning approaches is, e.g., also provided by Blalock et al. (2020); Gale et al. (2019); Hoefler et al. (2021); Liu et al. (2019c); Ye et al. (2018).

Overall, one of the predominant drawbacks of these methods is that they require laborious hyperparameter tuning, lack rigorous theoretical guarantees on the size and performance of the resulting compressed network, and/or conduct compression in a data-oblivious way. Consequently, these methods do not achieve optimal parameter efficiency or require significantly more training time in order to obtain optimally-pruned networks.

We discuss pruning and compression approaches most related to our proposed contributions in more detail below.

2.2.1 Unstructured Pruning

Weight pruning (LeCun et al., 1990) techniques aim to reduce the number of weights in a layer while approximately preserving its output. Such techniques hinge upon the idea that only a few dominant weights within a layer are required to approximately preserve the output. After assigning each weight a saliency score, e.g., its magnitude (Han et al., 2015a), the parameters of the network are usually sparsified by

deterministically removing those weights below a certain score (Frankle and Carbin, 2019; Guo et al., 2016; Han et al., 2015a; LeCun et al., 1990; Renda et al., 2020). Approaches of this type also include the works of Aghasi et al. (2017); Dong et al. (2017a); Iandola et al. (2016); Lebedev and Lempitsky (2016); Lin et al. (2017), where the desired sparsity is embedded as a constraint or via a regularizer into the training pipeline, and those of Gamboa et al. (2020); Lee et al. (2019); Lin et al. (2020); Molchanov et al. (2016, 2019); Yu et al. (2018b) that really on data-informed saliency criteria to prune the network.

The research proposed in this thesis overcomes two major drawbacks of prior work: (1) our proposed research exhibits rigorous theoretical guarantees of the effect that the discarded weights can have on the compressed network, which, to the best of our knowledge, is the first work to introduce a practical weight pruning algorithm with provable guarantees; and (2) our work extends to structured pruning with favorable empirical performance, which, unlike weight pruning, does not require specialized hardware and sparse linear algebra libraries in order to speed up inference.

2.2.2 Structured Pruning via Filter Pruning

Pruning entire neurons and filters directly shrinks the network leading to smaller storage requirements and improved inference-time performance on any hardware (Li et al., 2019b; Luo and Wu, 2018). Lately, these approaches were investigated in many papers (Chen et al., 2020; Dong et al., 2017b; He et al., 2019; Kang and Han, 2020; Li et al., 2016, 2019b; Liu et al., 2019b; Ye et al., 2018, 2020). Usually, filters are pruned by assigning an importance score to each neuron/filter, either solely weight-based (He et al., 2018, 2017) or data-informed (Yu et al., 2018b), and removing those with a score below a threshold. Most weight-based approach rely on taking the norm of the filters to assign filter importance and subsequently prune unimportant filers. These methods are data-oblivious heuristics that heavily rely on the assumption that filters with large weight magnitudes are more important, which may not hold in general (Ye et al., 2018). Moreover, Liu et al. (2019c) have noted that filter pruning may not be effective when applied on a per-layer basis without global consideration of the

resulting architecture.

In general, prior work on neuron and filter pruning has focused on approaches that either lack theoretical guarantees or a principled approach to allocating the sampling budget across layers, requiring tedious ablation studies or settling for a naive uniform allocation across the layers. In contrast, our proposed methods assign saliency scores to filters based on theoretical insights gained through our error analysis with accompanying error bounds. Subsequently, we leverage our theoretical error bounds to automatically identify important layers and allocate the user-specified prune ratio across layers. We can thus leverage our methods to automatically optimize the resulting architecture.

2.2.3 Low-rank Compression

Common approaches to low-rank compression entail leveraging tensor decompositions techniques such as Tucker-decomposition (Kim et al., 2015b), CP-decomposition (Lebedev et al., 2015), Tensor-Train (Garipov et al., 2016; Novikov et al., 2015) and others (Denil et al., 2013; Ioannou et al., 2017; Jaderberg et al., 2014). Other decomposition-like approaches include weight sharing, random projections, and feature hashing (Arora et al., 2018; Chen et al., 2015a,b; Shi et al., 2009; Ullrich et al., 2017; Weinberger et al., 2009). Alternatively, low-rank compression can be performed via matrix decomposition, e.g., singular value decomposition (SVD). To this end, the tensor is flattened before applying SVD as done by Denton et al. (2014); Sainath et al. (2013); Tukan et al. (2020b); Xue et al. (2013); Yu et al. (2017) among others. Chen et al. (2018a); Denton et al. (2014); Maalouf et al. (2021) also explore the use of subspace clustering before applying low-rank compression to each cluster to improve the approximation error. Notably, most prior work relies on some form of expensive approximation algorithm – even to just solve the per-layer low-rank compression, e.g., clustering or tensor decomposition. In this thesis, we instead focus on the global compression problem and show that simple compression techniques (SVD with channel slicing) are advantageous in this context as we can use them as efficient subroutines. We note that we can even extend our algorithm to multiple, different

types of per-layer decomposition. Our insights lead to novel state-of-the-art results for low-rank compressed networks.

2.2.4 Network-aware Compression

To determine the compression ratio of each layer, prior work suggests to account for compression during training (Alvarez and Salzmann, 2017; Ioannou et al., 2015; Wen et al., 2017; Xu et al., 2020), e.g., by training the network with a penalty that encourages the weight matrices to be low-rank or sparse. Others suggest to select layer ranks using variational Bayesian matrix factorization (Kim et al., 2015b). In their recent paper, Chin et al. (2020) suggest to produce an entire set of compressed networks with different accuracy/speed trade-offs. Our work was also inspired by a recent line of work towards automatically choosing or learning the rank of each layer (Gusak et al., 2019; Idelbayev and Carreira-Perpinán, 2020; Li and Shi, 2018; Tiwari et al., 2021; Zhang et al., 2015b,c). We take such approaches further and suggest a global compression framework that incorporates multiple decomposition techniques with more than one hyper-parameter per layer (number of subspaces and ranks of each layer). This approach increases the number of local minima in theory and helps improving the performance in practice. Moreover, we consider such a framework both in the context of structured pruning and low-rank compression highlighting novel types of connections between these types of compression.

2.2.5 Retraining of Pruned Networks

Virtually any pruning approach is combined with an appropriate training and/or retraining stage to regain the accuracy of the original network. For example, a potential prune pipeline with retraining may proceed as follows:

1. Prune weights of the trained network according to some criterion of importance;
2. Retrain the resulting network to regain the full accuracy;
3. Iteratively repeat steps 1 & 2 to further reduce the size.

Moreover, pruning can be performed before (Lee et al., 2019; Tanaka et al., 2020; Wang et al., 2020), during (Kusupati et al., 2020; Peste et al., 2021; Yu et al., 2018a; Zhu and Gupta, 2017), or after training (Han et al., 2015a; Singh and Alistarh, 2020), and repeated iteratively (Renda et al., 2020) as described above. Tuning hyperparameters for both training, pruning, and retraining is also critical for obtaining good performance in practice as previously explored by Frankle and Carbin (2019); Gale et al. (2019); Renda et al. (2020) among others.

Our work also exhibits improved performance when combined with more retraining and/or iterative prune-retrain strategies. However unlike prior work, our proposed methods exhibit tight error bounds that lead to improved performance in prune-only scenarios where no retraining is performed or no data is available to retrain the network. Our proposed methods also exhibit favorable performance compared to state-of-the-art approaches in scenarios where the amount of retraining is limited.

2.3 Generalization of Neural Networks

2.3.1 Generalization

The generalization properties of neural networks have been extensively investigated in various contexts (Bartlett et al., 2017; Dziugaite and Roy, 2017; Neyshabur et al., 2017a). However, as pointed out by Neyshabur et al. (2017b), current approaches to obtaining generalization bounds do not fully or accurately capture the empirical success of state-of-the-art neural network architectures. Recently, Arora et al. (2018); Nagarajan and Kolter (2019); Zhou et al. (2018a) highlighted the close connection between compressibility and generalization of neural networks. Arora et al. (2018) presented a compression method based on the Johnson-Lindenstrauss (JL) Lemma (Johnson and Lindenstrauss, 1984) and proved generalization bounds based on succinct reparameterizations of the original neural network. Pruning is hereby viewed as a form of noise injection into the network. By quantifying the (unpruned) network's ability to withstand random noise, they characterize the prunability of the

network to establish generalization bounds.

Building upon the work of Arora et al. (2018), we extend our theoretical coreset results to establish novel generalization bounds. Unlike the method of Arora et al. (2018), which exhibits guarantees of the compressed network's performance only on the set of training points, our method's guarantees hold (probabilistically) for any random point drawn from the data distribution. In addition, we establish that our method can ε-approximate the neural network output neuron-wise, which is stronger than the norm-based guarantee of Arora et al. (2018). We also empirically investigate the generalization ability of pruned networks under changes to the input distribution. Unlike prior work, we establish a principled understanding of the performance-size trade-off in neural networks under distribution changes. Specifically, for the first time we show that as a result of pruning, the network's ability to withstand noise and other types of data corruption is diminished. In other words, the network's ability to generalize is "traded" in exchange for compactness even though the nominal test accuracy, i.e., its nominal ability to generalize, is maintained.

2.3.2 Robustness

Our work builds on and extends previous work that investigates the robustness of pruned networks. Recently, Gamboa et al. (2020); Guo et al. (2018); Wang et al. (2018); Ye et al. (2019); Zhao et al. (2018) investigated the effects of adversarial inputs on pruned networks, however, the resulting evidence is inconclusive. While Gamboa et al. (2020); Guo et al. (2018) report that adversarial robustness may improve or remain the same for pruned networks, Wang et al. (2018); Ye et al. (2019); Zhao et al. (2018) report decreased robustness for pruned networks. Concurrently, the work of Hooker et al. (2019) investigated whether certain class accuracies are more affected than others. In contrast to prior work, we investigate both the functional similarities of pruned networks and the task-specific prune potential, i.e., the maximal prune ratio possible without loss in accuracy. Based on the findings of this thesis, we highlight the need to assess pruned networks across a wide variety of tasks to safely deploy networks due to the unpredictable nature of a network's prune potential.

2.3.3 Robust Training and Pruning

Among others, Dhillon et al. (2018); Wijayanto et al. (2019) investigated pipelines that incorporate pruning and robust training to obtain simultaneously sparse and robust networks. Gui et al. (2019); Sehwag et al. (2019) use magnitude-based pruning (Han et al., 2015a) to train sparse, robust networks, while the work of Sehwag et al. (2020) incorporates the robustness objective into an optimization-based pruning procedure. The results presented in this thesis offer a complementary viewpoint to the findings of prior work in that we can indeed efficiently train pruned networks that are (adversarially) robust. However, for the first time we show that pruned networks are disproportionally more affected by distributional changes in the input regardless of the training procedure.

2.3.4 Implicit Regularization via Overparameterization

Our work also relates to the beneficial role of overparameterization in deep learning (Allen-Zhu et al., 2019; Zhang et al., 2016). Conventional wisdom (Du et al., 2019; Neyshabur et al., 2015, 2018, 2019) states that stochastic gradient methods used for training implicitly regularize the network. This in turn ensures that the network generalizes well despite the potential to severely overfit. Moreover, the findings of Belkin et al. (2019); Nakkiran et al. (2020) suggest that the implicit regularization potential increases with the parameter count. Complementary to prior work, our work thoroughly establishes that pruned networks suffer distinctly more from small shifts in the input data distribution compared to unpruned networks. This is possibly due to the decreased *implicit* regularization potential as a result of the lower parameter count. Finally, our findings highlight that *explicit* regularization in the form of robust training can help regain some of the robustness properties that would otherwise be lost.

2.4 Architecture Design and Search

The process of designing a suitable neural network architecture is a key ingredient in deep learning going back to the very early days of neural networks (Miller et al., 1989). Many break-through advancements in performance can be traced back to novel architectural designs, e.g., the advent of convolutional neural networks (Krizhevsky et al., 2012) during the ILSVRC challenge (Russakovsky et al., 2015) or the more recent advancements in attention layers (Vaswani et al., 2017) for natural language processing. These advancements usually stem from tedious, manual design iterations with very little automation mostly relying on scientific intuition rather than algorithmic and/or theoretical advancements (Liu et al., 2019a). For example, convolutional neural networks (CNNs) slowly improved over the years via numerous, small refinements including residual connections (He et al., 2016), batch normalization (Ioffe and Szegedy, 2015), fully-convolutional network architectures (Long et al., 2015), improved depth-width trade-offs (Zagoruyko and Komodakis, 2016), and separable convolutions (Chollet, 2017). And CNNs are still being improved up to this day, e.g., see the recent work of Brock et al. (2021).

Prior work has thus considered the idea of automatically optimizing the architecture design in order to minimize the human effort required, e.g., see the recent survey of Elsken et al. (2019). This process is commonly referred to as neural architecture search (NAS). NAS approaches include evolutionary algorithms (Miller et al., 1989), where the considered architectures are randomly evolved before being trained and evaluated, and reinforcement learning (Zoph and Le, 2016), where the final loss is considered as a reward signal to an agent that is being trained. Another line of work considers the idea of differentiable architecture search (Liu et al., 2019a), where relative importance scores for a set of architectural design features are learned via gradient descent. Approaches of this flavor also include proxy-less architecture search (Cai et al., 2018), where the weights and design features of the architectures are co-optimized simultaneously via gradient descent. Moreover, neural network pruning can be viewed as a light-weight form of (proxy-less) NAS, specifically in the context

of pruning layers at different prune ratios (Liu et al., 2019c).

Our work builds upon this observation by suggesting a fully-automated budget allocation procedure that allocates an optimal per-layer prune ratio. Unlike prior work, which predominantly relies on manual procedures or heuristics, our allocation procedure is rooted in our coresets-based error analysis leading to more optimally pruned network architectures. Beyond structured pruning, we develop coreset tools for decomposition-based compression methods. Hence, we obtain another degree of freedom to compress the parent architecture thus further improving upon the possible parameter efficiency-performance trade-off. We then show how pruning can lead to an enhanced understanding of neural architectures when designing continuous-depth neural networks (Chen et al., 2018b) by providing insights into the sparsity-generalization trade-off. In summary, our work attempts to shrink the gap between neural architecture search, which is fully-automated but computationally expensive, and a manual design procedure, which is computationally efficient but labor-intensive. We do so by highlighting how pruning can be leveraged into an efficient, light-weight architecture search procedure.

2.5 Continuous-depth Models

A continuous normalizing flow (CNF) (Chen et al., 2018b) efficiently (Grathwohl et al., 2019) maps a latent space to data via ordinary differential equations (ODEs), relaxing the strong constraints over *discrete* normalizing flows (Dinh et al., 2016; Durkan et al., 2019; Huang et al., 2020; Kingma and Dhariwal, 2018; Papamakarios et al., 2017; Rezende and Mohamed, 2015). CNFs enable learning flexible probabilistic models by arbitrarily-chosen neural network topologies. This is in part enabled by learning the derivative of an ODE that can encode complex dynamics. Moreover, using adaptive numerical solvers for ODEs we obtain variable-depth architectures on the fly. While recent works investigated ways to improve the efficiency of CNFs (Finlay et al., 2020; Grathwohl et al., 2019; Hasani et al., 2021a; Li et al., 2020), regularize the flows (Onken et al., 2020; Yang and Karniadakis, 2020), or solving some

of their shortcomings such as crossing trajectories (Dupont et al., 2019; Massaroli et al., 2020), less is understood about their inner dynamics during and post training.

In this thesis, we use pruning algorithms to investigate generalization properties of sparse neural ordinary differential equations (neural ODEs) and CNFs. In particular, we investigate how the inner dynamics and the modeling performance of a continuous flow vary if we methodologically prune its neural network architecture. Here, our main objective is to better understand the dynamics of CNFs in density estimation tasks as we increase the sparsity of the network. Finally, we show that pruning can improve generalization in neural ODEs.

Part I

Theoretical Foundations

Chapter 3

Sensitivity-informed Compression Bounds for Neural Networks

3.1 Overview

In this chapter, we consider the problem of sparsifying the parameters of a trained neural network in a principled way so that the output of the compressed neural network is approximately preserved. We introduce a neural network compression approach based on identifying and removing weighted edges with low relative importance via coresets, small weighted subsets of the original set that approximate the pertinent cost function. Our compression algorithm hinges on extensions of the traditional sensitivity-based coresets framework (Braverman et al., 2016; Langberg and Schulman, 2010). In this sense, the work presented in this chapter aims to provide analytical network compression bounds and highlight avenues towards practical pruning algorithms that are inspired by the theoretical guarantees. We further highlight the intrinsic connection between compression bounds and generalization bounds (Allen-Zhu et al., 2019; Arora et al., 2018, 2019; Baykal et al., 2019; Neyshabur et al., 2019; Zhou et al., 2018b).

3.1.1 Contributions

The main contributions of this chapter are as follows:

- A provable coreset approach to sparsifying neural network parameters via importance sampling based on a novel, empirical notion of sensitivity.
- Analytical results establishing guarantees on the approximation accuracy, size, and generalization of the compressed neural network.
- Evaluations on real-world data sets that demonstrate the practical effectiveness of our algorithm in compressing neural network parameters and validate our theoretical results.

3.1.2 Relevant Papers

The results presented in this chapter are based on the following joint papers with Cenk Baykal:

- Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.
- Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *SIAM Journal on Mathematics of Data Science (Under Review; arXiv preprint arXiv:1910.05422)*, 2021b.

For further context and additional results, we may also refer the interested reader to Cenk Baykal's doctoral thesis (Baykal, 2021).

3.1.3 Outline

We first define the neural network coreset problem in Section 3.2. Subsequently, we introduce the neuron-wise sparsification procedure in Section 3.3 and analyze its per-

formance in Section 3.4. The main compression theorem is presented in Section 3.5 and we present preliminary compression results on fully-connected networks in Section 3.6. We conclude the chapter with a discussion section (Section 3.7).

3.2 Problem Definition

3.2.1 Fully-connected Neural Networks

A feedforward fully-connected neural network (NN) with $L \in \mathbb{N}_+$ layers and parameters θ defines a mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$ for a given input $x \in \mathcal{X} \subseteq \mathbb{R}^d$ to an output $y \in \mathcal{Y} \subseteq \mathbb{R}^k$ as follows. Let $\eta^\ell \in \mathbb{N}_+$ denote the number of neurons in layer $\ell \in [L]$, where $[L] = \{1, \ldots, L\}$ denotes the index set, and where $\eta^0 = d$ and $\eta^L = k$ with slight abuse of notation. Further, let $\eta = \sum_{\ell=1}^{L} \eta^\ell$ and $\eta^* = \max_{\ell \in \{1,\ldots,L\}} \eta^\ell$. For layers $\ell \in \{1, \ldots, L\}$, let $W^\ell \in \mathbb{R}^{\eta^\ell \times \eta^{\ell-1}}$ be the weight matrix for layer ℓ with entries denoted by w_{ij}^ℓ, rows denoted by $w_i^\ell \in \mathbb{R}^{1 \times \eta^{\ell-1}}$, and $\theta = (W^1, \ldots, W^L)$. For notational simplicity, we assume that the bias is embedded in the weight matrix. Then for an input vector $x \in \mathbb{R}^d$, let $a^0 = x$ and $z^\ell = W^\ell a^{\ell-1} \in \mathbb{R}^{\eta^\ell}$, $\forall \ell \in \{1, \ldots, L\}$, where $a^{\ell-1} = \phi(z^{\ell-1}) \in \mathbb{R}^{\eta^{\ell-1}}$ denotes the activation. We consider the activation function to be the rectified linear unit (ReLU) function, i.e., $\phi(\cdot) = \max\{\cdot, 0\}$ (entry-wise, if the input is a vector). The output of the network for an input x is $f_\theta(x) = z^L$, and in particular, for classification tasks the prediction is $\operatorname{argmax}_{i \in [k]} f_\theta(x)_i = \operatorname{argmax}_{i \in [k]} z_i^L$. Note that sometimes we may refer to weights and neurons as edges and nodes instead.

3.2.2 Neural Network Coreset Problem

Consider the setting where a neural network $f_\theta(\cdot)$ has been trained on a training set of independent and identically distributed (i.i.d.) samples from a joint distribution on $\mathcal{X} \times \mathcal{Y}$, yielding parameters $\theta = (W^1, \ldots, W^L)$. We further denote the input points of a validation data set as $\mathcal{P} = \{x_i\}_{i=1}^n \subseteq \mathcal{X}$ and the marginal distribution over the input space $\mathcal{X}$ as $\mathcal{D}$. We define the size of the parameter tuple θ, $\operatorname{nnz}(\theta)$, to be the sum of the number of non-zero entries in the weight matrices $W^1, \ldots, W^L$.

For any given $\varepsilon, \delta \in (0,1)$, our overarching goal is to generate a reparameterization $\hat{\theta}$, yielding the neural network $f_{\hat{\theta}}(\cdot)$, using a randomized algorithm, such that the number of number of nonzero (nnz) parameters is significantly reduced, i.e., $\text{nnz}(\hat{\theta}) \ll \text{nnz}(\theta)$, and the neural network output $f_\theta(x)$, $x \sim \mathcal{D}$ can be approximated up to $1 \pm \varepsilon$ multiplicative error with probability greater than $1 - \delta$. We define the $1 \pm \varepsilon$ multiplicative error between two k-dimensional vectors $a, b \in \mathbb{R}^k$ as the following entry-wise bound: $a \in (1 \pm \varepsilon)b \Leftrightarrow a_i \in (1 \pm \varepsilon)b_i \, \forall i \in [k]$, and formalize the definition of an (ε, δ)-coreset as follows.

Definition 1 ((ε, δ)-coreset)**.** *Given user-specified $\varepsilon, \delta \in (0,1)$, a set of parameters $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ is an (ε, δ)-coreset for the network parameterized by θ if for $x \sim \mathcal{D}$, it holds that*

$$\mathop{\mathbb{P}}_{\hat{\theta}, x}\left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)\right) \geq 1 - \delta,$$

where $\mathbb{P}_{\hat{\theta},x}$ denotes a probability measure with respect to a random data point x and the output $\hat{\theta}$ generated by a randomized compression scheme.

3.3 Method

In this section, we introduce our neuron-wise sparsification procedure. Our method is based on an importance sampling scheme that extends traditional sensitivity-based coreset constructions to the application of compressing parameters in neural networks. We further highlight how our neuron-wise sparsification procedure can be used to compress entire networks.

3.3.1 Sparsifying Weights in Neurons

Our method hinges on the insight that a small subsample of the data points $\mathcal{S}$ of appropriate size (see Section 3.4 for details) can be used to gauge the relative importance of incoming weights to a neuron.

Our algorithm to sparsify a neuron's weights is presented in Algorithm 1. Note that we call the procedure separately for either the positive or negative weights in-

coming to the neuron. This separation step is required in order to obtain valid approximation guarantees as outlined in Section 3.4. The cached activations are used to compute the *sensitivity*, i.e., relative importance, of each considered weight $j \in \mathcal{W}$ of neuron $i \in [\eta^\ell]$, $\ell \in \{1, \ldots, L\}$ (Algorithm 1, Lines 1-3). The relative importance of each weight j is computed as the maximum (over $x \in \mathcal{S}$) ratio of the weight's contribution to the sum of contributions of all (positive or negative) weights. In other words, the sensitivity s_j of a weight j captures the highest (relative) impact j had on the output of neuron $i \in [\eta^\ell]$ in layer ℓ across all $x \in \mathcal{S}$.

The sensitivities are then used to compute an importance sampling distribution over the incoming weights (Lines 5-7). The intuition behind the importance sampling distribution is that if s_j is high, then the weight j is more likely to have a high impact on the output of neuron i, therefore we should keep the weight j with a higher probability. m weights are then sampled with replacement (Lines 8-9) and the sampled weights are then reweighed to ensure unbiasedness of our estimator (Lines 11-13).

3.3.2 Neural Network Sparsification

In order to sparsify all the weights in a network we can invoke Algorithm 1 separately for positive and negative weights of all neurons in the network. Given an appropriate notion for the size of $\mathcal{S}$ and the value of ε and δ we can sparsify each neuron in the network while obtaining a provably compressed network. The appropriate notion for ε, δ, and $\mathcal{S}$ will be clear from our analysis section and the corresponding network compression bounds, see Section 3.4 and 3.5, respectively.

3.4 Analysis

In this section, we establish the theoretical guarantees of our neural network compression method as outlined in Section 3.3.

Algorithm 1 SparsifyNeuron($\mathcal{W}, w, \varepsilon, \delta, \mathcal{S}, a(\cdot)$)

Input: $\mathcal{W} \subseteq [\eta^{\ell-1}]$: index set of either positive or negative weights; $w \in \mathbb{R}^{1\times\eta^{\ell-1}}$: row vector corresponding to the weights incoming to neuron $i \in [\eta^\ell]$ in layer $\ell \in \{1,\dots,L\}$; $\varepsilon, \delta \in (0,1)$: error and failure probability, respectively; $\mathcal{S} \subseteq \mathcal{P}$: subsample of the original point set; $a(\cdot)$: cached activations of previous layer for all $x \in \mathcal{S}$.
Output: $\hat{w}$: sparse weight vector.

for $j \in \mathcal{W}$ **do**
  $s_j \leftarrow \max_{x\in\mathcal{S}} \frac{w_j a_j(x)}{\sum_{k\in\mathcal{W}} w_k a_k(x)}$; ▷ Compute the sensitivity of each edge
end for
$S \leftarrow \sum_{j\in\mathcal{W}} s_j$;
for $j \in \mathcal{W}$ **do**
  $q_j \leftarrow \frac{s_j}{S}$; ▷ Generate the importance sampling distribution over the incoming edges
end for
$m \leftarrow \left\lceil \frac{8\,S\,K\log(8\,\eta/\delta)}{\varepsilon^2} \right\rceil$; ▷ Compute the number of required samples
$\mathcal{C} \leftarrow$ a multiset of m samples from $\mathcal{W}$ where each $j \in \mathcal{W}$ is sampled with probability q_j;

$\hat{w} \leftarrow (0,\dots,0) \in \mathbb{R}^{1\times\eta^{\ell-1}}$; ▷ Initialize the compressed weight vector
for $j \in \mathcal{C}$ **do**
  $\hat{w}_j \leftarrow \hat{w}_j + \frac{w_j}{m\,q_j}$; ▷ Entries are reweighted by $\frac{1}{m\,q_j}$ to ensure unbiasedness of our estimator
end for
return $\hat{w}$;

3.4.1 Preliminaries

Let $x \sim \mathcal{D}$ be a randomly drawn input point. We explicitly refer to the pre-activation and activation values at layer $\ell \in \{1,\dots,\ell\}$ with respect to the input $x \in \text{supp}(\mathcal{D})$, where $\text{supp}(\mathcal{D})$ denotes the support of the distribution $\mathcal{D}$, as $z^\ell(x)$ and $a^\ell(x)$, respectively. The values of $z^\ell(x)$ and $a^\ell(x)$ at each layer ℓ will depend on whether or not we compressed the previous layers $\ell' \in \{1,\dots,\ell\}$. To formalize this interdependency, we let $\hat{z}^\ell(x)$ and $\hat{a}^\ell(x)$ denote the respective quantities of layer ℓ when we replace the weight matrices $W^1,\dots,W^\ell$ in layers $1,\dots,\ell$ by $\hat{W}^1,\dots,\hat{W}^\ell$, respectively.

For the remainder of this section (Section 3.4) we let $\ell \in \{1,\dots,L\}$ be an any layer and let $i \in [\eta^\ell]$ be an arbitrary neuron in layer ℓ. For the sake of readability, we omit the the variable denoting the layer $\ell \in \{1,\dots,L\}$, the neuron $i \in [\eta^\ell]$, and the incoming edge index $j \in [\eta^{\ell-1}]$, whenever they are clear from the context. For example, when referring to the intermediate value of a neuron $i \in [\eta^\ell]$ in layer

$\ell \in \{1, \ldots, L\}$, $z_i^\ell(x) = \langle w_i^\ell, \hat{a}^{\ell-1}(x) \rangle \in \mathbb{R}$ with respect to a point x, we will simply write $z(x) = \langle w, a(x) \rangle \in \mathbb{R}$, where $w := w_i^\ell \in \mathbb{R}^{1 \times \eta^{\ell-1}}$ and $a(x) := a^{\ell-1}(x) \in \mathbb{R}^{\eta^{\ell-1} \times 1}$. Under this notation, the weight of an incoming edge j is denoted by $w_j \in \mathbb{R}$.

3.4.2 Empirical Sensitivity for Positive Weights

In the following, we introduce our notion of relative importance of a neuron's weight in an arbitrary layer, denoted by empirical sensitivity (ES). For ease of presentation, we will for now assume that all weights and input activations are entry-wise non-negative. We then show how we can estimate the empirical sensitivity for any data point $x \sim \mathcal{D}$ with high probability. Later on, we will derive an importance sampling bound from our notion of empirical sensitivity. We begin by defining empirical sensitivity.

Definition of empirical sensitivity

To this end, let $\mathcal{W} = \{j \in [\eta^{\ell-1}] : w_j > 0\} \subseteq [\eta^{\ell-1}]$ be the index set of incoming edges with positive weights. We quantify the relative importance of each edge as follows.

Definition 2 (Relative Importance)**.** *The importance of an incoming edge $j \in \mathcal{W}$ with respect to an input $x \in \text{supp}(\mathcal{D})$ is given by the function $g_j(x)$, where*

$$g_j(x) = \frac{w_j \, a_j(x)}{\sum_{k \in \mathcal{W}} w_k \, a_k(x)} \quad \forall j \in \mathcal{W}.$$

Note that $g_j(x)$ is a function of the random variable $x \sim \mathcal{D}$. We now present our first assumption that pertains to the cumulative distribution function (CDF) of the relative importance random variable.

Assumption 1. *There exist universal constants $K, K' > 0$ such that for all $j \in \mathcal{W}$, the CDF of the random variable $g_j(x)$ for $x \sim \mathcal{D}$, denoted by $F_j(\cdot)$, satisfies $F_j\left(M_j/K\right) \leq \exp\left(-1/K'\right)$, where $M_j = \min\{y \in [0,1] : F_j(y) = 1\}$.*

Assumption 1 is a technical assumption on the ratio of the weighted activations that will enable us to rule out pathological problem instances where the relative importance of each edge cannot be well-approximated using a small number of data

points $\mathcal{S} \subseteq \mathcal{P}$. Henceforth, we consider a uniformly drawn (without replacement) subsample $\mathcal{S} \subseteq \mathcal{P}$, where $|\mathcal{S}| = \lceil K' \log(8\,\eta\,\eta^*/\delta) \rceil$, and define the sensitivity of an edge as follows.

Definition 3 (Empirical Sensitivity)**.** *Let $\mathcal{S} \subseteq \mathcal{P}$ be a subset of distinct points from $\mathcal{P} \overset{i.i.d.}{\sim} \mathcal{D}^n$. Then, the sensitivity over positive edges $j \in \mathcal{W}$ directed to a neuron is defined as $s_j = \max_{x \in \mathcal{S}} g_j(x)$.*

Order Statistic Sampling

We now establish a couple of technical results that will quantify the accuracy of our approximations of edge importance (i.e., sensitivity).

Lemma 1. *Let $K, K' > 0$ be universal constants and let $\mathcal{D}$ be a distribution with CDF $F(\cdot)$ satisfying $F(M/K) \leq \exp(-1/K')$, where $M = \min\{x \in [0,1] : F(x) = 1\}$. Let $\mathcal{P} = \{X_1, \ldots, X_n\}$ be a set of $n = |\mathcal{P}|$ i.i.d. samples each drawn from the distribution $\mathcal{D}$. Let $X_{n+1} \sim \mathcal{D}$ be an i.i.d. sample. Then,*

$$\mathbb{P}\left(K \max_{X \in \mathcal{P}} X < X_{n+1}\right) \leq \exp(-n/K).$$

Proof. Let $X_{\max} = \max_{X \in \mathcal{P}}$; then,

$$\begin{aligned}
\mathbb{P}(K\,X_{\max} < X_{n+1}) &= \int_0^M \mathbb{P}(X_{\max} < x/K | X_{n+1} = x)\, d\,\mathbb{P}(x) \\
&= \int_0^M \mathbb{P}\left(X < x/K\right)^n \, d\,\mathbb{P}(x) && \text{since } X_1, \ldots, X_n \text{ are i.i.d.} \\
&\leq \int_0^M F(x/K)^n \, d\,\mathbb{P}(x) && \text{where } F(\cdot) \text{ is the CDF of } X \sim \mathcal{D} \\
&\leq F(M/K)^n \int_0^M d\,\mathbb{P}(x) && \text{by monotonicity of } F \\
&= F(M/K)^n \\
&\leq \exp(-n/K') && \text{CDF assumption,}
\end{aligned}$$

and this completes the proof. □

We now proceed to establish that the notion of empirical sensitivity is a good approximation for the relative importance. For this purpose, let the relative importance $\hat{g}_j(x)$ of an edge j after the previous layers have already been compressed be

$$\hat{g}_j(x) = \frac{w_j \, \hat{a}_j(x)}{\sum_{k \in \mathcal{W}} w_k \, \hat{a}_k(x)}.$$

Lemma 2 (Empirical Sensitivity Approximation)**.** *Let $\varepsilon \in (0, 1/2), \delta \in (0, 1)$, $\ell \in \{1, \ldots, L\}$, Consider a set $\mathcal{S} = \{x_1, \ldots, x_n\} \subseteq \mathcal{P}$ of size $|\mathcal{S}| \geq \lceil K' \log(8\,\eta\,\eta^*/\delta) \rceil$. Then, conditioned on the event $\mathcal{E}_{1/2}$ occurring, i.e., $\hat{a}(x) \in (1 \pm 1/2)a(x)$,*

$$\mathop{\mathbb{P}}_{x \sim \mathcal{D}} \left(\exists j \in \mathcal{W} : C\, s_j < \hat{g}_j(x) \mid \mathcal{E}_{1/2} \right) \leq \frac{\delta}{8\,\eta},$$

where $C = 3\,K$ and $\mathcal{W} \subseteq [\eta^{\ell-1}]$.

Proof. Consider an arbitrary $j \in \mathcal{W}$ and $x' \in \mathcal{S}$ corresponding to $g_j(x')$ with CDF $F_j(\cdot)$ and recall that $M = \min\{x \in [0,1] : F_j(x) = 1\}$ as in Assumption 1. Note that by Assumption 1, we have

$$F(M/K) \leq \exp(-1/K'),$$

and so the random variables $g_j(x')$ for $x' \in \mathcal{S}$ satisfy the CDF condition required by Lemma 1. Now let $\mathcal{E}$ be the event that $K\, s_j < g_j(x)$ holds. Applying Lemma 1, we obtain

$$\mathbb{P}(\mathcal{E}) = \mathbb{P}(K\, s_j < g_j(x)) = \mathbb{P}\left(K \max_{x' \in \mathcal{S}} g_j(x') < g_j(x) \right) \leq \exp(-|\mathcal{S}|/K').$$

Now let $\hat{\mathcal{E}}$ denote the event that the inequality $C s_j < \hat{g}_j(x) = \frac{w_j \, \hat{a}_j(x)}{\sum_{k \in \mathcal{W}} w_k \, \hat{a}_k(x)}$ holds and note that the right side of the inequality is defined with respect to $\hat{g}_j(x)$ and not $g_j(x)$. Observe that since we conditioned on the event $\mathcal{E}_{1/2}$, we have that $\hat{a}(x) \in (1 \pm 1/2)a(x)$.

Now assume that event $\hat{\mathcal{E}}$ holds and note that by the implication above, we have

$$C\, s_j < \hat{g}_j(x) = \frac{w_j\, \hat{a}_j(x)}{\sum_{k \in \mathcal{W}} w_k\, \hat{a}_k(x)} \leq \frac{(1 + 1/2) w_j\, a_j(x)}{(1 - 1/2) \sum_{k \in \mathcal{W}} w_k\, a_k(x)}$$
$$\leq 3 \cdot \frac{w_j\, a_j(x)}{\sum_{k \in \mathcal{W}} w_k\, a_k(x)} = 3\, g_j(x).$$

where the second inequality follows from the fact that $^{1+1/2}/_{1-1/2} \leq 3$. Moreover, since we know that $C \geq 3K$, we conclude that if event $\hat{\mathcal{E}}$ occurs, we obtain the inequality

$$3\, K\, s_j \leq 3\, g_j(x) \Leftrightarrow K\, s_j \leq g_j(x),$$

which is precisely the definition of event $\mathcal{E}$. Thus, we have shown the conditional implication $\left(\hat{\mathcal{E}} \mid \mathcal{E}_{1/2}\right) \Rightarrow \mathcal{E}$, which implies that

$$\mathbb{P}(\hat{\mathcal{E}} \mid \mathcal{E}_{1/2}) = \mathbb{P}(C\, s_j < \hat{g}_j(x) \mid \mathcal{E}_{1/2}) \leq \mathbb{P}(\mathcal{E})$$
$$\leq \exp(-|\mathcal{S}|/K').$$

Since our choice of $j \in \mathcal{W}$ was arbitrary, the bound applies for any $j \in \mathcal{W}$. Thus, we have by the union bound

$$\mathbb{P}(\exists j \in \mathcal{W} : C\, s_j < \hat{g}_j(x) \mid \mathcal{E}_{1/2}) \leq \sum_{j \in \mathcal{W}} \mathbb{P}(C\, s_j < \hat{g}_j(x) \mid \mathcal{E}_{1/2}) \leq |\mathcal{W}| \exp(-|\mathcal{S}|/K')$$
$$= \left(\frac{|\mathcal{W}|}{\eta^*}\right) \frac{\delta}{8\eta} \leq \frac{\delta}{8\eta}.$$

□

In practice, the set $\mathcal{S}$ referenced above is chosen to be a subset of the validation data.

3.4.3 Importance Sampling Bounds for Positive Weights

Under the positive weight assumption, we now establish approximation guarantees under the assumption that the weights are positive. We will leverage our notion

of empirical sensitivity to derive an importance sampling distribution over the set of weights incoming to a neuron and establish a corresponding approximation guarantee.

Positive Weight Sparsification

Our first lemma establishes a core result that relates the weighted sum with respect to the sparse row vector $\hat{w}$, $\sum_{k\in\mathcal{W}} \hat{w}_k\, \hat{a}_k(x)$, to the value of the of the weighted sum with respect to the ground-truth row vector w, $\sum_{k\in\mathcal{W}} w_k\, \hat{a}_k(x)$. We remark that there is randomness with respect to the randomly generated row vector $\hat{w}_i^\ell$, a randomly drawn input $x \sim \mathcal{D}$, and the function $\hat{a}(\cdot) = \hat{a}^{\ell-1}(\cdot)$ defined by the randomly generated matrices $\hat{W}^2, \ldots, \hat{W}^{\ell-1}$ in the previous layers. Unless otherwise stated, we will henceforth use the shorthand notation $\mathbb{P}(\cdot)$ to denote $\mathbb{P}_{\hat{w}^\ell, x}$. Moreover, for ease of presentation, we will first condition on the event $\mathcal{E}_{1/2}$ that $\hat{a}(x) \in (1 \pm 1/2)a(x)$ holds. This conditioning will simplify the preliminary analysis and will be removed in our subsequent results.

Lemma 3 (Positive-Weights Sparsification)**.** *Let $\varepsilon, \delta \in (0,1)$, and $x \sim \mathcal{D}$.* SparsifyNeuron$(\mathcal{W}, w, \varepsilon, \delta, \mathcal{S}, a(\cdot))$ *generates a row vector $\hat{w}$ such that*

$$\mathbb{P}\left(\sum_{k\in\mathcal{W}} \hat{w}_k\, \hat{a}_k(x) \notin (1\pm\varepsilon)\sum_{k\in\mathcal{W}} w_k\, \hat{a}_k(x) \;\mid\; \mathcal{E}_{1/2}\right) \leq \frac{3\delta}{8\eta}$$

where $\mathrm{nnz}(\hat{w}) \leq \left\lceil \frac{8\,S\,K\log(8\,\eta/\delta)}{\varepsilon^2} \right\rceil$, *and* $S = \sum_{j\in\mathcal{W}} s_j$.

Proof. Let $\varepsilon, \delta \in (0,1)$ be arbitrary. Moreover, let $\mathcal{C}$ be the coreset with respect to the weight indices $\mathcal{W} \subseteq [\eta^{\ell-1}]$ used to construct $\hat{w}$. Note that as in SparsifyNeuron, $\mathcal{C}$ is a multiset sampled from $\mathcal{W}$ of size $m = \left\lceil \frac{8\,S\,K\log(8\,\eta/\delta)}{\varepsilon^2} \right\rceil$, where $S = \sum_{j\in\mathcal{W}} s_j$ and $\mathcal{C}$ is sampled according to the probability distribution q defined by

$$q_j = \frac{s_j}{S} \qquad \forall j \in \mathcal{W}.$$

Let

$$\hat{z} = \sum_{k\in\mathcal{W}} \hat{w}_k\, \hat{a}_k(x)$$

be the approximate intermediate value corresponding to the sparsified matrix $\hat{w}$ and let

$$\tilde{z} = \sum_{k \in \mathcal{W}} w_k \, \hat{a}_k(x).$$

Now define $\mathcal{E}$ to be the (favorable) event that $\hat{z}$ ε-approximates $\tilde{z}$, i.e., $\hat{z} \in (1 \pm \varepsilon)\tilde{z}$, We will now show that the complement of this event, $\mathcal{E}^{\mathsf{c}}$, occurs with sufficiently small probability. Let $\mathcal{Z} \subseteq \text{supp}(\mathcal{D})$ be the set of *well-behaved* points, i.e., the points for which our empirical sensitivity approximately upper bounds the relative importance of the weight for a particular input point, defined as follows:

$$\mathcal{Z} = \{x' \in \text{supp}(\mathcal{D}) \, : \, \hat{g}_j(x') \leq C s_j \quad \forall j \in \mathcal{W}\},$$

where $C = 3\,K$. Let $\mathcal{E}_{\mathcal{Z}}$ denote the event that $x \in \mathcal{Z}$.

Conditioned on $\mathcal{E}_{\mathcal{Z}}$, event $\mathcal{E}^{\mathsf{c}}$ occurs with probability $\leq \frac{\delta}{4\eta}$: Let $x \sim \mathcal{D}$ such that $x \in \mathcal{Z}$ and let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be m samples from $\mathcal{W}$ with respect to distribution q as before. Define m random variables $T_{c_1}, \ldots, T_{c_m}$ such that for all $j \in \mathcal{C}$

$$T_j = \frac{w_j \, \hat{a}_j(x)}{m \, q_j} = \frac{S \, w_j \, \hat{a}_j(x)}{m \, s_j}. \tag{3.1}$$

For any $j \in \mathcal{C}$, we have for the conditional expectation of T_j:

$$\begin{aligned}
\mathbb{E}\left[T_j \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}\right] &= \sum_{k \in \mathcal{W}} \frac{w_k \, \hat{a}_k(x)}{m \, q_k} \cdot q_k \\
&= \sum_{k \in \mathcal{W}} \frac{w_k \, \hat{a}_k(x)}{m} \\
&= \frac{\tilde{z}}{m}.
\end{aligned}$$

Moreover, we also note that conditioning on the event $\mathcal{E}_{\mathcal{Z}}$ (i.e., the event that $x \in \mathcal{Z}$) does not affect the expectation of T_j. Let $T = \sum_{j \in \mathcal{C}} T_j = \hat{z}$ denote our approximation

and note that by linearity of expectation,

$$\mathbb{E}\left[T \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}\right] = \sum_{j \in \mathcal{C}} \mathbb{E}\left[T_j \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}\right] = \tilde{z}$$

Thus, $\hat{z} = T$ is an unbiased estimator of $\tilde{z}$; thus, we will henceforth refer to $\mathbb{E}[T]$ as simply $\tilde{z}$ for brevity.

For the remainder of the proof we will assume that $\tilde{z} > 0$, since otherwise, $\tilde{z} = 0$ if and only if $T_j = 0$ for all $j \in \mathcal{C}$ almost surely, which follows by the fact that $T_j \geq 0$ for all $j \in \mathcal{C}$ by definition of $\mathcal{W}$ and the non-negativity of the ReLU activation. Therefore, in the case that $\tilde{z} = 0$, it follows that $\mathbb{P}(|\hat{z} - \tilde{z}| > \varepsilon \tilde{z}) = \mathbb{P}(\hat{z} > 0) = \mathbb{P}(0 > 0) = 0$, which trivially yields the statement of the lemma.

We now proceed with the case where $\tilde{z} > 0$ and leverage the fact that $x \in \mathcal{Z}$[1] to establish that for all $j \in \mathcal{W}$:

$$\begin{aligned} C s_j \geq \hat{g}_j(x) = \frac{w_j \, \hat{a}_j(x)}{\sum_{k \in \mathcal{W}} w_k \, \hat{a}_k(x)} = \frac{w_j \, \hat{a}_j(x)}{\tilde{z}} \\ \Leftrightarrow \quad \frac{w_j \, \hat{a}_j(x)}{s_j} \leq C \, \tilde{z}. \end{aligned} \tag{3.2}$$

Utilizing the inequality established above, we bound the conditional variance of each T_j, $j \in \mathcal{C}$ as follows

$$\begin{aligned} \mathrm{Var}(T_j \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}) &\leq \mathbb{E}\left[(T_j)^2 \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}\right] \\ &= \sum_{k \in \mathcal{W}} \frac{(w_k \, \hat{a}_k(x))^2}{(m \, q_k)^2} \cdot q_k \\ &= \frac{S}{m^2} \sum_{k \in \mathcal{W}} \frac{(w_k \, \hat{a}_k(x))^2}{s_k} \\ &\leq \frac{S}{m^2} \left(\sum_{k \in \mathcal{W}} w_k \, \hat{a}_k(x) \right) C \, \tilde{z} \\ &= \frac{S \, C \, \tilde{z}^2}{m^2}. \end{aligned}$$

[1] Since we conditioned on the event $\mathcal{E}_{\mathcal{Z}}$.

Since T is a sum of (conditionally) independent random variables, we obtain

$$\mathrm{Var}(T \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}) = m \,\mathrm{Var}(T_j \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}) \leq \frac{S\,C\,\tilde{z}^2}{m}.$$

Now, for each $j \in \mathcal{C}$ let

$$\widetilde{T}_j = T_j - \mathbb{E}\left[T_j \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}\right] = T_j - \tilde{z} \quad \text{and} \quad \widetilde{T} = \sum_{j \in \mathcal{C}} \widetilde{T}_j.$$

Note that by the fact that we conditioned on $x \in \mathcal{Z}$ (event $\mathcal{E}_{\mathcal{Z}}$), we obtain by definition of T_j in (3.1) and the inequality (3.2):

$$T_j = \frac{S\,w_j\,\hat{a}_j(x)}{m\,s_j} \leq \frac{S\,C\,\tilde{z}}{m}. \tag{3.3}$$

We also have that $S \geq 1$ by definition. More specifically, using the fact that the maximum over a set is greater than the average and rearranging sums, we obtain

$$\begin{aligned} S &= \sum_{j \in \mathcal{W}} s_j = \sum_{j \in \mathcal{W}} \max_{x' \in \mathcal{S}} \; g_j(x') \\ &\geq \frac{1}{|\mathcal{S}|} \sum_{j \in \mathcal{W}} \sum_{x' \in \mathcal{S}} g_j(x') = \frac{1}{|\mathcal{S}|} \sum_{x' \in \mathcal{S}} \sum_{j \in \mathcal{W}} g_j(x') \\ &= \frac{1}{|\mathcal{S}|} \sum_{x' \in \mathcal{S}} 1 = 1. \end{aligned}$$

Thus, the inequality established in (3.3) with the fact that $S \geq 1$ we obtain an upper bound on the absolute value of the centered random variables:

$$|\widetilde{T}_j| = \left| T_j - \frac{\tilde{z}}{m} \right| \leq \frac{S\,C\,\tilde{z}}{m} = M, \tag{3.4}$$

which follows from the fact that:

if $T_j \geq \frac{\tilde{z}}{m}$: Then, by our bound in (3.3) and the fact that $\frac{\tilde{z}}{m} \geq 0$, it follows that

$$\left| \widetilde{T}_j \right| = T_j - \frac{\tilde{z}}{m} \leq \frac{S\,C\,\tilde{z}}{m} - \frac{\tilde{z}}{m} \leq \frac{S\,C\,\tilde{z}}{m}.$$

if $T_j < \frac{\tilde{z}}{m}$: Then, using the fact that $T_j \geq 0$ and $S \geq 1$, we obtain

$$\left|\widetilde{T}_j\right| = \frac{\tilde{z}}{m} - T_j \leq \frac{\tilde{z}}{m} \leq \frac{S\,C\,\tilde{z}}{m}.$$

Applying Bernstein's inequality to both $\widetilde{T}$ and $-\widetilde{T}$ we have by symmetry and the union bound,

$$\begin{aligned}
\mathbb{P}(\mathcal{E}^{\mathsf{c}} \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}) &= \mathbb{P}\left(|T - \tilde{z}| \geq \varepsilon \tilde{z} \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}\right) \\
&\leq 2\exp\left(-\frac{\varepsilon^2 \tilde{z}^2}{2\,\mathrm{Var}(T) + \frac{2\,\varepsilon\,\tilde{z} M}{3}}\right) \\
&\leq 2\exp\left(-\frac{\varepsilon^2 \tilde{z}^2}{\frac{2SC\,\tilde{z}^2}{m} + \frac{2S\,C\,\tilde{z}^2}{3m}}\right) \\
&= 2\exp\left(-\frac{3\,\varepsilon^2\,m}{8S\,C}\right) \\
&\leq \frac{\delta}{4\eta},
\end{aligned}$$

where the second inequality follows by our upper bounds on $\mathrm{Var}(T)$ and $\left|\widetilde{T}_j\right|$ and the fact that $\varepsilon \in (0,1)$, and the last inequality follows by our choice of $m = \left\lceil \frac{8\,S\,K\log(8\,\eta/\delta)}{\varepsilon^2} \right\rceil$. This establishes that for any $\hat{a}(\cdot)$ satisfying $x \in \mathcal{Z}$, the event $\mathcal{E}^{\mathsf{c}}$ occurs with probability at most $\frac{\delta}{4\eta}$.

Removing the conditioning on $\mathcal{E}_{\mathcal{Z}}$: We have by law of total probability

$$\begin{aligned}
\mathbb{P}(\mathcal{E} \mid \hat{a}(\cdot),\, \mathcal{E}_{1/2}) &\geq \int_{x \in \mathcal{Z}} \mathbb{P}(\mathcal{E} \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}) \mathop{\mathbb{P}}_{x \sim \mathcal{D}}(x = x \mid \hat{a}(\cdot),\, \mathcal{E}_{1/2})\, dx \\
&\geq \left(1 - \frac{\delta}{4\eta}\right) \int_{x \in \mathcal{Z}} \mathop{\mathbb{P}}_{x \sim \mathcal{D}}(x = x \mid \hat{a}(\cdot),\, \mathcal{E}_{1/2})\, dx \\
&= \left(1 - \frac{\delta}{4\eta}\right) \mathop{\mathbb{P}}_{x \sim \mathcal{D}}(\mathcal{E}_{\mathcal{Z}} \mid \hat{a}(\cdot),\, \mathcal{E}_{1/2}) \\
&\geq \left(1 - \frac{\delta}{4\eta}\right)\left(1 - \frac{\delta}{8\eta}\right) \\
&\geq 1 - \frac{3\delta}{8\eta}
\end{aligned}$$

where the second-to-last inequality follows from the fact that $\mathbb{P}(\mathcal{E}^{\mathsf{c}} \mid \mathcal{E}_{\mathcal{Z}}, \mathcal{E}_{1/2}) \leq \frac{\delta}{4\eta}$ as was established above and the last inequality follows by Lemma 2. This concludes the proof.

□

3.4.4 Importance Sampling Bounds for all Weights

We now relax the requirement that the weights are strictly positive and instead consider the following index sets that partition the weighted edges: $\mathcal{W}_+ = \{j \in [\eta^{\ell-1}] : w_j > 0\}$ and $\mathcal{W}_- = \{j \in [\eta^{\ell-1}] : w_j < 0\}$. We still assume that the incoming activations from the previous layers are positive and we discuss how to relax this assumption towards the end of the section. In order to establish sampling bounds for all weights, we thus consider sparsifying the positive and negative weights separately before re-combining the approximated outputs. In order to establish bounds on the overall approximation, we need to consider the additional error incurred from approximating the positive and negative weights separately.

Definition of Sign Complexity

To this end, we define $\Delta_i^\ell(x)$ for a point $x \sim \mathcal{D}$ and neuron $i \in [\eta^\ell]$ as

$$\Delta_i^\ell(x) = \frac{\sum_{k \in [\eta^{\ell-1}]} |w_{ik}^\ell \, a_k^{\ell-1}(x)|}{\left| \sum_{k \in [\eta^{\ell-1}]} w_{ik}^\ell \, a_k^{\ell-1}(x) \right|}.$$

Following a similar approach to Assumption 1 and the proof of Lemma 2 we can establish that we can effectively approximate $\Delta_i^\ell(x)$ for any point $x \sim \mathcal{D}$ via a finite set of samples from $\mathcal{D}$. To this end, we let the *sign complexity* of neuron i in layer ℓ be defined as

$$\hat{\Delta}^\ell = \left(\frac{1}{|\mathcal{S}|} \max_{i \in [\eta^\ell]} \sum_{x' \in \mathcal{S}} \Delta_i^\ell(x') \right) + \kappa,$$

where $\kappa = \sqrt{2\lambda_*} \left(1 + \sqrt{2\lambda_*} \log\left(8\,\eta\,\eta^*/\delta\right)\right)$. We will omit the formal proof for brevity and will refer the interested reader to the original paper (Baykal et al., 2019) for a more detailed discussion.

Neuron Approximation

We now proceed to establishing approximation bounds for individual neurons. Note that we consider establishing these bounds in the context of approximating all neurons simultaneously. This will ease the subsequent analysis of establishing network-wide approximation bounds.

To this end, for $\varepsilon \in (0,1)$ and $\ell \in \{1,\dots,L\}$, we let $\varepsilon' = \frac{\varepsilon}{2\,L}$ and define

$$\varepsilon_\ell = \frac{\varepsilon'}{\hat{\Delta}^{\ell\to}} = \frac{\varepsilon}{2\,L\prod_{k=\ell}^{L}\hat{\Delta}^{k}}.$$

To formalize the interlayer dependencies, for each $i \in [\eta^\ell]$ we let $\mathcal{E}_i^\ell$ denote the (desirable) event that

$$\hat{z}_i^\ell(x) \in (1 \pm 2\,\ell\,\varepsilon_{\ell+1})\, z_i^\ell(x)$$

holds, and let $\mathcal{E}^\ell = \cap_{i\in[\eta^\ell]}\mathcal{E}_i^\ell$ be the intersection over the events corresponding to each neuron in layer ℓ.

The following lemma establishes a key result for approximating the value of neuron given that the previous layer's neurons are approximated well, i.e., the event $\mathcal{E}_i^\ell$ holds.

Lemma 4 (Conditional Neuron Value Approximation)**.** *Let $\varepsilon,\delta \in (0,1)$, $\ell \in \{1,\dots,L\}$, $i \in [\eta^\ell]$, and $x \sim \mathcal{D}$. Invoking* SparsifyNeuron *for the positive and negative weights generates a row vector $\hat{w}_i^\ell = \hat{w}_i^{\ell+} - \hat{w}_i^{\ell-} \in \mathbb{R}^{1\times\eta^{\ell-1}}$ such that*

$$\mathbb{P}\left(\mathcal{E}_i^\ell \mid \mathcal{E}^{\ell-1}\right) = \mathbb{P}\left(\hat{z}_i^\ell(x) \in (1\pm 2\,\ell\,\varepsilon_{\ell+1})\, z_i^\ell(x) \mid \mathcal{E}^{\ell-1}\right) \geq 1 - \delta/\eta, \tag{3.5}$$

where $\varepsilon_\ell = \frac{\varepsilon'}{\hat{\Delta}^{\ell\to}}$ and $\operatorname{nnz}(\hat{w}_i^\ell) \leq \left\lceil \frac{8\,S\,K\log(8\,\eta/\delta)}{{\varepsilon_\ell}^2}\right\rceil + 1$, where $S = \sum_{j\in\mathcal{W}_+} s_j + \sum_{j\in\mathcal{W}_-} s_j$.

Proof. Let $\hat{w}_i^{\ell+}$ and $\hat{w}_i^{\ell-}$ denote the sparsified row vectors generated when SparsifyNeuron is invoked with first two arguments corresponding to $(\mathcal{W}_+, w_i^\ell)$ and $(\mathcal{W}_-, -w_i^\ell)$, respectively. We will at times omit including the variables for the neuron i and layer ℓ in the proofs for clarity of exposition, and for example, refer to $\hat{w}_i^{\ell+}$ and $\hat{w}_i^{\ell-}$ as simply $\hat{w}^+$ and $\hat{w}^-$, respectively.

Let $x \sim \mathcal{D}$ and define

$$\hat{z}^+(x) = \sum_{k \in \mathcal{W}_+} \hat{w}_k^+ \, \hat{a}_k(x) \geq 0 \qquad \text{and} \qquad \hat{z}^-(x) = \sum_{k \in \mathcal{W}_-} (-\hat{w}_k^-) \, \hat{a}_k(x) \geq 0$$

be the approximate intermediate values corresponding to the sparsified matrices $\hat{w}^+$ and $\hat{w}^-$; let

$$\tilde{z}^+(x) = \sum_{k \in \mathcal{W}_+} w_k \, \hat{a}_k(x) \geq 0 \qquad \text{and} \qquad \tilde{z}^-(x) = \sum_{k \in \mathcal{W}_-} (-w_k) \, \hat{a}_k(x) \geq 0$$

be the corresponding intermediate values with respect to the the original row vector w; and finally, let

$$z^+(x) = \sum_{k \in \mathcal{W}_+} w_k \, a_k(x) \geq 0 \qquad \text{and} \qquad z^-(x) = \sum_{k \in \mathcal{W}_-} (-w_k) \, a_k(x) \geq 0$$

be the true intermediate values corresponding to the positive and negative valued weights. Note that in this context, we have by definition

$$\begin{aligned}
\hat{z}_i^\ell(x) &= \langle \hat{w}, \hat{a}(x) \rangle = \hat{z}^+(x) - \hat{z}^-(x), \\
\tilde{z}_i^\ell(x) &= \langle w, \hat{a}(x) \rangle = \tilde{z}^+(x) - \tilde{z}^-(x), \quad \text{and} \\
z_i^\ell(x) &= \langle w, a(x) \rangle = z^+(x) - z^-(x),
\end{aligned}$$

where we used the fact that $\hat{w} = \hat{w}^+ - \hat{w}^- \in \mathbb{R}^{1 \times \eta^{\ell-1}}$.

Finally, let $\varepsilon, \delta \in (0,1)$ be arbitrary and let $\mathcal{W}_+ = \{j \in [\eta^{\ell-1}] : w_j > 0\}$ and $\mathcal{W}_- = \{j \in [\eta^{\ell-1}] : w_j < 0\}$. Let ε_ℓ be defined as before, $\varepsilon_\ell = \frac{\varepsilon'}{\hat{\Delta}^{\ell \to}}$, where $\hat{\Delta}^{\ell \to} = \prod_{k=\ell}^{L} \hat{\Delta}^k$ and $\hat{\Delta}^\ell = \left(\frac{1}{|\mathcal{S}|} \max_{i \in [\eta^\ell]} \sum_{x' \in \mathcal{S}} \Delta_i^\ell(x') \right) + \kappa$.

Observe that $w_j > 0 \; \forall j \in \mathcal{W}_+$ and similarly, for all $(-w_j) > 0 \; \forall j \in \mathcal{W}_-$. That is, each of index sets $\mathcal{W}_+$ and $\mathcal{W}_-$ corresponds to strictly positive entries in the arguments w_i^ℓ and $-w_i^\ell$, respectively passed into SparsifyNeuron. Observe that

since we conditioned on the event $\mathcal{E}^{\ell-1}$, we have

$$\begin{aligned}
2\,(\ell-2)\,\varepsilon_\ell &\leq 2\,(\ell-2)\,\frac{\varepsilon}{2\,L\prod_{k=\ell}^{L}\hat{\Delta}^k} \\
&\leq \frac{\varepsilon}{\prod_{k=\ell}^{L}\hat{\Delta}^k} \\
&\leq \frac{\varepsilon}{2^{L-\ell+1}} && \text{Since } \hat{\Delta}^k \geq 2 \quad \forall k \in \{\ell,\dots,L\} \\
&\leq \frac{\varepsilon}{2},
\end{aligned}$$

where the inequality $\hat{\Delta}^k \geq 2$ follows from the fact that

$$\begin{aligned}
\hat{\Delta}^k &= \left(\frac{1}{|\mathcal{S}|}\max_{i\in[\eta^\ell]}\sum_{x'\in\mathcal{S}}\Delta_i^\ell(x')\right)+\kappa \\
&\geq 1+\kappa && \text{Since } \Delta_i^\ell(x') \geq 1 \quad \forall x' \in \operatorname{supp}(\mathcal{D}) \text{ by definition} \\
&\geq 2.
\end{aligned}$$

we obtain that $\hat{a}(x) \in (1 \pm \varepsilon/2)a(x)$, where, as before, $\hat{a}$ and a are shorthand notations for $\hat{a}^{\ell-1} \in \mathbb{R}^{\eta^{\ell-1}\times 1}$ and $a^{\ell-1} \in \mathbb{R}^{\eta^{\ell-1}\times 1}$, respectively. This implies that $\mathcal{E}^{\ell-1} \Rightarrow \mathcal{E}_{1/2}$ and since $m = \left\lceil \frac{8\,S\,K\log(8\,\eta/\delta)}{\varepsilon^2} \right\rceil$ in Algorithm 1 we can invoke Lemma 3 with $\varepsilon = \varepsilon_\ell$ on each of the SparsifyNeuron invocations to conclude that

$$\mathbb{P}\left(\hat{z}^+(x) \notin (1\pm\varepsilon_\ell)\tilde{z}^+(x) \mid \mathcal{E}^{\ell-1}\right) \leq \mathbb{P}\left(\hat{z}^+(x) \notin (1\pm\varepsilon_\ell)\tilde{z}^+(x) \mid \mathcal{E}_{1/2}\right) \leq \frac{3\delta}{8\eta},$$

and

$$\mathbb{P}\left(\hat{z}^-(x) \notin (1\pm\varepsilon_\ell)\tilde{z}^-(x) \mid \mathcal{E}^{\ell-1}\right) \leq \frac{3\delta}{8\eta}.$$

Therefore, by the union bound, we have

$$\mathbb{P}\left(\hat{z}^+(x) \notin (1\pm\varepsilon_\ell)\tilde{z}^+(x) \text{ or } \hat{z}^-(x) \notin (1\pm\varepsilon_\ell)\tilde{z}^-(x) \mid \mathcal{E}^{\ell-1}\right) \leq \frac{3\delta}{8\eta} + \frac{3\delta}{8\eta} = \frac{3\delta}{4\eta}.$$

Moreover, we have with probability at most $\frac{\delta}{4\eta}$ that $\Delta_i^\ell(x) > \hat{\Delta}^\ell$ as formalized in our corresponding paper (Baykal et al., 2019). We omit further details for brevity. Thus, by the union bound over the failure events, we have that with probability at

least $1 - (3\delta/4\eta + \delta/4\eta) = 1 - \delta/\eta$ that **both** of the following events occur

$$1. \quad \hat{z}^+(x) \in (1 \pm \varepsilon_\ell)\tilde{z}^+(x) \;\text{ and }\; \hat{z}^-(x) \in (1 \pm \varepsilon_\ell)\tilde{z}^-(x) \tag{3.6}$$

$$2. \quad \Delta_i^\ell(x) \leq \hat{\Delta}^\ell \tag{3.7}$$

Recall that $\varepsilon' = \frac{\varepsilon}{2\,L}$, $\varepsilon_\ell = \frac{\varepsilon'}{\hat{\Delta}^{\ell\rightarrow}}$, and that event $\mathcal{E}_i^\ell$ denotes the (desirable) event that $\hat{z}_i^\ell(x)\,(1 \pm 2\,\ell\,\varepsilon_{\ell+1})\,z_i^\ell(x)$ holds, and similarly, $\mathcal{E}^\ell = \cap_{i\in[\eta^\ell]}\,\mathcal{E}_i^\ell$ denotes the vector-wise analogue where

$$\hat{z}^\ell(x)\,(1 \pm 2\,\ell\,\varepsilon_{\ell+1})\,z^\ell(x).$$

Let $k = 2\,(\ell - 1)$ and note that by conditioning on the event $\mathcal{E}^{\ell-1}$, i.e., we have by definition

$$\hat{a}^{\ell-1}(x) \in (1 \pm 2\,(\ell-1)\varepsilon_\ell)a^{\ell-1}(x) = (1 \pm k\,\varepsilon_\ell)a^{\ell-1}(x),$$

which follows by definition of the ReLU function. Recall that our overarching goal is to establish that

$$\hat{z}_i^\ell(x) \in (1 \pm 2\,\ell\varepsilon_{\ell+1})\,z_i^\ell(x),$$

which would immediately imply by definition of the ReLU function that

$$\hat{a}_i^\ell(x) \in (1 \pm 2\,\ell\varepsilon_{\ell+1})\,a_i^\ell(x).$$

Having clarified the conditioning and our objective, we will once again drop the index i from the expressions moving forward.

Proceeding from above, we have with probability at least $1 - \delta/\eta$

$$\begin{aligned}
\hat{z}(x) &= \hat{z}^+(x) - \hat{z}^-(x) \\
&\leq (1+\varepsilon_\ell)\,\tilde{z}^+(x) - (1-\varepsilon_\ell)\,\tilde{z}^-(x) && \text{By event (3.6)} \\
&\leq (1+\varepsilon_\ell)(1+k\,\varepsilon_\ell)\,z^+(x) - (1-\varepsilon_\ell)(1-k\,\varepsilon_\ell)\,z^-(x) && \text{Conditioning on } \mathcal{E}^{\ell-1}.
\end{aligned}$$

We can further simplify the above expression as

$$\begin{aligned}
\hat{z}(x) &\leq \left(1+\varepsilon_\ell(k+1)+k\varepsilon_\ell^2\right) z^+(x) + \left(-1+(k+1)\varepsilon_\ell - k\varepsilon_\ell^2\right) z^-(x) \\
&= \left(1+k\,\varepsilon_\ell^2\right) z(x) + (k+1)\,\varepsilon_\ell \left(z^+(x)+z^-(x)\right) \\
&= \left(1+k\,\varepsilon_\ell^2\right) z(x) + \frac{(k+1)\,\varepsilon'}{\prod_{k=\ell}^{L}\hat{\Delta}^k}\,\left(z^+(x)+z^-(x)\right) \\
&\leq \left(1+k\,\varepsilon_\ell^2\right) z(x) + \frac{(k+1)\,\varepsilon'}{\Delta_i^\ell(x)\,\prod_{k=\ell+1}^{L}\hat{\Delta}^k}\,\left(z^+(x)+z^-(x)\right) \qquad \text{By event (3.7)} \\
&= \left(1+k\,\varepsilon_\ell^2\right) z(x) + \frac{(k+1)\,\varepsilon'}{\prod_{k=\ell+1}^{L}\hat{\Delta}^k}\,|z(x)| \qquad \text{By } \Delta_i^\ell(x) = \frac{z^+(x)+z^-(x)}{|z(x)|} \\
&= \left(1+k\,\varepsilon_\ell^2\right) z(x) + (k+1)\,\varepsilon_{\ell+1}\,|z(x)|.
\end{aligned}$$

To upper bound the last expression above, we begin by observing that $k\varepsilon_\ell^2 \leq \varepsilon_\ell$, which follows from the fact that $\varepsilon_\ell \leq \frac{1}{2\,L} \leq \frac{1}{k}$ by definition. Moreover, we also note that $\varepsilon_\ell \leq \varepsilon_{\ell+1}$ by definition of $\hat{\Delta}^\ell \geq 1$. Now, we consider two cases.

Case of $z(x) \geq 0$: In this case, we have

$$\begin{aligned}
\hat{z}(x) &\leq \left(1+k\,\varepsilon_\ell^2\right) z(x) + (k+1)\,\varepsilon_{\ell+1}\,|z(x)| \\
&\leq (1+\varepsilon_\ell) z(x) + (k+1)\varepsilon_{\ell+1} z(x) \\
&\leq (1+\varepsilon_{\ell+1}) z(x) + (k+1)\varepsilon_{\ell+1} z(x) \\
&= (1+(k+2)\,\varepsilon_{\ell+1})\,z(x) = (1+2\,\ell\varepsilon_{\ell+1})\,z(x),
\end{aligned}$$

where the last equality follows by definition of $k = 2\,(\ell-1)$, which implies that $k+2 = 2\ell$. Thus, this establishes the desired upper bound in the case that $z(x) \geq 0$.

Case of $z(x) < 0$: Since $z(x)$ is negative, we have that

$$\begin{aligned}
\hat{z}(x) &\leq \left(1+k\,\varepsilon_\ell^2\right) z(x) + (k+1)\,\varepsilon_{\ell+1}\,|z(x)| \\
&\leq z(x) - (k+1)\varepsilon_{\ell+1} z(x) \\
&\leq (1-(k+1)\varepsilon_{\ell+1})\,z(x) \\
&\leq (1-(k+2)\varepsilon_{\ell+1})\,z(x) = (1-2\,\ell\varepsilon_{\ell+1})\,z(x),
\end{aligned}$$

and this establishes the upper bound for the case of $z(x)$ being negative.

Putting the results of the case by case analysis together, we have the upper bound of $\hat{z}(x) \leq z(x) + 2\,\ell\varepsilon_{\ell+1}|z(x)|$. The proof for establishing the lower bound for $z(x)$ is analogous to that given above, and yields $\hat{z}(x) \geq z(x) - 2\,\ell\varepsilon_{\ell+1}|z(x)|$. Putting both the upper and lower bound together, we have that with probability at least $1 - \frac{\delta}{\eta}$:

$$\hat{z}(x) \in (1 \pm 2\,\ell\varepsilon_{\ell+1})\, z(x),$$

and this completes the proof.

□

Remarks on Negative Activations

We note that up to now we assumed that the input $a(x)$, i.e., the activations from the previous layer, are strictly nonnegative. However, we can decompose the input into $a(x) = a_{\text{pos}}(x) - a_{\text{neg}}(x)$, where $a_{\text{pos}}(x) \geq 0 \in \mathbb{R}^{\eta^{\ell-1}}$ and $a_{\text{neg}}(x) \geq 0 \in \mathbb{R}^{\eta^{\ell-1}}$. Furthermore, we can define the sensitivity over the set of points $\{a_{\text{pos}}(x),\, a_{\text{neg}}(x) \mid x \in \mathcal{S}\}$ (instead of $\{a(x) \mid x \in \mathcal{S}\}$), and thus maintain the required nonnegativity of the sensitivities. Then, in the terminology of Lemma 4, we let

$$z^{+}_{\text{pos}}(x) = \sum_{k \in \mathcal{W}_{+}} w_k\, a_{\text{pos},k}(x) \geq 0 \qquad \text{and} \qquad z^{-}_{\text{neg}}(x) = \sum_{k \in \mathcal{W}_{-}} (-w_k)\, a_{\text{neg},k}(x) \geq 0$$

be the corresponding positive parts, and

$$z^{+}_{\text{neg}}(x) = \sum_{k \in \mathcal{W}_{+}} w_k\, a_{\text{neg},k}(x) \geq 0 \qquad \text{and} \qquad z^{-}_{\text{pos}}(x) = \sum_{k \in \mathcal{W}_{-}} (-w_k)\, a_{\text{pos},k}(x) \geq 0$$

be the corresponding negative parts of the preactivation of the considered layer, such that

$$z^{+}(x) = z^{+}_{\text{pos}}(x) + z^{-}_{\text{neg}}(x) \qquad \text{and} \qquad z^{-}(x) = z^{+}_{\text{neg}}(x) + z^{-}_{\text{pos}}(x).$$

We also let

$$\Delta^{\ell}_{i}(x) = \frac{z^{+}(x) + z^{-}(x)}{|z(x)|}$$

be as before, with $z^+(x)$ and $z^-(x)$ defined as above. Equipped with above definitions, we can rederive Lemma 4 analogously in the more general setting, i.e., with potentially negative activations. We also note that we require a slightly larger sample size now since we have to take a union bound over the failure probabilities of all four approximations (i.e. $\hat{z}^+_{\text{pos}}(x)$, $\hat{z}^-_{\text{neg}}(x)$, $\hat{z}^+_{\text{neg}}(x)$, and $\hat{z}^-_{\text{pos}}(x)$) to obtain the desired overall failure probability of δ/η.

3.5 Network Compression Bounds

The following core result establishes unconditional layer-wise approximation guarantees and culminates in our main compression theorem. We first establish layer-wise error guarantees before we proceed with stating the compression bounds for the entire network.

3.5.1 Layer-wise Approximation

The following corollary immediately follows from Lemma 4 and establishes a layer-wise approximation guarantee.

Corollary 5 (Conditional Layer-wise Approximation)**.** *Let $\varepsilon, \delta \in (0,1)$, $\ell \in \{1, \ldots, L\}$, and $x \sim \mathcal{D}$. Appropriately invoking* SparsifyNeuron *for each neuron in layer ℓ generates a sparse weight matrix $\hat{W}^\ell = \left(\hat{w}^\ell_1, \ldots, \hat{w}^\ell_{\eta^\ell}\right)^\top \in \mathbb{R}^{\eta^\ell \times \eta^{\ell-1}}$ such that*

$$\mathbb{P}(\mathcal{E}^\ell \mid \mathcal{E}^{\ell-1}) = \mathbb{P}\left(\hat{z}^\ell(x) \in (1 \pm 2\,\ell\,\varepsilon_{\ell+1})\, z^\ell(x) \mid \mathcal{E}^{\ell-1}\right) \geq 1 - \frac{\delta\,\eta^\ell}{\eta}, \tag{3.8}$$

where $\varepsilon_\ell = \frac{\varepsilon'}{\hat{\Delta}^{\ell\rightarrow}}$, $\hat{z}^\ell(x) = \hat{W}^\ell \hat{a}^\ell(x)$, and $z^\ell(x) = W^\ell a^\ell(x)$.

Proof. Since (3.5) established by Lemma 4 holds for any neuron $i \in [\eta^\ell]$ in layer ℓ and since $(\mathcal{E}^\ell)^\mathsf{c} = \cup_{i \in [\eta^\ell]} (\mathcal{E}^\ell_i)^\mathsf{c}$, it follows by the union bound over the failure events $(\mathcal{E}^\ell_i)^\mathsf{c}$ for all $i \in [\eta^\ell]$ that with probability at least $1 - \frac{\eta^\ell \delta}{\eta}$

$$\hat{z}^\ell(x) = \hat{W}^\ell \hat{a}^{\ell-1}(x) \in (1 \pm 2\,\ell\,\varepsilon_{\ell+1})\, W^\ell a^{\ell-1}(x) = (1 \pm 2\,\ell\,\varepsilon_{\ell+1})\, z^\ell(x).$$

□

The following lemma removes the conditioning on $\mathcal{E}^{\ell-1}$ and explicitly considers the (compounding) error incurred by generating coresets $\hat{W}^1, \ldots, \hat{W}^\ell$ for multiple layers.

Lemma 6 (Layer-wise Approximation)**.** *Let $\varepsilon, \delta \in (0,1)$, $\ell \in \{1, \ldots, L\}$, and $x \sim \mathcal{D}$. Appropriately invoking* SparsifyNeuron *for each neuron in layer ℓ generates a sparse weight matrix $\hat{W}^\ell \in \mathbb{R}^{\eta^\ell \times \eta^{\ell-1}}$ such that, for $\hat{z}^\ell(x) = \hat{W}^\ell \hat{a}^\ell(x)$,*

$$\mathop{\mathbb{P}}_{(\hat{W}^2, \ldots, \hat{W}^\ell),\, x}(\mathcal{E}^\ell) = \mathop{\mathbb{P}}_{(\hat{W}^2, \ldots, \hat{W}^\ell),\, x}\left(\hat{z}^\ell(x) \in (1 \pm 2\,\ell\,\varepsilon_{\ell+1})\, z^\ell(x)\right) \geq 1 - \frac{\delta \sum_{\ell'=2}^{\ell} \eta^{\ell'}}{\eta}.$$

Proof. Invoking Corollary 5, we know that for any layer $\ell' \in \{1, \ldots, L\}$,

$$\mathop{\mathbb{P}}_{\hat{W}^{\ell'},\, x,\, \hat{a}^{\ell'-1}(\cdot)}(\mathcal{E}^{\ell'} \mid \mathcal{E}^{\ell'-1}) \geq 1 - \frac{\delta\,\eta^{\ell'}}{\eta}. \tag{3.9}$$

We also have by the law of total probability that

$$\begin{aligned}\mathbb{P}(\mathcal{E}^{\ell'}) &= \mathbb{P}(\mathcal{E}^{\ell'} \mid \mathcal{E}^{\ell'-1})\,\mathbb{P}(\mathcal{E}^{\ell'-1}) + \mathbb{P}(\mathcal{E}^{\ell'} \mid (\mathcal{E}^{\ell'-1})^{\mathsf{c}})\,\mathbb{P}((\mathcal{E}^{\ell'-1})^{\mathsf{c}}) \\ &\geq \mathbb{P}(\mathcal{E}^{\ell'} \mid \mathcal{E}^{\ell'-1})\,\mathbb{P}(\mathcal{E}^{\ell'-1}) \end{aligned} \tag{3.10}$$

Repeated applications of (3.9) and (3.10) in conjunction with the observation that $\mathbb{P}(\mathcal{E}^1) = 1$[2] yield

$$\begin{aligned}\mathbb{P}(\mathcal{E}^\ell) &\geq \mathbb{P}(\mathcal{E}^{\ell'} \mid \mathcal{E}^{\ell'-1})\,\mathbb{P}(\mathcal{E}^{\ell'-1}) \\ &\;\;\vdots && \text{Repeated applications of (3.10)} \\ &\geq \prod_{\ell'=1}^{\ell} \mathbb{P}(\mathcal{E}^{\ell'} \mid \mathcal{E}^{\ell'-1}) \\ &\geq \prod_{\ell'=1}^{\ell} \left(1 - \frac{\delta\,\eta^{\ell'}}{\eta}\right) && \text{By (3.9)} \\ &\geq 1 - \frac{\delta}{\eta} \sum_{\ell'=1}^{\ell} \eta^{\ell'} && \text{By the Weierstrass Product Inequality,}\end{aligned}$$

[2]Since we do not compress the input layer.

where the last inequality follows by the Weierstrass Product Inequality[3] and this establishes the lemma. □

3.5.2 Network Compression

Having established the layer-wise approximation guarantee of our sampling scheme, we present our main theorem for neural network compression that establishes the desired approximation guarantee of our algorithm. The proof is based on appropriately invoking Lemma 6 to establish approximation guarantees for the entire network.

Theorem 7 (Network Compression). *For $\varepsilon, \delta \in (0,1)$, invoking* SparsifyNeuron *for each neuron in the network generates a set of parameters $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ of size*

$$\text{nnz}(\hat{\theta}) \leq \sum_{\ell=1}^{L} \sum_{i=1}^{\eta^\ell} \left(\left\lceil \frac{32\, L^2\, (\hat{\Delta}^{\ell\rightarrow})^2\, S_i^\ell\, K\, \log(8\,\eta/\delta)}{\varepsilon^2} \right\rceil + 1 \right)$$

in $\mathcal{O}\left(\eta\ \eta^*\ \log\left(\eta\,\eta^*/\delta\right)\right)$ time such that $\mathbb{P}_{\hat{\theta},\, x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)\right) \geq 1 - \delta$.

Proof. Invoking Lemma 6 with $\ell = L$, we have that for $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$,

$$\begin{aligned}
\mathop{\mathbb{P}}_{\hat{\theta},\, x}\left(f_{\hat{\theta}}(x) \in 2\, L\, \varepsilon_{L+1} f_\theta(x)\right) &= \mathop{\mathbb{P}}_{\hat{\theta},\, x}\left(\hat{z}^L(x) \in 2\, L\, \varepsilon_{L+1} z^L(x)\right) \\
&= \mathbb{P}(\mathcal{E}^L) \\
&\geq 1 - \frac{\delta \sum_{\ell'=1}^{L} \eta^{\ell'}}{\eta} \\
&= 1 - \delta,
\end{aligned}$$

[3]The Weierstrass Product Inequality (Doerr, 2018) states that for $p_1, \ldots, p_n \in [0,1]$,

$$\prod_{i=1}^{n} (1 - p_i) \geq 1 - \sum_{i=1}^{n} p_i.$$

where the last equality follows by definition of $\eta = \sum_{\ell=1}^{L} \eta^{\ell}$. Note that by definition,

$$\begin{aligned}\varepsilon_{L+1} &= \frac{\varepsilon}{2\,L \prod_{k=L+1}^{L} \hat{\Delta}^{k}} \\ &= \frac{\varepsilon}{2\,L},\end{aligned}$$

where the last equality follows by the fact that the empty product $\prod_{k=L+1}^{L} \hat{\Delta}^{k}$ is equal to 1. Thus, we have $2\,L\varepsilon_{L+1} = \varepsilon$, and so we conclude

$$\mathop{\mathbb{P}}_{\hat{\theta},\,x} \left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_{\theta}(x)\right) \geq 1 - \delta,$$

which, along with the sampling complexity of Algorithm 1 (Line 8), establishes the approximation guarantee provided by the theorem.

For the computational time complexity, we observe that the most time consuming operation is the per-neuron weight sparsification procedure. The asymptotic time complexity of each SparsifyNeuron invocation for each neuron $i \in [\eta^{\ell}]$ in layers $\ell \in \{1, \ldots, L\}$ is dominated by the relative importance computation for incoming edges (Algorithm 1, Lines 1-3). This can be done by evaluating $w_{ik}^{\ell} a_{k}^{\ell-1}(x)$ for all $k \in \mathcal{W}$ and $x \in \mathcal{S}$, for a total computation time that is bounded above by $\mathcal{O}\left(|\mathcal{S}|\,\eta^{\ell-1}\right)$ since $|\mathcal{W}| \leq \eta^{\ell-1}$ for each $i \in [\eta^{\ell}]$. Thus, SparsifyNeuron takes $\mathcal{O}\left(|\mathcal{S}|\,\eta^{\ell-1}\right)$ time. Summing the computation time over all layers and neurons in each layer, we obtain an asymptotic time complexity of $\mathcal{O}\big(\,|\mathcal{S}|\,\sum_{\ell=2}^{L} \eta^{\ell-1}\eta^{\ell}\big) \subseteq \mathcal{O}\left(|\mathcal{S}|\,\eta^{*}\,\eta\right)$. Since $|\mathcal{S}| \in \mathcal{O}(\log(\eta\,\eta^{*}/\delta))$, we conclude that the computational complexity our neural network compression algorithm is

$$\mathcal{O}\left(\eta\;\eta^{*}\;\log\left(\eta\,\eta^{*}/\delta\right)\right). \tag{3.11}$$

□

We note that we can obtain a guarantee for a set of n randomly drawn points by invoking Theorem 7 with $\delta' = \delta/n$ and union-bounding over the failure probabilities, while only increasing the sampling complexity logarithmically.

3.5.3 Generalization Bounds

As a corollary to our main results, we obtain novel generalization bounds for neural networks in terms of empirical sensitivity. Following the terminology of Arora et al. (2018), the expected margin loss of a classifier $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ parameterized by θ with respect to a desired margin $\gamma > 0$ and distribution $\mathcal{D}$ is defined by $L_\gamma(f_\theta) = \mathbb{P}_{(x,y)\sim\mathcal{D}_{\mathcal{X},\mathcal{Y}}}\left(f_\theta(x)_y \leq \gamma + \max_{i\neq y} f_\theta(x)_i\right)$. We let $\hat{L}_\gamma$ denote the empirical estimate of the margin loss. The following corollary follows directly from the argument presented by Arora et al. (2018) and Theorem 7.

Corollary 8 (Generalization Bounds)**.** *For any $\delta \in (0,1)$ and margin $\gamma > 0$, sparsifying the network in context of Theorem 7 generates weights $\hat{\theta}$ such that with probability at least $1-\delta$, the expected error $L_0(f_{\hat{\theta}})$ with respect to the points in $\mathcal{P} \subseteq \mathcal{X}$, $|\mathcal{P}| = n$, is bounded by*

$$L_0(f_{\hat{\theta}}) \leq \hat{L}_\gamma(f_\theta) + \widetilde{\mathcal{O}}\left(\sqrt{\frac{\max_{x\in\mathcal{P}} \|f_\theta(x)\|_2^2 \; L^2 \sum_{\ell=1}^{L} (\hat{\Delta}^{\ell\to})^2 \sum_{i=1}^{\eta^\ell} S_i^\ell}{\gamma^2 \, n}}\right).$$

3.6 Results

In this section, we evaluate the practical effectiveness of our compression algorithm on popular small-scale benchmark data sets and varying fully-connected trained neural network configurations in various prune-only scenarios. The purpose of these experiments is to highlight the practical effectiveness of our method to sparsify a given neural network when comparing against other benchmarks that provide approximation guarantees.

3.6.1 Experimental Setup

Our algorithms. We test three variations of our compression algorithm: (i) sole neuron sparsification (CoreNet), (ii) sparsification with neuron pruning (CoreNet+), and (iii) sparsification with neuron pruning and amplification (CoreNet++). For CoreNet+, we note that we remove neurons whenever the maximum activation of a

neuron is equal to 0 over all evaluations on $\mathcal{S}$. For CoreNet++, we run amplification on CoreNet+ as is common for randomized algorithms by constructing multiple approximations and keeping the approximation that achieves lowest error on a separate hold-out dataset from the validation set. More details and additional analysis may be found in the corresponding paper (Baykal et al., 2019).

Networks and datasets. We use *MNIST* (LeCun et al., 1998), *FashionMNIST* (Xiao et al., 2017), and *CIFAR-10* (Torralba et al., 2008) and varying fully-connected trained neural network configurations: 2 to 5 hidden layers, 100 to 1000 hidden units, either fixed hidden sizes or decreasing hidden size denoted by *pyramid* in the figures.

Baselines. We compare the effectiveness of our sampling scheme in reducing the number of non-zero parameters of a network, i.e., in sparsifying the weight matrices, to that of uniform sampling, singular value decomposition (SVD), and current state-of-the-art sampling schemes for matrix sparsification (Achlioptas et al., 2013; Drineas and Zouzias, 2011; Kundu and Drineas, 2014), which are based on matrix norms – ℓ_1 and ℓ_2 (Frobenius).

Setup All algorithms were implemented in Python using the PyTorch library (Paszke et al., 2017) and simulations were conducted on a computer with a 2.60 GHz Intel i9-7980XE processor (18 cores total) and 128 GB RAM. Training was performed for 30 epochs on the normalized data sets using an Adam optimizer with a learning rate of 0.001 and a batch size of 300. The test accuracies were roughly 98% (MNIST), 45% (CIFAR-10), and 96% (FashionMNIST), depending on the network architecture. To account for the randomness in the training procedure, for each data set and neural network configuration, we averaged our results across 4 trained neural networks. For comparison, we evaluated the average relative error in output (ℓ_1-norm) and average drop in classification accuracy relative to the accuracy of the uncompressed network. Both metrics were evaluated on a previously unseen test set.

3.6.2 Results

Results for varying architectures and datasets are depicted in Figures 3-1 and 3-2 for the average drop in classification accuracy and relative error (ℓ_1-norm), respectively. As apparent from Figure 3-1, we are able to compress networks to about 15% of their original size without significant loss of accuracy for networks trained on *MNIST* and *FashionMNIST*, and to about 50% of their original size for *CIFAR-10*.

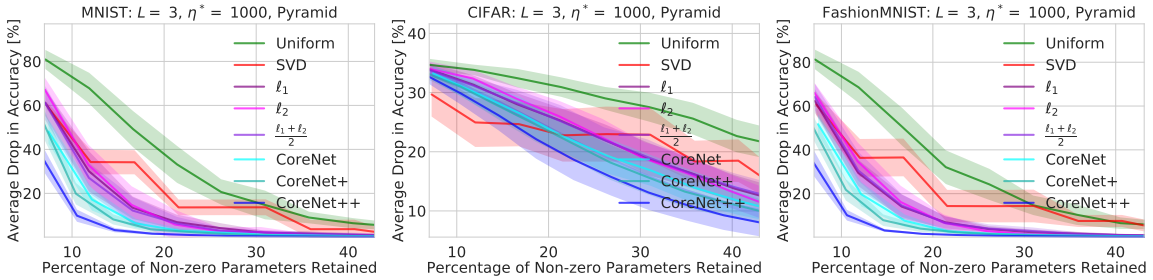


Figure 3-1: Evaluation of drop in classification accuracy after compression against the *MNIST*, *CIFAR-10*, and *FashionMNIST* datasets with varying number of hidden layers (L) and number of neurons per hidden layer (η^*). Shaded region corresponds to values within one standard deviation of the mean.

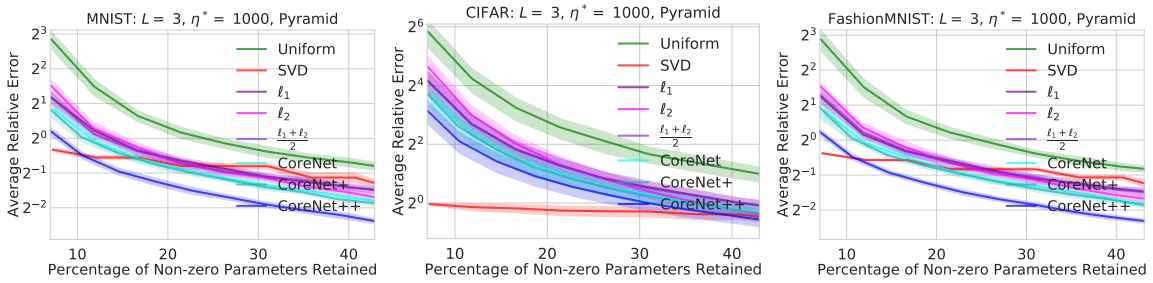


Figure 3-2: Evaluation of relative error after compression against the *MNIST*, *CIFAR-10*, and *FashionMNIST* datasets with varying number of hidden layers (L) and number of neurons per hidden layer (η^*).

The simulation results presented in this section validate our theoretical results established in Section 3.4. In particular, our empirical results indicate that we are able to outperform networks compressed via competing methods in matrix sparsification across the considered trials. The results presented in this section further suggest that empirical sensitivity can effectively capture the relative importance of neural network parameters, leading to a more informed importance sampling scheme.

3.7 Discussion

We present a coreset-based neural network compression algorithm for compressing the parameters of a trained, fully-connected neural network in a manner that approximately preserves the network's output. Our method and analysis extend traditional coreset constructions to the application of compressing parameters, which may be of independent interest. Our work distinguishes itself from prior approaches in that it establishes theoretical guarantees on the approximation accuracy and size of the generated compressed network. As a corollary to our analysis, we obtain generalization bounds for neural networks, which may provide novel insights on the generalization properties of neural networks. We empirically demonstrate the practical effectiveness of our compression algorithm on a variety of fully-connected neural network configurations and real-world data sets.

In subsequent chapters, we aim to extend our theoretical guarantees to other types of networks, namely convolutional neural networks (CNNs), and sparsification procedures, i.e., structured pruning. By building upon the theoretical tools and analysis techniques developed in this chapter, we aim to provide a simultaneously practical and theoretically-grounded approach to neural network pruning.

Chapter 4

Generalized Compression Bounds

4.1 Overview

In the previous chapter, we introduced network compression bounds in the context of weight sparsification for a network solely consisting of fully-connected layers. We will now highlight how we can generalize our theory of network compression to other types of layers and other types of sparsification. Specifically, in this chapter we generalize our compression bounds to convolutional layers when pruning entire filter and channel groups instead of individual weights from the network. Unlike weight pruning, filter pruning (or structured pruning) directly shrinks the underlying network architecture by reducing the width of each layer. This has important practical implications as well in terms how well we can leverage our network pruning algorithms to obtain efficient neural networks in practice (see Part II as well). We also note that neuron pruning in the context of fully-connected layers can be viewed as a special case of filter pruning, hence our guarantees hold regardless of the type of layer. The crux of our generalized theory lies in measuring the empirical sensitivity (ES) of the feature maps (or channels) generated by each of the filters in the previous layer, which will in turn enable us to quantify the importance of filters across layers.

4.1.1 Contributions

The main contributions of this chapter are as follows:

- An extension of our provable coreset approach in the context of filter and neuron pruning based on a generalized notion of empirical sensitivity.
- Analytical results establishing guarantees on the trade-off between approximation accuracy and size of the pruned neural network.
- An improved pruning framework via a mixture approach of sampling-based and deterministic pruning and accompanying analysis highlighting the improved sample efficiency.

4.1.2 Relevant Papers

The results presented in this chapter are based on the following paper:

- Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

4.1.3 Outline

We begin by formally re-introducing the neural network compression problem in the context of filter pruning as shown in Section 4.2. Subsequently, we discuss the pruning procedure in Section 4.3 and analyze its performance in Section 4.4. We discuss some implications of our theory in Section 4.5. We note that the purpose of this chapter is to derive the bounds of Chapter 3 in the context of filter pruning. For practical implications and empirical results, we refer the reader to Part II of this thesis.

4.2 Problem Definition

The set of parameters θ of a convolutional neural network (CNN) with L convolutional layers is a tuple of 4-dimensional weight matrices corresponding to each layer, i.e.,

$\theta = (W^1, \ldots, W^L)$, where W^ℓ denotes the 4-dimensional tensor in layer $\ell \in [L]$, W^ℓ_j filter $j \in [\eta^\ell]$, and η^ℓ the number of filters in layer ℓ. Moreover, let $W^{\ell+1}_{:j}$ be channel j of tensor $W^{\ell+1}$ that corresponds to filter W^ℓ_j. The set of parameters θ defines the mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$ from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. We consider the setting where a neural network $f_\theta(\cdot)$ has been trained on a training set of independent and identically distributed (i.i.d.) samples from a joint distribution defined on $\mathcal{X} \times \mathcal{Y}$, yielding parameters θ. We let $\mathcal{D}$ denote the marginal distribution over the input space $\mathcal{X}$ and define the size of the parameter tuple θ, $\mathrm{nnz}(\theta)$, to be the number of nonzero (nnz) entries in the weight tensors $W^1, \ldots, W^L$.

For an input $x \in \mathcal{X}$ to the network, we let $z^\ell(x)$ and $a^\ell(x) = \phi(z^\ell(x))$ denote the pre-activation and activation of layer ℓ, where ϕ is the activation function (applied entry-wise). The j^{th} feature map of layer ℓ is given by $a^\ell_j(x) = \phi(z^\ell_j(x))$. For a given input $x \in \mathcal{X}$, the output of the neural network with parameters θ is given by $f_\theta(x)$.

For any given $\varepsilon, \delta \in (0, 1)$, our overarching goal is to use a randomized algorithm to generate a small reparameterization $\hat{\theta}$ of θ such that $\mathrm{nnz}(\hat{\theta}) \ll \mathrm{nnz}(\theta)$ and for $x \sim \mathcal{D}$ the reference network output $f_\theta(x)$ can be approximated by $f_{\hat{\theta}}(x)$ up to $1 \pm \varepsilon$ entry-wise[1] multiplicative error with probability greater than $1 - \delta$.

Definition 4 ((ε, δ)-coreset)**.** *For $\varepsilon, \delta \in (0, 1)$, and a set of parameters $\theta = (W^1, \ldots, W^L)$, $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ such that $\mathrm{nnz}(\hat{\theta}) \ll \mathrm{nnz}(\theta)$, is an (ε, δ)-coreset for the original set of parameters θ if*

$$\mathop{\mathbb{P}}_{\hat{\theta}, x}\left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)\right) \geq 1 - \delta,$$

where $\mathbb{P}_{\hat{\theta},x}$ denotes the probability measure with respect to a random data point $x \sim \mathcal{D}$ and the output $\hat{\theta}$ generated by a randomized compression scheme.

4.3 Method

Algorithm 2 is the full algorithm for pruning features, i.e., neurons in fully-connected layers and channels in convolutional layers. Note that when pruning features/channels

[1]For two tensors T_1, T_2 of same dimensions, $T_1 \in (1 \pm \varepsilon)T_2$ denotes follow entry-wise bound: for each scalar entry t_1 in tensor T_1, $t_1 \in (1 \pm \varepsilon)t_2$, where t_2 is the corresponding entry in T_2.

in one layer we can also prune the corresponding neurons/filters in the previous layer. The pseudocode is organized for clarity of exposition rather than computational efficiency. Recall that θ is the full parameter set of the net, where $W^\ell \in \mathbb{R}^{\eta^\ell \times \eta^{\ell+1}}$ is the weight matrix between layers $\ell - 1$ and and ℓ. W_k^ℓ refers to the k^{th} neuron/filter of W^ℓ.

Algorithm 2 PRUNECHANNELS$(\theta, \ell, \mathcal{S}, \varepsilon, \delta)$

Input: θ: trained net; $\ell \in [L]$: layer; $\mathcal{S} \subset \text{supp}(\mathcal{D})$: sample of inputs; $\varepsilon \in (0,1)$: accuracy; $\delta \in (0,1)$: failure probability

Output: $\hat{W}^\ell$: filter-reduced weight tensor for layer ℓ; $\hat{W}^{\ell+1}$: channel reduced, weight tensor for layer $\ell + 1$

for $j \in [\eta^\ell]$ **do**
  for $i \in [\eta^{\ell+1}]$ and $\mathbf{x} \in \mathcal{S}$ **do**
    $I^+ \leftarrow \{j \in [\eta^\ell] : w_{ij}^{\ell+1} a_j^\ell(\mathbf{x}) \geq 0\}$
    $I^- \leftarrow [\eta^\ell] \setminus I^+$
    $g_{ij}^{\ell+1}(\mathbf{x}) \leftarrow \max_{I \in \{I^+, I^-\}} \frac{w_{ij}^{\ell+1} a_j^\ell(\mathbf{x})}{\sum_{k \in I} w_{ik}^{\ell+1} a_k^\ell(\mathbf{x})}$
  end for
  $s_j^\ell \leftarrow \max_{\mathbf{x} \in \mathcal{S}} \max_{i \in [\eta^{\ell+1}]} g_{ij}^{\ell+1}(\mathbf{x})$
end for
$S^\ell \leftarrow \sum_{j \in [\eta^\ell]} s_j^\ell$
for $j \in [\eta^\ell]$ **do**
  $p_j^\ell \leftarrow s_j^\ell / S^\ell$
end for
$K \leftarrow$ value from Assumption 2
$m \leftarrow \lceil (6 + 2\varepsilon)\, S^\ell\, K\, \log(2\,\eta^{\ell+1}/\delta) \varepsilon^{-2} \rceil$
$\mathcal{H} \leftarrow$ distribution on $[\eta^\ell]$ assigning probability p_j^ℓ to index j
$\hat{W}^\ell \leftarrow (0, \ldots, 0)$ ▷ same dimensions as W^ℓ
$\hat{W}^{\ell+1} \leftarrow (0, \ldots, 0)$ ▷ same dimensions as $W^{\ell+1}$
for $k \in [m]$ **do**
  $c(k) \leftarrow$ random draw from $\mathcal{H}$
  $\hat{W}_{c(k)}^\ell \leftarrow W_{c(k)}^\ell$ ▷ no reweighing or considering multiplicity of drawing index $c(k)$ multiple times
  $\hat{W}_{:c(k)}^{\ell+1} \leftarrow \hat{W}_{:c(k)}^{\ell+1} + \frac{W_{:c(k)}^{\ell+1}}{m p_{c(k)}}$ ▷ reweighing for unbiasedness of pre-activation in layer $\ell + 1$
end for
return $\hat{W}^\ell = [\hat{W}_1^\ell, \ldots, \hat{W}_{\eta^\ell}^\ell]$; $\hat{W}^{\ell+1} = [\hat{W}_{:1}^{\ell+1}, \ldots, \hat{W}_{:\eta^\ell}^{\ell+1}]$

4.4 Analysis

For notational simplicity, we will derive our theoretical results for linear layers, i.e., neuron pruning. We remind the reader that this result also applies to convolutional neural networks by taking channels of a weight tensor in place of neurons.

4.4.1 Channel Sparsification

Recall that $z_i^{\ell+1}(\mathbf{x})$ denotes the pre-activation of the i^{th} neuron in layer $\ell+1$ given input $\mathbf{x}$, and the activation $a_j^{\ell}(x) = \max\{0, z_j^{\ell}(\mathbf{x})\}$.

Definition 5 (Edge Sensitivity (Baykal et al., 2019))**.** *Fixing a layer $\ell \in [L]$, let $w_{ij}^{\ell+1}$ be the weight of edge $(j,i) \in [\eta^{\ell}] \times [\eta^{\ell+1}]$. The empirical sensitivity of weight entry $w_{ij}^{\ell+1}$ with respect to input $\mathbf{x} \in \mathcal{X}$ is defined to be*

$$g_{ij}^{\ell+1}(\mathbf{x}) = \max_{I \in \{I^+, I^-\}} \frac{w_{ij}^{\ell+1} a_j^{\ell}(\mathbf{x})}{\sum_{k \in I} w_{ik}^{\ell+1} a_k^{\ell}(\mathbf{x})}, \tag{4.1}$$

where $I^+ = \{j \in [\eta^{\ell}] : w_{ij}^{\ell+1} a_j^{\ell}(\mathbf{x}) \geq 0\}$ and $I^- = [\eta^{\ell}] \setminus I^+$ denote the set of positive and negative edges, respectively.

Algorithm 2 uses empirical sensitivity to compute the sensitivity of neurons on Lines 9-12.

Definition 6 (Neuron Sensitivity)**.** *The sensitivity of a neuron $j \in [\eta^{\ell}]$ in layer ℓ is defined as*

$$s_j^{\ell} = \max_{\mathbf{x} \in \mathcal{S}} \max_{i \in [\eta^{\ell+1}]} g_{ij}^{\ell+1}(\mathbf{x}) \tag{4.2}$$

In this section, we prove that Algorithm 2 yields a good approximation of the original net. We make the following assumption on the cumulative distribution function (CDF) of the input to ensure that the input distribution is not pathological.

Assumption 2. *There exist universal constants $K, K' > 0$ such that for any layer ℓ and all $j \in [\eta^{\ell}]$, the CDF of the random variable $\max_{i \in [\eta^{\ell+1}]} g_{ij}^{\ell+1}(x)$ for $x \sim \mathcal{D}$, denoted by $F_j(\cdot)$, satisfies $F_j\left(M_j/K\right) \leq \exp\left(-1/K'\right)$, where $M_j = \min\{y \in [0,1] : F_j(y) = 1\}$.*

Note that the analysis is carried out for the positive and negative elements of $W^{\ell+1}$ separately, which is also considered in the definition of sensitivity (Definition 5). For ease of exposition, we will thus assume that throughout the section $W^{\ell+1} \geq 0$ (element-wise), i.e., $I^+ = [\eta^\ell]$, and derive the results for this case. However, we note that we could equivalently assume $W^{\ell+1} \leq 0$ and the analysis would hold regardless. By considering both the positive and negative parts of $W^{\ell+1}$ in Definition 5 we can carry out the analysis for weight tensors with positive and negative elements.

Theorem 9. *Let $\varepsilon, \delta \in (0,1), \ell \in [L]$, and let $\mathcal{S}$ be a set of $\Theta(\log(\eta_*/\delta))$ i.i.d. samples drawn from $\mathcal{D}$. Then, $\hat{W}^{\ell+1}$ contains at most $\mathcal{O}(S^\ell \log(\eta_*/\delta)\varepsilon^{-2}))$ channels and for $x \sim \mathcal{D}$, with probability at least $1-\delta$, we have $\hat{z}^{\ell+1} \in (1 \pm \varepsilon) z^{\ell+1}$ (entry-wise), where $\eta_* = \max_{\ell \in [L]} \eta^\ell$.*

The remainder of this section builds towards proving Theorem 9. We begin by fixing a layer $\ell \in [L]$ and neuron $i \in [\eta^{\ell+1}]$. Consider the random variables $\{Y_k\}_{k \in [m]}$ where $Y_k(\mathbf{x}) = \frac{1}{m p_j} w_{ij}^{\ell+1} a_j^\ell(\mathbf{x})$ where Algorithm 2 selected index $j \in [\eta^\ell]$ on the k^{th} iteration of Line 19. Note that $z_i^{\ell+1}(\mathbf{x}) = \sum_{j \in [\eta^\ell]} w_{ij}^{\ell+1} a_j^\ell(\mathbf{x})$ and so we may also write $g_{ij}^{\ell+1}(\mathbf{x}) = w_{ij}^{\ell+1} a_j^\ell(\mathbf{x}) / z_i^{\ell+1}(\mathbf{x})$ when it is more convenient.

Lemma 10. *For each $\mathbf{x} \in \mathcal{X}$ and $k \in [m]$, $\mathbb{E}[Y_k(\mathbf{x})] = z_i^{\ell+1}(\mathbf{x})/m$.*

Proof. Y_j is drawn from distribution $\mathcal{H}$ defined on Line 15, so we compute the expectation directly.

$$\mathbb{E}[Y_j(\mathbf{x})] = \sum_{k \in [\eta^\ell]} \frac{w_{ik}^{\ell+1} a_k^\ell(\mathbf{x})}{m\, p_k} \cdot p_k = \frac{1}{m} \sum_{k \in [\eta^\ell]} w_{ik}^{\ell+1} a_k^\ell(\mathbf{x}) = \frac{z_i^{\ell+1}(\mathbf{x})}{m}$$

□

To bound the variance, we use the approach introduced in Chapter 3, where the main idea is to use the notion of empirical sensitivity to establish that a particular useful inequality holds with high probability over the randomness of the input point $x \sim \mathcal{D}$. Given that the inequality holds we can establish favorable bounds on the variance and magnitude of the random variables, which lead to a low sampling complexity.

For a random input point $\mathbf{x} \sim \mathcal{D}$, let $\mathcal{G}$ denote the event that the following inequality holds (for all neurons): $\max_{i \in [\eta^{\ell+1}]} g_{ij}^{\ell+1}(\mathbf{x}) \leq C\, s_j \quad \forall j \in [\eta^\ell]$, where $C = \max\{3K, 1\}$ and K is defined as in Assumption 2.

We now prove that under Assumption 2, event $\mathcal{G}$ occurs with high probability. From now on, to ease notation, we will drop certain superscripts/subscripts with the meaning is clear. For example, $z(\mathbf{x})$ will refer to $z_i^{\ell+1}(\mathbf{x})$.

Lemma 11. *If Assumption 2 holds, $\mathbb{P}(\mathcal{G}) > 1 - \delta/2\eta^\ell$. Here the probability is over the randomness of drawing $\mathbf{x} \sim \mathcal{D}$.*

Proof. Since $\max_{i \in [\eta^{\ell+1}]} g_{ij}(x)$ is a function of $x \sim \mathcal{D}$, for any $j \in [\eta^\ell]$ we can let D be a distribution over $\max_{i \in [\eta^{\ell+1}]} g_{ij}(x)$ and observe that since $s_j = \max_{\mathbf{x} \in \mathcal{S}} \max_{i \in [\eta^{\ell+1}]} g_{ij}(\mathbf{x})$, the negation of event $\mathcal{G}$ for a single neuron $j \in [\eta^\ell]$ can be expressed as the event

$$X > C \max_{k \in [|\mathcal{S}|]} X_k,$$

where $X \sim D$ and $X_1, \ldots, X_{|\mathcal{S}|} \overset{i.i.d.}{\sim} D$ since the points in $\mathcal{S}$ were drawn i.i.d. from $\mathcal{D}$. Invoking Lemma 8 from Baykal et al. (2019) in conjunction with Assumption 2, we obtain for any arbitrary j

$$\mathbb{P}(\max_{i \in [\eta^{\ell+1}]} g_{ij}(x) > C\, s_j) = \mathbb{P}(X > C \max_{k \in [|\mathcal{S}|]} X_k) \leq \exp(-|\mathcal{S}|/K')$$

with the K' from Assumption 2. Since our choice of neuron j was arbitrary, the inequality above holds for all neurons, therefore we can apply the union bound to obtain:

$$\begin{aligned}
\mathbb{P}_{x \sim \mathcal{D}}(\mathcal{G}) &= 1 - \mathbb{P}(\exists j \in [\eta^\ell] : \max_{i \in [\eta^{\ell+1}]} g_{ij}(x) > C\, s_j) \\
&\geq 1 - \sum_{j \in [\eta^\ell]} \mathbb{P}(\max_{i \in [\eta^{\ell+1}]} g_{ij}(x) > C\, s_j) \\
&\geq 1 - \eta^\ell \exp(-|\mathcal{S}|/K') \\
&\geq 1 - \frac{\delta}{2\eta^{\ell+1}}
\end{aligned}$$

where the last line follows from the fact that $|\mathcal{S}| \geq \left\lceil K' \log\left(2\,\eta^{\ell}\eta^{\ell+1}/\delta\right)\right\rceil$. □

Lemma 12. *For any* $\mathbf{x}$ *such that event* $\mathcal{G}$ *occurs, then* $|Y_k(\mathbf{x}) - \mathbb{E}\left[Y_k(\mathbf{x})\right]| \leq CSz/m$. *Here the expectation is over the randomness of Algorithm 2.*

Proof. Recall that $S = \sum_{j\in[\eta^{\ell}]} s_j$. Let neuron $j \in [\eta^{\ell}]$ be selected on iteration k of Line 19. For any $k \in [m]$ we have:

$$Y_k(\mathbf{x}) = \frac{w_{ij}a_j(\mathbf{x})}{mp_j} = S\,\frac{w_{ij}a_j(\mathbf{x})}{m\,s_j} \leq C\,S\,\frac{w_{ij}a_j(\mathbf{x})}{m\,\max_{i'}\,g_{i'j}(\mathbf{x})} \leq C\,S\,\frac{w_{ij}a_j(\mathbf{x})}{m\,g_{ij}(\mathbf{x})} = \frac{C\,S\,z}{m},$$

where the first inequality follows by the inequality of event $\mathcal{G}$, the second by the fact that $\max_{i'} g_{i'j}(\mathbf{x}) \geq g_{ij}(\mathbf{x})$ for any i, and the third equality by definition of $g_{ij}(\mathbf{x}) = w_{ij}a_j(\mathbf{x})/z(\mathbf{x})$. This implies that $|Y_k - \mathbb{E}\left[Y_k\right]| = \left|Y_k - \frac{z}{m}\right| \in [-z/m, CSz/m]$ by Lemma 10 and since $Y_k \geq 0$. The result follows since $C, S \geq 1$. □

Lemma 13. *For any* $\mathbf{x}$ *such that event* $\mathcal{G}$ *occurs, then* $Var(Y_k(\mathbf{x})) \leq CSz^2/m^2$. *Here the expectation is over the randomness of Algorithm 2.*

Proof. We can use the same inequality obtained by conditioning on $\mathcal{G}$ to bound the variance of our estimator.

$$\begin{aligned}
\mathrm{Var}(Y_k(\mathbf{x})) &= \mathbb{E}\left[Y_k^2(\mathbf{x})\right] - \left(\mathbb{E}\left[Y_k(\mathbf{x})\right]\right)^2 \\
&\leq \mathbb{E}\left[Y_k^2(\mathbf{x})\right] \\
&= \sum_{j\in[\eta^{\ell}]} \left(\frac{w_{ij}a_j(\mathbf{x})}{m\,p_j}\right)^2 \cdot p_j && \text{by definition of } Y_k \\
&= \frac{S}{m^2} \sum_{j\in[\eta^{\ell}]} \frac{(w_{ij}a_j(\mathbf{x}))^2}{s_j} && \text{since } p_j = s_j/S \\
&\leq \frac{CS}{m^2} \sum_{j\in[\eta^{\ell}]} \frac{(w_{ij}a_j(\mathbf{x}))^2}{\max_{i'\in[\eta^{\ell+1}]}\,g_{i'j}(\mathbf{x})} && \text{by occurrence of event } \mathcal{G} \\
&\leq \frac{CS\,z}{m^2} \sum_{j\in[\eta^{\ell}]} w_{ij}a_j(\mathbf{x}) && \text{since } g_{ij}(\mathbf{x}) = w_{ij}a_j(\mathbf{x})/z(\mathbf{x}) \\
&= \frac{CSz^2}{m^2}.
\end{aligned}$$

□

We are now ready to prove Theorem 9.

Proof of Theorem 9. Recall the form of Bernstein's inequality that, given random variables $X_1, \ldots, X_m$ such that for each $k \in [m]$ we have $\mathbb{E}[X_k] = 0$ and $|X_k| \leq M$ almost surely, then

$$\mathbb{P}\left(\sum_{k\in[m]} X_k \geq t\right) \leq \exp\left(\frac{-t^2/2}{\sum_{k\in[m]} \mathbb{E}[X_k^2] + Mt/3}\right)$$

We apply this with $X_k = Y_k - \frac{z}{m}$. We must take the probability with respect to the randomness of both drawing $\mathbf{x} \sim \mathcal{D}$ and Algorithm 2. By Lemma 10, $E[X_k] = 0$. Let us assume that event $\mathcal{G}$ occurs. By Lemma 12, we may set $M = CSz/m$. By Lemma 13, $\sum_{k\in[m]} \mathbb{E}[X_k^2] \leq CSz^2/m$. We will apply the inequality with $t = \varepsilon z$.

Observe that $\sum_{k\in[m]} X_k = \hat{z} - z$. Plugging in these values, and taking both tails of the inequality, we obtain:

$$\begin{aligned}
\mathbb{P}(|\hat{z} - z| \geq \varepsilon z \,:\, \mathcal{G}) &\leq 2\exp\left(\frac{-\varepsilon^2 z^2/2}{CSz^2/m + CS\varepsilon z^2/3m}\right) \\
&= 2\exp\left(-\frac{\varepsilon^2 m}{SK(6+2\varepsilon)}\right) && \text{since } C \leq 3K \\
&\leq \frac{\delta}{2\eta^{\ell+1}} && \text{by definition of } m
\end{aligned}$$

Removing dependence on event $\mathcal{G}$, we write:

$$\begin{aligned}
\mathbb{P}(|\hat{z} - z| \geq \varepsilon z) \geq \mathbb{P}(|\hat{z} - z| \geq \varepsilon z \,:\, \mathcal{G})\,\mathbb{P}(\mathcal{G}) &\geq \left(1 - \frac{\delta}{2\eta^{\ell+1}}\right)\left(1 - \frac{\delta}{2\eta^{\ell+1}}\right) \\
&\geq 1 - \frac{\delta}{\eta^{\ell+1}}
\end{aligned}$$

where we have applied Lemma 11. This implies the result for any single neuron, and the theorem follows by application of the union bound over all $\eta^{\ell+1}$ neurons in layer ℓ. □

4.4.2 Main Compression Theorem

Having established layer-wise approximation guarantees as in Section 4.4.1, all that remains to establish guarantees on the output of the entire network is to carefully propagate the error through the layers as was shown in Chapter 3. For each $i \in [\eta^{\ell+1}]$ and $\ell \in [L]$, define

$$\tilde{\Delta}_i^\ell(x) = \left(z_i^+(x) + z_i^-(x)\right) / |z_i(x)|,$$

where $z_i^+(x) = \sum_{k \in I^+} w_{ik}^{\ell+1} a_k^\ell(\mathbf{x})$ and $z_i^-(x) = \sum_{k \in I^-} w_{ik}^{\ell+1} a_k^\ell(\mathbf{x})$ are positive and negative components of $z_i^{\ell+1}(x)$, respectively, with I^+ and I^- as in Algorithm 2. For each $\ell \in [L]$, let Δ^ℓ be a constant defined as a function of the input distribution $\mathcal{D}$ [2], such that with high probability over $x \sim \mathcal{D}$, $\Delta^\ell \geq \max_{i \in [\eta^{\ell+1}]} \Delta_i^\ell$. Finally, let $\Delta^{\ell\rightarrow} = \prod_{k=\ell}^{L} \Delta^k$.

Generalizing Theorem 9 to obtain a layer-wise bound and applying error propagation bounds of Chapter 3, we establish our main compression theorem below.

Theorem 14. *Let $\varepsilon, \delta \in (0,1)$ be arbitrary, let $\mathcal{S} \subset \mathcal{X}$ denote the set of $\lceil K' \log(4\eta/\delta) \rceil$ i.i.d. points drawn from $\mathcal{D}$, and suppose we are given a network with parameters $\theta = (W^1, \ldots, W^L)$. Consider the set of parameters $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ generated by pruning channels of θ according to Algorithm 2 for each $\ell \in [L]$. Then, $\hat{\theta}$ satisfies $\mathbb{P}_{\hat{\theta},\, x \sim \mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)\right) \geq 1 - \delta$, and the number of filters in $\hat{\theta}$ is bounded by $\mathcal{O}\left(\sum_{\ell=1}^{L} \frac{L^2 (\Delta^{\ell\rightarrow})^2 S^\ell \log(\eta/\delta)}{\varepsilon^2}\right)$.*

4.4.3 Extension to Filters

To extend our algorithm to convolutional neural networks, we need to consider the fact that there is implicit weight sharing involved by definition of the filters. Intuitively speaking, to measure the importance of a feature map (i.e. neuron) in the case of a fully-connected network we consider the maximum impact it has on the preactivation $z^{\ell+1}(x)$. In the case of convolutions the same intuition holds, that is we want to

[2] If $\Delta_i(x)$ is a sub-Exponential random variable (Vershynin, 2016) with parameter $\lambda = O(1)$, then for δ failure probability: $\Delta^\ell \mathcal{O}(\mathbb{E}_{x \sim \mathcal{D}}[\max_i \Delta_i(x)] + \log(1/\delta))$ (Baykal et al., 2019; Vershynin, 2016)

capture the maximum contribution of a feature map $a_j^\ell(x)$, which is now a two-dimensional image instead of a scalar neuron, to the pre-activation $z^{\ell+1}(x)$ in layer $\ell + 1$. Thus, to adapt our algorithm to prune channels, we modify the definition of sensitivity slightly, by also taking the maximum over the patches $p \in \mathcal{P}$ (i.e., sliding windows created by convolutions). In this context, each activation $a_j^\ell(x)$ is also associated with a patch $p \in \mathcal{P}$, which we denote by a_{jp}^ℓ. In particular, the slight change is the following:

$$s_j^\ell = \max_{x \in \mathcal{S}} \max_{i \in [\eta^{\ell+1}]} \max_{p \in \mathcal{P}} \frac{\langle w_{ij}^{\ell+1}, a_{jp}^\ell(x) \rangle}{\sum_{k \in [\eta^\ell]} \langle w_{ik}^{\ell+1}, a_{kp}^\ell(x) \rangle},$$

where $a_{\cdot p}$ corresponds to the activation window associated with patch $p \in \mathcal{P}$. Also note that now $w_{ij}^{\ell+1}$ denotes the kernel of filter i for channel j and we thus have to take a dot product with the corresponding patch. Everything else remains the same and the proofs are analogous.

4.4.4 Boosting Sampling via Deterministic Choices

Importance sampling schemes, such as the one described above, are powerful tools with numerous applications in Big Data settings, ranging from sparsifying matrices (Achlioptas et al., 2013; Baykal et al., 2019; Drineas and Zouzias, 2011; Kundu and Drineas, 2014; Tropp et al., 2015) to constructing coresets for machine learning problems (Bachem et al., 2017; Braverman et al., 2016; Feldman and Langberg, 2011). However, by the nature of the exponential decay in probability associated with importance sampling schemes, sampling schemes perform truly well when the sampling pool and the number of samples is sufficiently large (Tropp et al., 2015). However, under certain conditions on the sampling distribution, the size of the sampling pool, and the size of the desired sample m, it has been observed that deterministically picking the m samples corresponding to the highest m probabilities may yield an estimator that incurs lower error (McCurdy, 2018; Papailiopoulos et al., 2014).

To this end, consider a hybrid scheme that picks k indices deterministically (without reweighing) and samples m' indices. More formally, let $\mathcal{C}_{\text{det}} \subseteq [n]$ be the set of

k unique indices (corresponding to weights) that are picked deterministically, and define

$$\hat{z}_{\text{det}} = \sum_{j \in \mathcal{C}_{\text{det}}} w_{ij} a_j,$$

where we note that the weights are not reweighed. Now let $\mathcal{C}_{\text{rand}}$ be a set of m' indices sampled from the remaining indices i.e., sampled from $[n] \setminus \mathcal{C}_{\text{det}}$, with probability distribution $q = (q_1, \ldots, q_n)$. To define the distribution q, recall that the original distribution p is defined to be $p_i = s_i/S$ for each $i \in [n]$. Now, q is simply the normalized distribution resulting from setting the probabilities associated with indices in $\mathcal{C}_{\text{det}}$ to be 0, i.e.,

$$q_i = \begin{cases} \frac{s_i}{S - S_k} & \text{if } i \notin \mathcal{C}_{\text{det}}, \\ 0 & \text{otherwise} \end{cases},$$

where $S_k = \sum_{j \in \mathcal{C}_{\text{det}}} s_j$ is the sum of sensitivities of the entries that were deterministically picked.

Instead of doing a combinatorial search over all $\binom{n}{k}$ choices for the deterministic set $\mathcal{C}_{\text{det}}$, for computational efficiency, we found that setting $\mathcal{C}_{\text{det}}$ to be the indices with the top k sensitivities was the most likely set to satisfy the condition above.

We state the general theorem below.

Theorem 15. *It is better to keep k feature maps, $\mathcal{C}_{\text{det}} \subseteq [\eta^\ell]$, $|\mathcal{C}_{\text{det}}| = k$, deterministically and sample $m' = \left\lceil (6 + 2\varepsilon)\,(S^\ell - S_k^\ell)\,K\,\log(8\eta_*/\delta)\varepsilon^{-2} \right\rceil$ features from $[\eta^\ell] \setminus \mathcal{C}_{\text{det}}$ if*

$$\sum_{j \notin \mathcal{C}_{\text{det}}} \left(1 - \frac{s_j}{S - S_k}\right)^{m'} > \sum_{j=1}^{\eta^\ell} \left(1 - \frac{s_j}{S}\right)^{m} + \sqrt{\frac{\log(2/\delta)(m + m')}{2}},$$

where $m = \left\lceil (6 + 2\varepsilon)\,S^\ell\,K\,\log(4\eta_*/\delta)\varepsilon^{-2} \right\rceil$, $S_k = \sum_{j \in \mathcal{C}_{\text{det}}} s_j$ and $\eta_* = \max_\ell \eta^\ell$.

Proof. Let $m \geq \left\lceil (6 + 2\varepsilon)\,S^\ell\,K\,\log(4\eta_*/\delta)\varepsilon^{-2} \right\rceil$ as in Theorem 9 and note that from Theorem 9, we know that if $\hat{z}$ is our approximation with respect to sampled set of indices, $\mathcal{C}$, we have

$$\mathbb{P}(\mathcal{E}) \leq \delta$$

where $\mathcal{E}$ is the event that the inequality

$$\left|\hat{z}_i^{\ell+1}(x) - z_i^{\ell+1}(x)\right| \leq \varepsilon z_i^{\ell+1}(x) \quad \forall i \in [\eta^{\ell+1}]$$

holds. Henceforth, we will let $i \in [\eta^{\ell+1}]$ be an arbitrary neuron and, similar to before, consider the problem of approximating the neuron's value $z_i^{\ell+1}(x)$ (subsequently denoted by z) by our approximating $\hat{z}_i^{\ell+1}(x)$ (subsequently denoted by $\hat{z}$).

Similar to our previous analysis of our importance sampling scheme, we let $\mathcal{C}_{\text{rand}} = \{c_1, \ldots, c_{m'}\}$ denote the multiset of m' neuron indices that are sampled with respect to distribution q and for each $j \in [m']$ define $Y_j = \hat{w}_{ic_j} a_{c_j}$ and let $Y = \sum_{j \in [m']} Y_j$. For clarity of exposition, we define $\hat{z}_{\text{rand}} = Y$ be our approximation with respect to the random sampling procedure, i.e.,

$$\hat{z}_{\text{rand}} = \sum_{j \in \mathcal{C}_{\text{rand}}} \hat{w}_{ij} a_j = Y.$$

Thus, our estimator under this scheme is given by $\hat{z}' = \hat{z}_{\text{det}} + \hat{z}_{\text{rand}}$

Now we want to analyze the sampling complexity of our new estimator $\hat{z}'$ so that $\mathbb{P}(|\hat{z}' - z| \geq \varepsilon z) \leq \delta/2$.

Establishing the sampling complexity for sampling with respect to distribution q is almost identical to the proof of Theorem 9. First, note that $\mathbb{E}\left[\hat{z}' \mid \mathbf{x}\right] = \hat{z}_{\text{det}} + \mathbb{E}\left[\hat{z}_{\text{rand}} \mid \mathbf{x}\right]$ since $\hat{z}_{\text{det}}$ is a constant (conditioned on a realization $\mathbf{x}$ of $x \sim \mathcal{D}$). Now note that for any $j \in [m']$

$$\begin{aligned}
\mathbb{E}\left[Y_j \mid \mathbf{x}\right] &= \sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} \hat{w}_{ik} a_k \cdot q_k \\
&= \frac{1}{m'} \sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} w_{ik} a_k \\
&= \frac{z - \hat{z}_{\text{det}}}{m'},
\end{aligned}$$

and so $\mathbb{E}\left[\hat{z}_{\text{rand}} \mid \mathbf{x}\right] = \mathbb{E}\left[Y \mid \mathbf{x}\right] = z - \hat{z}_{\text{det}}$.

This implies that $\mathbb{E}\left[\hat{z}'\right] = \hat{z}_{\text{det}} + (z - \hat{z}_{\text{det}}) = z$, and so our estimator remains

unbiased. This also yields

$$\begin{aligned}|Y - \mathbb{E}[Y \mid \mathbf{x}]| &= |\hat{z}_{\text{rand}} - \mathbb{E}[\hat{z}_{\text{rand}}]| = |\hat{z}_{\text{rand}} + \hat{z}_{\text{det}} - z| \\ &= |\hat{z}' - z|,\end{aligned}$$

which implies that all we have to do to bound the failure probability of the event $|z' - z| \geq \varepsilon z$ is to apply Bernstein's inequality to our estimator $\hat{z}_{\text{rand}} = Y$, just as we had done in the proof of Theorem 9. The only minor change is the variance and magnitude of the random variables Y_k for $k \in [m']$ since the distribution is now with respect to q and not p. Proceeding as in the proof of Lemma 12, we have

$$\begin{aligned}\hat{w}_{ij} a_j(\mathbf{x}) = \frac{w_{ij} a_j(\mathbf{x})}{m' \, q_j} &= (S - S_k) \, \frac{w_{ij} a_j(\mathbf{x})}{m' \, s_j} \\ &\leq \frac{(S - S_k) C \, z}{m'}.\end{aligned}$$

Now, to bound the magnitude of the random variables note that

$$\mathbb{E}[Y_j \mid \mathbf{x}] = \frac{z - \hat{z}_{\text{det}}}{m'} = \frac{1}{m'} \sum_{j \notin \mathcal{C}_{\text{det}}} w_{ij} a_j \leq \frac{(S - S_k) C \, z}{m'}.$$

The result above combined with this fact yields for the magnitude of the random variables

$$R' = \max_{j \in [m']} |Y_j - \mathbb{E}[Y_j \mid \mathbf{x}]| \leq \frac{(S - S_k) C \, z}{m'},$$

where we observe that the only relative difference to the bound of Lemma 12 is the term $S - S_k$ appears, where $S_k = \sum_{j \in \mathcal{C}_{\text{det}}} s_j$, instead of S[3]

[3]and of course the sampling complexity is m' instead of m

Similarly, for the variance of a single Y_j

$$\begin{aligned}\operatorname{Var}(Y_j \mid \mathbf{x}, \mathcal{G}) &\leq \sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} \frac{(w_{ik} a_k(\mathbf{x}))^2}{m'^2 \, q_k} \\ &= \frac{S - S_k}{m'^2} \sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} \frac{(w_{ik} a_k(\mathbf{x}))^2}{s_k} \\ &\leq \frac{C(S - S_k)\, z}{m'^2} \sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} w_{ik} a_k(\mathbf{x}) \\ &\leq \frac{C(S - S_k) z^2 \, \min\{1, C(S - S_k)\}}{m'^2},\end{aligned}$$

where the last inequality follows by the fact that $\sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} w_{ik} a_k(\mathbf{x}) \leq z$ and by the sensitivity inequality from the proof of Lemma 13

$$\sum_{k \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} w_{ik} a_k(\mathbf{x}) \leq Cz \sum_{j \in [\eta^\ell] \setminus \mathcal{C}_{\text{det}}} s_j = Cz(S - S_k).$$

This implies by Bernstein's inequality and the argument in proof of Theorem 9 that if we sample

$$m' = \left\lceil (6 + 2\varepsilon)\,(S^\ell - S_k^\ell)\, K \, \log(8\eta_*/\delta)\varepsilon^{-2} \right\rceil$$

times from the distribution q, then we have

$$\mathbb{P}(|\hat{z}' - z| \geq \varepsilon z) \leq \delta/2.$$

Now let $p = (p_1, \ldots, p_n)$ be the probability distribution and let $\mathcal{C}$ denote the multi-set of indices sampled from $[n]$ when m samples are taken from $[n]$ with respect to distribution p. For each index $j \in [n]$ let $U_j(m, p) = \mathbb{1}[j \in \mathcal{C}]$ be the indicator random variable of the event that index j is sampled at least once and let $U(m, p) = \sum_{i=j}^{n} U_j(m, p)$. Note that U is a random variable that denotes the number of unique samples that result from the sampling process described above, and its expectation

is given by

$$\begin{aligned}\mathbb{E}\left[U(m,p)\right] &= \sum_{j=1}^{n} \mathbb{E}\left[U_j(m,p)\right] = \sum_{j=1}^{n} \mathbb{P}(i \in \mathcal{C}) \\ &= \sum_{j=1}^{n} \mathbb{P}(j \text{ is sampled at least once}) \\ &= \sum_{j=1}^{n} \left(1 - \mathbb{P}(j \text{ is not sampled})\right) \\ &= n - \sum_{j=1}^{n} (1-p_j)^m.\end{aligned}$$

Now we want to establish the condition for which $U(m',q) < U(m,p)$, which, if it holds, would imply that the number of distinct weights that we retain with the deterministic + sampling approach is lower and still achieves the same error and failure probability guarantees, making it the overall better approach. To apply a strong concentration inequality, let $\mathcal{C}' = \mathcal{C}_{\text{det}} \cup \mathcal{C}_{\text{rand}} = \{c'_1, \ldots, c'_k, c'_{k+1}, \ldots, c'_{m'}\}$ denote the set of indices sampled from the deterministic + sampling (with distribution q) approach, and let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be the indices of the random samples obtained by sampling from distribution p. Let $f(c'_1, \ldots, c'_{m'}, c_1, \ldots, c_m)$ denote the difference $U(m',q) - U(m,p)$ in the number of unique samples in $\mathcal{C}'$ and $\mathcal{C}$. Note that f satisfies the bounded difference inequality with Lipschitz constant 1 since changing the index of any single sample in $\mathcal{C} \cup \mathcal{C}'$ can change f by at most 1. Moreover, there are $m' + m$ random variables, thus, applying McDiarmid's inequality (van Handel, 2014), we obtain

$$\mathbb{P}(\mathbb{E}\left[U(m,p) - U(m',q)\right] - (U(m,p) - U(m',q)) \geq t) \leq \exp\left(-\frac{-2t^2}{(m+m')}\right),$$

this implies that for $t = \sqrt{\frac{\log(2/\delta)(m+m')}{2}}$,

$$\mathbb{E}\left[U(m,p) - U(m',q)\right] \leq U(m,p) - U(m',q) + t$$

with probability at least $1 - \delta/2$. Thus, this means that if $E[U(m,p)] - \mathbb{E}\left[U(m',q)\right] > t$,

then $U(m,p) > U(m',q)$.

More specifically, recall that

$$\mathbb{E}\left[U(m,p)\right] = n - \sum_{j=1}^{n}(1-p_j)^m = n - \sum_{j=1}^{n}\left(1-\frac{s_j}{S}\right)^m$$

and

$$\begin{aligned}
\mathbb{E}\left[U(m',q)\right] &= k + \sum_{j:q_j>0}\left(1-(1-q_j)^{m'}\right) \\
&= k + (n-k) - \sum_{j:q_j>0}(1-q_j)^{m'} \\
&= n - \sum_{j:q_j>0}(1-q_j)^{m'} \\
&= n - \sum_{j\notin\mathcal{C}_{\text{det}}}\left(1-\frac{s_j}{S-S_k}\right)^{m'}
\end{aligned}$$

Thus, rearranging terms, we conclude that it is better to conduct the deterministic + sampling scheme if

$$\sum_{j\notin\mathcal{C}_{\text{det}}}\left(1-\frac{s_j}{S-S_k}\right)^{m'} > \sum_{j=1}^{n}\left(1-\frac{s_j}{S}\right)^{m} + \sqrt{\frac{\log(2/\delta)(m+m')}{2}}.$$

Putting it all together, and conditioning on the above inequality holding, we have by the union bound

$$\mathbb{P}\left(|\hat{z}'-z| \geq \varepsilon z \cup U(m',q) > U(m,p)\right) \leq \delta,$$

this implies that with probability at least $1-\delta$: (i) $\hat{z}' \in (1\pm\varepsilon)z$ and (ii) $U(m',q) < U(m,p)$, implying that the deterministic + sampling approach ensures the error guarantee holds with a smaller number of unique samples, leading to better compression. □

4.5 Discussion

In this chapter, we highlight that our theoretical techniques developed in Chapter 3 can readily generalize to other types of pruning and other types of networks. This provides some interesting avenues for subsequent work in the sense that we can now develop practical pruning algorithms for large-scale, deep neural networks in practical settings, such as deep ResNets (He et al., 2016) on ImageNet (Russakovsky et al., 2015). We have not shown empirical results for our generalized approach to neural network compression yet and will dive into details of how we can effectively leverage our layer-wise error guarantees in the next part of this thesis (Part II). Moreover, our theoretical results indicate that our sensitivity-based framework can serve as a modular and flexible proving technique for various types of neural networks, including potentially new and even more complex architectures such as Transformers (Vaswani et al., 2017).

Part II

Efficient Neural Networks

Chapter 5

Provable Filter Pruning

5.1 Overview

In this chapter, we introduce a data-informed algorithm for pruning redundant filters in convolutional neural networks (CNNs) while incurring minimal loss in the network's accuracy (see Figure 5-1 for an overview). At the heart of our method lies a novel definition of filter importance, i.e., filter *sensitivity*, that is computed by using a small batch of input points as introduced in Chapter 4. There, we prove that by evaluating the relative contribution of each filter to the output of the layer, we can accurately capture its importance with respect to the other filters in the network. We also show that sampling filters with probabilities proportional to their sensitivities leads to an importance sampling scheme with low variance, which enables us to establish rigorous guarantees on the size and performance of the resulting pruned network.

Moreover, our analysis helps bridge the notions of compressibility and importance of each network layer: layers that are more compressible are less important for preserving the output of the original network, and vice-versa. Hence, we obtain and introduce a fully-automated sample size allocation procedure for properly identifying and preserving critical network structures as a corollary.

Unlike weight pruning approaches that lead to irregular sparsity patterns – requiring specialized libraries or hardware to enable computational speedups – our approach compresses the original network to a slimmer subnetwork by pruning fil-

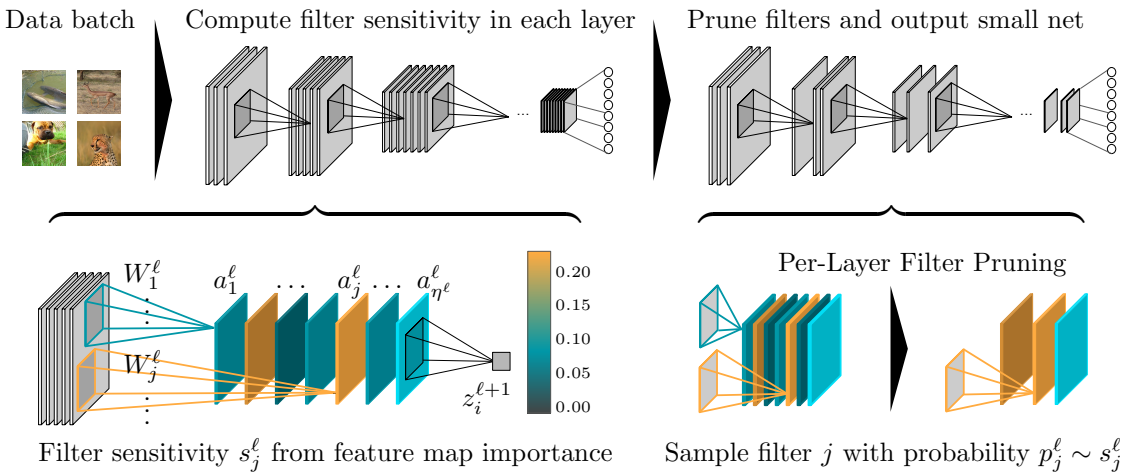


Figure 5-1: Overview of our pruning method. We use a small batch of data points to quantify the relative importance s_j^ℓ of each filter W_j^ℓ in layer ℓ by considering the importance of the corresponding feature map $a_j^\ell = \phi(z_j^\ell)$ in computing the output $z^{\ell+1}$ of layer $\ell + 1$, where $\phi(\cdot)$ is the non-linear activation function. We then prune filters by sampling each filter j with probability proportional to s_j^ℓ and removing the filters that were not sampled. We invoke the filter pruning procedure each layer to obtain the pruned network (the *prune* step); we then retrain the pruned network (*retrain* step), and repeat the *prune-retrain* cycle iteratively.

ters, which enables accelerated inference with any off-the-shelf deep learning library and hardware. We evaluate and compare the effectiveness of our approach in pruning a diverse set of network architectures trained on real-world data sets. Our empirical results show that our approach generates sparser and more efficient models with minimal loss in accuracy when compared to those generated by state-of-the-art filter pruning approaches.

5.1.1 Contributions

The main contributions of this chapter are as follows:

- A novel pruning framework for filter pruning based on our generalized notion of sensitivity including sampling-based and deterministic pruning methods.

- A layer-wise allocation procedure to optimally allocate a per-layer prune budget based on our layer-wise theoretical error guarantees.

- Evaluations on large-scale data sets and networks that demonstrate the practical effectiveness of our algorithm in compressing neural networks via filter pruning.

5.1.2 Relevant Papers

The results presented in this chapter are based on the following paper:

- Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

5.1.3 Outline

We introduce our approach to filter pruning in Section 5.2 including a summary of the theoretical results that our pruning method is based on and discussions about the sampling-based and derandomized version of our algorithm. In Section 5.3, we introduce our method to budget allocation, i.e., the process of assigning an optimal prune ratio to each layer of a network based on our theoretical error guarantees. We then present empirical results on various benchmarks in Section 5.4 and discuss our contributions in Section 5.5.

5.2 Filter Pruning

In this section, we introduce the network pruning problem and outline our sampling-based filter pruning procedure and its theoretical properties. We revisit the notion of *empirical sensitivity (ES)* from Part I to quantify the importance of each filter using a small set of input points. We summarize how our importance criterion enables us to construct a low-variance importance sampling distribution over the filters in each layer. We conclude by showing that our approach can eliminate a large fraction of filters while ensuring that the output of each layer is approximately preserved. We also discuss how we can further boost the performance of our algorithm by choosing filters deterministically based on our theoretical insights from Section 4.4.4.

5.2.1 Preliminaries

Analogous to Chapter 4, we consider a trained L-layer network with parameters $\theta = (W^1, \ldots, W^L)$, where W^ℓ denotes the 4-dimensional tensor in layer $\ell \in [L]$, W_j^ℓ filter $j \in [\eta^\ell]$, and η^ℓ the number of filters in layer ℓ. Moreover, let $W_{:j}^{\ell+1}$ be channel j of tensor $W^{\ell+1}$ that corresponds to filter W_j^ℓ. We let $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}^k$ denote the input and output space, respectively. The marginal distribution over the input space is given by $\mathcal{D}$. For an input $x \in \mathcal{X}$ to the network, we let $z^\ell(x)$ and $a^\ell(x) = \phi(z^\ell(x))$ denote the pre-activation and activation of layer ℓ, where ϕ is the activation function (applied entry-wise). The j^{th} feature map of layer ℓ is given by $a_j^\ell(x) = \phi(z_j^\ell(x))$ (see Figure 5-1). For a given input $x \in \mathcal{X}$, the output of the neural network with parameters θ is given by $f_\theta(x)$.

Our overarching goal is to prune filters from each layer $\ell \in [L]$ by random sampling to generate a compact reparameterization of θ, $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$, where the number of filters in the pruned weight tensor $\hat{W}^\ell$ is a small fraction of the number of filters in the original (uncompressed) tensor W^ℓ. Let $\text{nnz}(\theta)$ denote the number of nonzero (nnz) parameters in the network, i.e., the sum of the number of weights over each $W^\ell \in (W^1, \ldots, W^L)$.

Pruning Objective For a given $\varepsilon, \delta \in (0, 1)$, our objective is to generate a compressed network with parameters $\hat{\theta}$ such that $\text{nnz}(\hat{\theta}) \ll \text{nnz}(\theta)$ and $\mathbb{P}_{x \sim \mathcal{D}, \hat{\theta}}(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)) \geq 1 - \delta$, where $f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)$ denotes an entry-wise guarantee over the output neurons $f_{\hat{\theta}}(x), f_\theta(x) \in \mathcal{Y}$.

5.2.2 Sampling-based Filter Pruning

Our sampling-based filter pruning algorithm for an arbitrary layer $\ell \in [L]$ is depicted as Algorithm 3. The sampling procedure takes as input the set of η^ℓ *channels* in layer $\ell + 1$ that constitute the weight tensor $W^{\ell+1}$, i.e., $W^{\ell+1} = [W_{:1}^{\ell+1}, \ldots, W_{:\eta^\ell}^{\ell+1}]$ as well as the desired relative error and failure probability, $\varepsilon, \delta \in (0, 1)$, respectively. In Line 2 we construct the importance sampling distribution over the feature maps corresponding to the channels by leveraging the *empirical sensitivity* of each feature

Algorithm 3 PRUNECHANNELS($W^{\ell+1}, \varepsilon, \delta, s^\ell$)

Input: $W^{\ell+1} = [W^{\ell+1}_{:1}, \ldots, W^{\ell+1}_{:\eta^\ell}]$: original channels; ε: relative error; δ: failure probability; s^ℓ: feature map sensitivities as in (5.1)
Output: $\hat{W}^{\ell+1}$: pruned channels

1: $S^\ell \leftarrow \sum_{j \in [\eta^\ell]} s^\ell_j$ ▷ where s^ℓ_j is as in (5.1)
2: $p^\ell_j \leftarrow s^\ell_j / S^\ell \quad \forall j \in [\eta^\ell]$
3: $m^\ell \leftarrow \lceil (6 + 2\varepsilon)\, S^\ell\, K\, \log(4\eta_*/\delta)\varepsilon^{-2} \rceil$
4: $\hat{W}^{\ell+1} \leftarrow [0, \ldots, 0]$ ▷ same dimensions as $W^{\ell+1}$
5: **for** $k \in [m^\ell]$ **do**
6: $\quad c(k) \leftarrow$ random draw from $p^\ell = (p^\ell_1, \ldots, p^\ell_{\eta^\ell})$
7: $\quad \hat{W}^{\ell+1}_{:c(k)} \leftarrow \hat{W}^{\ell+1}_{:c(k)} + W^{\ell+1}_{:c(k)} / m^\ell p^\ell_{c(k)}$
8: **end for**
9: **return** $\hat{W}^{\ell+1} = [\hat{W}^{\ell+1}_{:1}, \ldots, \hat{W}^{\ell+1}_{:\eta^\ell}]$;

map $j \in [\eta^\ell]$ as defined in (5.1) and explained in detail in the following subsections. Note that we initially prune *channels* from $W^{\ell+1}$, but as we prune channels from $W^{\ell+1}$ we can simultaneously prune the corresponding *filters* in W^ℓ.

We subsequently set the sample complexity m^ℓ as a function of the given error (ε) and failure probability (δ). This is done in order to ensure that, after the pruning (i.e., sampling) procedure, the *approximate* output – with respect to the sampled channels $\hat{W}^{\ell+1}$ – of the layer will approximate the *true* output of the layer – with respect to the original tensor – up to a multiplicative factor of $(1 \pm \varepsilon)$, with probability at least $1 - \delta$. Intuitively, more samples are required to achieve a low specified error ε with low failure probability δ, and vice-versa. We then proceed to sample m^l times with replacement according to distribution p^ℓ (Lines 5-8) and reweigh each sample by a factor that is inversely proportional to its sample probability to obtain an *unbiased* estimator for the layer's output (see below). The unsampled channels in $W^{\ell+1}$ – and the corresponding filters in W^ℓ – are subsequently discarded, leading to a reduction in the layer's size.

5.2.3 A Tightly-concentrated Estimator

We now turn our attention to analyzing the influence of the sampled channels $\hat{W}^{\ell+1}$ (as in Algorithm 3) on layer $\ell+1$. For ease of presentation, we will henceforth assume

that the layer is linear[1] and will omit explicit references to the input x whenever appropriate. Note that the *true* pre-activation of layer $\ell+1$ is given by $z^{\ell+1} = W^{\ell+1}a^{\ell}$, and the *approximate* pre-activation with respect to $\hat{W}^{\ell+1}$ is given by $\hat{z}^{\ell+1} = \hat{W}^{\ell+1}a^{\ell}$. By construction of $\hat{W}^{\ell+1}$ in Algorithm 3, we equivalently have for each entry $i \in [\eta^{\ell+1}]$

$$\hat{z}_i^{\ell+1} = \frac{1}{m}\sum_{k=1}^{m} Y_{ik}, \quad \text{where} \quad Y_{ik} = W_{ic(k)}^{\ell+1}\frac{a_{c(k)}^{\ell}}{p_{c(k)}^{\ell}}, c(k) \sim p \quad \forall k.$$

By reweighing our samples, we obtain an unbiased estimator for each entry i of the true pre-activation output, i.e., $\mathbb{E}\left[\hat{z}_i^{\ell+1}\right] = z_i^{\ell+1}$ – which follows by the linearity of expectation and the fact that $\mathbb{E}\left[Y_{ik}\right] = z_i^{\ell+1}$ for each $k \in [m]$ –, and so we have for the entire vector $\mathbb{E}_{\hat{W}^{\ell+1}}[\hat{z}^{\ell+1}] = z^{\ell+1}$. So far, we have shown that in expectation, our channel sampling procedure incurs zero error owing to its unbiasedness. However, our objective is to obtain a high probability bound on the entry-wise deviation $\left|\hat{z}_i^{\ell+1} - z_i^{\ell+1}\right|$ for each entry i, which implies that we have to show that our estimator $\hat{z}_i^{\ell+1}$ is highly concentrated around its mean $z_i^{\ell+1}$. To do so, we leverage the following standard result.

Theorem 16 (Bernstein's inequality (Vershynin, 2016))**.** *Let $Y_1, \ldots, Y_m$ be a sequence of m i.i.d. random variables satisfying $\max_{k\in[m]} |Y_k - \mathbb{E}\left[Y_k\right]| \leq R$, and let $Y = \sum_{k=1}^{m} Y_k$ denote their sum. Then, for every $\varepsilon \geq 0$, $\delta \in (0,1)$, we have that $\mathbb{P}\left(|Y/m - \mathbb{E}\left[Y_k\right]| \geq \varepsilon\,\mathbb{E}\left[Y_k\right]\right) \leq \delta$ for*

$$m \geq \frac{\log(2/\delta)}{(\varepsilon E[Y_k])^2}\left(Var(Y_k) + \frac{2}{3}\varepsilon\,\mathbb{E}\left[Y_k\right]R\right).$$

Letting $i \in [\eta^{\ell+1}]$ be arbitrary and applying Theorem 16 to the mean of the random variables $(Y_{ik})_{k\in[m]}$, i.e., to $\hat{z}_i^{\ell+1}$, we observe that the number of samples required for a sufficiently high concentration around the mean is highly dependent on the magnitude and variance of the random variables $(Y_{ik})_k$. By definition of Y_{ik}, observe that these expressions are explicit functions of the sampling distribution p^{ℓ}. Thus, to minimize[2] the number of samples required to achieve high concentration we require a

[1] The extension to CNNs follows directly as outlined Section 4.4

[2] We define the minimization with respect to sample complexity from Theorem 16, which serves

judiciously defined sampling distribution that simultaneously minimizes both R_i and $\mathrm{Var}(Y_{ik})$. For example, the naive approach of uniform sampling, i.e., $p_j^\ell = 1/\eta^\ell$ for each $j \in [\eta^\ell]$ also leads to an unbiased estimator, however, for uniform sampling we have $\mathrm{Var}(Y_{ik}) \approx \eta^\ell \, \mathbb{E}\,[Y_{ik}]^2$ and $R_i \approx \eta^\ell \max_k(w_{ik}^{\ell+1} a_k^\ell)$ and so $\mathrm{Var}(Y_{ik}), R \in \Omega(\eta^\ell)$ in the general case, leading to a linear sampling complexity $m \in \Omega(\eta^\ell)$ by Theorem 16.

5.2.4 Empirical Sensitivity

To obtain a better sampling distribution, we leverage the notion of empirical sensitivity (ES) to prune channels as detailed in Chapter 4. Specifically, for $W^{\ell+1} \geq 0$ (the generalization can be found in Section 4.4) we let the sensitivity s_j^ℓ of feature map j in ℓ be defined as

$$s_j^\ell = \max_{x \in \mathcal{S}} \max_{i \in [\eta^{\ell+1}]} \frac{w_{ij}^{\ell+1} a_j^\ell(x)}{\sum_{k \in [\eta^\ell]} w_{ik}^{\ell+1} a_k^\ell(x)}, \tag{5.1}$$

where $\mathcal{S}$ is a set of t independent and identically distributed (i.i.d.) points drawn from $\mathcal{D}$. Intuitively, the sensitivity of feature map $j \in [\eta^\ell]$ is the maximum (over $i \in [\eta^{\ell+1}]$) relative impact that feature map j had on any pre-activation in the next layer $z_i^{\ell+1}$. We then define the probability of sampling each channel as in Algorithm 3: $j \in [\eta^\ell]$ as $p_j = s_j^\ell / S^\ell$, where $S^\ell = \sum_j s_j^\ell$ is the sum of sensitivities. Under a mild assumption on the distribution – that is satisfied by a wide class of distributions, such as the Uniform, Gaussian, Exponential, among others – of activations (Assumption 2 in Section 4.4), ES enables us to leverage the inherent stochasticity in the draw $x \sim \mathcal{D}$ and establish (see Lemmas 11, 12, and 13 in Section 4.4) that with high probability (over the randomness in $\mathcal{S}$ and x) that

$$\mathrm{Var}(Y_{ik}) \in \Theta(S \, \mathbb{E}\,[Y_{ik}]^2) \quad \text{and} \quad R \in \Theta(S \, \mathbb{E}\,[Y_{ik}]) \qquad \forall i \in [\eta^{\ell+1}]$$

and that the sampling complexity is given by $m \in \Theta(S \, \log(2/\delta) \, \varepsilon^{-2})$ by Theorem 16.

We note that ES does not require knowledge of the data distribution $\mathcal{D}$ and is easy to compute in practice by randomly drawing a small set of input points $\mathcal{S}$ from the validation set and passing the points in $\mathcal{S}$ through the network. This stands in contrast

as a sufficiently good proxy as Bernstein's inequality is tight up to logarithmic factors (Tropp et al., 2015).

with the sensitivity framework used in state-of-the-art coresets constructions (Bachem et al., 2017; Braverman et al., 2016), where the sensitivity is defined to be with respect to the supremum over all $x \in \text{supp}(\mathcal{D})$ in (5.1) instead of a maximum over $x \in \mathcal{S}$. As also noted in Part I, ES inherently considers data points that are likely to be drawn from the distribution $\mathcal{D}$ in practice, leading to a more practical and informed sampling distribution with lower sampling complexity.

Our insights from the discussion in this section culminate in the core theorem (Theorem 9) that we derive in Chapter 4, which establishes that the pruned channels $\hat{W}^{\ell+1}$ (corresponding to pruned filters in W^ℓ) generated by Algorithm 3 is such that the output of layer $\ell + 1$ is well-approximated for each entry. We state it below for completeness.

Theorem 9. *Let $\varepsilon, \delta \in (0,1), \ell \in [L]$, and let $\mathcal{S}$ be a set of $\Theta(\log(\eta_*/\delta))$ i.i.d. samples drawn from $\mathcal{D}$. Then, $\hat{W}^{\ell+1}$ contains at most $\mathcal{O}(S^\ell \log(\eta_*/\delta)\varepsilon^{-2}))$ channels and for $x \sim \mathcal{D}$, with probability at least $1-\delta$, we have $\hat{z}^{\ell+1} \in (1 \pm \varepsilon) z^{\ell+1}$ (entry-wise), where $\eta_* = \max_{\ell \in [L]} \eta^\ell$.*

Theorem 9 can be generalized to hold for all weights and applied iteratively to obtain layer-wise approximation guarantees for the output of each layer. The resulting layer-wise error can then be propagated through the layers to obtain a guarantee on the final output of the compressed network. In particular, applying the error propagation bounds of Chapter 3, we establish our main compression theorem below. The proofs and additional details can be found in Section 4.4.

Theorem 14. *Let $\varepsilon, \delta \in (0,1)$ be arbitrary, let $\mathcal{S} \subset \mathcal{X}$ denote the set of $\lceil K' \log(4\eta/\delta) \rceil$ i.i.d. points drawn from $\mathcal{D}$, and suppose we are given a network with parameters $\theta = (W^1, \ldots, W^L)$. Consider the set of parameters $\hat{\theta} = (\hat{W}^1, \ldots, \hat{W}^L)$ generated by pruning channels of θ according to Algorithm 2 for each $\ell \in [L]$. Then, $\hat{\theta}$ satisfies $\mathbb{P}_{\hat{\theta},\, x \sim \mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \varepsilon) f_\theta(x)\right) \geq 1 - \delta$, and the number of filters in $\hat{\theta}$ is bounded by $\mathcal{O}\left(\sum_{\ell=1}^{L} \frac{L^2 (\Delta^{\ell \to})^2 S^\ell \log(\eta/\delta)}{\varepsilon^2}\right)$.*

5.2.5 Derandomized Filter Pruning

In Section 4.4.4 we highlight how partially derandomizing our approach can yield to better theoretical compression bounds. Specifically, we derive the below theoretical result based on keeping the top-k sensitivities.

Theorem 15. *It is better to keep k feature maps, $\mathcal{C}_{\text{det}} \subseteq [\eta^\ell]$, $|\mathcal{C}_{\text{det}}| = k$, deterministically and sample $m' = \lceil (6+2\varepsilon)\,(S^\ell - S^\ell_k)\,K\,\log(8\eta_*/\delta)\varepsilon^{-2} \rceil$ features from $[\eta^\ell] \setminus \mathcal{C}_{\text{det}}$ if*

$$\sum_{j \notin \mathcal{C}_{\text{det}}} \left(1 - \frac{s_j}{S - S_k}\right)^{m'} > \sum_{j=1}^{\eta^\ell} \left(1 - \frac{s_j}{S}\right)^{m} + \sqrt{\frac{\log(2/\delta)(m+m')}{2}},$$

where $m = \lceil (6+2\varepsilon)\,S^\ell\,K\,\log(4\eta_*/\delta)\varepsilon^{-2} \rceil$, $S_k = \sum_{j \in \mathcal{C}_{\text{det}}} s_j$ and $\eta_* = \max_\ell \eta^\ell$.

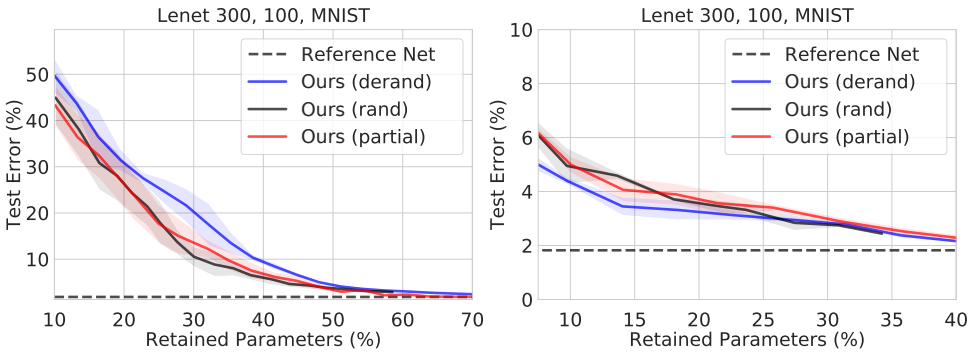
These insights prompted us to consider three variations of our filter pruning approach:

1. No derandomization ("rand"): We apply Algorithm 3 and sample channels with probability proportional to their sensitivity.

2. Partial derandomization ("partial"): We apply Theorem 15 as a preprocessing step to keep the top k channels and then sample from the rest according to Algorithm 3.

3. Complete derandomization ("derand"): We simply keep the top channels until our sampling budget is exhausted.

The results of our evaluations on a LeNet300-100 architecture trained on MNIST can be seen in Figure 5-2. As visible from Figure 5-2(a), the process of partial derandomization does not impact the performance of our algorithm, while the complete derandomization of our algorithm has a slightly detrimental effect on the performance. This is in accordance with Theorem 15, which predicts that that it is best to only partially derandomize the sampling procedure. However, after we retrain the network, the additional error incurred by the complete derandomization is negligible as shown in Figure 5-2(b). Moreover, it appears that – especially for extremely low



(a) Before retraining (b) After retraining

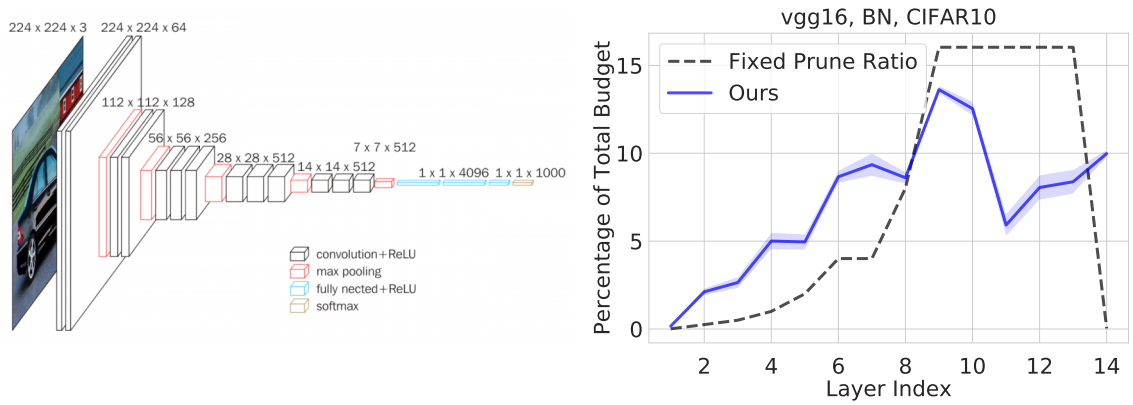
Figure 5-2: The performance of our approach on a LeNet300-100 architecture trained on MNIST with no derandomization (denoted by "rand"), with partial derandomization (denoted by "partial"), and with complete derandomization (denoted by "derand"). The plot in (a) and (b) show the resulting test accuracy for various percentage of retained parameters $1 -$ (pruneratio) before and after retraining, respectively. The additional error of the derandomized algorithm can be neglected in practical settings, especially after retraining.

sampling regime – the completely derandomized approach seems to incur a slight performance boost relative to the other approaches. We suspect that simply keeping the top channels may have a positive side effect on the optimization landscape during retraining.

Consequently, for our subsequent experimental evaluations we use the completely derandomized version of our algorithm for both implementational simplicity and the ability to potentially perform better after retraining, which is an aspect of the pruning procedure that our analysis currently does not consider.

5.3 Optimal Budget Allocation

In the previous sections, we establish the sampling complexity of our filter pruning scheme for any user-specified ε and δ. However, in practice, it is more common for the practitioner to specify the desired pruning ratio, which specifies the resulting size of the pruned model. Given this sampling budget, a practical question that arises is how to optimally ration the sampling budget across the network's layers to minimize the error of the pruned model. A naive approach would be to uniformly allocate the



(a) VGG16 architecture

(b) Budget Allocation for VGG16

Figure 5-3: Early layers of VGG are relatively harder to approximate due to their large spatial dimensions as shown in (a). Our error bounds naturally bridge layer compressibility and importance and enable us to automatically allocate relatively more samples to early layers and less to latter layers as shown in (b). The final layer – due to its immediate influence on the output – is also automatically assigned a large portion of the sampling budget.

sampling budget N so that the same ratio of filters is kept in each layer. However, this allocation scheme implicitly assumes that each layer of the network is of equal importance to retaining the output, which is virtually never the case in practice, as exemplified by Figure 5-3(a).

It turns out that our analytical bounds on the sample complexity per layer (m^ℓ in Algorithm 3) naturally capture the importance of each layer. The key insight lies in bridging the compressibility and importance of each layer: if a layer is not very important, i.e., it does not heavily influence the output of the network, then we expect it to be highly compressible, and vice-versa. This intuition is precisely captured by our sampling complexity bounds that quantify the difficulty of a layer's compressibility.

We leverage this insight to formulate a simple binary search procedure for judiciously allocating the sampling budget N as follows. Let $\delta \in (0, 1)$ be user-specified, pick a random $\varepsilon > 0$, and compute the sampling complexity m^ℓ as in Algorithm 3 together with the resulting layer size n^ℓ. If $\sum_\ell n^\ell = N$, we are done, otherwise, continue searching for an appropriate ε on a smaller interval depending on whether $\sum_\ell n^\ell$ is greater or less than N. The allocation generated by this procedure (see Figure 5-3(b)

for an example) ensures that the maximum layer-wise error incurred by pruning is at most ε.

In other words, our procedure minimizes the maximum (theoretical) error across layers. Specifically, let $\varepsilon^\ell(m^\ell)$ denote our theoretical error guarantee for layer ℓ as a function of the number of samples. Then our budget allocation procedure may be stated as the following optimization procedure

$$\min_{m^1,\dots,m^L} \quad \max_{\ell\in[L]} \varepsilon^\ell(m^\ell) \quad \text{s. t.} \quad \sum_{\ell\in[L]} n^\ell(m^\ell) \leq N, \tag{5.2}$$

where $n^\ell(m^\ell)$ denotes the layer size as a function of the number of filters/samples. Interestingly enough, our approach to budget allocation is independent of the underlying pruning scheme as long as we can describe the layer-wise error incurred by pruning as a function of the number of parameters we want to keep. We denote the resulting pruning algorithm by Provable Filter Pruning (PFP).

5.4 Results

In this section, we evaluate and compare our algorithm's performance to that of state-of-the-art pruning schemes in generating compact networks that retain the predictive accuracy of the original model. Our evaluations show that our approach generates significantly smaller and more efficient models compared to those generated by competing methods. Our results demonstrate the practicality and wide-spread applicability of our proposed approach: across all of our experiments, our algorithm took on the order of a minute to prune a given network[3], required no manual tuning of its hyper-parameters, and performed consistently well across a diverse set of pruning scenarios.

[3]Excluding the time required for the retraining step, which was approximately the same across all methods

5.4.1 Experimental Setup

For our experimental evaluations, we considered a variety of datasets (MNIST, CIFAR-10, ImageNet) and neural network architectures (LeNet, VGG, ResNet, WideResNet, DenseNet) and compared against several state-of-the-art filter pruning methods. We conducted all experiments on either a single NVIDIA RTX 2080Ti with 11GB RAM or a NVIDIA Tesla V100 with 16GB RAM and implemented them in PyTorch (Paszke et al., 2017). Retraining with ImageNet was conducted on a cluster of 8 NVIDIA Tesla V100 GPUs.

Our algorithm only requires two inputs in practice: the desired pruning ratio (PR) and failure probability $\delta \in (0, 1)$, since the number of samples in each layer is automatically assigned by our allocation procedure described in Section 5.3. Following the conventional data partitioning ratio, we reserve 90% of the training data set for training and the remaining 10% for the validation set (Lee et al., 2019).

For each scenario, we prune the original (pre-trained) network with a target prune ratio using the respective pruning algorithm and fine-tune the network by retraining for a specified number of epochs. We repeat this procedure iteratively to obtain various target prune ratios and report the percentage of parameters pruned (PR) and the percentage of FLOP reduction (FR) for each target prune ratio. The target prune ratio follows a hyperharmonic sequence where the i^{th} PR is determined by $1 - 1/(i+1)^{\alpha}$, where α is an experiment-dependent tuning parameter. We conduct the prune-retrain cycle for a range of $10 - 20$ target prune ratios, and report the highest PR and FR for which the compressed network achieves commensurate accuracy, i.e., when the pruned model's test accuracy is within 0.5% of the original model. The quantities reported are averaged over 3 trained models for each scenario, unless stated otherwise.

5.4.2 Comparison Methods

We compare our algorithm to that of the following filter pruning algorithms that we implemented and ran alongside our algorithm: Filter Thresholding (FT) (Li et al., 2016), SoftNet (He et al., 2018), and ThiNet (Luo et al., 2017). We note that FT

and SoftNet are both (weight) magnitude-based filter pruning algorithms, and this class of pruning schemes has recently been reported to be state-of-the-art (Gale et al., 2019; Pitas et al., 2019; Yu et al., 2018b). These methods were re-implemented for our own experiments to ensure an objective comparison method between the methods and we deployed the same iterative pruning and fine-tune strategy as is used in our method. Moreover, we considered a fixed pruning ratio of filters in each layers as none of the competing methods provide an automatic procedure to detect relative layer importance and allocate samples accordingly. Thus, the differentiating factor between the competing methods is their respective pruning step that we elaborate upon below.

Filter Thresholding (FT) (Li et al., 2016) Consider the set of filters $W^\ell = [W_1^\ell, \ldots, W_{\eta^\ell}^\ell]$ in layer ℓ and let $\left\|W_j^\ell\right\|_{2,2}$ denote the entry-wise ℓ_2-norm of W_j^ℓ (or Frobenius norm). Consider a desired sparsity level of $t\%$, i.e., we want to keep only $t\%$ of the filters. We then simply keep the filters with the largest norm until we satisfy our desired level of sparsity.

SoftNet (He et al., 2018) The pruning procedure of He et al. (2018) is similar in nature to the work of Li et al. (2016) except the saliency score used is the entrywise ℓ_1-norm $\left\|W_j^\ell\right\|_{1,1}$ of a filter map W_j^ℓ. During their fine-tuning scheme they allow pruned filters to become non-zero again and then repeat the pruning procedure. As for the other comparisons, however, we only employ a one-shot prune and fine-tune scheme.

ThiNet (Luo et al., 2017) Unlike the previous two approaches, which compute the saliency score of the filter W_j^ℓ by looking at its entry-wise norm, the method of Luo et al. (2017) iteratively and greedily chooses the feature map (and thus corresponding filter) that incurs the least error in an absolute sense in the pre-activation of the next layer. That is, initially, the method picks filter j^* such that $j^* = \text{argmin}_{j \in [\eta^\ell]} \max_{x \in \mathcal{S}} \left| z^{\ell+1}(x) - z_{[j]}^{\ell+1}(x) \right|$, where $z^{\ell+1}(x)$ denotes the pre-activation of layer $\ell + 1$ for some input data point x, $z_{[j]}^{\ell+1}(x)$ the pre-activation when only con-

sidering feature map j in layer ℓ, and $\mathcal{S}$ a set of input data points. We note that this greedy approach is quadratic in both the size η^ℓ of layer ℓ and the size $|\mathcal{S}|$ of the set of data points $\mathcal{S}$, thus rendering it very slow in practice. In particular, we only use a set $\mathcal{S}$ of cardinality comparable to our own method, i.e., around 100 data points in total. On the other hand, Luo et al. report to use 100 data points per output class resulting in 1000 data points for CIFAR-10.

Additional comparisons to other state-of-the-art channel and filter pruning methods can be found in Table 5.5 and Table 5.7 for CIFAR-10 and ImageNet, respectively.

5.4.3 LeNet Architectures on MNIST

As our first experiment, we evaluate the performance of our pruning algorithm and the comparison methods on LeNet300-100 (LeCun et al., 1998), a fully-connected network with two hidden layers of size 300 and 100 hidden units, respectively, and its convolutional counterpart, LeNet-5 (LeCun et al., 1998), which consists of two convolutional layers and two fully-connected layers. Both networks were trained on MNIST using the hyper-parameters specified in Table 5.1.

Table 5.1: We report the hyperparameters used during MNIST training, pruning, and fine-tuning for the LeNet architectures. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During fine-tuning we used the same hyperparameters except for the ones indicated in the lower part of the table.

		LeNet-300-100	LeNet-5
Train	test error	1.59	0.72
	loss	cross-entropy	cross-entropy
	optimizer	SGD	SGD
	epochs	40	40
	batch size	64	64
	LR	0.01	0.01
	LR decay	0.1@{30}	0.1@{25, 35}
	momentum	0.9	0.9
	weight decay	1.0e-4	1.0e-4
Prune	δ	1.0e-12	1.0e-12
	α	not iterative	1.18
Fine-tune	epochs	30	40
	LR decay	0.1@{20, 28}	0.1@{25, 35}

Table 5.2: The prune ratio and the corresponding test error of the sparsest network – with commensurate accuracy – generated by each algorithm.

	Prune Method	Test Error (%)	Prune Ratio (%)
LeNet-300-100	Unpruned	1.59	
	Ours	+0.41	**84.32**
	FT	+0.35	81.68
	SoftNet	+0.41	81.69
	ThiNet	+10.58	75.01
LeNet-5	Unpruned	0.72	
	Ours	+0.35	**92.37**
	FT	+0.47	85.04
	SoftNet	+0.40	80.57
	ThiNet	+0.12	58.17

Table 5.2 depicts the performance of each pruning algorithm in attaining the sparsest possible network that achieves commensurate accuracy for the LeNet architectures. In both scenarios, our algorithm generates significantly sparser networks compared to those generated by the competing filter pruning approaches. In fact, the pruned LeNet-5 model generated by our algorithm by removing filters achieves a prune ratio of $\approx 90\%$, which is even competitive with the accuracy of the sparse models generated by state-of-the-art *weight pruning* algorithms (Lee et al., 2019) [4]. In addition to evaluating the sparsity of the generated models subject to the commensurate accuracy constraint, we also investigated the performance of the pruning algorithms for extreme (i.e., around 5%) pruning ratios (see Figure 5-4(a)). We see that our algorithm's performance relative to those of competing algorithms is strictly better for a wide range of target prune ratios. For LeNet-5 Figure 5-4(a) shows that our algorithm's favorable performance is even more pronounced at extreme sparsity levels (at $\approx 95\%$ prune ratio).

[4]Weight pruning approaches can generate significantly sparser models with commensurate accuracy than can filter pruning approaches since the set of feasible solutions to the problem of filter pruning is a subset of the feasible set for the weight pruning problem

5.4.4 Convolutional Neural Networks on CIFAR-10

Next, we evaluated the performance of each pruning algorithm on significantly larger and deeper Convolutional Neural Networks trained on the CIFAR-10 data set: VGG16 with BatchNorm (Simonyan and Zisserman, 2015), ResNet20, ResNet56, ResNet110 (He et al., 2016), DenseNet22 (Huang et al., 2017), and WideResNet16-8 (Zagoruyko and Komodakis, 2016). For CIFAR-10 experiments, we use the standard data augmentation techniques: padding 4 pixels on each side, random crop to 32x32 pixels, and random horizontal flip. We summarize the complete set of hyperparameters for the various networks in Table 5.3.

Table 5.3: We report the hyperparameters used during training, pruning, and fine-tuning for various convolutional architectures on CIFAR-10. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During fine-tuning we used the same hyperparameters except for the ones indicated in the lower part of the table. $\{30, \ldots\}$ denotes that the learning rate is decayed every 30 epochs.

		VGG16	ResNet20/56/110	DenseNet22	WRN-16-8
Train	test error	7.11	8.59/7.05/6.43	10.07	4.81
	loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
	optimizer	SGD	SGD	SGD	SGD
	epochs	300	182	300	200
	batch size	256	128	64	128
	LR	0.05	0.1	0.1	0.1
	LR decay	0.5@{30, ...}	0.1@{91, 136}	0.1@{150, 225}	0.2@{60, ...}
	momentum	0.9	0.9	0.9	0.9
	Nesterov	✗	✗	✓	✓
	weight decay	5.0e-4	1.0e-4	1.0e-4	5.0e-4
Prune	δ	1.0e-16	1.0e-16	1.0e-16	1.0e-16
	α	1.50	0.50/0.79/0.79	0.40	0.36
Fine-tune	epochs	150	182	300	200

Our results are summarized in Table 5.4 and Figure 5-4. Similar to the results reported in Table 5.2 in the previous subsection, Table 5.4 shows that our method is able to achieve the most sparse model with minimal loss in predictive power relative to the original network. Furthermore, by inspecting the values reported for percentage of FLOP reduction (FR), we observe that the models generated by our approach are not only more sparse in terms of the number of total parameters, but also more efficient in terms of the inference time complexity.

Table 5.4: Overview of the pruning performance of each algorithm for various CNN architectures. For each algorithm and network architecture, the table reports the prune ratio (PR, %) and pruned FLOPs ratio (FR, %) of pruned models when achieving test accuracy within 0.5% of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). Our results indicate that our pruning algorithm generates smaller and more efficient networks with minimal loss in accuracy, when compared to competing approaches.

[%]	Orig.	Ours			FT			SoftNet			ThiNet		
	Err.	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR
ResNet20	8.60	+0.49	**62.67**	45.46	+0.43	42.65	44.59	+0.50	46.42	**49.40**	+2.10	32.90	32.73
ResNet56	7.05	+0.28	**88.98**	**84.42**	+0.48	81.46	82.73	+0.36	81.46	82.73	+1.28	50.08	50.06
ResNet110	6.43	+0.36	**92.07**	**89.76**	+0.17	86.38	87.39	+0.34	86.38	87.39	+0.92	49.70	50.39
VGG16	7.11	+0.50	**94.32**	**85.03**	+1.11	80.09	80.14	+0.81	63.95	63.91	+2.13	63.95	64.02
DenseNet22	10.07	+0.46	**56.44**	**62.66**	+0.32	29.31	30.23	+0.21	29.31	30.23	+4.36	50.76	51.06
WRN16-8	4.83	+0.46	**66.22**	**64.57**	+0.40	24.88	24.74	+0.14	16.93	16.77	+0.35	14.18	14.09

Figure 5-4 depicts the performance of the evaluated algorithms for various levels of prune ratios. Once again, we see the consistently better performance of our algorithm in generating sparser models that approximately match or exceed the predictive accuracy of the original uncompressed network. In addition, Table 5.5 provides further comparisons to state-of-the-art filter pruning methods where we compare the performance of our approach to the results for various ResNets and VGG16 reported directly in the respective papers. The comparisons in Table 5.5 reaffirm that our algorithm can consistently generate simultaneously sparser and more accurate networks compared to competing methods.

In view of our results from the previous subsection, the results shown in Table 5.4, Figure 5-4, and Table 5.5 highlight the versatility and broad applicability of our method, and seem to suggest that our approach fares better relative to the compared algorithms on more challenging pruning tasks that involve large-scale networks. We suspect that these favorable properties are explained by the data-informed evaluations of filter importance and the corresponding theoretical guarantees of our algorithm – which enable robustness to variations in network architecture and data distribution.

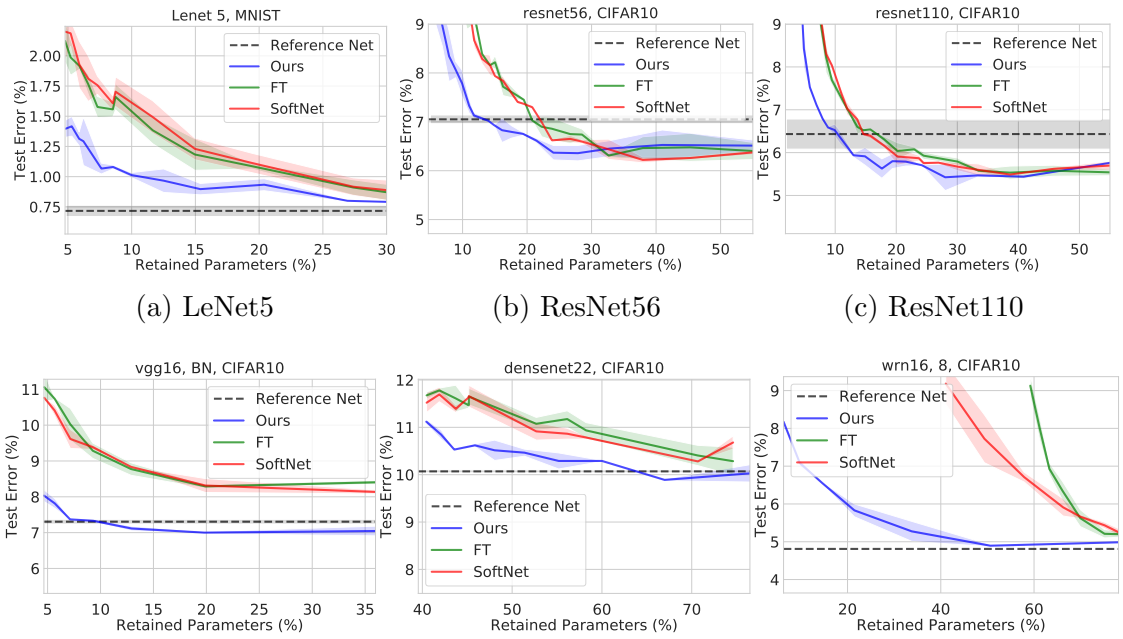
Table 5.5: The performance of our algorithm and that of state-of-the-art filter pruning algorithms on modern CNN architectures trained on CIFAR-10. The reported results for the competing algorithms were taken directly from the corresponding papers. For each network architecture, the best performing algorithm for each evaluation metric, i.e., Pruned Err., Err. Diff, PR, and FR, is shown in **bold**. The results show that our algorithm consistently outperforms state-of-the-art pruning approaches in nearly all of the relevant pruning metrics.

Model	Method	Orig. Err. (%)	Pruned Err. (%)	Err. Diff. (%)	PR (%)	FR (%)
ResNet20	Ours (within 0.5% err.)	8.60	9.09	+0.49	**62.67**	45.46
	Ours (orig. err.)	8.60	**8.64**	**+0.04**	43.16	32.10
	Ours (lowest err.)	8.60	**8.64**	**+0.04**	43.16	32.10
	He et al. (2018) (SoftNet)	7.80	8.80	+1.00	N/A	29.30
	He et al. (2019)	7.80	9.56	+1.76	N/A	**54.00**
	Ye et al. (2018)	8.00	9.10	+1.10	37.22	N/A
	Lin et al. (2020)	7.52	9.72	+2.20	40.00	N/A
ResNet56	Ours (within 0.5% err.)	7.05	7.33	+0.28	**88.98**	**84.42**
	Ours (orig. err.)	7.05	7.02	-0.03	86.00	80.76
	Ours (lowest err.)	7.05	6.36	**-0.69**	72.10	67.41
	Li et al. (2016) (FT)	6.96	6.94	-0.02	13.70	27.60
	He et al. (2018) (SoftNet)	6.41	6.65	+0.24	N/A	52.60
	He et al. (2019)	6.41	6.51	+0.10	N/A	52.60
	He et al. (2017)	7.20	8.20	+1.00	N/A	50.00
	Li et al. (2019b)	6.28	6.60	+0.32	78.10	50.00
	Lin et al. (2020)	5.49	**5.97**	+0.48	40.00	N/A
ResNet110	Ours (within 0.5% err.)	6.43	6.79	+0.36	**92.07**	**89.76**
	Ours (orig. err.)	6.43	6.35	-0.08	89.15	86.97
	Ours (lowest err.)	6.43	**5.42**	**-1.01**	71.98	68.94
	Li et al. (2016) (FT)	6.47	6.70	+0.23	32.40	38.60
	He et al. (2018) (SoftNet)	6.32	6.14	-0.18	N/A	40.80
	He et al. (2019)	6.32	6.16	-0.16	N/A	52.30
	Dong et al. (2017b)	6.37	6.56	+0.19	N/A	34.21
VGG16	Ours (within 0.5% err.)	7.28	7.78	+0.50	**94.32**	**85.03**
	Ours (orig. err.)	7.28	7.17	-0.11	87.06	70.32
	Ours (lowest err.)	7.28	7.06	**-0.22**	80.02	59.21
	Li et al. (2016) (FT)	6.75	6.60	-0.15	64.00	34.20
	Huang et al. (2018b)	7.23	7.83	+0.60	83.30	45.00
	He et al. (2019)	6.42	6.77	+0.35	N/A	35.90
	Li et al. (2019b)	5.98	**6.18**	+0.20	78.20	76.50

5.4.5 Convolutional Neural Networks on ImageNet

We consider pruning convolutional neural networks of varying size (ResNet18, ResNet50, and ResNet101) trained on the ImageNet (Russakovsky et al., 2015) data set. The hyper-parameters used for training and for our pruning algorithm are shown in Table 5.6. For this dataset, we considered two scenarios: (i) iterative pruning without retraining and (ii) iterative prune-retrain with a limited amount of iterations given the resource-intensive nature of the experiments.

In the first scenario, we evaluate the baseline effectiveness of each pruning algo-



(a) LeNet5 (b) ResNet56 (c) ResNet110

(d) VGG16 (e) DenseNet22 (f) WRN16-8

Figure 5-4: The accuracy of the generated pruned models for the evaluated pruning schemes for various target prune ratios. Note that the x axis is the percentage of **parameters retained**, i.e., $(1 - \text{pruneratio})$. ThiNet was omitted from the plots for better readability. Our results show that our approach generates pruned networks with minimal loss in accuracy even for high prune ratios. Shaded regions correspond to values within one standard deviation of the mean.

rithm *without fine-tuning* by applying the same iterative prune-scheme, but without the retraining step. The results of these evaluations can be seen in Figure 5-5. Figure 5-5 shows that our algorithm outperforms the competing approaches in generating compact, more accurate networks. We suspect that by reevaluating the data-informed filter importance (empirical sensitivity) after each iteration our approach is capable of more precisely capturing the inter-dependency between layers that alter the relative importance of filters and layers with each pruning step. This is in contrast to competing approaches, which predominantly rely on weight-based criteria of filter importance, and thus can only capture this inter-dependency after retraining (which subsequently alters the magnitude of the weights).

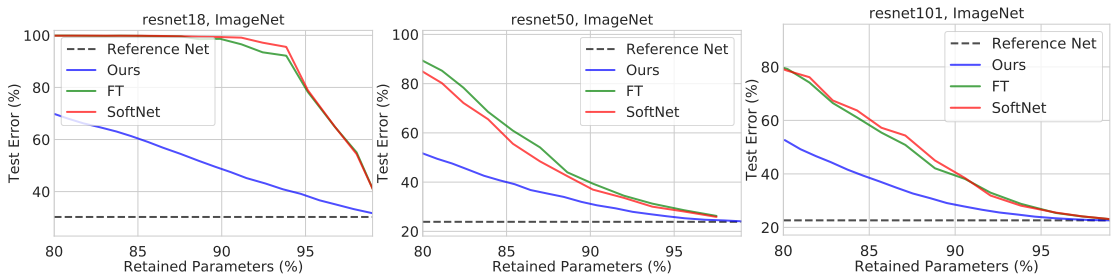
Next, we consider pruning the networks using the standard iterative prune-retrain procedure as before with only a limited number of iterations (2-3 iterations per reported experiment). The results of our evaluations are reported in Table 5.7 with

Table 5.6: The hyper-parameters used for training and pruning residual networks trained on the ImageNet data set.

		ResNet18/50/101
Train	top-1 test error	30.26/23.87/22.63
	top-5 test error	10.93/7.13/6.45
	loss	cross-entropy
	optimizer	SGD
	epochs	90
	batch size	256
	LR	0.1
	LR decay	0.1@{30, 60}
	momentum	0.9
	Nesterov	✗
	weight decay	1.0e-4
Prune	δ	1.0e-16
	α	0.43/0.50/0.50
Fine-tune	epochs	90

Table 5.7: Comparisons of the performance of various pruning algorithms on ResNets trained on ImageNet (Russakovsky et al., 2015). The reported results for the competing algorithms were taken directly from the corresponding papers. For each network architecture, the best performing algorithm for each evaluation metric, i.e., Pruned Err., Err. Diff, PR, and FR, is shown in bold.

Model	Method	Top-1 Err. (%)			Top-5 Err. (%)			PR (%)	FR (%)
		Orig.	Pruned	Diff.	Orig.	Pruned	Diff.		
Resnet18	Ours (within 4.0% top-1)	30.26	34.35	+4.09	10.93	13.25	+2.32	**60.48**	**43.12**
	Ours (within 2.0% top-1)	30.26	32.62	+2.36	10.93	12.09	+1.16	43.80	29.30
	Ours (lowest top-1 err.)	30.26	**31.34**	**+1.08**	10.93	**11.43**	**+0.50**	31.03	19.99
	He et al. (2018) (SoftNet)	29.72	32.90	+3.18	10.37	12.22	+1.85	N/A	41.80
	He et al. (2019)	29.72	31.59	+1.87	10.37	11.52	+1.15	N/A	41.80
	Dong et al. (2017b)	30.02	33.67	+3.65	10.76	13.06	+2.30	N/A	33.30
Resnet50	Ours (within 1.0% top-1)	23.87	24.79	+0.92	7.13	7.57	+0.45	**44.04**	30.05
	Ours (lowest top-1 err.)	23.87	**24.09**	**+0.22**	7.13	**7.19**	**+0.06**	18.01	10.82
	He et al. (2018) (SoftNet)	23.85	25.39	+1.54	7.13	7.94	+0.81	N/A	41.80
	Luo et al. (2017) (ThiNet)	27.12	27.96	+0.84	8.86	9.33	+0.47	33.72	36.39
	He et al. (2019)	23.85	25.17	+1.32	7.13	7.68	+0.55	N/A	53.50
	He et al. (2017)	N/A	N/A	N/A	7.80	9.20	+1.40	N/A	50.00
	Luo and Wu (2018)	23.85	25.24	+1.39	7.13	7.85	+0.72	N/A	48.70
	Liu et al. (2019b)	23.40	24.60	+1.20	N/A	N/A	N/A	N/A	**51.22**
Resnet101	Ours (within 1.0% top-1)	22.63	23.57	+0.94	6.45	6.89	+0.44	**50.45**	**45.08**
	Ours (lowest top-1 err.)	22.63	23.22	+0.59	6.45	6.74	+0.29	33.04	29.38
	He et al. (2018) (SoftNet)	22.63	**22.49**	**-0.14**	6.44	**6.29**	**-0.15**	N/A	42.20
	He et al. (2019)	22.63	22.68	+0.05	6.44	6.44	+0.00	N/A	42.20
	Ye et al. (2018)	23.60	24.73	+1.13	N/A	N/A	N/A	47.20	42.69



(a) ResNet18 (b) ResNet50 (c) ResNet101

Figure 5-5: The results of our evaluations of the algorithms in the *prune-only* scenario, where the network is iteratively pruned down to a specified target prune ratio and the fine-tuning step is omitted. Note that the x axis is the percentage of parameters retained, i.e., $(1 - \text{pruneratio})$.

respect to the following metrics: the resulting error of the pruned network (Pruned Err.), the difference in model classification error (Err. Diff), the percentage of parameters pruned (PR), and the FLOP Reduction (FR). We would like to highlight that – despite the limited resources used during the experiments – our method is able to produce compressed networks that are as accurate and compact as the models generated by competing approaches (obtained by significantly more prune-retrain iterations than allotted to our algorithm).

5.4.6 Application to Real-time Regression Tasks

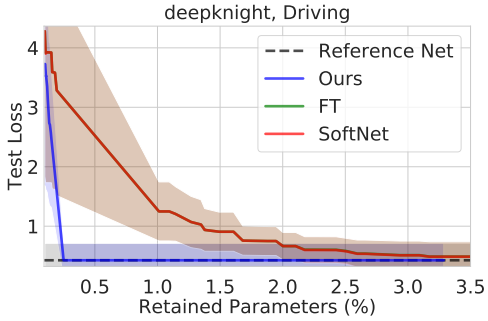
Real-time applications of neural networks, such as their use in autonomous driving scenarios, require network models that are not only highly accurate, but also highly efficient, i.e., fast, when it comes to inference time complexity (Amini et al., 2018). Model compression, and in particular, filter pruning has potential to generate compressed networks capable of achieving both of these objectives. To evaluate and compare the effectiveness of our method on pruning networks intended for regression tasks and real-time systems, we evaluated the various pruning approaches on the *DeepKnight* network (Amini et al., 2018), a regression network deployed on an autonomous vehicle in real time to predict the steering angle of the human driver (see Table 5.8 for experimental details).

Figure 5-6 depicts the results of our evaluations and comparisons on the *Deep-*

Table 5.8: We report the hyperparameters used for training and pruning the driving network of Amini et al. (2018) together with the provided data set. No fine-tuning was conducted for this architecture.

		Deepknight
Train	test loss	4.9e-5
	loss	MSE
	optimizer	Adam
	epochs	100
	batch size	32
	LR	1e-4
	LR decay	0.1@{50, 90}
	momentum	0.9
	weight decay	1.0e-4
Prune	δ	1.0e-32
	α	not iterative
Fine-tune	epochs	0

(a) Example driving image from Amini et al. (2018)



(b) Pruning performance before retraining

Figure 5-6: The performance of our approach on a regression task used to infer the steering angle for an autonomous driving task (Amini et al., 2018). (a) An exemplary image taken from the data set. (b) The performance of our pruning procedure before retraining evaluated on the test loss and compared to competing filter pruning methods. Note that the x axis is percentage of parameters retained, i.e., $1 - (\text{pruneratio})$.

Knight network *without* the fine-tuning step. We omitted the iterative fine-tuning step for this scenario and instead evaluated the test loss for various prune ratios because (i) the evaluated algorithms were able to generate highly accurate models without the retraining step and (ii) in order to evaluate and compare the performance of solely the core pruning procedure. Similar to the results obtained in the preceding pruning scenarios, Figure 5-6 shows that our method consistently outperforms competing approaches for all of the specified prune ratios.

5.5 Discussion

Empirical Results. In addition to the favorable empirical results of our algorithm, our approach exhibits various advantages over competing methods that manifest themselves in our empirical evaluations. For one, our algorithm does not require any additional hyper-parameters other than the pruning ratio and the desired failure probability. Given these sole two parameters, our approach automatically allocates the number of filters to sample for each layer. This alleviates the need to perform time-intensive ablation studies (He et al., 2018) and to resort to uninformed (i.e., uniform) sample allocation strategies, e.g., removing the same percentage of filters in each layer (Li et al., 2016), which fails to consider the non-uniform influence of each layer on the network's output (see Section 5.3). Moreover, our algorithm is simple-to-implement and computationally efficient both in theory and practice: the computational complexity is dominated by the $|\mathcal{S}|$ forward passes required to compute the sensitivities ($|\mathcal{S}| \leq 256$ in practical settings) and in practice, our algorithm takes on the order of a minute to prune the network.

Theoretical Error Guarantees. We present – to the best of our knowledge – the first filter pruning algorithm that generates a pruned network with theoretical guarantees on the size and performance of the generated network. Our method is data-informed, simple-to-implement, and efficient both in theory and practice. Our approach can also be broadly applied to varying network architectures and data sets with minimal hyper-parameter tuning. This stands in contrast to existing filter pruning approaches that are generally data-oblivious, rely on heuristics for evaluating the parameter importance, or require tedious hyper-parameter tuning. Our empirical evaluations on popular network architectures and data sets reaffirm the favorable theoretical properties of our method and demonstrate its practical effectiveness in obtaining sparse, efficient networks. We envision that besides its immediate use for pruning state-of-the-art models, our approach can also be used as a sub-procedure in other deep learning applications, e.g., for identifying winning lottery tickets (Frankle and Carbin, 2019) and for efficient architecture search (Liu et al., 2019c).

Global compression. Moreover, we also identified a modular compression framework that considers both the layer-wise error guarantees as well as global considerations of how much each individual layer should be pruned in order to achieve the optimal performance-size trade-off in a given network (see Section 5.3). Our main insight hereby hinges on the observation that we can leverage our layer-wise error guarantees to optimize over the per-layer prune ratios to obtain a more optimally pruned network. We will further build upon this insight in our next chapter, Chapter 6, where we consider the neural network compression problem in the context of low-rank decomposition techniques.

Chapter 6

Automatic Layer-wise Decomposition

6.1 Overview

Neural network compression entails taking an existing model and reducing its computational and memory footprint in order to enable the deployment of large-scale networks in resource-constrained environments. In the previous chapters, we approach this problem from the perspective of pruning either individual weights or structures such as neurons and filters from the underlying weight tensors of each layer.

Complementary to this approach, *low-rank compression* aims at decomposing a layer's weight tensor into a tuple of smaller low-rank tensors. Such compression techniques may build upon the rich literature on low-rank decomposition and its numerous applications outside deep learning such as dimensionality reduction (Laparra et al., 2015) or spectral clustering (Peng et al., 2015). Moreover, low-rank compression can be readily implemented in any machine learning framework by replacing the existing layer with a set of smaller layers without the need for, e.g., sparse linear algebra support.

In this sense, just like our approach to filter pruning (Chapter 5) we can readily obtain efficiency gains from the network since we can replace the large, dense layers by smaller, but still dense layers. Moreover, by considering low-rank compression we

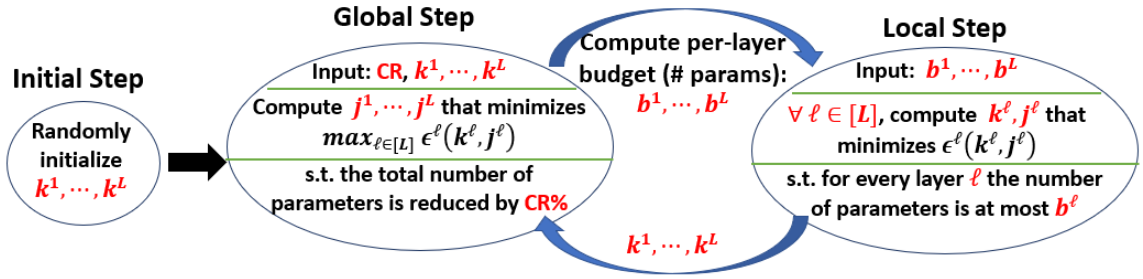


Figure 6-1: **ALDS Overview**. The framework consists of a global and local step to obtain an optimal per-layer low-rank compression. We first randomly initialize the number of subspaces for each layer. We then optimize for the optimal per-layer, per-subspace rank by minimizing the maximum relative compression error (global step). Given a per-layer budget, we then optimize for the number of low-rank subspaces in each layer (local step). Both steps are repeated iteratively until convergence.

can gain insights about the dimensionality of the underlying feature space.

Within deep learning, we encounter two related, yet distinct challenges when applying low-rank compression. On the one hand, each layer should be efficiently decomposed (the "local step") and, on the other hand, we need to balance the amount of compression in each layer in order to achieve a desired overall compression ratio with minimal loss in the predictive power of the network (the "global step"). While the "local step", i.e., designing the most efficient layer-wise decomposition method, has traditionally received lots of attention (Denton et al., 2014; Garipov et al., 2016; Jaderberg et al., 2014; Kim et al., 2015b; Lebedev et al., 2015; Novikov et al., 2015), the "global step" has only recently been the focus of attention in research, e.g., see the recent works of Alvarez and Salzmann (2017); Idelbayev and Carreira-Perpinán (2020); Xu et al. (2020).

Building upon our insights from Chapter 5, we design a framework that simultaneously accounts for both the local (layer-wise compression) and global step (budget allocation). Our proposed solution, termed *Automatic Layer-wise Decomposition Selector (ALDS)*, addresses this challenge by iteratively optimizing for each layer's decomposition method (local step) and the low-rank compression itself while accounting for the maximum error incurred across layers (global step). In that sense, it also provides a generalization of our pruning framework presented in the previous chapter by decoupling the local and global step during the optimization procedure.

6.1.1 Contributions

Our contributions can be summarized as follows:

- An efficient, layer-wise decomposition framework relying on a straightforward decomposition of each layer that is based on singular value decomposition (SVD), where each layer's weight tensor is first folded and decomposed via SVD followed by encoding the resulting pair of matrices as two separate layers.
- A generalization of our layer-wise decomposition via an automatically-determined splitting of the weight matrix into multiple subsets for enhanced low-rank decomposition.
- A global framework (ALDS) that optimally determines the type of decomposition (number of subspaces) and the optimal per-layer low-rank compression based on minimizing the maximum relative error incurred.
- Extensive experimental evaluations on multiple benchmarks, models, and datasets, including large-scale datasets, establishing the competitive performance of our algorithm relative to existing approaches.

6.1.2 Relevant Papers

The results presented in this chapter are based on the following paper:

- Lucas Liebenwein, Alaa Maalouf, Oren Gal, Dan Feldman, and Daniela Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. In *Advances in Neural Information Processing Systems (Under Review; arXiv preprint arXiv:2107.11442)*, 2021c.

6.1.3 Outline

The chapter is structured as follows. In Section 6.2 we introduce our approach to low-rank compression and the global consideration that lead to our main contribution, i.e., an error-aware global compression framework. Subsequently, we highlight

our experimental results in Section 6.3 including necessary experimental details for reproducibility. Additional results are provided in Appendix A. We conclude this chapter with a discussion in Section 6.4.

6.2 Method

In this section, we introduce our compression framework consisting of a layer-wise decomposition method (Section 6.2.2), a global selection mechanism to simultaneously compress all layers of a network (Section 6.2.3), and an optimization procedure (ALDS) to solve the selection problem (Section 6.2.4). Before diving into the details of our methods, we present some preliminaries pertaining to the use of low-rank compression techniques in the context of neural networks (Section 6.2.1).

6.2.1 Preliminaries

Our layer-wise compression technique hinges upon the insight that any linear layer may be cast as a matrix multiplication, which enables us to rely on SVD as compression subroutine. Focusing on convolutions we show how such a layer can be recast as matrix multiplication. Similar approaches have been used by Denton et al. (2014); Idelbayev and Carreira-Perpinán (2020); Wen et al. (2017) among others.

Convolution to matrix multiplication. For a given convolutional layer of f filters, c channels, $\kappa_1 \times \kappa_2$ kernel and an input feature map with c features, each of size $m_1 \times m_2$, we denote by $\mathcal{W} \in \mathbb{R}^{f \times c \times \kappa_1 \times \kappa_2}$ and $\mathcal{X} \in \mathbb{R}^{c \times m_1 \times m_2}$ the weight tensor and input tensor, respectively. Moreover, let $W \in \mathbb{R}^{f \times c\kappa_1\kappa_2}$ denote the unfolded matrix operator of the layer constructed from $\mathcal{W}$ by flattening the c kernels of each filter into a row and stacking the rows to form a matrix. Finally, let p denote the total number of sliding blocks and $X \in \mathbb{R}^{c\kappa_1\kappa_2 \times p}$ denote the unfolded input matrix, which is constructed from the input tensor $\mathcal{X}$ as follows: while simulating the convolution by sliding $\mathcal{W}$ along $\mathcal{X}$ we extract the sliding local blocks of $\mathcal{X}$ across all channels by flattening each block into a $c\kappa_1\kappa_2$-dimensional column vector and concatenating them

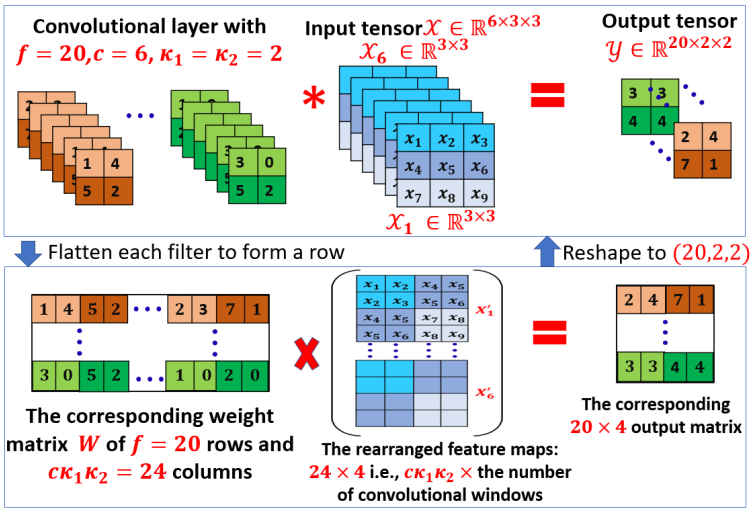


Figure 6-2: **Convolution to matrix multiplication.** A convolutional layer of $f = 20$ filters, $c = 6$ channels, and 2×2 kernel ($\kappa_1 = \kappa_2 = 2$). The input tensor shape is $6 \times 3 \times 3$. The corresponding weight matrix has $f = 20$ rows (one row per filter) and 24 columns ($c \times \kappa_1 \times \kappa_2$), as for the corresponding feature matrix, it has 24 rows and 4 columns, the 4 here is the number of convolution windows (i.e., number of pixels/entries in each of the output feature maps). After multiplying those matrices, we reshape them to the desired shape to obtain the desired output feature maps.

together to form X. As illustrated in Figure 6-2, we may now express the convolution $\mathcal{Y} = \mathcal{W} * \mathcal{X}$ as the matrix multiplication $Y = WX$, where $\mathcal{Y} \in \mathbb{R}^{f\times p_1\times p_2}$ and $Y \in \mathbb{R}^{f\times p}$ correspond to the tensor and matrix representation of the output feature maps, respectively, and p_1, p_2 denote the spatial dimensions of $\mathcal{Y}$. The equivalence of $\mathcal{Y}$ and Y can be easily established via an appropriate reshaping operation since $p = p_1 p_2$.

Efficient tensor decomposition via SVD. Equipped with the notion of correspondence between convolution and matrix multiplication our goal is to decompose the layer via its matrix operator $W \in \mathbb{R}^{f\times c\kappa_1\kappa_2}$. To this end, we compute the j-rank approximation of W using SVD and factor it into a pair of smaller matrices $U \in \mathbb{R}^{f\times j}$ and $V \in \mathbb{R}^{j\times c\kappa_1\kappa_2}$. We may then replace the original convolution, represented by W, by two smaller convolutions, represented by V and U for the first and second layer, respectively. Just like for the original layer, we can establish an equivalent convolution layer for both U and V as depicted in Figure 6-3. To establish the equivalence we note that (a) every row of the matrices V and U corresponds to a flattened filter of the respective convolution, and (b) the number of channels in each layer is equal to

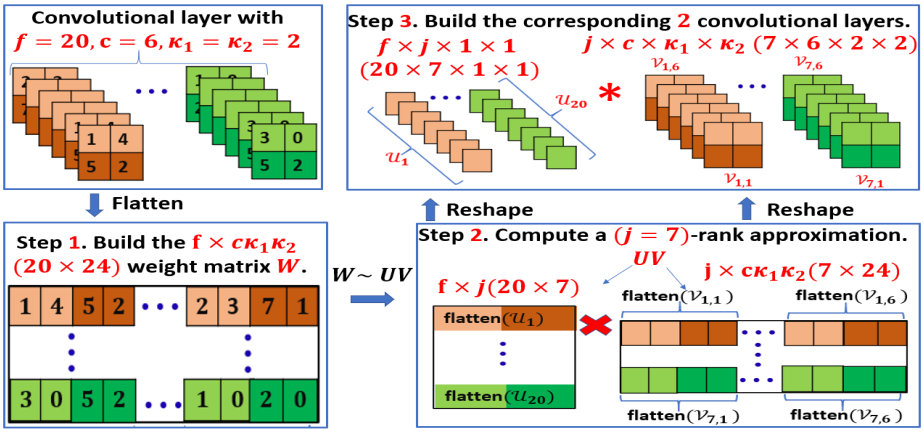


Figure 6-3: **Low-rank decomposition for convolutional layers via SVD.** The given convolution, c.f. Figure 6-2, has 20 filters, each of shape $6 \times 2 \times 2$, resulting in a total of 480 parameters. After extracting the corresponding weight matrix $W \in \mathbb{R}^{20\times 24}$ $(f \times c\kappa_1\kappa_2)$, we compute its $(j = 7)$-rank decomposition to obtain the pair of matrices $U \in \mathbb{R}^{20\times 7}$ $(f \times j)$ and $V \in \mathbb{R}^{7\times 24}$ $(j \times c\kappa_1\kappa_2)$. Those matrices are encoded back as a pair of convolutional layers, the first (corresponding to V) has $j = 7$ filters, $c = 6$ channels and a 2×2 $(\kappa_1 \times \kappa_2)$ kernel, whereas the second (corresponding to U) is a 1×1 convolution of $f = 20$ filters, and $j = 7$ channels. The resulting layers have 308 parameters.

the number of channels in its corresponding input tensor. Hence, the first layer, which is represented by $V \in \mathbb{R}^{j\times c\kappa_1\kappa_2}$ has j filters, c.f. (a), each consisting of c channels, c.f. (b), with kernel size $\kappa_1 \times \kappa_2$. The second layer corresponding to U has f filters, c.f. (a), j channels, c.f. (b), and a 1×1 kernel and may be equivalently represented as the tensor $\mathcal{U} \in \mathbb{R}^{f\times j\times 1\times 1}$. Note that the number of weights is reduced from $fc\kappa_1\kappa_2$ to $j(f + c\kappa_1\kappa_2)$.

6.2.2 Local Layer Compression

We detail our low-rank compression scheme for convolutional layers below and note that it readily applies to fully-connected layers as well as a special case of convolutions with a 1×1 kernel.

Compressing Convolutions via SVD

Given a convolutional layer of f filters, c channels, and a $\kappa_1 \times \kappa_2$ kernel we denote the corresponding weight tensor by $\mathcal{W} \in \mathbb{R}^{f\times c\times \kappa_1\times \kappa_2}$. Following Denton et al. (2014); Idelbayev and Carreira-Perpinán (2020); Wen et al. (2017) and others, we can then

interpret the layer as a linear layer of shape $f \times c\kappa_1\kappa_2$ and the corresponding rank j-approximation as two subsequent linear layers of shape $f \times j$ and $j \times c\kappa_1\kappa_2$. Mapped back to convolutions, this corresponds to a $j \times c \times \kappa_1 \times \kappa_2$ convolution followed by a $f \times j \times 1 \times 1$ convolution.

Multiple Subspaces

Following the intuition outlined in Section 6.1 we propose to cluster the columns of the layer's weight matrix into $k \geq 2$ separate subspaces before applying SVD to each subset. To this end, we may consider any clustering method, such as k-means or projective clustering (Chen et al., 2018a; Maalouf et al., 2021). However, such methods require expensive approximation algorithms which would limit our ability to incorporate them into an optimization-based compression framework as outlined in Section 6.2.3. In addition, arbitrary clustering may require re-shuffling the input tensors which could lead to significant slow-downs during inference. We instead opted for a simple clustering method, namely *channel slicing*, where we simply divide the c input channels of the layer into k subsets each containing at most $\lceil c/k \rceil$ consecutive input channels. Unlike other methods, channel slicing is efficiently implementable, e.g., as grouped convolutions in PyTorch (Paszke et al., 2017) and ensures practical speed-ups subsequent to compressing the network.

Remarks on Clustering Methods to Choose Subspaces

As shortly mentioned above, one can cluster the columns of the corresponding weight matrix W, instead of clustering the channels of the convolutional layer. Here, the channel clustering can be defined as constraint clustering of these columns, where columns which include entries that correspond to the same kernel (e.g., the first 4 columns in W from Figure 6-3) are guaranteed to be in the same cluster.

This generalization is easily adaptable to other clustering methods that generate a wider set of solutions, e.g., the known **k-means**. An intuitive choice for our case could be **projective clustering** and its variants. The goal of projective clustering is to compute a set of k subspaces, each of dimension j, that minimizes the sum of *squared*

distances from each column in W to its *closest* subspace from this set. Then, we can partition the columns of W into k subsets according to their nearest subspace from this set. This is a natural extension of SVD that solves this problem for the case of $k = 1$. However, this problem is known to be NP-hard, hence expensive approximation algorithms are required to solve it, or alternatively, a local minimum solution can be obtained using the Expectation-maximization (EM) method (Dempster et al., 1977).

Furthermore, and probably more importantly, all of these methods cannot be considered as structured compression since arbitrary clustering may require re-shuffling the input tensors which could lead to significant slow-downs during inference. For example, when compressing a fully-connected layer, the arbitrary clustering may result in nonconsecutive neurons from the first layer that are connected to the same neuron in the second layer, while neurons that are between them are not. Hence, these layers can only have a large, sparse instead of a small, dense representation.

To this end, we choose to use **channel slicing**, i.e., we simply split the channels of the convolutional layer into k chunks, where each chunk has at most c/k consecutive channels. Splitting the channels into consecutive subsets (without allowing any arbitrary clustering) and applying the factorization on each one results in a structurally compressed layer without the need of special software/hardware support. Furthermore, this approach is the fastest among all the others.

Finally, while other approaches may give a better initial guess for a compressed network in theory, in practice this is not the case; see Figure 6-8. We see that in practice, our method improve upon state-of-the-art techniques and obtains smaller networks with higher accuracy without the use of those complicated approaches that may result in sparse but not smaller network.

Overview of Per-layer Decomposition

In summary, for given integers $j, k \geq 1$ and a 4D tensor $\mathcal{W} \in \mathbb{R}^{f \times c \times \kappa_1 \times \kappa_2}$ representing a convolution the per-layer compression method proceeds as follows:

1. Partition the channels of the convolutional layer into k subsets, where each

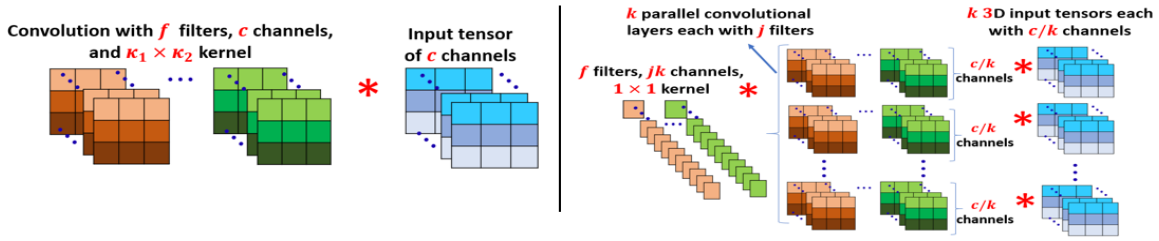


Figure 6-4: **Left: 2D convolution. right: decomposition used for ALDS.** For a $f \times c \times \kappa_1 \times \kappa_2$ convolution with f filters, c channels, and $\kappa_1 \times \kappa_2$ kernel, our per-layer decomposition consists: (1) k parallel $j \times c/k \times \kappa_1 \times \kappa_2$ convolutions; (2) a single $f \times kj \times 1 \times 1$ convolution applied on the first layer's (stacked) output.

subset has at most $\lceil c/k \rceil$ consecutive channels, resulting in k convolutional tensors $\{\mathcal{W}_i\}_{i=1}^k$ where $\mathcal{W}_i \in \mathbb{R}^{f \times c_i \times \kappa_1 \times \kappa_2}$, and $\sum_{i=1}^k c_i = c$.

2. DECOMPOSE each tensor $\mathcal{W}_i$, $i \in [k]$, by building the corresponding weight matrix $W_i \in \mathbb{R}^{f \times c_i \kappa_1 \kappa_2}$, c.f. Figure 6-4, computing its j-rank approximation, and factoring it into a pair of smaller matrices U_i of f rows and j columns and V_i of j rows and $c_i \kappa_1 \kappa_2$ columns.

3. REPLACE the original layer in the network by 2 layers. The first consists of k parallel convolutions, where the i^{th} parallel layer, $i \in [k]$, is described by the tensor $\mathcal{V}_i \in \mathbb{R}^{j \times c_i \times \kappa_1 \times \kappa_2}$ which can be constructed from the matrix V_i (j filters, c_i channels, $\kappa_1 \times \kappa_2$ kernel). The second layer is constructed by reshaping each matrix U_i, $i \in [k]$, to obtain the tensor $\mathcal{U}_i \in \mathbb{R}^{f \times j \times 1 \times 1}$, and then channel stacking all k tensors $\mathcal{U}_1, \cdots, \mathcal{U}_k$ to get a single tensor of shape $f \times kj \times 1 \times 1$.

The decomposed layer is depicted in Figure 6-4. The resulting layer pair has $jc\kappa_1\kappa_2$ and jfk parameters, respectively, which implies a parameter reduction from $fc\kappa_1\kappa_2$ to $j(fk + c\kappa_1\kappa_2)$.

6.2.3 Global Network Compression

In the previous section, we introduced our layer compression scheme. We note that in practice we usually want to compress an entire network consisting of L layers up to a pre-specified relative reduction in parameters, i.e., up to some desired compression

ratio (**CR**). However, it is generally unclear how much each layer $\ell \in [L]$ should be compressed in order to achieve the desired **CR** while incurring a minimal increase in loss. Unfortunately, this optimization problem is NP-complete as we would have to check every combination of layer compression resulting in the desired **CR** in order to optimally compress each layer. On the other hand, simple heuristics, e.g., constant per-layer compression ratios, may lead to sub-optimal results, see Section 6.3. To this end, we propose an efficiently solvable global compression framework based on minimizing the maximum relative error incurred across layers. We describe each component of our optimization procedure in greater detail below.

The Layer-wise Relative Error as Proxy for the Overall Loss

Since the true cost (the additional loss incurred after compression) would result in an NP-complete problem, we replace the true cost by a more efficient proxy. Specifically, we consider the maximum relative error $\varepsilon := \max_{\ell \in [L]} \varepsilon^\ell$ across layers, where ε^ℓ denotes the theoretical maximum relative error in the ℓ^{th} layer as described in Theorem 17 below. We choose to minimize this particular cost because: (i) minimizing the maximum relative error ensures that no layer incurs an unreasonably large error that might otherwise get propagated or amplified; (ii) relying on a relative instead of an absolute error notion is preferred as scaling between layers may arbitrarily change, e.g., due to batch normalization, and thus the absolute scale of layer errors may not be indicative of the increase in loss; and (iii) the per-layer relative error has been shown to be intrinsically linked to the theoretical compression error, e.g., see the work of Arora et al. (2018) and Part I of this thesis, thus representing a natural proxy for the cost.

Definition of the Per-layer Relative Error

Let $\mathcal{W}^\ell \in \mathbb{R}^{f^\ell \times c^\ell \times \kappa_1^\ell \times \kappa_2^\ell}$ and $W^\ell \in \mathbb{R}^{f^\ell \times c^\ell \kappa_1^\ell \kappa_2^\ell}$ denote the weight tensor and corresponding folded matrix of layer ℓ, respectively. The per-layer relative error ε^ℓ is hereby defined as the relative difference in the operator norm between the matrix $\hat{W}^\ell$ (that corresponds to the compressed weight tensor $\hat{\mathcal{W}}^\ell$) and the original weight

matrix W^ℓ in layer ℓ, i.e,.

$$\varepsilon^\ell \coloneqq \|\hat{W}^\ell - W^\ell\|/\|W^\ell\|. \tag{6.1}$$

Note that while in practice our method decomposes the original layer into a set of separate layers (see Section 6.2.2), for the purpose of deriving the resulting error we re-compose the compressed layers into the overall matrix operator $\hat{W}^\ell$, i.e., $\hat{W}^\ell = [U_1^\ell V_1^\ell \cdots U_{k^\ell}^\ell V_{k^\ell}^\ell]$, where $U_i^\ell V_i^\ell$ is the factorization of the ith cluster (set of columns) in the ℓth layer, for every $\ell \in [L]$ and $i \in [k^\ell]$, see supplementary material for more details. We note that the operator norm $\|\cdot\|$ for a convolutional layer thus signifies the maximum relative error incurred for an individual output patch ("pixel") across all output channels.

Derivation of Relative Error Bounds

We now derive an error bound that enables us to describe the per-layer relative error in terms of the compression hyperparameters j^ℓ and k^ℓ, i.e., $\varepsilon^\ell = \varepsilon^\ell(k^\ell, j^\ell)$. This will prove useful later on as we have to repeatedly query the relative error in our optimization procedure. The error bound is described in the following.

Theorem 17. *Given a layer matrix W^ℓ and the corresponding low-rank approximation $\hat{W}^\ell$, the relative error $\varepsilon^\ell \coloneqq \|\hat{W}^\ell - W^\ell\|/\|W^\ell\|$ is bounded by*

$$\varepsilon^\ell \leq \sqrt{k}/\alpha_1 \cdot \max_{i\in[k]} \alpha_{i,j+1}, \tag{6.2}$$

where $\alpha_{i,j+1}$ is the $j+1$ largest singular value of the matrix W_i^ℓ, for every $i \in [k]$, and $\alpha_1 = \|W^\ell\|$ is the largest singular value of W^ℓ.

Proof. First, we recall the matrices $W_1^\ell, \cdots, W_k^\ell$ and we denote the SVD factorization for each of them by: $W_i^\ell = \tilde{U}_i^\ell \tilde{\Sigma}_i^\ell \tilde{V}_i^\ell$. Now, observe that for every $i \in [k]$, the matrix $\hat{W}_i^\ell$ is the j-rank approximation of W_i^ℓ. Hence, the SVD factorization of $\hat{W}_i^\ell$ can be writen as $\hat{W}_i^\ell = \tilde{U}_i^\ell \hat{\Sigma}_i^\ell \tilde{V}_i^{\ell^T}$, where $\hat{\Sigma}_i^\ell \in \mathbb{R}^{f\times d}$ is a diagonal matrix such that its first j-diagonal entries are equal to the first j-entries on the diagonal of $\tilde{\Sigma}_i^\ell$, and the rest

are zeros. Hence,

$$W^\ell - \hat{W}^\ell = [W_1^\ell - \hat{W}_1^\ell, \cdots, W_k^\ell - \hat{W}_k^\ell] = [\tilde{U}_1^\ell(\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \cdots, \tilde{U}_k^\ell(\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell]$$
$$= [\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell] \operatorname{diag}\left((\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \ldots, (\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell\right). \tag{6.3}$$

By (6.3) and by the triangle inequality, we have that

$$\left\|W^\ell - \hat{W}^\ell\right\| \leq \left\|\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]\right\| \left\|\operatorname{diag}\left((\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \ldots, (\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell\right)\right\|. \tag{6.4}$$

Now, we observe that

$$\left\|\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]\right\|^2 = \left\|\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]^T\right\| = \|\operatorname{diag}(k, \ldots, k)\| = k. \tag{6.5}$$

Finally, we show that

$$\left\|\operatorname{diag}\left((\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \ldots, (\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell\right)\right\| = \max_{i\in[k]} \left\|(\tilde{\Sigma}_i^\ell - \hat{\Sigma}_i^\ell)\tilde{V}_i^\ell\right\| \tag{6.6}$$
$$= \max_{i\in[k]} \left\|(\tilde{\Sigma}_i^\ell - \hat{\Sigma}_i^\ell)\right\| = \max_{i\in[k]} \alpha_{i,j+1}, \tag{6.7}$$

where the second equality holds since the columns of V are orthogonal and the last equality holds according to the Eckhart-Young-Mirsky Theorem (Theorem 2.4.8 of Golub and Van Loan (2013)). Plugging (6.7) and (6.5) into (6.4) concludes the proof. □

Resulting Network Size

Let $\theta = \{\mathcal{W}^\ell\}_{\ell=1}^L$ denote the set of weights for the L layers and note that the number of parameters in layer ℓ is given by $\left|\mathcal{W}^\ell\right| = f^\ell c^\ell \kappa_1^\ell \kappa_2^\ell$ and $|\theta| = \sum_{\ell\in[L]} |\mathcal{W}^\ell|$. Moreover, note that $|\hat{\mathcal{W}}^\ell| = j^\ell(k^\ell f^\ell + c^\ell \kappa_1^\ell \kappa_2^\ell)$ if decomposed, $\hat{\theta} = \{\hat{\mathcal{W}}^\ell\}_{\ell=1}^L$, and $\left|\hat{\theta}\right| = \sum_{\ell\in[L]} \left|\hat{\mathcal{W}}^\ell\right|$. The overall compression ratio is thus given by $1 - |\hat{\theta}|/|\theta|$ where we neglected other parameters for ease of exposition. Observe that the layer budget $\left|\hat{\mathcal{W}}^\ell\right|$ is fully determined by k^ℓ, j^ℓ just like the error bound.

Global Network Compression

Putting everything together we obtain the following formulation for the optimal per-layer budget:

$$\begin{aligned} \varepsilon_{opt} = \min_{\{j^\ell, k^\ell\}_{\ell=1}^{L}} \quad & \max_{\ell \in [L]} \varepsilon^\ell(k^\ell, j^\ell) \\ \text{subject to} \quad & 1 - \left|\hat{\theta}(k^1, j^1, \ldots, k^L, j^L)\right| / |\theta| \leq \mathbf{CR}, \end{aligned} \tag{6.8}$$

where **CR** denotes the desired overall compression ratio. Thus optimally allocating a per-layer budget entails finding the optimal number of subspaces k^ℓ and ranks j^ℓ for each layer constrained by the desired overall **CR**.

6.2.4 Automatic Layer-wise Decomposition Selector (ALDS)

Overview

We propose to solve (6.8) by iteratively optimizing $k^1, \ldots, k^L$ and $j^1, \ldots, j^L$ until convergence akin of an EM-like algorithm as shown in Algorithm 4 and Figure 6-1. Specifically, for a given set of weights θ and desired compression ratio **CR** we first randomly initialize the number of subspaces $k^1, \ldots, k^L$ for each layer (Line 2). Based on given values for each k^ℓ we then solve for the optimal ranks $j^1, \ldots, j^L$ such that the overall **CR** is satisfied (Line 4). Note that the maximum error ε is minimized if all errors are equal. Thus solving for the ranks in Line 4 entails guessing a value for ε, computing the resulting network size, and repeating the process until the desired **CR** is satisfied, e.g. via binary search. Subsequently, we re-assign the number of subspaces k^ℓ for each layer by iterating through the finite set of possible values for k^ℓ (Line 7) and choosing the one that minimizes the relative error for the current layer budget b^ℓ (computed in Line 6). We then iteratively repeat both steps until convergence (Lines 3-8). To improve the quality of the local optimum we initialize the procedure with multiple random seeds (Lines 1-11) and pick the allocation with the lowest error (Line 12). We note that we make repeated calls to our decomposition subroutine (i.e. SVD; Lines 4, 7) highlighting the necessity for it to be efficient.

Algorithm 4 ALDS(θ, **CR**, n_{seed})

Input: θ: network parameters; **CR**: overall compression ratio; n_{seed}: number of random seeds to initialize

Output: $k^1, \ldots, k^L$: number of subspaces for each layer; $j_1, \ldots, j^L$: desired rank per subspace for each layer

1: **for** $i \in [n_{\text{seed}}]$ **do**
2: $k^1, \ldots, k^L \leftarrow$ RandomInit()
3: **while** not converged **do**
4: $j^1, \ldots, j^L \leftarrow$ OptimalRanks(**CR**, $k^1, \ldots, k^L$) ▷ Global step: choose such that $\varepsilon^1 = \ldots = \varepsilon^L$
5: **for** $\ell \in [L]$ **do**
6: $b^\ell \leftarrow j^\ell(k^\ell f^\ell + c^\ell \kappa_1^\ell \kappa_2^\ell)$ ▷ resulting layer budget
7: $k^\ell \leftarrow$ OptimalSubspaces(b^ℓ) ▷ Local step: minimize error bound for a given layer budget
8: **end for**
9: **end while**
10: $\varepsilon_i =$ RecordError($k^1, \ldots, k^L, j^1, \ldots, j^L$)
11: **end for**
12: **return** $k^1, \ldots, k^L, j^1, \ldots, j^L$ from $i_{\text{best}} = \text{argmin}_i \, \varepsilon_i$

Implementation Details

Our suggested algorithm aims at minimizing the maximum relative error $\varepsilon := \max_{\ell \in [L]} \varepsilon^\ell$ across the L layers of the network as a proxy for the true cost, where ε^ℓ is the theoretical maximum relative error in the ℓ^{th}:

$$\varepsilon^\ell := \frac{\left\| \hat{W}^\ell - W^\ell \right\|}{\|W^\ell\|}.$$

Through Algorithm 4, for every $\ell \in [L]$ we need to repeatedly compute ε^ℓ as a function of j^ℓ and k^ℓ. At Line 4, we are given a guess for the optimal values of $k^1, \ldots, k^L$, and our goal is to compute the values $j^1, \ldots, j^L$ such that the resulting errors $\varepsilon^1, \ldots, \varepsilon^L$ are (approximately) equal in order to minimize the maximum error $\max_{\ell \in [L]} \varepsilon^\ell$ while achieving the desired global compression ratio. To this end, we guess a value for ε and for given $k^1, \ldots, k^L$ pick the corresponding $j^1, \ldots, j^L$ such that ε constitutes a tight upper bound for the relative error in each layer. Based on the now resulting budget (and consequently compression ratio) we can now improve our

guess of ε, e.g., via binary search or other types of root finding algorithms, until we convergence to a value of ε that corresponds to our desired overall compression ratio.

Subsequently, for each layer we are given specific values of k^ℓ and j^ℓ, which implies that we are given a budget b^ℓ for every layer $\ell \in [L]$. Subsequently, we re-assign the number of subspaces k^ℓ and their ranks j^ℓ for each layer by iterating through the finite set of possible values for k^ℓ (Line 7) and choosing the combination of j^ℓ, k^ℓ that minimizes the relative error for the current layer budget b^ℓ (computed in Line 6).

We then iteratively repeat both steps until convergence (Lines 3-8).

Hence, instead of computing the cost of each layer at each step, we can save a lookup table that stores the errors ε^ℓ for the possible values of k^ℓ and j^ℓ of each layer. For every layer $\ell \in [L]$, we iterate over the finite set of values of k^ℓ, and we split the matrix W^ℓ to k^ℓ matrices (according to the channel slicing approach that is explained in Section 6.2.2), then we compute the SVD-factorization of each matrix from these k^ℓ matrices, and finally, compute ε^ℓ that corresponds to a specific j^ℓ (k^ℓ is already given) in $O(fd)$ time, where f is the number of rows in the weight matrix that corresponds to the ℓth layer and d is the number of columns.

Furthermore, instead of computing each option of ε^ℓ in $O(fd)$ time, we use the derived upper bound to compute it in $O(k)$ time and saving it in the lookup table. This is done to ensure a more efficient implementation of the lookup table, and having this table ensures a more efficient implementation of Algorithm 4.

Extensions

Here, we use SVD with multiple subspaces as per-layer compression method. However, we note that ALDS can be readily extended to any desired *set* of low-rank compression techniques. Specifically, we can replace the local step of Line 7 by a search over different methods, e.g., Tucker decomposition, PCA, or other SVD compression schemes, and return the best method for a given budget. In general, we may combine ALDS with any low-rank compression as long as we can efficiently evaluate the per-layer error of the compression scheme. Note that this essentially equips us with a framework to automatically choose the per-layer decomposition technique fully

automatically.

To this end, we test an extension of ALDS where in addition to searching over multiple values of k^ℓ we simultaneously search over various flattening schemes to convert a convolutional tensor to a matrix before applying SVD.

As before, let $\mathcal{W} \in \mathbb{R}^{f \times c \times \kappa_1 \times \kappa_2}$ denote the weight tensor for a convolutional layer with f filters, c input channels, and a $\kappa_1 \times \kappa_2$ kernel. Moreover, let j denote the desired rank of the decomposition. We consider the following schemes to automatically search over:

- SCHEME 0: flatten the tensor to a matrix of shape $f \times c\kappa_1\kappa_2$. The decomposed layers correspond to a $j \times c \times \kappa_1 \times \kappa_2$-convolution followed by a $f \times j \times 1 \times 1$-convolution. This is the same scheme as used in ALDS.

- SCHEME 1: flatten the tensor to a matrix of shape $f\kappa_1 \times c\kappa_2$. The decomposed layers correspond to a $j \times c \times 1 \times \kappa_2$-convolution followed by a $f \times j \times \kappa_1 \times 1$-convolution.

- SCHEME 2: flatten the tensor to a matrix of shape $f\kappa_2 \times c\kappa_1$. The decomposed layers correspond to a $j \times c \times \kappa_1 \times 1$-convolution followed by a $f \times j \times 1 \times \kappa_2$-convolution.

- SCHEME 3: flatten the tensor to a matrix of shape $f\kappa_1\kappa_2 \times c$. The decomposed layers correspond to a $j \times c \times 1 \times 1$-convolution followed by a $f \times j \times \kappa_1 \times \kappa_2$-convolution.

We denote this method by ALDS+ and provide preliminary results in Section 6.3.5. We note that since ALDS+ is a generalization of ALDS its performance is at least as good as the original ALDS. Moreover, our preliminary results actually suggest that the extension clearly improves upon the empirical performance of ALDS.

6.3 Results

6.3.1 Experimental Setup

Our experimental evaluations are based on a variety of network architectures, data sets, and compression pipelines. In the following, we provide all necessary hyperparameters to reproduce our experiments for each of the datasets and respective network architectures.

All networks were trained, compressed, and evaluated on a compute cluster with NVIDIA Titan RTX and NVIDIA RTX 2080Ti GPUs. The experiments were conducted with PyTorch 1.7 and our code is fully open-sourced.

All networks are trained according to the hyperparameters outlined in the respective original papers. During retraining, we reuse the same hyperparameters.

Moreover, each experiment is repeated 3 times and we report mean and mean, standard deviation in the tables and figures, respectively.

For each data set, we use the publicly available development set as test set and use a 90%/5%/5% split on the train set to obtain a separate train and *two* validation sets. One validation set is used for data-dependent compression methods, e.g., PCA (Zhang et al., 2015a); the other set is used for early stopping during training.

Networks and datasets

We study various standard network architectures and data sets. Particularly, we test our compression framework on ResNet20 (He et al., 2016), DenseNet22 (Huang et al., 2017), WRN16-8 (Zagoruyko and Komodakis, 2016), and VGG16 (Simonyan and Zisserman, 2015) on CIFAR-10 (Torralba et al., 2008); ResNet18 (He et al., 2016), and AlexNet (Krizhevsky et al., 2012) on ImageNet (Russakovsky et al., 2015); and on Deeplab-V3 (Chen et al., 2017) with a ResNet50 backbone on Pascal VOC segmentation data (Everingham et al., 2015).

Baselines

We compare ALDS to a diverse set of low-rank compression techniques. Specifically, we have implemented PCA (Zhang et al., 2015b), SVD with energy-based layer allocation (SVD-Energy) following Alvarez and Salzmann (2017); Wen et al. (2017), and simple SVD with constant per-layer compression (Denton et al., 2014). Additionally, we also implemented the recent learned rank selection mechanism (L-Rank) of Idelbayev and Carreira-Perpinán (2020). Finally, we implemented two recent filter pruning methods, i.e., FT of Li et al. (2016) and PFP from Chapter 5, as alternative compression techniques for densely compressed networks. Each baseline comparison method is described in more detail below:

1. PCA (Zhang et al., 2015a) decomposes each layer based on principle component analysis of the pre-activation (output of linear layer). We implement the symmetric, linear version of their method. The per-layer compression ratio is based on the greedy solution for minimizing the product of the per-layer energy, where the energy is defined as the sum of singular values in the compressed layer, see Equation (14) of Zhang et al. (2015a).

2. SVD-Energy (Alvarez and Salzmann, 2017; Wen et al., 2017) decomposes each layer via matrix folding akin to our SVD-based decomposition. The per-layer compression ratio is found by keeping the relative energy reduction constant across layers, where energy is defined as the sum of squared singular values.

3. SVD (Denton et al., 2014) decomposes each layer via matrix folding akin to our SVD-based decomposition. However, we hereby fix $k^\ell = 1$ for all layers $\ell \in [L]$ in order to provide a nominal comparison akin of "standard" tensor decomposition. The per-layer compression ratio is kept constant across all layers.

4. L-Rank (Idelbayev and Carreira-Perpinán, 2020) decomposes each layer via matrix folding akin to our SVD-based decomposition. The per-layer compression is determined by minimizing a joint cost objective of the energy and the

computational cost of each layer, see Equation (5) of Idelbayev and Carreira-Perpinán (2020) for details.

5. FT (Li et al., 2016) prunes the filters (or neurons) in each layer with the lowest element-wise ℓ_2-norm. The per-layer compression ratio is set manually (constant in our implementation).

6. PFP (Chapter 5) prunes the channels with the lowest sensitivity, where the data-dependent sensitivities are based on a provable notion of channel pruning. The per-layer prune ratio is determined based on the associated theoretical error guarantees.

Additional comparisons on ImageNet are provided in Section 6.3.3.

Retraining

For our experiments, we study one-shot and iterative learning rate rewinding inspired by Renda et al. (2020) for various amounts of retraining. In particular, we consider the following unified compress-retrain pipeline across all methods:

1. Train for e epochs according to the standard training schedule for the respective network.

2. Compress the network according to the chosen method.

3. Retrain the network for r epochs using the training hyperparameters from epochs $[e - r, e]$.

4. Iteratively repeat 1.-3. after projecting the decomposed layers back (optional).

Reporting Metrics

We report Top-1, Top-5, and IoU test accuracy as applicable for the respective task. For each compressed network we also report the compression ratio, i.e., relative reduction, in terms of parameters and floating point operations denoted by CR-P and

Table 6.1: The experimental hyperparameters for training, compression, and retraining for the tested CIFAR-10 network architectures. "LR" and "LR decay" hereby denote the learning and the (multiplicative) learning rate decay, respectively, that is deployed at the epochs as specified. "$\{x, \dots\}$" indicates that the learning rate is decayed every x epochs.

	Hyperparameters		VGG16	Resnet20	DenseNet22	WRN-16-8
CIFAR-10	(Re-)Training	Test accuracy (%)	92.81	91.4	89.90	95.19
		Loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
		Optimizer	SGD	SGD	SGD	SGD
		Epochs	300	182	300	200
		Warm-up	10	5	10	5
		Batch size	256	128	64	128
		LR	0.05	0.1	0.1	0.1
		LR decay	0.5@{30, ...}	0.1@{91, 136}	0.1@{150, 225}	0.2@{60, ...}
		Momentum	0.9	0.9	0.9	0.9
		Nesterov	✗	✗	✓	✓
		Weight decay	5.0e-4	1.0e-4	1.0e-4	5.0e-4
	Compression	α	0.80	0.80	0.80	0.80
		n_{seed}	15	15	15	15

CR-F, respectively. Each experiment was repeated 3 times and we report mean and standard deviation.

Relevant Hyperparameters

CIFAR-10. All relevant hyperparameters are outlined in Table 6.1. We add a warmup period in the beginning where we linearly scale up the learning rate from 0 to the nominal learning rate to ensure proper training performance in distributed training settings (Goyal et al., 2017). During training we use the standard data augmentation strategy for CIFAR: (1) zero padding from 32x32 to 36x36; (2) random crop to 32x32; (3) random horizontal flip; (4) channel-wise normalization. During inference only the normalization (4) is applied. The compression ratios are chosen according to a geometric sequence with the common ratio denoted by α in Table 6.1, i.e., the compression ratio for iteration i is determined by $1 - \alpha^i$. The compression parameter n_{seed} denotes the number of seeds used to initialize Algorithm 4 for compressing with ALDS.

ImageNet. We report the relevant hyperparameters in Table 6.2. For ImageNet, we consider the networks architectures Resnet18 (He et al., 2016) and AlexNet (Krizhevsky

Table 6.2: The experimental hyperparameters for training, compression, and retraining for the tested ImageNet network architectures. "LR" and "LR decay" hereby denote the learning and the (multiplicative) learning rate decay, respectively, that is deployed at the epochs as specified. "$\{x, \ldots\}$" indicates that the learning rate is decayed every x epochs.

	Hyperparameters		ResNet18	AlexNet
ImageNet	(Re-)Training	Top 1 Test accuracy (%)	69.64	57.30
		Top 5 Test accuracy (%)	88.98	80.20
		Loss	cross-entropy	cross-entropy
		Optimizer	SGD	SGD
		Epochs	90	90
		Warm-up	5	5
		Batch size	256	256
		LR	0.1	0.1
		LR decay	0.1@{30, 60, 80}	0.1@{30, 60, 80}
		Momentum	0.9	0.9
		Nesterov	✗	✗
		Weight decay	1.0e-4	1.0e-4
	Compression	α	0.80	0.80
		n_{seed}	15	15

et al., 2012). During training we use the following data augmentation: (1) randomly resize and crop to 224x224; (2) random horizontal flip; (3) channel-wise normalization. During inference, we use a center crop to 224x224 before (3) is applied.

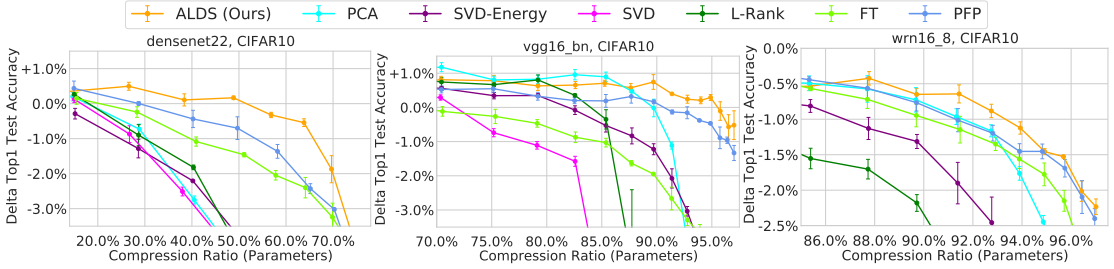
Pascal VOC. We consider the segmentation task from Pascal VOC 2012 (Everingham et al., 2015). We augment the nominal data training data using the extra labels as provided by Hariharan et al. (2011). As network architecture we consider a DeeplabV3 (Chen et al., 2017) with ResNet50 backbone pre-trained on ImageNet. During training we use the following data augmentation pipeline: (1) randomly resize (256x256 to 1024x1024) and crop to 513x513; (2) random horizontal flip; (3) channel-wise normalization. During inference, we resize to 513x513 exactly before the normalization (3) is applied. The experimental hyperparameters are summarized in Table 6.3.

Table 6.3: The experimental hyperparameters for training, compression, and retraining for the tested VOC network architecture. "LR" and "LR decay" hereby denote the learning and the learning rate decay, respectively. Note that the learning rate is polynomially decayed after each step.

	Hyperparameters		DeeplabV3-ResNet50
Pascal VOC 2012 – Segmentation	(Re-)Training	IoU Test accuracy (%)	69.84
		Top 1 Test accuracy (%)	94.25
		Loss	cross-entropy
		Optimizer	SGD
		Epochs	45
		Warm-up	0
		Batch size	32
		LR	0.02
		LR decay	(1 - "step"/"total steps")$^{0.9}$
		Momentum	0.9
		Nesterov	✗
		Weight decay	1.0e-4
	Compression	α	0.80
		n_{seed}	15

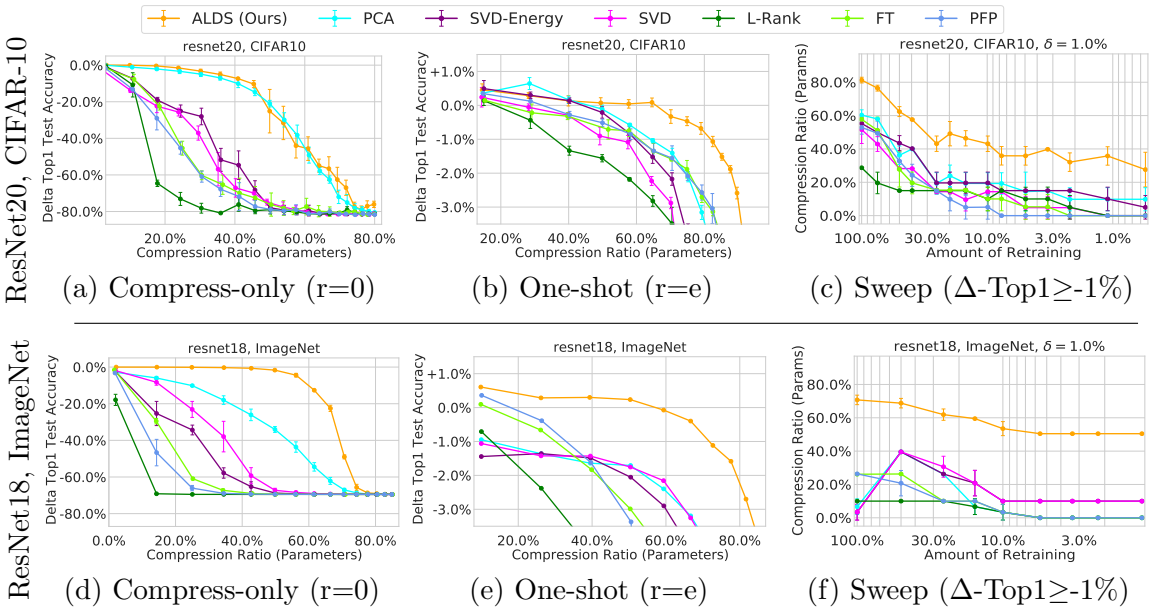
6.3.2 One-shot Compression with Baselines

We train reference networks on CIFAR-10, ImageNet, and VOC, and then compress and retrain the networks *once* with $r = e$ for various baseline comparisons and compression ratios. In Figure 6-5, we provide results for DenseNet22, VGG16, and WRN16-8 on CIFAR-10. Notably, our approach is able to outperform existing baselines approaches across a wide range of tested compression ratios. Specifically, in the region where the networks incur only minimal drop in accuracy (Δ-Top1$\geq -1\%$) ALDS is particularly effective.



(a) DenseNet22 (b) VGG16 (c) WRN16-8

Figure 6-5: One-shot compress+retrain experiments on CIFAR-10 with baseline comparisons.



(a) Compress-only (r=0) (b) One-shot (r=e) (c) Sweep (Δ-Top1≥-1%)

(d) Compress-only (r=0) (e) One-shot (r=e) (f) Sweep (Δ-Top1≥-1%)

Figure 6-6: The size-accuracy trade-off for various compression ratios, methods, and networks. Compression was performed after training and networks were re-trained once for the indicated amount (**one-shot**). (a, b, d, e): the difference in test accuracy for fixed amounts of retraining. (c, f): the maximal compression ratio with less-than-1% accuracy drop for variable amounts of retraining.

Moreover, we tested ALDS on ResNet20 (CIFAR-10) and ResNet18 (ImageNet) as shown in Figure 6-6. For these experiments, we performed a grid search over both multiple compression ratios and amounts of retraining. Here, we highlight that ALDS outperforms baseline approaches even with significantly less retraining. On Resnet 18 (ImageNet) ALDS can compress over 50% of the parameters with minimal retraining (1% retraining) and a less-than-1% accuracy drop compared to the best comparison methods (40% compression with 50% retraining).

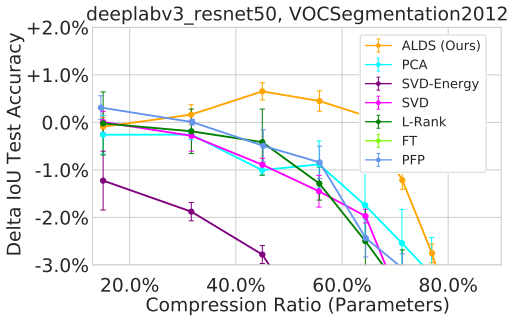


Figure 6-7: One-shot compress+retrain for DeeplabV3-ResNet50 on VOC.

Finally, we tested the same setup on a DeeplabV3 with a ResNet50 backbone trained on Pascal VOC 2012 segmentation data, see Figure 6-7. We note that ALDS consistently outperforms other baselines methods in this setting as well (60% CR-P

Table 6.4: Baseline results for Δ-Top1≥−0.5% for one-shot. Results coincide with Figures 6-5, 6-6, 6-7.

	Model	Metric	ALDS (Ours)	PCA	SVD-Energy	SVD	L-Rank	FT	PFP
CIFAR-10	ResNet20 Top1: 91.39	Δ-Top1	-0.47	-0.11	-0.21	-0.29	-0.44	-0.32	-0.28
		CR-P, CR-F	**74.91**, **67.86**	49.88, 48.67	49.88, 49.08	39.81, 38.95	28.71, 54.89	39.69, 39.57	40.28, 30.06
	VGG16 Top1: 92.78	Δ-Top1	-0.11	-0.02	-0.08	+0.29	-0.35	-0.47	-0.47
		CR-P, CR-F	**95.77**, **86.23**	89.72, 85.84	82.57, 81.32	70.35, 70.13	85.38, 75.86	79.13, 78.44	94.87, 84.76
	DenseNet22 Top1: 89.88	Δ-Top1	-0.32	+0.20	-0.29	+0.13	+0.26	-0.24	-0.44
		CR-P, CR-F	**56.84**, **61.98**	14.67, 34.55	15.16, 19.34	15.00, 15.33	14.98, 35.21	28.33, 29.50	40.24, 43.37
	WRN16-8 Top1: 89.88	Δ-Top1	-0.42	-0.49	-0.41	-0.96	-0.45	-0.32	-0.44
		CR-P, CR-F	**87.77**, 79.90	85.33, **83.45**	64.75, 60.94	40.20, 39.97	49.86, 58.00	82.33, 75.97	85.33, 80.68
ImageNet	ResNet18 Top1: 69.62, Top5: 89.08	Δ-Top1, Top5	-0.40, -0.05	-0.95,-0.37	-1.49, -0.64	-1.75, -0.72	-0.71, -0.23	+0.10, +0.42	-0.39, -0.08
		CR-P, CR-F	**66.70**, **43.51**	9.99, 12.78	39.56, 40.99	50.38, 50.37	10.01, 32.64	9.86, 11.17	26.35, 17.96
VOC	DeeplabV3 IoU: 91.39 Top1: 99.34	Δ-IoU, Top1	+0.14, -0.15	-0.26, -0.02	-1.88, -0.47	-0.28, -0.18	-0.42, -0.09	-4.30, -0.91	-0.49, -0.21
		CR-P, CR-F	**64.38**, **64.11**	55.68, 55.82	31.61, 32.27	31.64, 31.51	44.99, 45.02	15.00, 15.06	45.17, 43.93

vs. 20% without accuracy drop).

Our one-shot results are again summarized in Table 6.4 where we report CR-P and CR-F for Δ-Top1≥−0.5%. We note that pruning usually takes on the order of seconds and minutes for CIFAR and ImageNet, respectively, which is usually faster than even a single training epoch.

6.3.3 ImageNet Benchmarks

Next, we test our framework on two common ImageNet benchmarks, ResNet18 and AlexNet. We follow the compress-retrain pipeline outlined in the beginning of the section and repeat it iteratively to obtain higher compression ratios. Specifically, after retraining and before the next compression step we project the decomposed layers back to the original layer. This way, we avoid recursing on the decomposed layers.

Our results are reported in Table 6.5 where we compare to a wide variety of available compression benchmarks (results were adapted directly from the respective papers). The middle part and bottom part of the table for each network are organized into low-rank compression and filter pruning approaches, respectively. Note that the reported differences in accuracy (Δ-Top1 and Δ-Top5) are relative to our baseline accuracies. On ResNet18 we can reduce the number of FLOPs by 65% with minimal drop in accuracy compared to the best competing method (MUSCO, 58.67%). With

Table 6.5: AlexNet and ResNet18 Benchmarks on ImageNet. We report Top-1, Top-5 accuracy and percentage reduction in terms of parameters and FLOPs denoted by CR-P and CR-F, respectively. Best results with less than 0.5% accuracy drop are bolded.

	Method	Δ-Top1	Δ-Top5	CR-P (%)	CR-F (%)
ResNet18, Top1, 5: 69.64%, 88.98%	ALDS (Ours)	+0.41	+0.37	66.70	42.70
	ALDS (Ours)	**-0.38**	**+0.04**	**75.00**	**64.50**
	ALDS (Ours)	-0.90	-0.25	78.50	71.50
	ALDS (Ours)	-1.37	-0.56	80.60	76.30
	MUSCO (Gusak et al., 2019)	**-0.37**	**-0.20**	N/A	58.67
	TRP1 (Xu et al., 2020)	-4.18	-2.5	N/A	44.70
	TRP1+Nu (Xu et al., 2020)	-4.25	-2.61	N/A	55.15
	TRP2+Nu (Xu et al., 2020)	-4.3	-2.37	N/A	68.55
	PCA (Zhang et al., 2015b)	-6.54	-4.54	N/A	29.07
	Expand (Jaderberg et al., 2014)	-6.84	-5.26	N/A	50.00
	PFP (Liebenwein et al., 2020)	-2.26	-1.07	43.80	29.30
	SoftNet (He et al., 2018)	-2.54	-1.2	N/A	41.80
	Median (He et al., 2019)	-1.23	-0.5	N/A	41.80
	Slimming (Liu et al., 2017)	-1.77	-1.19	N/A	28.05
	Low-cost (Dong et al., 2017b)	-3.55	-2.2	N/A	34.64
	Gating (Hua et al., 2018)	-1.52	-0.93	N/A	37.88
	FT (He et al., 2017)	-3.08	-1.75	N/A	41.86
	DCP (Zhuang et al., 2018)	-2.19	-1.28	N/A	47.08
	FBS (Gao et al., 2018)	-2.44	-1.36	N/A	49.49
AlexNet, Top1, 5: 57.30%, 80.20%	ALDS (Ours)	+0.10	+0.45	92.00	76.10
	ALDS (Ours)	-0.21	-0.36	93.0	77.9
	ALDS (Ours)	**-0.41**	**-0.54**	**93.50**	**81.4**
	Tucker (Kim et al., 2015a)	N/A	-1.87	N/A	62.40
	Regularize (Tai et al., 2015)	N/A	-0.54	N/A	74.35
	Coordinate (Wen et al., 2017)	N/A	-0.34	N/A	62.82
	Efficient (Kim et al., 2019)	-0.7	-0.3	N/A	62.40
	L-Rank (Idelbayev et al., 2020)	**-0.13**	**-0.13**	N/A	**66.77**
	NISP (Yu et al., 2018b)	-1.43	N/A	N/A	67.94
	OICSR (Li et al., 2019a)	-0.47	N/A	N/A	53.70
	Oracle (Ding et al., 2019)	-1.13	-0.67	N/A	31.97

a slightly higher drop in accuracy (-1.37%) we can even compress 76% of FLOPs. On AlexNet, our framework finds networks with -0.21% and -0.41% difference in accuracy with over 77% and 81% fewer FLOPs. This constitutes a more-than-10% improvement in terms of FLOPs compared to current state-of-the-art (L-Rank) for similar accuracy drops.

6.3.4 Ablation Study

In order to gain a better understanding of the various aspects of our method we consider an ablation study where we selectively turn off various features of ALDS. Specifically, we compare the full version of ALDS to the following variants:

1. ALDS-Error solves for the optimal ranks (Line 4 of Algorithm 4) for a desired set of values for $k^1, \ldots, k^L$. We test $k^\ell = 3, \forall \ell \in [L]$. This variant tests the benefits of varying the number of subspaces compared to fixing them to a desired value.

2. SVD-Error corresponds to ALDS-Error with $k^\ell = 1, \forall \ell \in [L]$. This variants tests the benefits of having multiple subspaces in the first places in the context error-based allocation of the per-layer compression ratio.

3. ALDS-Simple picks the ranks in each layer for a desired set of values of $k^1, \ldots, k^L$ such that the per-layer compression ratio is constant. We test $k^\ell = 3, \forall \ell \in [L]$, and $k^\ell = 5, \forall \ell \in [L]$. This variant tests the benefits of allocating the per-layer compression ratio according to the layer error compared to a simple constant heuristic.

4. Messi proceeds like ALDS-Simple but replaces the subspace clustering with projective clustering (Maalouf et al., 2021). We test $k^\ell = 3, \forall \ell \in [L]$. This variant tests the disadvantages of having a simple subspace clustering technique (channel slicing) compared to using a more sophisticated technique.

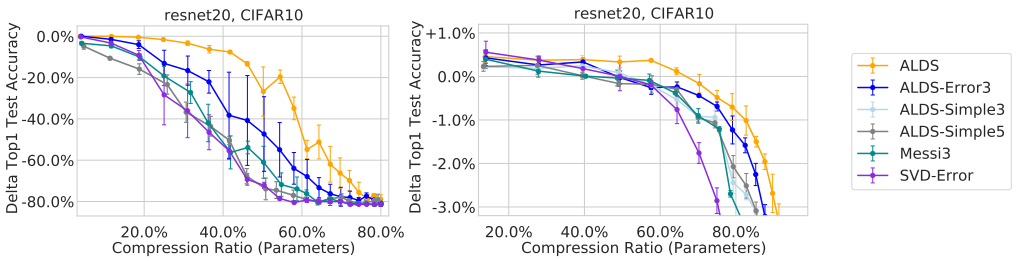
We note that ALDS-Simple with $k^\ell = 1, \forall \ell \in [L]$ corresponds to the SVD comparison method from the previous sections.

We study the variations on a ResNet20 trained on CIFAR-10 in two settings: compression only and one-shot compress+retrain. The results are presented in Figures 6-8. We highlight that the complete variant of our algorithm (ALDS) consistently outperforms the weaker variants providing empirical evidence on the effectiveness of each of the core components of ALDS.

We note that varying the number of subspaces for each layer in order to optimally assign a value of k^ℓ in each layer is crucial in improving our performance. This is apparent from the comparison between ALDS, ALDS-Error, and SVD-Error: having a fixed value for k yields sub-optimal results.

Picking an appropriate notion of cost (maximum relative error) is furthermore preferred over simple heuristics such a constant per-layer compression ratio. Specifically, the main difference between ALDS-Error and ALDS-Simple is the way how the ranks are determined for a given set of k's: ALDS-Error optimizes for the error-based cost function while ALDS-Simple relies on a simple constant per-layer compression ratio heuristic. In practice, ALDS-Error outperforms ALDS-Simple across all tested scenarios.

Finally, we test the disadvantages of using a simple subspace clustering method. To this end, we compare ALDS-Simple and Messi for fixed values of k. While in some scenarios, particularly without retraining, Messi provides modest improvements over ALDS-Simple, the improvement is negligible for most settings. Moreover, note that Messi requires an expensive approximation algorithm as explained in Section 6.2.2. This would in turn prevent us from incorporating Messi into the full ALDS framework in a computationally efficient manner. However, as apparent from the ablation study we exhibit the most performance gains for features related to global considerations instead of local, per-layer improvements. In addition, we should also note that Messi does not emit a structured reparameterization thus requires specialized software or hardware to obtain speed-ups. Consequently, we may conclude that channel slicing is the appropriate clustering technique in our context.



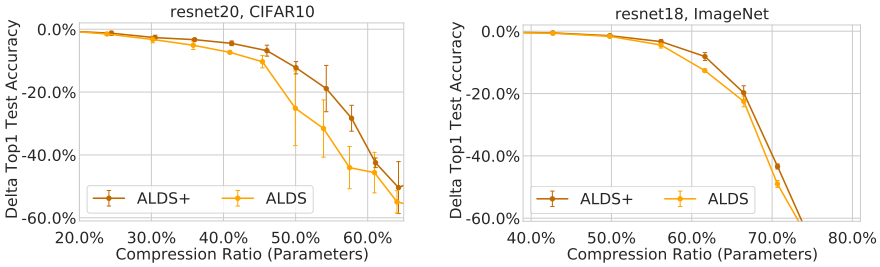
(a) no retraining ($r = 0$) (b) one-shot ($r = e$)

Figure 6-8: The difference in test accuracy ("Delta Top1 Test Accuracy") for various target compression ratios, ALDS-based/ALDS-related methods, and networks on CIFAR-10.

6.3.5 Extensions of ALDS

We test and compare ALDS with ALDS+ (see Section 6.2.4) to investigate the performance gains we can obtain from generalizing our local step to search over multiple decomposition schemes. We run one-shot compress-only experiments on ResNet20 (CIFAR-10) and ResNet18 (ImageNet).

The results are shown in Figure 6-9. We find that ALDS+ can significantly increase the performance-size trade-off compared to our standards ALDS method. This is expected since by generalizing the local step of ALDS we are increasing the search space of possible decomposition solution. Using our ALDS framework we can efficiently and automatically search over the increases solution space. We envision that our observations will invigorate future research into the possibility of not only choosing the optimal per-layer compression ratio but also the optimal compression scheme.



(a) ResNet20 (CIFAR-10) (b) ResNet18 (ImageNet)

Figure 6-9: The difference in test accuracy ("Delta Top1 Test Accuracy") for various target compression ratios, ALDS-based/ALDS-related methods, and networks on CIFAR-10. The networks were compressed once and not retrained afterwards.

6.4 Discussion

Practical benefits. By conducting a wide variety of experiments across multiple datasets and networks we have shown the effectiveness and versatility of our compression framework compared to existing methods. The runtime of ALDS is negligible compared to retraining and it can thus be efficiently incorporated into compress-retrain pipelines.

ALDS as modular compression framework. By separately considering the low-rank compression scheme for each layer (local step) and the actual low-rank compression (global step) we have provided a framework that can efficiently search over a set of desired hyperparameters that describe the low-rank compression. Our framework can be viewed as a natural generalization of the type of pruning problem we have encountered in the previous chapters and may serve as a potent plug-and-play solution for multiple different types of pruning or compression.

Error bounds lead to global insights. At the core of our contribution is our error analysis that enables us to link the global and local aspects of layer-wise compression techniques. We leverage our error bounds in practice to compress networks more effectively via an automated rank selection procedure without additional tedious hyperparameter tuning. However, we also have to rely on a proxy definition (maximum relative error) of the compression error to enable a tractable solution that we can implement efficiently. We hope these observations invigorate future research into compression techniques that come with tight error bounds – potentially even considering retraining – which can then naturally be wrapped into a global compression framework.

Part III

Applications

Chapter 7

Pruning Beyond Test Accuracy

7.1 Overview

In previous chapters, we focus our efforts on finding the most efficient parameterization of a given neural network under the constraint that we want to achieve approximately commensurate test accuracy compared to the original, unpruned network. Recall that a prototypical pruning pipeline with retraining may consist of the following steps:

1. Prune weights ("unstructured pruning") or filters/neurons ("structured pruning") from the trained network according to some criterion of importance;

2. Retrain the resulting network to regain the full accuracy;

3. Iteratively repeat steps 1 & 2 to further reduce the size.

In other words, successfully pruning a network entails identifying and removing the redundant parameters. Moreover, starting from a pre-trained, overparameterized network with good performance (as opposed to a small, randomly initialized network) ensures that the pruned network maintains the same level of performance as its uncompressed counterpart.

In this chapter, we revisit these common assumptions and rigorously assess how pruning using state-of-the-art prune-retrain techniques (Baykal et al., 2021b; Lieben-

wein et al., 2020; Renda et al., 2020) affects the function represented by a neural network, including the similarities and disparities exhibited by a pruned network with respect to its unpruned counterpart.

We formalize the notion of (functional) similarities between networks by introducing novel types of classification-based functional distance metrics. Using these metrics, we test the hypothesis that pruned models are (functionally) similar to their (unpruned) parent network and can be reliably distinguished from separately trained networks. We term the network's ability to be pruned for a particular task without performance decrease its *prune potential*, i.e., the maximal prune ratio for which the pruned network maintains its original performance, and test a network's prune potential under various tasks. The prune potential provides insights into the amount of overparameterization the network exhibits for a particular task and thus serves as a useful indicator of how much of the network can be safely pruned.

Our findings. We find that the pruned models are functionally similar to the uncompressed parent model, which enables us to distinguish the parent of a pruned network for a range of prune ratios. Despite the similarity between the pruned network and its parent, we observe that the prune potential of the network varies significantly for a large number of tasks. That is, a pruned model may be of similar predictive power as the original one when it comes to test accuracy, but may be much more brittle when faced with out of distribution data points. This raises concerns about deploying pruned models on the basis of accuracy alone, in particular for safety-critical applications such as autonomous driving (Schwarting et al., 2020), where unforeseen, out-of-distribution, or noisy data points commonly arise. Our insights, which hold even when considering robust training objectives, underscore the need to consider task-specific evaluation metrics during pruning, prior to the deployment of a pruned network to, e.g., safety-critical systems. These results also question the common assumption that there exists a significant amount of "redundant" parameters to begin with and provide a robust framework to measure the amount of genuine overparameterization in networks.

Guidelines. Based on these observations we formulate a set of easy-to-follow guidelines to pruning in practice:

1. Don't prune if unexpected shifts in the data distribution may occur during deployment.

2. Prune moderately if you have partial knowledge of the distribution shifts during training and pruning.

3. Prune to the full extent if you can account for all shifts in the data distribution during training and pruning.

4. Maximize the prune potential by explicitly considering data augmentation during retraining.

7.1.1 Contributions

- We propose novel functional distance metrics for classification-based neural networks and investigate the functional similarities between the pruned network and its unpruned counterpart.

- We propose the notion of *prune potential*, i.e., the maximal prune ratio (model sparsity) at which the pruned network can achieve commensurate performance, as a quantifiable means to estimate the overparameterization of a network and show that it is significantly lower on challenging inference tasks.

- We provide a unified framework to establish task-specific guidelines that help practitioners assess the effects of pruning during the design and deployment of neural networks in practice.

- We conduct experiments across multiple data sets, architectures, and pruning methods showing that our observations hold across common pruning benchmarks and real-world scenarios.

7.1.2 Relevant Papers

The results presented in this chapter are based on the following paper:

- Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 93–138, 2021a.

7.1.3 Outline

In Section 7.2 we introduce our approach to pruning and the type of networks and datasets we consider. We provide additional experimental hyperparameters and nominal prune-accuracy results in Section B.1. Subsequently, we investigate the functional similarities between pruned networks and their unpruned counterpart in Section 7.3 to formalize our intuition that pruned networks are similar. We provide a detailed description of these results in Section B.2. We then focus on the main part of the experimental analysis in Section 7.4, where we investigate how pruned networks behave under various changes in the input data distribution. A more detailed description with additional results is provided in Section B.3. Finally, we consider avenues to improve the robustness of pruned networks in Section 7.5 with additional details provided in Section B.4. We conclude the chapter with a discussion in Section 7.6.

7.2 Methodology

7.2.1 Pruning Setup

For our experiments, we consider a variety of network architectures, data sets, and pruning methods as outlined below. Our pruning pipeline, see Algorithm 5, is based on iterative pruning and retraining following Renda et al. (2020). It is simple, network-agnostic, and widely used; hence we opted to choose it as representative pruning pipeline.

Table 7.1: Overview of the pruning methods evaluated. Here, $a(x)$ denotes the activation of the corresponding layer with respect to a sample input x to the network.

Type	Method	Data-informed	Sensitivity	Scope
Unstructured (Weights)	WT: Weight Thresholding (Renda et al., 2020)	✗	$\lvert W_{ij}\rvert$	Global
	SiPP: Sensitivity-informed Pruning (Baykal et al., 2021b)	✓	$\propto \lvert W_{ij} a_j(x)\rvert$	Global
Structured (Neurons/Filters)	FT: Filter Thresholding (Renda et al., 2020)	✗	$\lVert W_{i:}\rVert_1$	Local
	PFP: Provable Filter Pruning (Liebenwein et al., 2020)	✓	$\propto \lVert W_{:j} a(x)\rVert_\infty$	Local

Data sets and network architectures. We consider CIFAR-10 (Torralba et al., 2008), ImageNet (Russakovsky et al., 2015), and Pascal VOC segmentation data (Everingham et al., 2015) as data sets. We consider ResNets 18/56/110 (He et al., 2016), WRN16-8 (Zagoruyko and Komodakis, 2016), DenseNet22 (Huang et al., 2017), and VGG16 (Simonyan and Zisserman, 2015) on CIFAR-10; ResNet18 and 101 (He et al., 2016) on ImageNet; and a DeeplabV3-ResNet50 (Chen et al., 2017) on VOC.

Training. For all networks, we apply the standard training parameters as indicated in the respective papers. We apply the linear scaling rule of Goyal et al. (2017) when training on multiple GPUs in parallel including warm-up. All hyperparameter settings with their numerical values are listed in Section B.1. All networks are trained once to completion before pruning (Line 2 of Algorithm 5).

Pruning. We consider multiple unstructured and structured pruning methods, where we prune individual weights and filters/neurons, respectively, see Table 7.1 for an overview. We perform pruning by updating a binary mask indicating whether the corresponding weight is active or pruned (Line 5 of Algorithm 5).

Unstructured pruning. The weight pruning approaches we consider follow a global pruning strategy: (1) globally sort the weights according to their relative importance, i.e., sensitivity, and (2) prune $r_{\text{prune}}\%$ of the weights with the lowest sensitivity. In particular, we study two methods to compute the sensitivity of weights, weight thresholding (Renda et al., 2020) and Sensitivity-informed Provable Pruning (SiPP) (Baykal et al., 2021b). Weight Thresholding (WT) is a simple heuristic, originally introduced by Han et al. (2015a) and re-purposed by Renda et al. (2020), that defines the sensitivity of a weight as the magnitude of the weight. SiPP, on

the other hand, is a data-informed approach with provable guarantees to computing weight sensitivities (Baykal et al., 2021b). The approach uses a small batch of input points $\mathcal{S} \subseteq \mathcal{P}$, e.g., from the validation set, to evaluate the saliency of each network parameter. This is done by incorporating the corresponding (sample) activations, $a(x)$, $x \in \mathcal{S}$, along with the weight into the importance computation (see Table 7.1).

Algorithm 5 PRUNERETRAIN(n_{cycles}, r_{prune}, n_{train}, ρ_{train})

Input: n_{cycles}: number of prune-retrain cycles; r_{prune}: relative prune ratio; n_{train}: number of train epochs; ρ_{train}: training hyper-parameters
Output: c: pruning mask, θ: parameters of the pruned network

$\theta_0 \leftarrow$ RANDOMINIT()
$\theta \leftarrow$ TRAIN$(\theta_0, n_{\text{train}}, \rho_{\text{train}})$
$c \leftarrow 1^{|\theta_0|}$ ▷ binary mask for the parameters
for $i \in [n_{\text{cycles}}]$ **do**
  $c \leftarrow$ PRUNE$(c \odot \theta, r_{\text{prune}})$ ▷ Prune r_{prune}% of the remaining parameters.
  $\theta \leftarrow$ TRAIN$(c \odot \theta, n_{\text{train}}, \rho_{\text{train}})$
end for
return c, θ

Structured pruning. The filter/neuron pruning approaches we consider follow a two-step strategy: (1) allocate a per-layer prune ratio satisfying the overall prune ratio and (2) prune the filters with lowest sensitivity in each layer. We study Filter Thresholding (FT) as used by Renda et al. (2020) and Provable Filter Pruning (PFP), see Chapter 5. FT, as originally introduced by He et al. (2018); Li et al. (2016) and used here analogous to Renda et al. (2020), uses the filter norm to evaluate its sensitivity. Layer allocation is performed manually and we deploy a uniform prune ratio across layers to avoid further hyperparameters. PFP (Chapter 5) is an extension of SiPP that evaluates filter sensitivity as the maximum sensitivity of the channel in the next layer (ℓ_∞-norm of the corresponding weight sensitivity). PFP uses the associated theoretical error guarantees to optimally allocate the layer-wise budget.

Retraining. We retrain the network with the exact training hyperparameters as is common (Baykal et al., 2021b; Liebenwein et al., 2020; Renda et al., 2020). Specifically, we re-use the same learning rate schedule and retrain for the same amount of

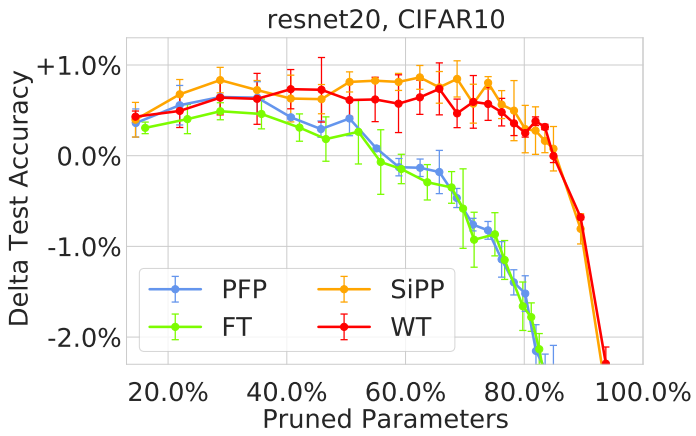


Figure 7-1: The accuracy of the generated pruned models for the evaluated pruning schemes for various target prune ratios using iterative fine-tuning.

epochs (Line 6 of Algorithm 5). After retraining, we iteratively repeat the pruning procedure to obtain even smaller networks (Lines 4-7 of Algorithm 5).

Prune results. Figure 7-1 shows an exemplary test accuracy curve of a Resnet20 (CIFAR-10) across different target prune ratios for iterative pruning. The remaining prune results are summarized in the appendix (Section B.1). We note while PruneRetrain may be more computationally expensive than other pruning pipelines, it is network-agnostic and produces state-of-the-art pruning results (Renda et al., 2020).

7.2.2 Experiments Roadmap

Our observations stem from multiple experiments that can be clustered into one set of experiments pertaining to understanding the functional similarities (i.e., functional distance) of pruned networks and one set pertaining to the prune potential on a variety of image-based classification tasks. First, we compare subsets of pixels that are sufficiently informative for driving the network's decision. For this, we use the feature-wise (pixel-wise) selection mechanism of Carter et al. (2019, 2020). We also investigate how pruned networks behave under ℓ_∞-bounded random noise. Second, we assess the ability of pruned networks to generalize to out-of-distribution (o.o.d.) test data sets that contain random noise (ℓ_∞-bounded) and common corruptions (Hendrycks and

Dietterich, 2019; Recht et al., 2018, 2019) including weather, contrast, and brightness changes. In all experiments, we compare the performance, i.e., the accuracy on the various test sets, of pruned networks with those of their unpruned counterparts as well as a separately trained unpruned network. Each experiment is repeated 3 times and we report mean and standard deviation (error bars). We first provide key insights for each experiment before diving deeper into the details in the subsequent sections.

7.3 Function Distance

Given a pruned model with commensurate test accuracy relative to the parent (uncompressed) network, can we conclude that the *function* represented by the pruned model is similar to the parent network for unforeseen data points? In this section, we investigate the extent to which the pruned and parent model are functionally similar under two distinct metrics: informative features and noise resilience. Our findings show that pruned networks are more functionally similar to their original network than a separately trained, unpruned network underscoring the intuition that pruned networks remain functionally similar to their unpruned counterpart.

7.3.1 Methodology

Comparison of informative features. We compare features (pixels) that are informative for the decision-making of each model. Specifically, for a network $f_\theta(x)$ with parameters θ and input $x \in \mathbb{R}^n$ we want to find an input mask $m \in \{0,1\}^n$ such that $f_\theta(x) \approx f_\theta(m \odot x)$, i.e.,

$$m = \underset{\|m\|_0 \leq (1-B)n}{\operatorname{argmin}} \|f_\theta(x) - f_\theta(m \odot x)\| \tag{7.1}$$

for some sparsity level B. To approximately solve (7.1) we use the greedy backward selection algorithm (`BackSelect`) of Carter et al. (2019, 2020). The procedure iteratively masks the least informative pixel (i.e., the pixel which if masked would reduce the confidence of the prediction of the correct label by the smallest amount) to obtain

a sorting of the pixels in order of increasing importance. After sorting, we can remove the bottom $B\%$ of pixels.

Given two networks $f_\theta(\cdot)$ and $f_{\hat{\theta}}(\cdot)$, we can then measure the difference between the functions by switching up the respective input masks m and $\hat{m}$ to see the change in the output, i.e., $\|f_\theta(\hat{m} \odot x) - f_\theta(m \odot x)\|$ and vice versa. If one model can make a confident and correct prediction on the pixels that were informative to another model, the models may have similar decision-making strategies. We apply this strategy to identify subsets of informative pixels across a sample of 2000 CIFAR-10 test images. For each image, we compute the subset of informative pixels, i.e., the input mask m, for an unpruned network, five pruned networks (of increasing prune ratio) derived from that network, and a separate, unpruned network of the same type. We probe whether the informative pixels from one model are also informative to the other models for a sparsity level of 90%.

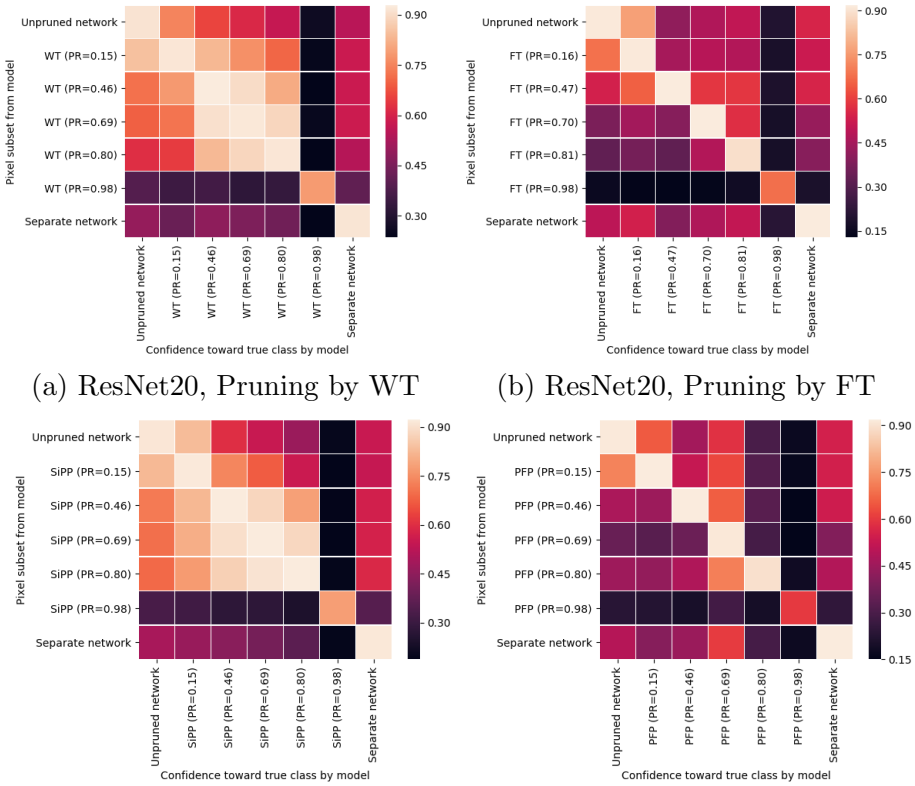
Noise similarities. We consider injecting l_∞-bounded, random noise into the test data and we compare the predicted labels between the pruned networks and their unpruned counterpart to investigate the behavior in local neighborhood of points. Specifically, for two networks $f_\theta(\cdot)$ and $f_{\hat{\theta}}(\cdot)$ with parameters θ and $\hat{\theta}$, respectively, we consider the expected number of matching label prediction and the expected norm difference of the output with noise ε, i.e.,

$$\mathbb{E}_{x' \sim \mathcal{D} + \mathcal{U}^n(-\varepsilon, \varepsilon)} \left[\operatorname{argmax} f_\theta(x') = \operatorname{argmax} f_{\hat{\theta}}(x') \right]$$

and

$$\mathbb{E}_{x' \sim \mathcal{D} + \mathcal{U}^n(-\varepsilon, \varepsilon)} \left\| f_\theta(x') - f_{\hat{\theta}}(x') \right\|_2 ,$$

respectively. We test the noise similarity of networks for a random subset of 1000 test images for 100 repetitions of random noise injection and average over the results.



(a) ResNet20, Pruning by WT

(b) ResNet20, Pruning by FT

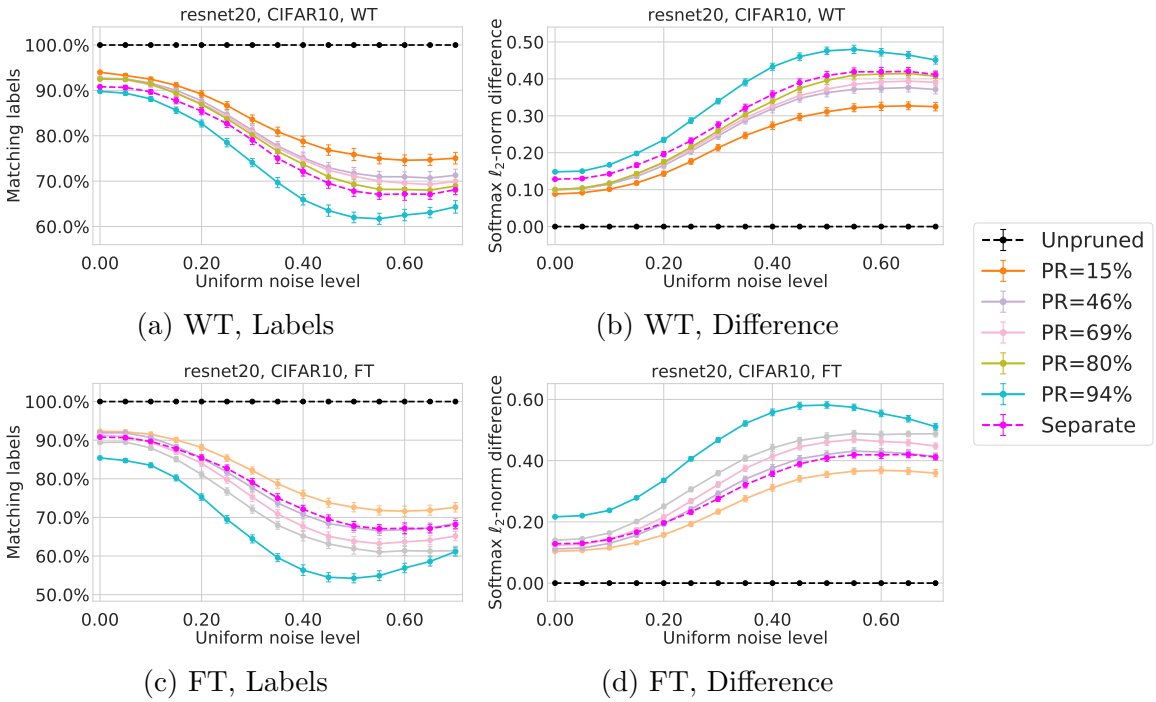
(c) ResNet20, Pruning by SiPP

(d) ResNet20, Pruning by PFP

Figure 7-2: Heatmap of confidences on informative pixels from pruned ResNet20 models. Y-axis is the model used to generate 10% pixel subsets of 2000 sampled CIFAR-10 test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a)Weight Thresholding (WT), (b) Filter Thresholding (FT), (c) Sensitivity-informed Provable Pruning (SiPP), (d) Provable Filter Pruning (PFP).

7.3.2 Results

Comparison of informative features. Figure 7-2 shows heatmaps of mean confidence on masked images containing only the 10% most informative features as ordered by `BackSelect` from an unpruned network (ResNet20 on CIFAR-10), five pruned networks, and a separate, unpruned network. For each masked image containing only the informative features (found with respect to the predicted class), we evaluate the confidence toward the true class for all models to reveal whether such features are informative to the other models. We find features informative to the unpruned network



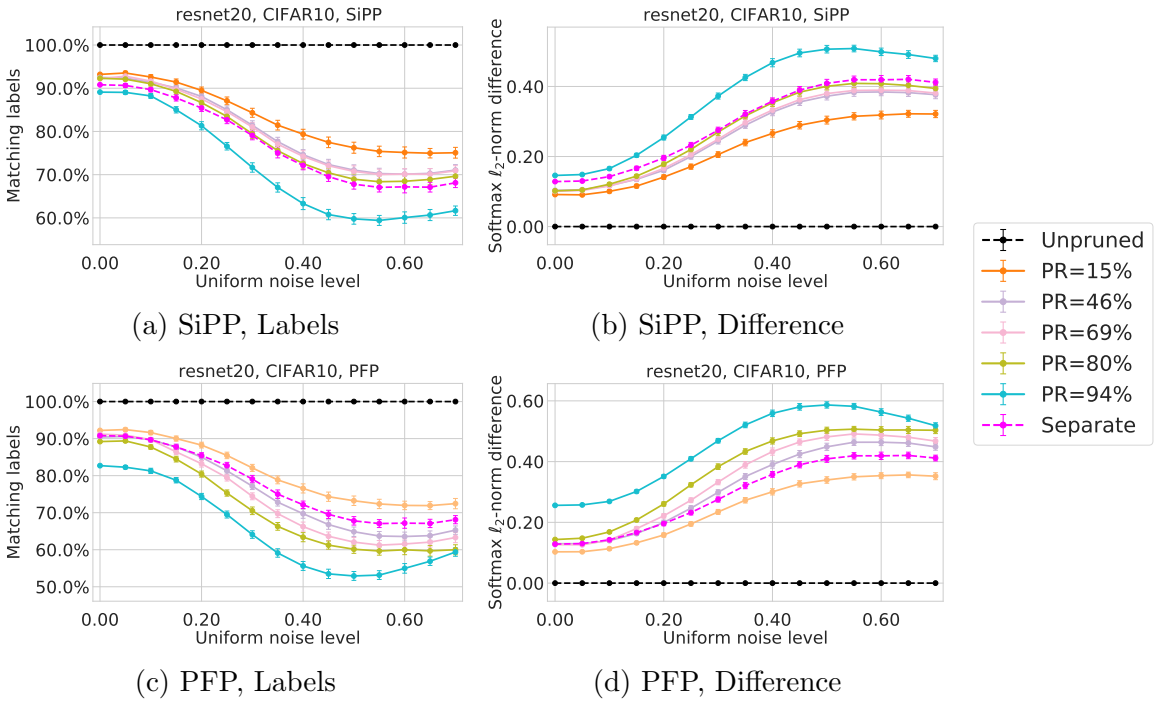
(a) WT, Labels (b) WT, Difference

(c) FT, Labels (d) FT, Difference

Figure 7-3: The functional similarities between pruned ResNet20 models and their unpruned parent. We consider the difference in the output after injecting various amounts of noise into the input, see (a), (b) and (c), (d) for networks weight-pruned with WT and filter-pruned with FT, respectively. The differences between a separately trained network and the unpruned parent is also shown. The plots depict the difference measured as the percentage of matching predictions and as norm-based difference in the output after applying softmax, see (a), (c) and (b), (d), respectively.

suffice for confident predictions by the pruned networks derived from it, but do not suffice for prediction by the separate, unpruned network. We also find that informative features from pruned networks can be used for prediction by the original network, suggesting these models employ a similar decision-making process. In general, our results suggest weight-pruned networks maintain higher confidence on parent features than do filter-pruned networks. For models pruned beyond commensurate accuracy (PR = 0.98 in Figure 7-2), the informative features are no longer predictive.

Noise similarities. In Figures 7-3a, 7-3c the percentage of matching label predictions of WT- and FT-pruned ResNet20 networks are shown with respect to their unpruned counterpart for multiple noise levels. We can conclude that the predictions of the pruned networks tend to correlate with the predictions of the unpruned parent



(a) SiPP, Labels

(b) SiPP, Difference

(c) PFP, Labels

(d) PFP, Difference

Figure 7-4: The functional similarities between pruned ResNet20 models and their unpruned parent.

– clearly more than the predictions from a separately trained, unpruned network. We also consider the overall difference in the ℓ_2-norm between pruned and unpruned networks of the softmax output where we observe similar trends, see Figures 7-3b, 7-3d. We conduct the same experiments with SiPP- and PFP-pruned networks as well (see Figure 7-4) to further validate our findings. These results indicate that the decision boundaries of the pruned network tend to remain close to those of the unpruned network implying that during retraining properties of the original function are maintained. We note that the correlation decreases as we prune more corroborating our intuitive understanding of pruning.

7.4 Pruning under Distribution Changes

We highlighted that pruned networks behave functionally similarly, however, ultimately the performance is measured in terms of the loss or accuracy on previously unseen data. In this section, we investigate how pruned networks behave in the pres-

ence of shifts in the data distribution, including noise, weather, and other corruptions. While it is commonly known that out-of-distribution (o.o.d.) data can harm the performance of neural networks (Madry et al., 2018), we specifically investigate whether pruned network suffer *disproportionately more* from o.o.d. data compared to their parent network. Answering this question affirmatively has profound implications on the practical deployment of pruned networks, specifically for safety-critical systems.

To this end, we define a network's *prune potential* to be the maximal prune ratio for which the pruned network achieves similar loss (up to margin δ) compared to the unpruned one for data sampled from distribution $\mathcal{D}$.

Definition 7 (Prune Potential)**.** *Given a neural network $f_\theta(x)$ with parameters θ, input-label pair $(x, y) \sim \mathcal{D}$, and loss function $\ell(\cdot, \cdot)$ the* prune potential $P(\theta, \mathcal{D})$ *for some margin δ is given by*

$$\begin{aligned} P(\theta, \mathcal{D}) = \max_{c \in \{0,1\}^{|\theta|}} & 1 - {\|c\|_0}/{\|\theta\|_0} \\ \textit{subject to} & \\ \mathbb{E}_{(x,y)\sim\mathcal{D}} & \left[\ell(y, f_{c \odot \hat{\theta}}(x)) - \ell(y, f_\theta(x))\right] \leq \delta, \end{aligned} \tag{7.2}$$

where $\|\cdot\|_0$ denotes the number of nonzero elements, and c and $\hat{\theta}$ denote the prune mask and parameters, respectively, obtained from PruneRetrain *(Algorithm 5).*

The prune potential $\mathcal{P}(\theta, \mathcal{D})$ thus indicates how much of the network can be safely pruned with minimal additional loss incurred. In other words, it indicates to what degree the pruned network can *maintain* the performance of the parent network. As an additional benefit the prune potential may act as a robust measure to gauge the *overparameterization* of a network in the presence of distribution shifts.

Moreover, we define a network's *excess loss* to be the additional loss incurred under distributional changes of the input.

Definition 8 (Excess Loss)**.** *Given a neural network $f_\theta(\cdot)$ with parameters θ, training distribution $\mathcal{D}$ from which we can sample input-label pairs $(x, y) \sim \mathcal{D}$, test distribution $\mathcal{D}'$ from which we can also sample input-label pairs, and loss function $\ell(\cdot, \cdot)$, the* excess

loss $e(\theta, \mathcal{D}')$ *is given by*

$$e(\theta, \mathcal{D}') = \mathop{\mathbb{E}}_{(x',y')\sim\mathcal{D}'} \ell(y', f_\theta(x')) - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \ell(y, f_\theta(x)).$$

The excess loss hereby indicates the expected performance drop of the network for distribution changes, which we can evaluate for various unpruned and pruned parameter sets for a given network architecture to understand to what extend the excess loss varies.

7.4.1 Methodology

We choose test error (indicator loss function) to evaluate the prune potential and excess loss (excess error). We evaluate the constraint of (7.2) for a margin of $\delta = 0.5\%$.

We compare the prune potentials $p = P(\theta, \mathcal{D})$ and $p' = P(\theta, \mathcal{D}')$ for two distributions $\mathcal{D}$ and $\mathcal{D}'$ to assess whether pruning up to the prune potential p implies that we can also safely prune up to p for $\mathcal{D}'$. Specifically, the difference $p - p'$ in prune potential can indicate how much the prune potential varies and thus whether it is safe to prune the network up to its full potential p when the input is instead drawn from $\mathcal{D}'$. We note that in practice we may only have access to $\mathcal{D}$ but not $\mathcal{D}'$. Thus in order to safely prune a network up to some prune ratio p it is crucial to understand to what degree the prune potential may vary for shifts in the distribution.

We also compare the excess error $e = e(\theta, \mathcal{D}')$ and $\hat{e} = e(c \odot \hat{\theta}, \mathcal{D}')$ for an unpruned and pruned network with parameters θ and $c \odot \hat{\theta}$, respectively. Note that the difference in excess error, $\hat{e} - e$, quantifies the additional error incurred by a pruned network under distribution changes compared to the additional error incurred by an unpruned network. Ideally, the difference $\hat{e} - e$ should be zero across all prune ratios, which would imply that the prune-accuracy trade-off for nominal data is indicative of the trade-off for o.o.d. data.

We evaluate the prune potential and excess error using nominal test data (train distribution $\mathcal{D}$) and o.o.d. test data (test distribution $\mathcal{D}'$). Specifically, we consider o.o.d. data with random noise following Section 7.3.1, and o.o.d. data corrupted using

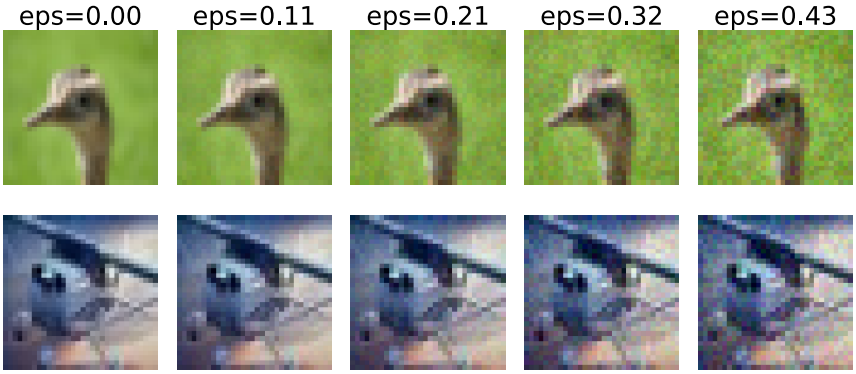


Figure 7-5: Example images from the CIFAR-10 test dataset that were used in this study with various levels of noise injected. A human test subject can classify the images equally well despite the noise present.

state-of-the-art corruption techniques, i.e., CIFAR-10.1 (Recht et al., 2018), CIFAR-10-C (Hendrycks and Dietterich, 2019) for CIFAR-10; ObjectNet (Barbu et al., 2019), ImageNet-C (Hendrycks and Dietterich, 2019) for ImageNet; and VOC-C (Michaelis et al., 2019) for VOC. For CIFAR-10-C, ImageNet-C, VOC-C we choose severity level 3 out of 5. The prune potential is evaluated separately for each corruption while the excess error is evaluated by averaging over all corruptions (test distribution).

7.4.2 Results

Noise

We evaluated the prune potential of multiple CIFAR-10 networks for various noise levels, the results of which are shown in Figure 7-6. Initially, the network exhibits high prune potential, similar to the prune potential on the original test data (noise level 0.0). However, as we increase the noise injected into the image the prune potential rapidly drops to 0%. As shown, most networks' prune potential based on noise exhibit similar properties. This is particularly discomforting as the noise does not deteriorate a human's ability to classify the images correctly as can be seen from Figure 7-5. These results highlight we may not be able to significantly prune networks if maintaining performance on slightly harder data is the goal.

However, we note that depending on the architecture the specific prune potential may be less affected. Specifically, we note that the prune potential of WideResNets

(Figure 7-6, bottom right) seems to be mosstly unaffected by noise across all tested prune methods. In contrast to the other networks, WideResNets are wider and less deep, which may provide a possible explanation for the observed behaviors. Due to the wide spatial dimensions of each network, the noise may be better absorbed since it may spread across the width of the layer. Moreover, the reduced depth may help in avoiding positive amplification of the noise as depth increases. From this observation, we may conclude that WideResNets are indeed overparameterized and henceforth the prune potential remains unaffected for slight perturbations in the input.

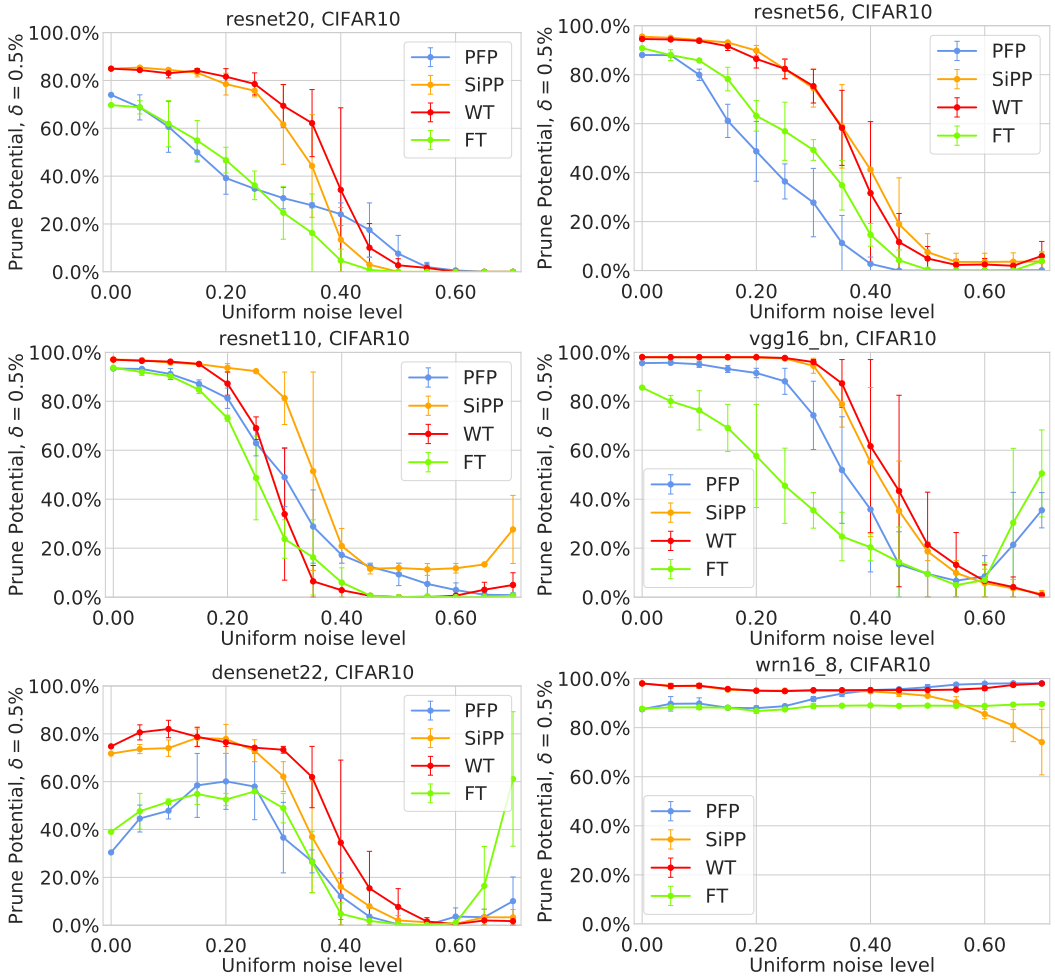


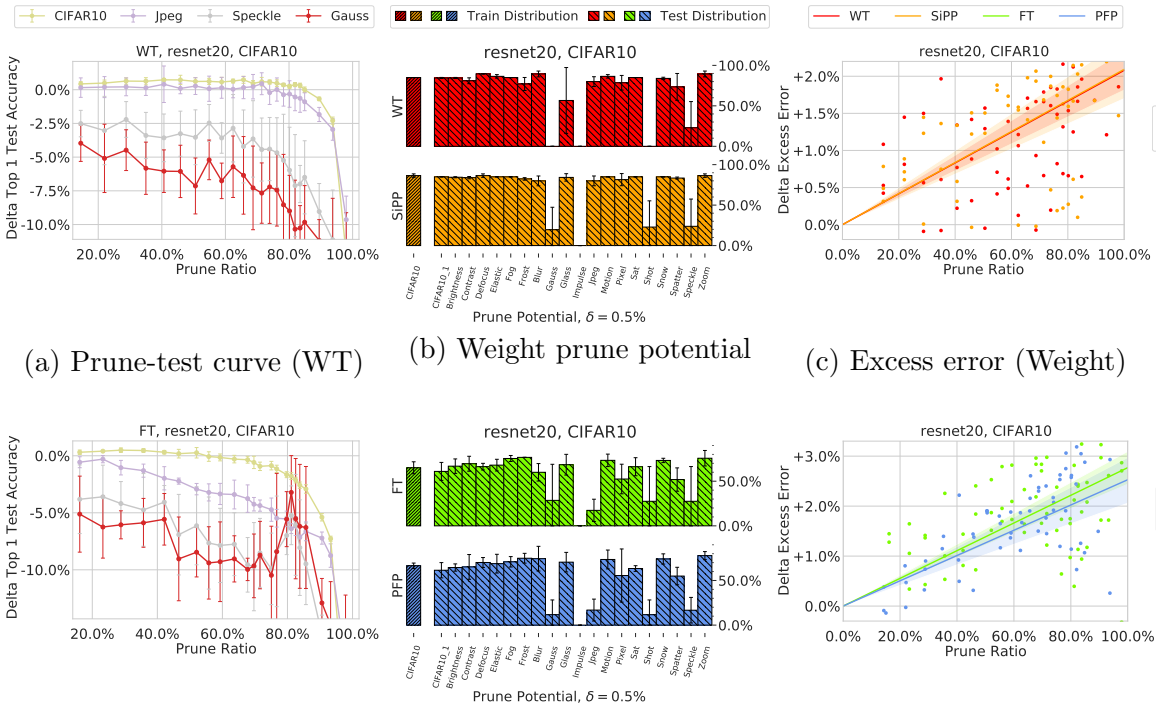
Figure 7-6: The prune potential (%) achievable over various levels of noise injected into the input on different network architectures trained and pruned on CIFAR-10.

Prune-accuracy curves for corruptions

Separately, we investigated the prune potential for image corruptions based on the CIFAR-10-C, ImageNet-C. In Figures 7-7a and 7-7d we show the test accuracy of pruned networks across various target prune ratios for a subset of CIFAR-10-C corruptions for a ResNet20 pruned with WT and FT, respectively. In particular, for some simpler corruptions, such as Jpeg, the prune-accuracy curves closely resembles the original CIFAR-10 curve, while for metrics such as Speckle and Gauss the curve indicates a noticeable accuracy drop across all target prune ratios. Moreover, the prune-accuracy curve becomes more unpredictable and less stable as indicated by the significantly higher variance of the resulting accuracy. We thus conclude that the achievable accuracy of the network depends on the pruning method and the target prune ratio, however, we observe an equally strong dependence on the chosen test metric. Consequently, this affects the prune potential of the network highlighting the sensible trade-off between generalization performance and prune potential.

Prune potential for corruptions

For each corruption we then extracted the resulting prune potential from the prune-accuracy curves, see Figures 7-7b and 7-7e for weight pruning and filter pruning, respectively. In particular, for corruptions, such as Gauss, Impulse, or Shot, we observe that the network's prune potential hits (almost) 0% implying that any form of pruning may adversely affect the network's performance under such circumstances. We repeated the same experiment for a ResNet18 trained on ImageNet and tested on ImageNet-C, see Figure 7-8. Noticeably, the network exhibits significantly higher variance in the prune potential across different corruptions compared to the networks tested on CIFAR-10. This effect is also more pronounced for filter pruning methods. In Section B.3.1, we provide results for additional networks for both CIFAR-10 and ImageNet corroborating our findings presented here.



(a) Prune-test curve (WT) (b) Weight prune potential (c) Excess error (Weight)

(d) Prune-test curve (FT) (e) Filter prune potential (f) Excess error (Filter)

Figure 7-7: The prune potential for a ResNet20 on CIFAR-10-C test datasets. We observe that depending on the type of corruption the network has significantly less prune potential than when measured w.r.t. the nominal CIFAR-10 test accuracy.

Choice of δ

We additionally investigate how the prune potential is affected by our choice of δ (see Section B.3.2). While the actual value of the prune potential is naturally affected by δ, we find that our observations of the resulting trends remain unaffected by our particular choice of δ. Hence, we simply choose $\delta = 0.5\%$ uniformly across all experiments reflecting the requirement that our pruned network should be close in accuracy to the parent network while allowing some slack to increase the prune potential.

Excess error

In contrast to the prune potential, the difference in excess error enables us to quantify across multiple prune ratios how much *additional error* is incurred by the pruned network on top of the unpruned network's excess error when tested on o.o.d. data.

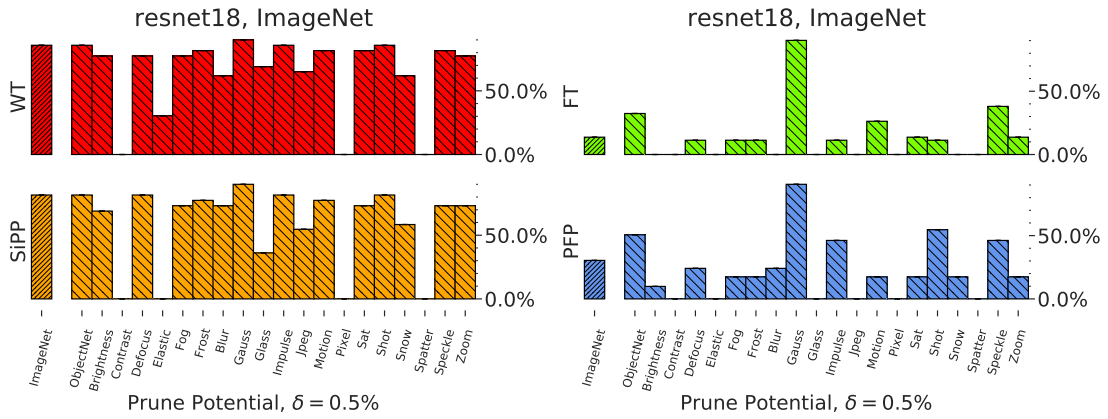


Figure 7-8: Prune potential of a ResNet18 (ImageNet).

A non-zero difference thus indicates how the prune-accuracy curve changes under distribution changes. In other words, the difference in excess error quantifies the difference in error between the pruned and unpruned network *on top* of the difference in error that is incurred for nominal test data (train distribution).

We evaluated the difference in excess error between pruned and unpruned ResNet20 networks trained on CIFAR-10 for various prune ratios (see Figures 7-7c, 7-7f). Note that by definition the excess error is 0% for a prune ratio of 0%. We observe that the difference in excess error can reach upwards of 2% and 3% for weight and filter pruning, respectively. Moreover, the higher the prune ratio the more variance we can observe indicating that the pruned network's behavior becomes less predictable overall. These observations strongly indicate that pruned networks suffer disproportionally more from o.o.d. data across a wide spectrum of prune ratios and that the additional performance drop on o.o.d. data is positively correlated with the prune ratio. In other words, while current pruning techniques achieve commensurate accuracy for high prune ratios on nominal test data, the same pruning techniques do not maintain commensurate accuracy for even small prune ratios on o.o.d. test data. Additional results are presented in Section B.3.3.

Measuring overparameterization.

The results presented so far in this section highlight that the prune potential on nominal test data does not reliably indicate the overall performance of the pruned

network. This may lead to novel insights into understanding the amount of overparameterization in deep networks. In Section B.3.4 we summarize the prune potential across all tested data distributions and networks as a way to gauge the amount of overparameterization of a network. A subset of the results are shown in Table 7.2. While some networks' prune potentials are significantly affected by changes in the distribution, other networks' prune potentials are virtually unaffected. Take for example the weight prune potential on nominal test data (training distribution) of a VGG16 and a WRN16-8, which is around 98% for both. However, when both networks are evaluated on o.o.d. test data (test distribution) they exhibit distinctly different behaviors. While the WRN16-8's prune potential remains fairly stable at around 95% (3% drop), the VGG16's prune potential falls to 80% (18% drop).

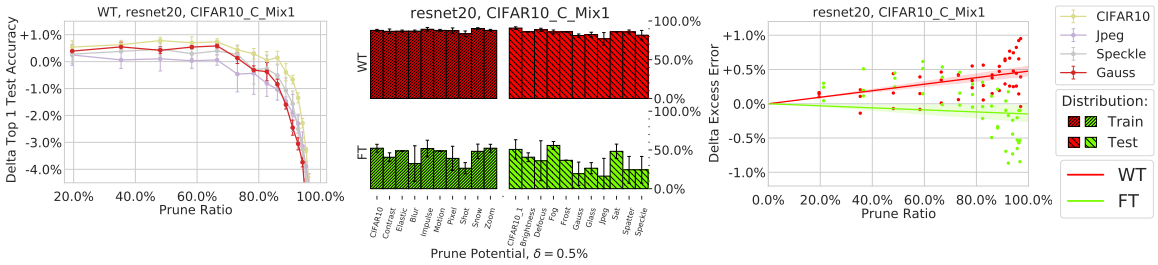
Overall, these results illustrate the fact that the prune potential may act as a robust measure of a network's genuine overparameterization in theory, and may also be helpful in informing the practitioner on the extent of pruning that should be conducted prior to deployment in practice.

Table 7.2: The prune potential of various networks trained on CIFAR-10 (upper part) and ImageNet (lower part) evaluated on the train and test distribution, which consist of nominal data and the average over all corruptions, respectively.

Model	Method	Prune Potential (%)		
		Train Dist.	Test Dist.	Diff.
ResNet20	WT	84.9 ± 0.0	**66.7 ± 3.3**	-18.2
	FT	65.0 ± 6.7	**55.3 ± 4.8**	-9.7
VGG16	WT	98.0 ± 0.0	**80.9 ± 2.2**	-17.1
	FT	85.4 ± 2.4	**66.3 ± 0.5**	-19.1
WRN16-8	WT	98.0 ± 0.0	**95.7 ± 0.7**	-2.3
	FT	86.2 ± 1.3	**75.7 ± 4.1**	-10.5
ResNet18	WT	85.8	**63.6**	-22.2
	FT	13.7	**13.5**	-0.2

7.5 Towards Robust Pruning

Our experiments raise the question whether the decreased performance of pruned networks is a limitation of our current pruning and training techniques or whether it



(a) WT, prune-test (b) Prune pot. (WT, FT) (c) Excess error (WT, FT)

Figure 7-9: The prune potential (b) and excess error (c) of a ResNet20 shown for corruptions that were included (train distribution) and excluded (test distribution) during training. The prune-accuracy curves in (a) are shown for corruptions from the test distribution.

is an inherent limitation of pruned, i.e., smaller, networks themselves. To this end, we investigated whether training (and retraining) in a robust manner can benefit the pruned network and minimize the effects we observed previously.

7.5.1 Methodology

To test the hypothesis that we can regain some of the robustness properties of the unpruned network we repeated the experiments of Section 7.4 but during (re-)training we incorporated a randomly chosen fixed subset of nine corruptions from CIFAR-10-C and ImageNet-C into the data augmentation pipeline. That is, every time we sample an image from the train set during (re-)training we choose an image corruption (or no corruption) uniformly at random to corrupt the image effectively altering the train distribution that the network sees. The remaining corruptions are not used during training and make up the new test distribution. Additional experimental details are provided in Appendix B.4.

7.5.2 Results

Prune-accuracy curves and prune potential. In Figure 7-9a we show the prune-accuracy curves for three corruptions from the test distribution for WT-pruned ResNet20s. Compared to the results in Figure 7-7a where we did not perform robust (re-)training we observe that the prune-accuracy curves are much more stable and

that the prune-accuracy curve on nominal test data (CIFAR-10) is more predictive of the others. However, the results on the evaluation of the prune potential as shown in Figure 7-9b for weight and filter pruning reveal that even in this setting the prune potential can be significantly lower (or exhibit high variation over multiple trials) for some of the corruptions from the test distribution. These observations further corroborate our findings but also highlight the beneficial effects of robust training in efficiently alleviating some of the short-comings (see Appendix B.4 for a complete exposition of the results).

Excess error. Similar trends can also be observed when considering the difference in excess error as shown in Figure 7-9c. While we can reduce the correlation between prune ratio and excess error, we can not entirely eliminate it. Moreover, we can still observe high variations in the excess error confirming the sensible trade-off between generalization performance and prune potential.

Implicit regularization. In this section, we show that we can regain much of the prune potential even under distribution changes, at least when we can incorporate these additional data points into our training pipeline. For example, the weight prune potential for a ResNet20 for both the nominal and robust training scenario is around 85%. Consequently, we argue that both pruned and unpruned networks have sufficient capacity to represent the underlying distribution given that the training is performed using an appropriate optimization pipeline.

Specifically, previous work noted that overparameterized networks may benefit from implicit regularization when optimized with a stochastic optimizer and that more parameters amplify this effect. We can confirm these observations in the sense that pruned networks suffer disproportionally more from o.o.d. data than unpruned networks with more parameters. That is, unpruned network exhibit more implicit regularization through stochastic gradient descent (SGD) leading to more robustness. However, we can regain some of the robustness properties by adding *explicit* regularization during training in the form of data augmentation. We can thus "trade"

the implicit regularization potential which we lose by removing parameters for explicit regularization through data augmentation.

Choice of test distribution. While our results suggest that robust training indeed improves the generalization of pruned networks for o.o.d. data, we would like to emphasize that our conclusion intrinsically hinges upon on the choice of train and test distribution. While we did strictly separate the corruptions used during train and test time, these corruptions can be loosely categorized into four types, i.e. noise, blur, weather, digital, all of which are present in both the train and test distribution. Therefore, we suspect that for significantly different corruption models (or adversarial inputs) we may observe more significant trade-offs resembling the results of Section 7.4 where we performed nominal (re-)training. This is a consequence of requiring additional explicit regularization since the explicit regularization must be modeled.

7.6 Discussion

Weight vs filter pruning. We find that across all tested corruptions filter pruning methods are more error-prone and have lower prune potential compared to weight pruning. We conjecture this trend stems from the fact that structured pruning is overall a harder problem, implying that structurally pruned networks are less capable of maintaining the properties of the parent network when compared to those generated by weight-based pruning.

Genuine overparameterization. In light of our results we conjecture that while the high capacity of modern networks may not be strictly necessary to achieve high test accuracy – since pruned networks with commensurate accuracy exist –, the "excess" capacity of these networks may be beneficial to maintaining other crucial properties of the network, such as its ability to perform well on unforeseen or out-of-distribution data. This challenges the common wisdom that modern networks are overparameterized and, hence, contain redundant parameters that can be pruned in

a straight-forward manner without "loss of performance." Unlike prior work that has predominantly pointed to the test accuracy as a gauge for overparameterization (and thus the ability to prune a network), we hypothesize that a more robust and accurate measure of *genuine* overparameterization is one that not only considers test accuracy, but also the minimum (or average) prune potential over a variety of tasks. Studying the prune potential is thus not only useful to study the ability to safely prune a network but also has the positive side-effect of establishing a robust measure of network overparameterization.

Implicit regularization. Our studies reveal that the amount of overparameterization is not only a function of the task and the network size but also a function of the *training procedure*. Specifically, we can prune the network more if we explicitly regularize the network during (re-)training thus increasing the "genuine" overparameterization of the network which implies a higher prune potential. However, with fewer parameters (due to pruning more) we trade in some of the implicit regularization potential from SGD. Since implicit regularization is not necessarily model-based we can only regain the robustness of the pruned network for known, i.e. modeled, distribution changes.

Generalization-aware pruning. Based on our results we formulate a set of guidelines for pruning in practice as shown in Section 7.1. We argue that in order to reliably and robustly deploy pruned networks especially in the context of safety-critical systems we should not only designate a *hold-out data set* (test set) but also a *hold-out data distribution* (test distribution). By assessing the performance of the pruned network on data from the train and test distribution, we can then quantify the effect of pruning in a way that can unearth some of the short-comings that are *lost in pruning* and are not apparent from a plain prune-accuracy curve on nominal test data. Following our framework, a practitioner will be able to more reliably assess whether the pruned network can be considered as performant (in a robust sense) as the unpruned network.

Broader Impact. In this chapter, we study the impact of pruning when deploying neural networks in real-world conditions. This is of particular importance since the main motivation to prune neural networks lies within training neural networks that are simultaneously efficient and accurate. Henceforth, these networks can then be deployed in resource-constrained environments, such as robotics, to achieve tasks that could otherwise only be computed on large-scale compute infrastructure.

However, we show that pruned neural networks do not necessarily perform on par with their unpruned parent but rather that they exhibit significant performance decreases depending on slight variations within their assigned task. This raises concerns with regards to the ability to find effectively pruned architectures with current network pruning techniques. With the increasing amount of applications for deep learning including on small-scale devices, we have to vigilantly monitor and consider the effects, brittleness, and potential biasedness of small-scale neural networks at an even higher degree than for regular deep networks.

Conclusion. We have investigated the effects of the pruning process on the functional properties of the pruned network relative to its uncompressed counterpart. Our empirical results suggest that pruned models are functionally similar to the their uncompressed counterparts but that, despite this similarity, the prune potential of the network varies significantly on a task-dependent basis: the prune potential decreases significantly as the difficulty of the inference task increases. Our findings underscore the need to consider task-specific evaluation metrics beyond test accuracy prior to deploying a pruned network and provide novel insights into understanding the amount of network overparameterization in deep learning. We envision that our framework may invigorate further work towards rigorously understanding the inherent model size-performance trade-off and help practitioners in adequately designing and pruning network architectures in a task-specific manner.

Chapter 8

Pruning Continuous-depth Models

8.1 Overview

Continuous normalizing flows (CNFs) (Chen et al., 2018b) efficiently (Grathwohl et al., 2019) map a latent space to data via ordinary differential equations (ODEs), relaxing the strong constraints over *discrete* normalizing flows (Dinh et al., 2016; Durkan et al., 2019; Huang et al., 2020; Kingma and Dhariwal, 2018; Papamakarios et al., 2017; Rezende and Mohamed, 2015). CNFs enable learning flexible probabilistic models by arbitrarily-chosen neural network topologies. While recent works investigated ways to improve CNFs' efficiency (Finlay et al., 2020; Grathwohl et al., 2019; Li et al., 2020), regularize the flows (Onken et al., 2020; Yang and Karniadakis, 2020), or solving their fundamental shortcomings such as crossing trajectories (Dupont et al., 2019; Massaroli et al., 2020), less is understood about their inner dynamics during and post training.

In this chapter, we set out to use standard pruning algorithms to investigate generalization properties of sparse neural ordinary differential equations (neural ODEs) and CNFs. In particular, we investigate how the inner dynamics and the modeling performance of a continuous flow varies if we methodologically prune its neural network architecture. Reducing unnecessary weights of a neural network (pruning) (Han et al., 2015b; Hassibi and Stork, 1993; LeCun et al., 1990; Li et al., 2016) without loss of accuracy results in smaller network size (Hinton et al., 2015; Liebenwein

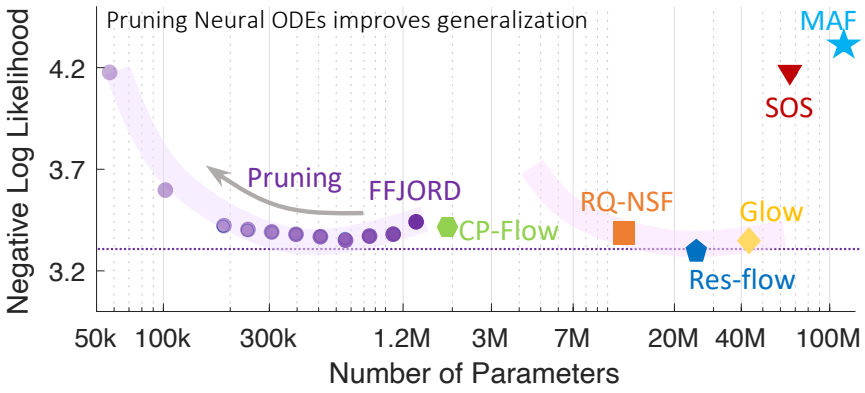
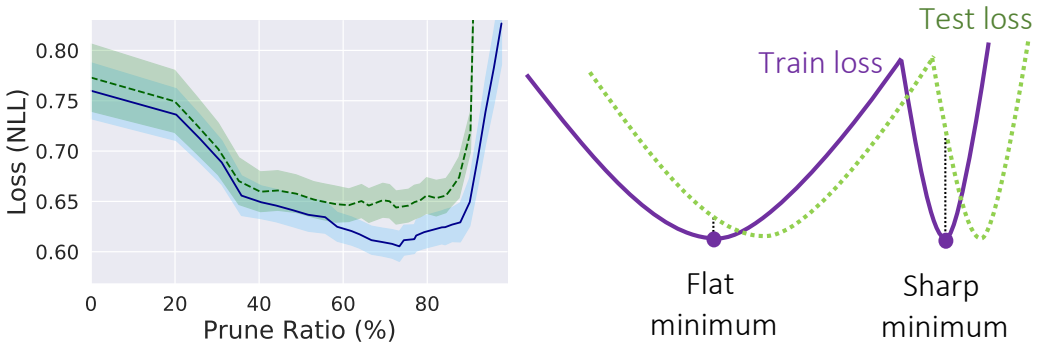


Figure 8-1: Pruning neural ODEs improves their generalization with at least 1 order of magnitude less parameters. CIFAR-10 density estimation. Values and methods are described in Table 8.6.

et al., 2021a), computational efficiency (Luo et al., 2017; Molchanov et al., 2016; Yang et al., 2017), faster inference (Frankle and Carbin, 2019), and enhanced interpretability (Baykal et al., 2019, 2021b; Lechner et al., 2020a; Liebenwein et al., 2020). Here, our main objective is to better understand CNFs' dynamics in density estimation tasks as we increase network sparsity and to show that pruning can improve generalization in neural ODEs.

Unlike in Chapter 7, where we highlight a potential short-coming of pruned neural networks in terms of generalization ability, here we investigate the ability of pruning to *improve* generalization in the context of time-continuous, recurrent neural networks. Specifically, we highlight an application of pruning that enables improved generalization performance compared to prior work with significantly less parameters as shown in Figure 8-1 for example.

Pruning improves generalization in neural ODEs. Our results consistently suggest that a certain ratio of pruning of fully-connected neural ODEs leads to lower empirical risk in density estimation tasks, thus obtaining better generalization. We validate this observation on a large series of experiments with increasing dimensionality. See an example here in Figure 8-2a.



(a) Pruned Generalization. (b) Flat vs. sharp minima.

Figure 8-2: Improved generalization through pruning. 8-2a: pruning enhances generalization of continuous-depth models. Structured pruning (green), unstructured pruning (blue). More details in Section 8.5. 8-2b: flat minima result in better generalization compared to sharp minima. Pruning neural ODEs flattens the loss around local minima. Figure 8-2b is reproduced from Keskar et al. (2017).

Pruning flattens the loss surface of neural ODEs. Additionally, we conduct a Hessian-based empirical investigation on the objective function of the flows-under-test in density estimation tasks to better understand why pruning results in better generalization. We find that for neural ODEs, pruning decreases the value of the Hessian's eigenvalues, and as a result, flattens the loss which leads to better generalization, c.f., Keskar et al. (2017) (Figure 8-2b).

Pruning helps avoiding mode-collapse in generative modeling. In a series of multi-modal density estimation tasks, we observe that densely connected CNFs often get stuck in a sharp local minimum (See Figure 8-2b) and as a result, cannot properly distinguish different modes of data. This phenomena is known as mode-collapse. Once we sparsify the flows, the quality of the density estimation task increases significantly and consequently mode-collapse does not occur.

Pruning finds minimal and efficient neural ODE representations. Our framework finds highly optimized and efficient neural ODE architectures via pruning. In many instances, we can reduce the parameter count by 70-98% (6x-50x compression rate). Notably, one cannot directly train such sparse and efficient continuous-depth models from scratch.

8.1.1 Contributions

In short, our contributions for this chapter are as follows:

- A generic pruning framework for unstructured and structured pruning of time-continuous neural networks, including neural ODEs and CNFs.
- An extensive experimental analysis that highlights the improved generalization performance of sparse, i.e., pruned, neural ODEs in classification and generative modelling task on low-dimensional toy data and high-dimensional tabular data.
- A generalization of our sparsity-inducing pruning framework to generative modelling of images highlighting a loss-size trade-off with over an order-of-magnitude improvement over prior work.
- A Hessian-based analysis of the training of sparse flows that indicates an improved, i.e., flattened, loss landscape of pruned neural ODEs further corroborating our findings.

8.1.2 Relevant Papers

The results presented in this chapter are based on the following paper:

- Lucas Liebenwein, Ramin Hasani, Alexander Amini, and Daniela Rus. Sparse flows: Pruning continuous-depth models. In *Advances in Neural Information Processing Systems (Under Review; arXiv preprint arXiv:2106.12718)*, 2021b.

8.1.3 Outline

In the following, we provide the necessary background on neural ODEs and continuous normalizing flows in Section 8.2. Next, we introduce our pruning framework in Section 8.3 and provide the necessary hyperparameters to instantiate our pruning framework for the considered networks and datasets in Section 8.4. Our main results and experimental analysis are presented in Section 8.5 and we conclude the chapter with a discussion in Section 8.6.

8.2 Background

In this section, we describe the necessary background topics to construct our framework. We show how to perform generative modeling by continuous depth models using the change of variables formula.

Generative modeling via change of variables. The change of variables formula uses an invertible mapping $f : \mathbb{R}^D \to \mathbb{R}^D$, to wrap a normalized base distribution $p_z(\mathbf{z})$, to specify a more complex distribution. In particular, given $z \sim p_z(\mathbf{z})$, a random variable, the log density for function $f(\mathbf{z}) = \mathbf{x}$ can be computed by Grathwohl et al. (2019):

$$\log p_x(\mathbf{x}) = \log p_z(\mathbf{z}) - \log \det \left| \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right|, \tag{8.1}$$

where the Jacobian of f is determined by $\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}$. While theoretically (8.1) demonstrates a straightforward way to find the log density, from a practical standpoint computation of the Jacobian determinant has a time complexity of $\mathcal{O}(D^3)$. Restricting neural network architectures can make the computation of this term more tractable. Examples include designing normalizing flows (Berg et al., 2018; Papamakarios et al., 2017; Rezende and Mohamed, 2015), autoregressive transformations (Durkan et al., 2019; Jaini et al., 2019; Kingma et al., 2016; Müller et al., 2019; Oliva et al., 2018a; Wehenkel and Louppe, 2019), partitioned transformations (Dinh et al., 2016; Kingma and Dhariwal, 2018), universal flows (Kong and Chaudhuri, 2020; Teshima et al., 2020), and the use of optimal transport theorem (Huang et al., 2020).

Alternative to these discrete transformation algorithms, one can construct a generative model similar to (8.1), and declare f by a continuous-time dynamics (Chen et al., 2018b; Grathwohl et al., 2019; Lechner et al., 2020b). Given a sample from the base distribution, one can parametrize an ordinary differential equations (ODEs) by a function $f(\mathbf{z}(t), t, \theta)$, and solve the ODE to obtain the observable data. When f is a neural network, the system is called a neural ODE (Chen et al., 2018b).

Neural ODEs. More formally, a neural ODE is defined by finding the solution to the initial value problem (IVP): $\frac{\partial \mathbf{z}(t)}{\partial t} = f(\mathbf{z}(t), t, \theta)$, $\mathbf{z}(t_0) = \mathbf{z}_0$, with $\mathbf{z}_0 \sim p_{z_0}(\mathbf{z}_0)$, to get $\mathbf{z}(t_n)$ the desired output observations at a terminal integration step n (Chen et al., 2018b).[1]

Continuous normalizing flows. If $\mathbf{z}(t_n)$ is set to our observable data, given samples from the base distribution $\mathbf{z}_0 \sim p_{z_0}(\mathbf{z}_0)$, the neural ODE described above forms a continuous normalizing flow (CNF). CNFs modify the change in log density by the left hand-side differential equation and as a result the total change in log-density by the right hand-side equation (Chen et al., 2018b; Grathwohl et al., 2019):

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right), \quad \log p(\mathbf{z}(t_n)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt. \tag{8.2}$$

The system of two differential equations (the neural ODE ($\frac{\partial \mathbf{z}(t)}{\partial t} = f(\mathbf{z}(t), t, \theta)$) and (8.2) can then be solved by automatic differentiation algorithms such as backpropagation through time (Rumelhart et al., 1986) or the adjoint sensitivity method (Chen et al., 2018b; Pontryagin, 2018). Computation of $\mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right)$ costs $\mathcal{O}(D^2)$. A method called the Free-form Jacobian of Reversible Dynamics (FFJORD) (Grathwohl et al., 2019) improved the cost to $\mathcal{O}(D)$ by using the Hutchinson's trace estimator (Adams et al., 2018; Hutchinson, 1989). Thus, the trace of the Jacobian can be estimated by: $\mathrm{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) = \mathbb{E}_{p(\boldsymbol{\varepsilon})}\left[\boldsymbol{\varepsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\varepsilon}\right]$, where $p(\varepsilon)$ is typically set to a Gaussian or Rademacher distribution (Grathwohl et al., 2019). Throughout the paper, we investigate the properties of FFJORD CNFs by pruning their neural network architectures.

8.3 Pruning Neural ODEs

We enable sparsity in neural ODEs and CNFs by removing, i.e., pruning, redundant weights from the underlying neural network architecture during training. Pruning

[1] One can design a more expressive representation (Hasani et al., 2020; Vorbach et al., 2021) of continuous-depth models by using the second-order approximation of the neural ODE formulation (Hasani et al., 2021b). This representation might give rise to a better neural flows which will be the focus of our continued effort.

can tremendously improve the parameter efficiency of neural networks across numerous tasks, such as computer vision (Liebenwein et al., 2020) and natural language processing (Maalouf et al., 2021).

8.3.1 A General Framework for Training Sparse Flows

Our approach to training Sparse Flows is inspired by *iterative learning rate rewinding*, a recently proposed and broadly adopted pruning framework as used by Liebenwein et al. (2020); Renda et al. (2020) among others.

In short, our pruning framework proceeds by first training an unpruned, i.e., dense network to obtain a warm initialization for pruning. Subsequently, we proceed by iteratively pruning and retraining the network until we either obtain the desired level of sparsity, i.e., prune ratio or when the loss for a pre-specified hold-out dataset (validation loss) starts to deteriorate (early stopping).

We note that our framework is readily applicable to any continuous-depth model and not restricted to FFJORD-like models. Moreover, we can account for various types of pruning, i.e., unstructured pruning of weights and structured pruning of neurons or filters. An overview of the framework is provided in Algorithm 6 and we provide more details below.

8.3.2 From Dense to Sparse Flows

Train a dense flow for a warm initialization. To initiate the training process, we first train a densely-connected network to obtain a warm initialization (Line 2 of Algorithm 6). We use Adam with a fixed step learning decay schedule and weight decay in some instances. Numerical values for each network and dataset are provided in the supplementary material. Based on the warm initialization, we start pruning the network.

Prune for Sparse Flow. For the prune step (Line 5 of Algorithm 6) we either consider unstructured or structured pruning, i.e., weight or neuron/filter pruning,

Table 8.1: Pruning Methods.

	Unstructured Pruning (Han et al., 2015a)	Structured Pruning (Li et al., 2016)
Target	Weights	Neurons and Filters
Importance Score	$\|W_{ij}\|$	$\|W_{i:}\|_1$
Scope	Global	Local

Algorithm 6 SparseFlow(f, Φ_{train}, PR, e)

Input: f: neural ODE model with parameter set θ; Φ_{train}: hyper-parameters for training; PR: relative prune ratio; e: number of training epochs per prune-cycle.
Output: $f(\cdot;\hat{\theta})$: Sparse Flow; m: sparse connection pattern.

1: $\theta_0 \leftarrow$ RandomInit()
2: $\theta \leftarrow$ Train($\theta_0, \Phi_{\text{train}}, e$) ▷ Initial training stage with dense neural ODE ("warm start").
3: $m \leftarrow 1^{|\theta_0|}$ ▷ Initialize binary mask indicating neural connection pattern.
4: **while** validation loss of Sparse Flow decreases **do**
5: $m \leftarrow$ Prune($m \odot \theta, PR$) ▷ Prune PR% of the *remaining* parameters and update binary mask.
6: $\theta \leftarrow$ Train($m \odot \theta, \Phi_{\text{train}}, e$) ▷ Continue training with updated connection pattern.
7: **end while**
8: $\hat{\theta} \leftarrow m \odot \theta$
9: **return** $f(\cdot;\hat{\theta})$, m

respectively. At a fundamental level, unstructured pruning aims at inducing sparsity into the parameters of the flow while structured pruning enables reducing the dimensionality of each flow layer. For unstructured pruning, we use magnitude pruning (Han et al., 2015a)), where we prune weights across all layers (global) with magnitudes below a pre-defined threshold. For structured pruning, we use the ℓ_1-norm of the weights associated with the neuron/filter and prune the structures with lowest norm for constant per-layer prune ratio (local) as proposed by Li et al. (2016). See Table 8.1 for an overview.

Train the Sparse Flow. Following the pruning step, we re-initiate the training with the new sparsity pattern and the unpruned weights (Line 6 of Algorithm 6).

Note that we generally follow the same hyperparameters as during the initial, dense training stage.

Iterate for increased sparsity and performance. Naturally, we can iteratively repeat the PRUNE and TRAIN step to further sparsify the flow (Lines 4-7 of Algorithm 6). Moreover, the resulting sparsity-performance trade-off is affected by the total number of iterations, the relative prune ratio PR per iteration, and the amount of training between PRUNE steps. We find that a good trade-off is to keep the amount of training constant across all iterations and tune it such that the initial, dense flow is essentially trained to (or close to) convergence. Depending on the difficulty of the task and the available compute resources we can then adapt the per-iteration prune ratio PR. Note that the overall relative sparsity after n iterations is given by $(1 - PR)^n$. Detailed hyperparameters for each experiment are provided in the supplementary material.

8.4 Experimental Setup

We provide the necessary hyperparameters to reproduce our experiments below. For each set of experiments (Toy, Tabular, Images) we summarize the architecture of the unpruned model and relevant hyperparameters pertaining to the training/pruning process. For the experiments on the toy datasets, we based our code on the TorchDyn library (Poli et al., 2020a). For the experiments on the tabular datasets and image experiments, we based our code on the official code repository of FFJORD (Grathwohl et al., 2019).

8.5 Experiments

We perform a diverse set of experiments demonstrating the effect of pruning on the generalization capability of continuous-depth models. Our experiments include pruning ODE-based flows in density estimation tasks with increasing complexity, as well

Table 8.2: **Toy Dataset** Hyperparameters.

Hyperparameters		**Gaussians**	**GaussianSpiral**	**Spiral**	**Moon**
Architecture	Layers	2	4	4	2
	Hidden Size	128	64	64	128
	Activation	Sigmoid	Sigmoid	Sigmoid	Tanh
	Divergence	Hutchison	Hutchison	Hutchison	Hutchison
Solver	Type	Dopri	Dopri	Dopri	Dopri
	Rel. tol.	1.0e-5	1.0e-5	1.0e-5	1.0e-4
	Abs. tol	1.0e-5	1.0e-5	1.0e-5	1.0e-4
	Backprop.	Adjoint	Adjoint	Adjoint	Adjoint
(Re-)Training	Optimizer	AdamW	AdamW	AdamW	Adam
	Epochs	100	100	100	50
	Batch size	1024	1024	1024	128
	LR	5.0e-3	5.0e-2	5.0e-2	1.0e-2
	β_1	0.9	0.9	0.9	0.9
	β_2	0.999	0.999	0.999	0.999
	Weight decay	1.0e-5	1.0e-2	1.0e-6	1.0e-4
Pruning	PR	10%	10%	10%	10%

as pruning neural ODEs in supervised inference tasks. The density estimation tasks were conducted on flows equipped with Free-form Jacobian of Reversible Dynamics (FFJORD) (Grathwohl et al., 2019), using adaptive ODE solvers (Dormand and Prince, 1980). We used two code bases (FFJORD from Grathwohl et al. (2019) and TorchDyn (Poli et al., 2020a)) over which we implemented our pruning framework.

Baselines. In complex density estimation tasks we compare the performance of Sparse Flows to a variety of baseline methods including: FFJORD (Grathwohl et al., 2019), masked autoencoder density estimation (MADE) (Germain et al., 2015), Real NVP (Dinh et al., 2016), masked autoregressive flow (MAF) (Papamakarios et al., 2017), Glow (Kingma and Dhariwal, 2018), convex potential flows (CP-Flow) (Huang et al., 2020), transformation autoregressive networks (TAN) (Oliva et al., 2018b), neural autoregressive flows (NAF) (Huang et al., 2018a), and sum-of-squares polynomial flow (SOS) (Jaini et al., 2019).

Table 8.3: **Tabular Datasets** Hyperparameters.

Hyperparameters		**Power**	**Gas**	**Hepmass**	**Miniboone**	**Bsds300**
Architecture		Please refer to Table 4, Appendix B.1 of Grathwohl et al. (2019).				
Solver		Please refer to Appendix C of Grathwohl et al. (2019).				
(Re-)Training	Optimizer	Adam	Adam	Adam	Adam	Adam
	Epochs	100	30	400	400	100
	Batch size	10000	1000	10000	1000	10000
	LR	1.0e-3	1.0e-3	1.0e-3	1.0e-3	1.0e-3
	LR step	0.1@{90, 97}	0.1@{25, 28}	0.1@{250, 295}	0.1@{300, 350}	0.1@{96, 99}
	β_1	0.9	0.9	0.9	0.9	0.9
	β_2	0.999	0.999	0.999	0.999	0.999
	Weight decay	1.0e-6	1.0e-6	1.0e-6	1.0e-6	1.0e-6
Pruning	PR	25%	25%	22%	22%	25%

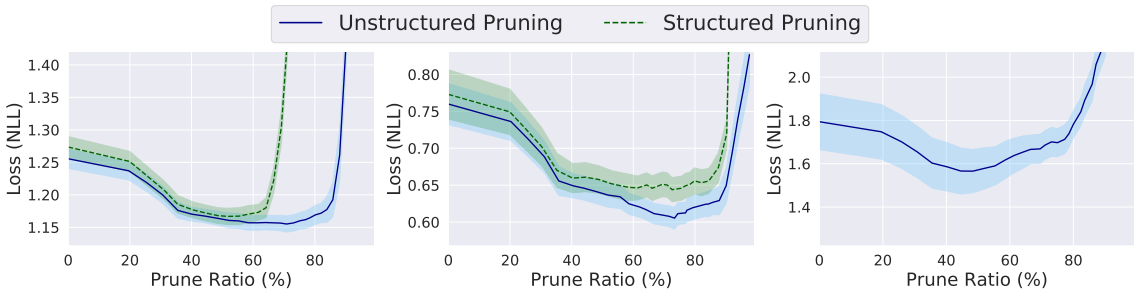
Table 8.4: **Image Datasets** Hyperparameters.

Hyperparameters		**MNIST**	**CIFAR-10**
Architecture		Please refer to Appendix B.1 (multi-scale) of Grathwohl et al. (2019).	
Solver		Please refer to Appendix C of Grathwohl et al. (2019).	
(Re-)Training	Optimizer	Adam	Adam
	Epochs	50	50
	Batch size	200	200
	LR	1.0e-3	1.0e-3
	LR step	0.1@{45}	0.1@{45}
	β_1	0.9	0.9
	β_2	0.999	0.999
	Weight decay	0.0	0.0
Pruning	PR	22%	22%

8.5.1 Density Estimation on 2D Data

In the first set of experiments, we train FFJORD on a multi-modal Gaussian distribution, a multi-model set of Gaussian distributions placed orderly on a spiral as well as a spiral distribution with sparse regions. Figure 8-4 (first row) illustrates that densely connected flows (prune ratio = 0%) might get stuck in sharp local minima and as a result induce mode collapse (Srivastava et al., 2017). Once we perform unstructured pruning, we observe that the quality of the density estimation in all tasks considerably improves, c.f. Figure 8-4 (second and third rows). If we continue sparsifying the flows, depending on the task at hand, the flows get disrupted again.

Therefore, there is a certain threshold for pruning flows required to avoid generative modeling issues such as mode-collapse in continuous flows. We validate this



(a) Gaussians (b) Gaussian Spiral (c) Spiral

Figure 8-3: Negative log likelihood of Sparse Flow as function of prune ratio.

observation by plotting the negative log-likelihood loss as a function of the prune ratio in all three tasks with both unstructured and structured pruning. As shown in Figure 8-3, we confirm that sparsity in flows improves the performance of continuous normalizing flows.

We further explore the inner dynamics of the flows between unpruned and pruned networks on the multi-modal case, with the aim of understanding how pruning enhances density estimation performance. Figure 8-5 represents the vector-field constructed by each flow to model 6-Gaussians independently. We observe that sparse flows with PR of 70% attract the vector-field directions uniformly towards the mean of each Gaussian distribution. In contrast, unpruned flows do not exploit this feature and contain converging vectors in between Gaussians. This is how the mode-collapse occurs.

8.5.2 Density Estimation on Real Data – Tabular

We scale our experiments to a set of five real-world tabular datasets (prepared based on the instructions given by Papamakarios et al. (2017) and Grathwohl et al. (2019)) to verify our empirical observations about the effect of pruning on the generalizability of continuous normalizing flows. Table 8.5 summarizes the results. We observe that sparsifying FFJORD flows substantially improves their performance in all 5 tasks. In particular, we gain up to 42% performance gain in the POWER, 35% in GAS, 12% in HEPMASS, 5% in MINIBOONE and 19% in BSDS300.

More importantly, this is achieved with flows with 1 to 3 orders of magnitude

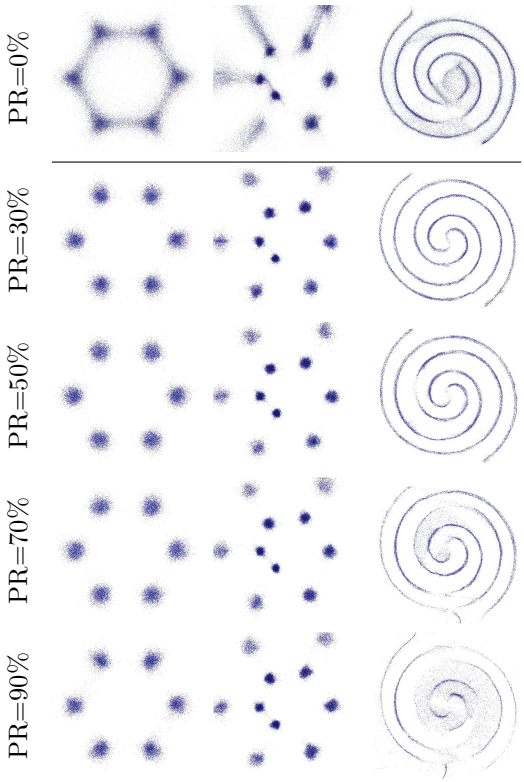


Figure 8-4: Unstructured pruning of FFJORD (PR= Prune ratio).

less parameters compared to other advanced flows. On MINIBOONE dataset for instance, we found a sparse flow with only 4% of its original network that outperforms its densely-connected FFJORD flow. On MINIBOONE, Autoregressive flows (NAF) and sum-of-squares models (SOS) which outperform all other models possess 8.03 and 6.87 million parameters. In contrast, we obtain a Sparse Flow with only 32K parameters that outperform all models except NAF and SOS.

Let us now look at the loss versus prune-ratio trends in all experiments to conclude our empirical observations on real-world tabular datasets. As shown in Figure 8-7, we observe that pruning considerably improves the performance of flows at larger scale as well.

8.5.3 Density Estimation on Real-Data – Vision

Next, we extend our experiments to density estimation for image datasets, MNIST and CIFAR10. We observe a similar case on generative modeling with both datasets,

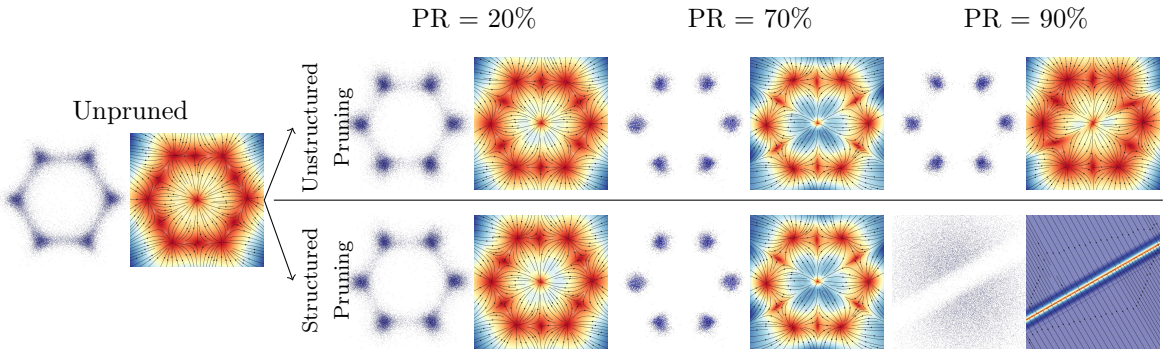
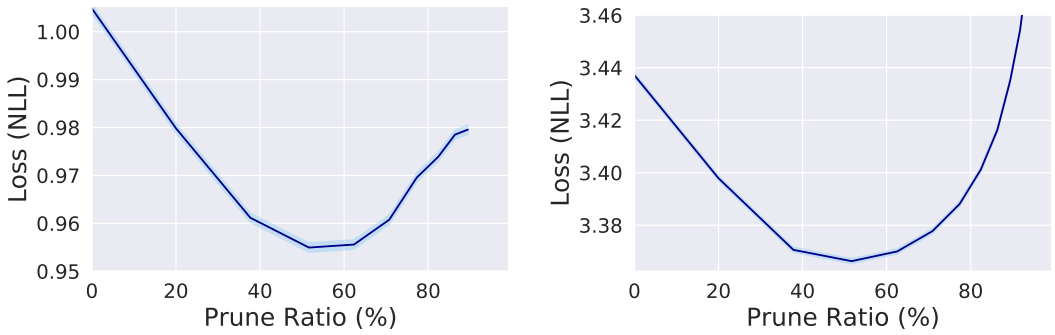


Figure 8-5: Multi-modal Gaussian flow and pruning. We observe that Sparse Flows attract the vector-field directions uniformly towards the mean of each Gaussian distribution, while an unpruned flow does not exploit this feature and contains converging vectors in between Gaussians.

where pruned flows outperform densely-connected FFJORD flows. On MNIST, a sparse FFJORD flow with 63% of its weights pruned outperforms all other benchmarks. Compared to the second best flow (Residual flow), our sparse flow contains 70x less parameters (234K vs 16.6M). On CIFAR10, we achieve the second best performance with over 38x less parameters compared to Residual flows which performs best.



(a) MNIST.

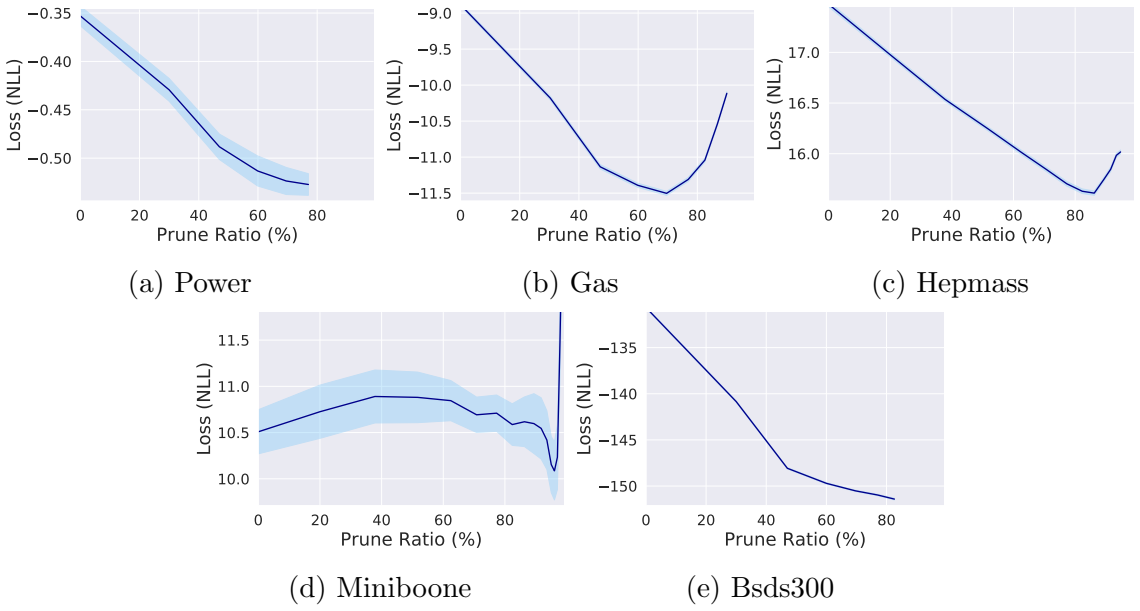
(b) CIFAR10.

Figure 8-6: Loss vs. prune ratio for MNIST and CIFAR10. Unstructured Pruning is applied.

Furthermore, on CIFAR10, Glow with 44 million parameters performs on par with our sparse FFJORD network with 657k parameters. It is worth mentioning that FFJORD obtains this results by using a simple Gaussian prior, while Glow takes advantage of learned base distribution (Grathwohl et al., 2019). Figure 8-6 illustrates

Table 8.5: Negative test log-likelihood (NLL) in nats of tabular datasets from (Papamakarios et al., 2017) and corresponding architecture size in number of parameters (#params). Sparse Flow (based on FFJORD) with lowest NLL and competing baseline with lowest NLL are bolded.

Model	Power		Gas		Hepmass		Miniboone		BSDS300	
	nats	#params	nats	#params	nats	#params	nats	#params	nats	#params
MADE (Germain et al., 2015)	3.08	6K	-3.56	6K	20.98	147K	15.59	164K	-148.85	621K
Real NVP (Dinh et al., 2016)	-0.17	212K	-8.33	216K	18.71	5.46M	13.84	5.68M	-153.28	22.3M
MAF (Papamakarios et al., 2017)	-0.24	59.0K	-10.08	62.0K	17.70	1.47M	11.75	1.64M	-155.69	6.21M
Glow (Kingma and Dhariwal, 2018)	-0.17	N/A	-8.15	N/A	18.92	N/A	11.35	N/A	-155.07	N/A
CP-Flow (Huang et al., 2020)	-0.52	5.46M	-10.36	2.76M	16.93	2.92M	10.58	379K	-154.99	2.15M
TAN (Oliva et al., 2018b)	**-0.60**	N/A	**-12.06**	N/A	**13.78**	N/A	11.01	N/A	**-159.80**	N/A
NAF (Huang et al., 2018a)	**-0.62**	**451K**	**-11.96**	**443K**	15.09	10.7M	**8.86**	**8.03M**	-157.73	42.3M
SOS (Jaini et al., 2019)	**-0.60**	**212K**	**-11.99**	**256K**	15.15	4.43M	**8.90**	**6.87M**	-157.48	9.09M
FFJORD (Grathwohl et al., 2019)	-0.35	43.3K	-8.58	279K	17.53	547K	10.50	821K	-128.33	6.70M
Sparse Flow	-0.45	30K	-10.79	194K	16.53	340K	10.84	397K	-145.62	4.69M
	-0.50	23K	-11.19	147K	15.82	160K	10.81	186K	-148.72	3.55M
	-0.53	**13K**	**-11.59**	**85K**	**15.60**	**75K**	**9.95**	**32K**	-150.45	2.03M
	-0.52	10K	-11.47	64K	15.99	46K	10.54	18K	**-151.34**	**1.16M**



(a) Power (b) Gas (c) Hepmass

(d) Miniboone (e) Bsds300

Figure 8-7: Negative log-likelihood versus prune ratio on tabular datasets with unstructured pruning.

the improvement of the loss (negative log-likelihood) in density estimation on image datasets as a result of pruning neural ODEs. Around 60% sparsity of a continuous normalizing flow leads to better generative modeling compared to densely structured flows.

Table 8.6: Negative test log-likelihood (NLL) in bits/dim for image datasets and corresponding architecture size in number of parameters (#params). Sparse Flow (based on FFJORD) with lowest NLL and competing baseline with lowest NLL are bolded.

Model	**MNIST**		**CIFAR-10**	
	bits/dim	#params	bits/dim	#params
MADE (Germain et al., 2015)	1.41	1.20M	5.80	11.5M
Real NVP (Dinh et al., 2016)	1.05	N/A	3.49	N/A
MAF (Papamakarios et al., 2017)	1.91	12.0M	4.31	115M
Glow (Kingma and Dhariwal, 2018)	1.06	N/A	3.35	44.0M
CP-Flow (Huang et al., 2020)	1.02	2.90M	3.40	1.90M
TAN (Oliva et al., 2018b)	1.19	N/A	3.98	N/A
SOS (Jaini et al., 2019)	1.81	17.2M	4.18	67.1M
RQ-NSF (Durkan et al., 2019)		N/A	3.38	11.8M
Residual Flow (Chen et al., 2019)	**0.97**	**16.6M**	**3.28**	**25.2M**
FFJORD (Grathwohl et al., 2019))	1.01	801K	3.44	1.36M
Sparse Flows (PR=20%)	0.97	641K	3.38	1.09M
Sparse Flows (PR=38%)	0.96	499K	3.37	845K
Sparse Flows (PR=52%)	0.95	387K	**3.36**	**657K**
Sparse Flows (PR=63%)	**0.95**	**302K**	3.37	510K
Sparse Flows (PR=71%)	0.96	234K	3.38	395K
Sparse Flows (PR=77%)	0.97	182K	3.39	308K
Sparse Flows (PR=82%)	0.98	141K	3.40	239K
Sparse Flows (PR=86%)	0.97	109K	3.42	186K

8.5.4 Pruning Flattens the Loss Surface

What could be the potential reason for the enhanced generalization of pruned CNFs besides their ability to resolve mode-collapse which we observed before? To investigate this further, we conducted a Hessian-based analysis on the flows-under-test. Dissecting the properties of the Hessian by Eigenanalysis allows us to gain useful insights about the behavior of neural networks (Erichson et al., 2021; Ghorbani et al., 2019; Hochreiter and Schmidhuber, 1997; Lechner and Hasani, 2020; Sagun et al., 2017). We use PyHessian (Yao et al., 2020) tool-set to analyze the Hessian H w.r.t. the parameters of the CNF. This enables us to study the curvature of the loss function as the eigenvalues of the Hessian determines (locally) the loss gradients' changes.

Larger Hessian eigenvalues therefore, stand for sharper curvatures and their sign

Table 8.7: Eigenanalysis of the Hessian H in terms of the largest eigenvalue (λ_{max}), trace (tr), and condition number (κ) of pruned and unpruned continuous normalizing flows on the mixture of Gaussian task. Numbers are normalized with respect to the unpruned flow.

Model	NLL	$\lambda_{max}(H)$	tr(H)	$\kappa(H)$
Unpruned FFJORD	1.309	1.000	1.000	1.000
Sparse Flows (PR=20%)	1.163	0.976	0.858	0.825
Sparse Flows (PR=60%)	1.125	0.356	0.583	0.717
Sparse Flows (PR=70%)	**1.118**	**0.295**	**0.340**	**0.709**

identifies upward or downward curvatures. In Table 8.7, we report the maximum eigenvalue of the Hessian λ_{max}, Hessian's Trace, and the condition number $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$.[2] Smaller λ_{max} and tr(H) indicates that our normalizing flow found a flatter minimum. As shown in Figure 8-2b, a flat minimum leads to a better generalization error as opposed to a sharp minimum. We find that up to a certain prune ratio, the maximum of the Hessian decreases and increases thereafter. Therefore, we claim that pruned continuous flows finds flatter local minimum, therefore, it generalize better than their unpruned version. Moreover, the Hessian condition number κ could be an indicator of the robustness and efficiency of a deep model (Bottou and Bousquet, 2008). Smaller κ corresponds to obtaining a more robust learned agent. We observe that κ also follows the same trend and up to a certain prune ratio it shrinks and then increases again confirming our hypothesis.

8.5.5 On the Robustness of Decision Boundaries

While pruning improves performance, it is imperative to investigate their robustness properties in constructing decision boundaries (Lechner et al., 2021). For feedforward deep models it was recently shown that although pruned networks perform as well as their unpruned version, their robustness significantly decreases due to smaller network size (Liebenwein et al., 2021a). Is this also the case for neural ODEs?

We design an experiment to investigate this. We take a simple 2-Dimensional

[2]This experiment was inspired by the Hessian-based robustness analysis performed by Erichson et al. (2021).

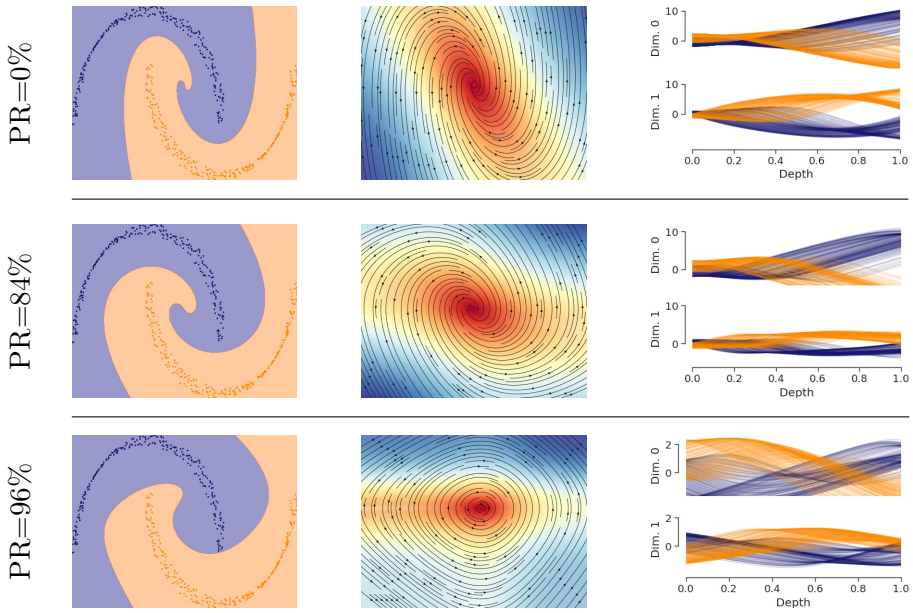


Figure 8-8: Robustness of decision boundaries for pruned networks. Column 1 is the decision boundary. Column 2 = state-space, and column 3 = the flow of data points.

moon dataset and perform classification using unpruned and pruned neural ODEs.[3] This experiment (shown in Figure 8-8) demonstrates another valuable property of neural ODEs: We observe that neural ODE instances pruned up to 84%, manage to establish a safe decision boundary for the two half moon classes. This insight about neural ODEs which was obtained by our pruning framework is another testimony of the merits of using neural ODE in decision-critical applications. Additionally, we observe that the decision-boundary gets very close to one of the classes for a network pruned up to 96% which reduces robustness. Though even this network instance provide the same classification accuracy compared to the densely connected version.

The state-space plot illustrates that a neural ODE's learned vector-field becomes edgier as we keep pruning the network. Nonetheless, we observe that the distribution of the vector-field's intensity (the color map in the second column) get attenuated as we go further away from its center, in highly pruned networks. This indicates that classification near the decision boundary is more sensitive to perturbations in extremely pruned networks.

[3]This experiment is performed using the TorchDyn library (Poli et al., 2020a).

8.6 Discussion

We show the effectiveness of pruning for continuous neural networks. Pruning improves generalization performance of continuous normalizing flows in density estimation tasks at scale. Additionally, pruning allows us to obtain performant minimal network instances with at least one order of magnitude less parameter count. By providing key insights about how pruning improves generative modeling and inference, we enabled the design of better neural ODE instances.

Ensuring sparse Hessian computation for sparse continuous-depth models. As we prune neural ODE instances, their weight matrices will contain zero entries. However the Hessian with respect to those zero entries is not necessarily zero. Therefore, when we compute the eigenvalues of the Hessian, we must ensure to make the decomposition vector sets the Hessian of the pruned weights to zero before performing our eigenanalysis.

What are the limitations of Sparse Flows? Similar to any ODE-based learning system, the computational efficiency of Sparse Flows is highly determined by the choice of their ODE solvers, data and model parameters. As the complexity of any of these fronts increases, the number of function evaluations for a given task increases. Thus we might have a slow training process. This computational overhead can be relaxed in principle by the use of efficient ODE solvers (Poli et al., 2020b) and flow regularization schemes (Massaroli et al., 2020).

What design notes did we learn from applying pruning to neural ODEs? Our framework suggested that to obtain a generalizable sparse neural ODE representation, the choice of activation function is important. In particular activations that are Lipschitz continuous, monotonous, and bounded are better design choices for density estimation tasks (Akiba et al., 2019; Hasani et al., 2019a,b). Moreover, we find that the generalizability of sparse neural ODEs is more influenced by their neural network's width than their depth (number of layers). Furthermore, pruning neural

ODEs allows us to obtain better hyperparameters for the optimization problem by setting a trade-off between the value of the learning rate and weight decay.

In summary, we hope to have shown compelling evidence for the effectiveness of having sparsity in ODE-based flows.

Chapter 9

Conclusion

9.1 Summary

In this thesis, we present multiple avenues towards designing more capable and more efficient deep learning systems. We take a holistic approach in the sense that we initially analyze neural networks from first principles to derive analytical bounds on the performance-efficiency trade-offs in modern deep learning architectures. We then take a deep dive into translating our theoretical findings into practical algorithms that can generate efficient neural networks and applications thereof.

We contribute a novel set of techniques to analyze neural networks that are rooted in coresets and provide an intuitive analytical tool box in terms of empirical sensitivity, i.e., a novel measure of the relative importance of individual structures in the network (Part I). Moreover, our sensitivity-based framework can serve as a modular and flexible proving technique for various types of network architectures.

We then derive practical algorithms from our theoretical insights for obtaining efficient networks using filter pruning and low-rank compression (Part II). To this end, we build upon our analytical bounds that characterize the emerging trade-off between size and approximation error of individual network layers. Thereafter, we develop state-of-the-art methods that aim at optimally compressing individual layers while accounting for the resulting approximation error in the network. We find that these insights lead to more efficient and performant architectures in practice.

Finally, we study the effects of pruning beyond common benchmarks to analyze the potential of pruning in a broader spectrum of prospective applications (Part III). On the one hand, we find that pruning may adversely affect the generalization ability of the network to generalize to out-of-distribution input data. We empirically show that this phenomenon is linked to the intrinsic relation between the stochastic training procedure and the overparameterized optimization landscape. On the other hand, we provide compelling evidence that pruning can improve the convergence of training by removing redundant parameters and thus help in the training procedure to find better-generalizing local optima. We empirically analyze these observations in the context of continuous-depth models. Their underlying computation graph is significantly more complex given the recurrent nature of these models and pruning can thus act as potent regularizer during training.

9.2 Lessons Learned

Below, we provide an informal platform to discuss some of the key insights that the author gathered over the course of the thesis research.

Theory-informed Empiricism

The research presented in this thesis very much starts out with the premise of developing a theoretical framework to describe the performance-size trade-off in neural networks. We analyze how we can downsample neural network parameters and provide accompanying relative error bounds with high probability. We also draw connections to the theory of generalization, which in some sense can be seen as the theoretical "sibling" of pruning. In generalization theory, we aim to describe the required complexity of the underlying architectures for a given learning task. In pruning, we aim to optimize for the architecture and accompanying weights with lowest complexity for a given learning task.

In that sense, our theoretical results help us gain important insights into what is theoretically possible in the analysis of neural networks. One of the main insights,

we gain from our theoretical analysis is that we can often times accurately capture and describe local, layer-wise behavior of the network. However, analyzing the entire network architecture often time leads to highly inaccurate bounds given that we have to repeatedly consider upper bounds for the resulting error accumulated across many layers. This problem is exacerbated by the highly flexible configuration space of networks and the resulting diversity in architectures.

Likewise in practice, we observe that pruning individual layers is a manageable task. However, understanding how an architecture should be optimally pruned as a whole is a notoriously difficult task that we can only solve approximately optimal even with today's compute capabilities. These connections between some of the weaknesses in the theoretical analysis and corresponding empirical observations can enable us to leverage our theory to design better and more-informed pruning algorithms.

To this end, we leverage our tight error analysis of individual layers to devise efficient per-layer pruning strategy. Specifically, we are still able to leverage our theoretical description of the per-layer pruning procedure in terms of the relative error guarantees to devise approximately optimal pruning strategies for the entire network.

Unfortunately, we cannot simply use our global network analysis to inform our pruning strategy in the same manner. We are hereby limited by our analysis as the resulting error bounds often times only constitute a fairly loose upper bound. That is where our main insight comes into play: Our theoretical analysis indicates that the overall error in the network can be described as a weighted sum, or as some function more generally speaking, of the relative error in individual layers. We then use this insight to design better pruning algorithms in practice that are based on optimizing a convex objective that is a function of the relative error in individual layers.

This example illustrates that we can often times develop and design performant algorithms in practice with insights developed from the accompanying theory even in research domains like deep learning that are mostly driven by empirical results. Unlike some typical problems in computer science (graph algorithms, sorting algorithms, and so many more) however, we cannot necessarily hope to develop a theoretical

description that directly translates into performant algorithms. Such connections, as seen with the research presented in this thesis, are often times more entangled and require a careful trade-off with some empirical observations and intuition.

Empiricism-informed Theory

In the same spirit that theory can guide our empirical performance improvements, empiricism can also lead to better approaches in theory. In this thesis, our empirical insights have guided our theoretical exploration as well.

A key observation of our early experiments is that heuristics can achieve a close-to-optimal performance on unstructured pruning. Specifically, using magnitude pruning, i.e., dropping the weights with lowest absolute value up to a desired threshold, leads to consistent state-of-the-art results in terms of the performance-size trade-off across numerous benchmarks and pruning setups.

However, at the same time, there is a gap in the literature of how to optimally prune network architectures in a structured manner, e.g., by removing entire channels or low-rank decomposing individual layers. As a consequence, prior work often relies on manual ablation studies, large-scale grid searches, or other types of error-and-trial approaches to better assess the trade-offs. While this is useful to gain insights for particular use cases, such observations hardly generalize beyond the investigated network architectures.

This conclusion has led us to analyze different types of pruning, mainly neuron and channel pruning, using our empirical sensitivity framework. As part of the research presented in this thesis, we show how we can generalize our theoretical analysis to channel pruning and devise a practical channel pruning algorithm. In our experimental evaluations, we see that our theory-guided approach leads to clear improvements in terms of the performance-size trade-off for neural networks.

In this particular scenario, empirical observations enabled us to understand for what type of problems our intuition about the problem – and thus our ability to devise suitable heuristics – was sufficient to design effective algorithms (cf. weight pruning) and where it was not (cf. channel pruning). More generally speaking, empiricism can

provide valuable insights into understanding where a deeper theoretical understanding is required in order to obtain sufficiently performant algorithms.

A Robust Experimental Platform

While conducting the thesis research, the author co-developed a robust and scalable experiment platform for large-scale testing of our algorithms. This greatly reduced the friction between our empirical observations and theoretical advancements as well as enabled us to quickly iterate on our algorithms. While developing robust software might be a large up-front cost to pay in a fast-paced research environment, it greatly improves upon the quality of research in the long run.

At the end of the day, deep learning always imposes a significant experimental overhead as we experiment with different networks, datasets, pruning pipelines, or hyperparameters. Without the ability to quickly test a variety of configurations, we might be misguided, overly focus on aspects of the research that do not lead to significant improvements, or overfit to a particular scenario.

On the other hand, having a robust and standardized experimental platform may lead to new levels of abstractions that in turn may inform new research ideas and fuel breakthroughs in the field. This has already led to significant advancements in recent years, e.g., the advent of Transformers in NLP (Brown et al., 2020) that was enabled by the ability to train neural networks in a highly distributed fashion. We envision that the trend towards standardization and abstraction of common deep learning practices will be crucial in facilitating future advancements in artificial intelligence.

9.3 Closing Remarks

With the push towards global digitalization, we have established the foundations for harnessing and leveraging big data across numerous aspects of our personal and professional lives. Machine learning and, in particular, deep learning are powerful techniques that are increasingly becoming drivers of innovation to turn these enormous amounts of data into consequential insights and so far we have only scratched

the surface of what may be achievable down the road. By reducing the computational cost and consequently the required resources, we can significantly lower the entry barriers to deep learning and broadening the range of potential applications. With the methods presented in this thesis, we aspire to contribute an impactful and valuable set of techniques towards realizing this vision.

Funding

This research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This research was further supported in part by The Boeing Company, the U.S. National Science Foundation (NSF) under Awards 1723943 and 1526815, Office of Naval Research (ONR) Grant N00014-18-1-2830, Microsoft, and JP Morgan Chase.

Appendix A

Appendix: Automatic Layer-wise Decomposition

We provide additional results of our experimental evaluations for Chapter 6. Specifically, we provide tabularized results of our benchmark experiments presented in Section 6.3.2.

Table A.1: The maximal compression ratio for which the drop in test accuracy is at most some pre-specified δ on CIFAR-10. The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded.

	Model	Prune Method	$\delta = 0.0\%$			$\delta = 0.5\%$			$\delta = 1.0\%$			$\delta = 2.0\%$			$\delta = 3.0\%$		
			Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F
CIFAR10	ResNet20 Top1: 91.39	ALDS	+0.09	**64.58**	**55.95**	-0.47	**74.91**	**67.86**	-0.68	**79.01**	**71.59**	-1.88	**87.68**	**83.23**	-2.59	**89.65**	**85.32**
		PCA	+0.16	39.98	38.64	-0.11	49.88	48.67	-0.58	58.04	57.21	-1.41	70.54	70.78	-2.11	75.23	76.01
		SVD-Energy	+0.14	40.22	39.38	-0.21	49.88	49.08	-0.83	57.95	57.15	-1.52	64.76	64.10	-2.17	70.47	70.01
		SVD	+0.24	14.36	15.34	-0.29	39.81	38.95	-0.90	49.19	50.21	-1.08	57.47	57.80	-2.88	70.14	71.31
		L-Rank	+0.14	15.00	29.08	-0.44	28.71	54.89	-0.44	28.71	54.89	-1.56	49.87	72.57	-2.82	64.81	80.80
		FT	+0.15	15.29	16.66	-0.32	39.69	39.57	-0.75	57.77	55.85	-1.88	74.89	71.76	-2.71	79.29	76.74
		PFP	+0.12	28.74	20.56	-0.28	40.28	30.06	-0.85	58.26	46.94	-1.56	70.49	59.78	-2.57	79.28	69.27
	VGG16 Top1: 92.78	ALDS	+0.29	**94.89**	**83.94**	-0.11	**95.77**	**86.23**	-0.52	**97.01**	**88.95**	-0.52	**97.03**	88.95	-0.52	**97.03**	88.95
		PCA	+0.47	87.74	81.05	-0.02	89.72	85.84	-0.02	89.72	85.84	-1.12	91.37	89.57	-1.12	91.37	89.57
		SVD-Energy	+0.35	79.21	78.70	-0.08	82.57	81.32	-0.83	87.74	85.36	-1.22	89.71	87.13	-2.08	91.37	88.58
		SVD	+0.29	70.35	70.13	+0.29	70.35	70.13	-0.74	75.18	75.13	-1.58	82.58	82.39	-1.58	82.58	82.39
		L-Rank	+0.35	82.56	69.67	-0.35	85.38	75.86	-0.35	85.38	75.86	-0.35	85.38	75.86	-0.35	85.38	75.86
		FT	+0.17	64.81	62.16	-0.47	79.13	78.44	-0.87	82.61	82.41	-1.95	89.69	89.91	-2.66	91.35	**91.68**
		PFP	+0.16	89.73	74.61	-0.47	94.87	84.76	-0.96	96.40	88.38	-1.33	97.02	**90.25**	-1.33	97.02	90.25
	DenseNet22 Top1: 89.88	ALDS	+0.17	**48.85**	**51.90**	-0.32	**56.84**	**61.98**	-0.54	**63.83**	**69.68**	-1.87	**69.67**	**74.48**	-1.87	**69.67**	**74.48**
		PCA	+0.20	14.67	34.55	+0.20	14.67	34.55	-0.73	28.83	57.02	-0.73	28.83	57.02	-2.75	40.51	70.03
		SVD-Energy				-0.29	15.16	19.34	-0.29	15.16	19.34	-1.28	28.62	33.26	-2.21	40.20	44.72
		SVD	+0.13	15.00	15.33	+0.13	15.00	15.33	-0.87	26.73	27.41	-0.87	26.73	27.41	-2.51	37.99	39.25
		L-Rank	+0.26	14.98	35.21	+0.26	14.98	35.21	-0.90	28.67	63.55	-1.82	40.33	73.45	-1.82	40.33	73.45
		FT	+0.15	15.49	16.70	-0.24	28.33	29.50	-0.24	28.33	29.50	-1.46	51.10	51.03	-2.40	64.12	63.09
		PFP	+0.00	28.68	32.60	-0.44	40.24	43.37	-0.70	49.67	51.94	-1.36	58.20	58.21	-2.43	65.17	64.50
	WRN16-8 Top1: 95.21	ALDS	+0.05	28.67	13.00	-0.42	**87.77**	79.90	-0.88	**92.75**	87.39	-1.53	**95.69**	92.50	-2.23	**97.01**	**95.51**
		PCA	+0.14	15.00	7.98	-0.49	85.33	**83.45**	-0.96	91.33	**90.23**	-1.76	93.90	**93.15**	-2.45	94.87	94.30
		SVD-Energy	+0.29	15.01	6.92	-0.41	64.75	60.94	-0.81	85.38	83.52	-1.90	91.38	90.04	-2.46	92.77	91.58
		SVD							-0.96	40.20	39.97	-1.63	70.48	70.49	-1.63	70.48	70.49
		L-Rank	+0.25	14.99	6.79	-0.45	49.86	58.00	-0.88	75.20	82.26	-1.70	87.73	92.03	-2.18	89.72	93.51
		FT	+0.03	**64.54**	**61.53**	-0.32	82.33	75.97	-0.95	89.70	83.52	-1.78	94.91	90.82	-2.86	96.42	93.33
		PFP	+0.05	57.92	54.74	-0.44	85.33	80.68	-0.77	89.71	85.16	-1.69	95.65	92.60	-2.40	96.96	94.36

Table A.2: The maximal compression ratio for which the drop in test accuracy is at most $\delta = 1.0\%$ for ResNet20 (CIFAR-10) for various amounts of retraining (as indicated). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded.

	Model	Prune Method	$r = 0\% \, e$			$r = 5\% \, e$			$r = 10\% \, e$			$r = 25\% \, e$			$r = 50\% \, e$			$r = 100\% \, e$		
			Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F	Top1 Acc.	CR-P	CR-F
CIFAR10	ResNet20 Top1: 91.39	ALDS	-0.13	**14.82**	**7.03**	-0.53	**35.87**	**26.27**	-0.73	**43.12**	**33.65**	-0.65	**43.14**	**33.33**	-0.86	**62.39**	**54.40**	-0.88	**81.29**	**74.23**
		PCA							-0.74	19.31	18.64	-0.70	19.34	18.44	-0.59	36.21	35.19	-0.74	60.29	59.81
		SVD-Energy				-0.64	14.99	14.09	-0.70	19.61	18.81	-0.59	19.61	18.81	-0.73	43.46	42.49	-0.46	55.25	54.59
		SVD							-0.83	14.36	15.34	-0.58	14.36	15.34	-0.69	28.21	29.11	-0.77	51.58	51.52
		L-Rank										-0.64	15.00	29.08	-0.33	15.00	29.08	-0.44	28.71	54.89
		FT										-0.67	15.29	16.66	-0.69	27.76	28.40	-0.75	57.77	55.85
		PFP										-0.77	14.88	9.61	-0.83	32.71	23.85	-0.54	52.89	42.04

Table A.3: The maximal compression ratio for which the drop in test accuracy is at most some pre-specified δ on ResNet18 (ImageNet). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded.

	Model	Prune Method	$\delta = 0.0\%$			$\delta = 0.5\%$			$\delta = 1.0\%$			$\delta = 2.0\%$			$\delta = 3.0\%$		
			Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F
ImageNet	ResNet18 Top1: 69.64 Top5: 88.98	ALDS	+0.24/+0.39	**50.33**	**23.62**	-0.40/-0.05	**66.70**	**43.51**	-0.40/-0.05	**66.70**	**43.51**	-1.58/-0.55	**77.62**	**59.51**	-2.70/-1.39	**81.69**	**66.68**
		PCA							-0.95/-0.37	9.99	12.78	-1.71/-0.85	50.42	51.68	-2.40/-1.06	59.38	60.04
		SVD-Energy										-1.49/-0.64	39.56	40.99	-2.90/-1.38	59.43	60.65
		SVD										-1.75/-0.72	50.38	50.37	-2.16/-0.85	59.36	59.33
		L-Rank							-0.71/-0.23	10.01	32.64	-0.71/-0.23	10.01	32.64	-2.38/-1.38	26.24	59.82
		FT	+0.10/+0.42	9.86	11.17	+0.10/+0.42	9.86	11.17	-0.66/-0.17	26.08	26.44	-1.83/-0.73	39.88	38.13	-2.99/-1.41	50.33	47.18
		PFP	+0.36/+0.51	10.09	7.35	-0.39/-0.08	26.35	17.96	-0.39/-0.08	26.35	17.96	-1.62/-0.71	39.66	27.71	-1.62/-0.71	39.66	27.71

Table A.4: The maximal compression ratio for which the drop in test accuracy is at most $\delta = 1.0\%$ for ResNet18 (ImageNet) for various amounts of retraining (as indicated). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded.

	Model	Prune Method	$r = 0\% \, e$			$r = 5\% \, e$			$r = 10\% \, e$			$r = 25\% \, e$			$r = 50\% \, e$			$r = 100\% \, e$		
			Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F	Top1/5 Acc.	CR-P	CR-F
ImageNet	ResNet18 Top1: 69.62 Top5: 89.08	ALDS	-0.54/-0.24	**39.57**	**15.20**	-0.48/-0.24	**50.46**	**23.70**	-0.72/-0.30	**53.50**	**26.90**	-0.64/-0.31	**61.90**	**36.58**	-0.40/-0.23	**68.78**	**47.23**	-0.73/-0.31	**70.73**	**49.85**
		PCA							-inf/-inf	3.33	3.99	-0.80/-0.38	26.21	27.53	-0.76/-0.51	39.53	40.45	-inf/-inf	6.65	8.14
		SVD-Energy				-0.28/-0.14	10.00	11.05	-0.25/-0.12	10.00	11.05	-0.55/-0.25	26.24	27.14	-0.66/-0.33	39.56	40.48	-inf/-inf	3.33	3.66
		SVD				-0.32/-0.13	9.98	9.94	-0.19/-0.07	9.98	9.94	-0.71/-0.34	30.63	30.82	-0.59/-0.32	39.53	39.51	-inf/-inf	3.33	3.31
		L-Rank							-inf/-inf	3.34	10.88	-0.40/-0.23	10.01	32.40	-0.16/+0.03	10.01	32.40	-0.72/-0.26	10.01	32.40
		FT							-inf/-inf	3.36	3.75	-0.21/-0.15	9.95	10.78	-0.83/-0.46	26.29	26.57	-0.66/-0.32	26.12	26.62
		PFP							-inf/-inf	3.34	2.37	-0.14/-0.13	9.96	7.72	-0.37/-0.31	20.76	15.14	-0.38/-0.15	26.35	19.14

Table A.5: The maximal compression ratio for which the drop in test accuracy is at most some pre-specified δ on DeeplabV3-ResNet50 (Pascal VOC2012). The table reports compression ratio in terms of parameters and FLOPs, denoted by CR-P and CR-F, respectively. When the desired δ was not achieved for any compression ratio in the range the fields are left blank. The top values achieved for CR-P and CR-F are bolded.

	Model	Prune Method	$\delta = 0.0\%$			$\delta = 0.5\%$			$\delta = 1.0\%$			$\delta = 2.0\%$			$\delta = 3.0\%$		
			IoU/Top1 Acc.	CR-P	CR-F	IoU/Top1 Acc.	CR-P	CR-F	IoU/Top1 Acc.	CR-P	CR-F	IoU/Top1 Acc.	CR-P	CR-F	IoU/Top1 Acc.	CR-P	CR-F
VOCSegmentation2012	DeeplabV3-ResNet50 IoU: 68.16 Top1: 94.25	ALDS	+0.14/-0.15	**64.38**	**64.11**	+0.14/-0.15	**64.38**	**64.11**	+0.14/-0.15	**64.38**	**64.11**	-1.22/-0.36	**71.36**	**70.89**	-2.76/-0.61	**76.96**	**76.37**
		PCA				-0.26/-0.02	31.59	31.63	-0.88/-0.24	55.68	55.82	-1.74/-0.39	64.33	64.54	-2.54/-0.46	71.29	71.63
		SVD-Energy										-1.88/-0.47	31.61	32.27	-2.78/-0.62	44.99	45.60
		SVD	+0.01/-0.02	14.99	14.85	-0.28/-0.18	31.64	31.51	-0.89/-0.25	45.02	44.95	-1.97/-0.50	64.42	64.42	-1.97/-0.50	64.42	64.42
		L-Rank				-0.42/-0.09	44.99	45.02	-0.42/-0.09	44.99	45.02	-1.29/-0.33	55.74	56.01	-2.50/-0.57	64.39	64.82
		FT															
		PFP	+0.01/-0.05	31.79	30.62	-0.49/-0.21	45.17	43.93	-0.84/-0.32	55.78	54.61	-0.84/-0.32	55.78	54.61	-2.43/-0.61	64.47	63.41

Appendix B

Appendix: Pruning Beyond Test Accuracy

We provide additional details and results pertaining to our experimental setup and results of Chapter 7.

B.1 Detailed Methodology and Prune Results

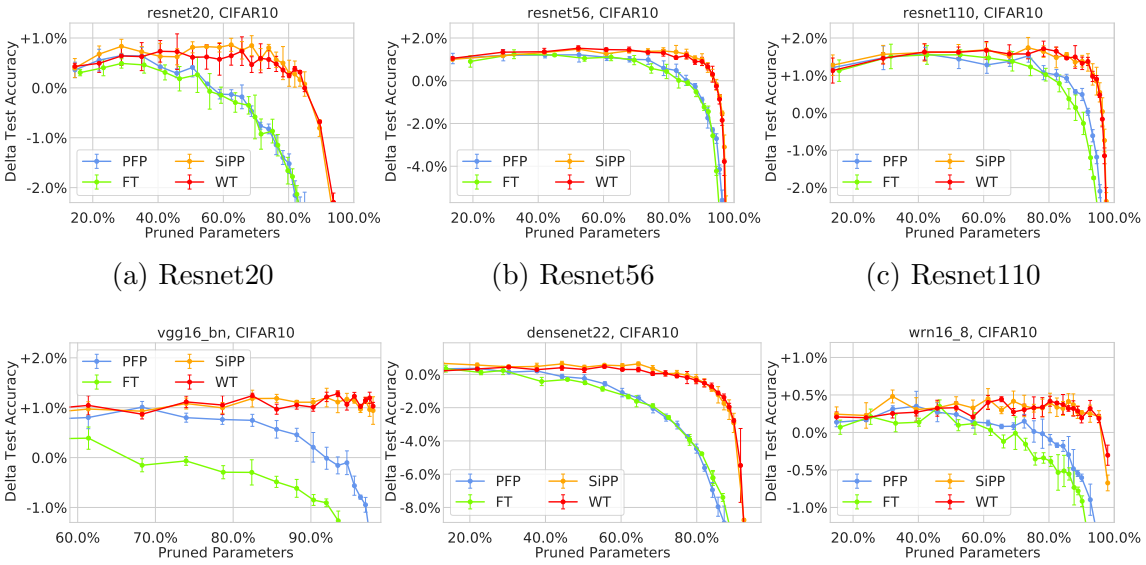
Our experimental evaluations are based on a variety of neural network architectures including ResNets (He et al., 2016), VGGs (Simonyan and Zisserman, 2014), DenseNets (Huang et al., 2017), and WideResNets (Zagoruyko and Komodakis, 2016) trained on CIFAR-10 (Torralba et al., 2008) and ImageNet (Russakovsky et al., 2015). We also conduct experiments on a DeeplabV3 (Chen et al., 2017) with a ResNet50 backbone trained on the Pascal VOC 2011 segmentation data set (Everingham et al., 2015). In the following section we outline the experimental details of the experiments on which we base our observations. All networks were trained and evaluated on a compute cluster with NVIDIA RTX 2080Ti and NVIDIA Titan RTX, and the experiments were implemented in PyTorch (Paszke et al., 2017). For each trained network, we summarize the hyperparameters and the resulting prune results on the nominal test data.

Table B.1: We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on CIFAR-10. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs. During retraining we used the same hyperparameters. $\{30, \ldots\}$ denotes that the learning rate is decayed every 30 epochs.

		VGG16	Resnet20/56/110	DenseNet22	WRN-16-8
Train	test error	7.19	8.6/7.19/6.43	10.10	4.81
	loss	cross-entropy	cross-entropy	cross-entropy	cross-entropy
	optimizer	SGD	SGD	SGD	SGD
	epochs	300	182	300	200
	warm-up	5	5	5	5
	batch size	256	128	64	128
	LR	0.05	0.1	0.1	0.1
	LR decay	0.5@{30, ...}	0.1@{91, 136}	0.1@{150, 225}	0.2@{60, ...}
	momentum	0.9	0.9	0.9	0.9
	Nesterov	✗	✗	✓	✓
	weight decay	5.0e-4	1.0e-4	1.0e-4	5.0e-4
Prune	γ	1.0e-16	1.0e-16	1.0e-16	1.0e-16
	α	0.85	0.85	0.85	0.85

B.1.1 Experimental Setup for CIFAR-10

All hyperparameters for training, retraining, and pruning are outlined in Table B.1. For training CIFAR-10 networks we used the training hyperparameters outlined in the respective original papers, i.e., as described by He et al. (2016), Simonyan and Zisserman (2014), Huang et al. (2017), and Zagoruyko and Komodakis (2016) for ResNets, VGGs, DenseNets, and WideResNets, respectively. For retraining, we did not change the hyperparameters and repurposed the training hyperparameters following the approaches of Liebenwein et al. (2020); Renda et al. (2020). We added a warmup period in the beginning where we linearly scale up the learning rate from 0 to the nominal learning rate. Iterative pruning is conducted by repeatedly removing the same ratio of parameters (denoted by α in Table B.1). The prune parameter γ describes the failure probability of the (provable) randomized pruning algorithms SiPP and PFP. We refer the reader to the respective papers for more details, see the papers by Baykal et al. (2021b) and Liebenwein et al. (2020) for SiPP and PFP, respectively.



(a) Resnet20 (b) Resnet56 (c) Resnet110

(d) VGG16 (e) DenseNet22 (f) WRN16-8

Figure B-1: The difference in test accuracy to the uncompressed network for the generated pruned models trained on CIFAR-10 for the evaluated pruning schemes for various target prune ratios.

B.1.2 Pruning Performance on CIFAR-10

Below we provide the results regarding the achievable test accuracy of pruned networks across multiple target prune ratios. Figure B-1 indices the results for various networks trained on CIFAR-10 using an iterative schedule to prune them. In Table B.2, we indicate the maximal prune ratio (PR) and the maximal ratio of reduced flops (FR) for which the network achieves commensurate accuracy (within 0.5% of the original accuracy). We note that the performance of our pruned networks is competitive with state-of-the-art pruning results (Baykal et al., 2021b; Han et al., 2015a; Liebenwein et al., 2020; Renda et al., 2020). For ResNet20 for example, we are able to prune the network to 85% sparsity while maintaining the original test error (-0.02% test error), see Table B.2.

B.1.3 Experimental Setup on ImageNet

The hyperparameters for the ImageNet pruning experiments are summarized in Table B.3. We consider pruned convolutional neural networks derived from Resnet18

Table B.2: Overview of the pruning performance of each algorithm for various CNN architectures evaluated on the CIFAR data set. For each algorithm and network architecture, the table reports the prune ratio (PR, %) and the ratio of flop reduction (FR, %) of pruned models when achieving test accuracy within $\delta = 0.5\%$ of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). The top values for the error and either PR (for weight-based) or FR (for filter-based algorithms) are bolded, respectively.

Model	Orig.	WT			SiPP			FT			PFP		
	Err.	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR
Resnet20	8.60	-0.02	**84.92**	81.04	-0.08	**84.92**	78.78	-0.33	52.06	24.98	-0.10	44.9	**31.27**
Resnet56	7.19	-0.53	92.91	94.31	-0.30	**93.30**	93.90	-0.38	82.71	65.50	-0.11	84.31	**73.65**
Resnet110	6.73	-0.10	**95.76**	96.73	-0.42	95.36	95.88	-0.25	86.71	72.07	-0.25	90.27	**82.42**
VGG16	7.19	-1.01	97.87	91.24	-0.82	**98.00**	88.45	-0.38	61.39	56.17	-0.15	90.30	**72.03**
DenseNet22	10.10	-0.09	71.38	76.81	-0.20	**73.16**	76.60	+0.21	43.55	42.95	-0.04	46.18	**51.86**
WRN16-8	4.81	-0.18	95.22	92.89	-0.21	**95.30**	92.03	+0.13	76.76	71.03	-0.08	78.79	**74.51**

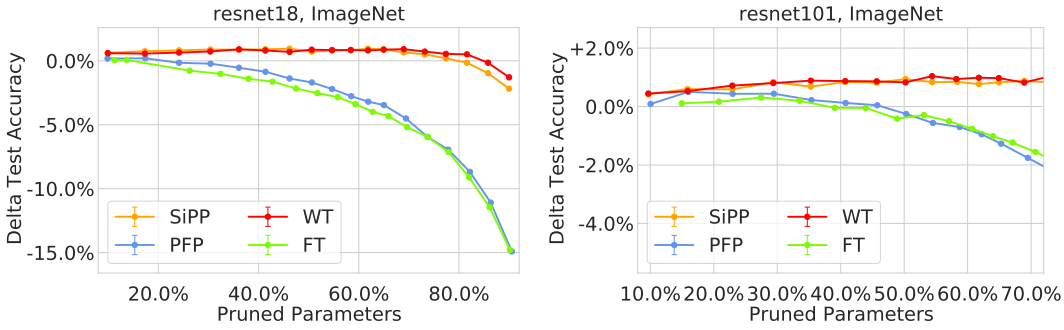
Table B.3: We report the hyperparameters used during training, pruning, and retraining for various convolutional architectures on ImageNet. LR hereby denotes the learning rate and LR decay denotes the learning rate decay that we deploy after a certain number of epochs.

		ResNet18/101
Train	top-1 test error	30.26/22.63
	top-5 test error	10.93/6.45
	loss	cross-entropy
	optimizer	SGD
	epochs	90
	warm-up	5
	batch size	256
	LR	0.1
	LR decay	0.1@{30, 60, 80}
	momentum	0.9
	Nesterov	✗
	weight decay	1.0e-4
Prune	γ	1.0e-16
	α	0.90

and Resnet101. As in the case of the CIFAR-10 experiments, we re-purpose the training schedule from the original ResNet paper (He et al., 2016) for both training and retraining. For multi-gpu training we use the linear scaling rule of (Goyal et al., 2017) to scale up the learning rate and we use learning rate warm, where we linearly scale up the learning rate from 0 to the nominal learning rate.

B.1.4 Pruning Performance on ImageNet

The results of our pruning experiments are summarized in Figure B-2 and Table B.4. Given the computationally expensive nature of ImageNet experiments, we stopped the experiments once the pruned network did not achieve commensurate accuracy anymore (instead of going to extreme prune ratios where the performance decays further). Specifically, we show the achievable test accuracy on nominal ImageNet data for various target prune ratios in Figure B-2. In Table B.4 we additionally report the maximal prune ratio (PR) and ratio of removed flops (FR) for which the pruned network achieves commensurate accuracy (i.e. within 0.5% of the unpruned network's accuracy). Just as in the case of CIFAR-10, our results are competitive with those reported in state-of-the-art papers (Han et al., 2015a; Liebenwein et al., 2020; Renda et al., 2020).



(a) Resnet18 (b) Resnet101

Figure B-2: The accuracy of the generated pruned models trained on ImageNet for the evaluated pruning schemes for various target prune ratios.

Table B.4: Overview of the pruning performance of each algorithm for various CNN architectures trained and evaluated on the ImageNet data set. For each algorithm and network architecture, the table reports the prune ratio (PR, %) and the ratio of flop reduction (FR, %) of pruned models when achieving test accuracy within $\delta = 0.5\%$ of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). The top values for the error and either PR (for weight-based) or FR (for filter-based algorithms) are bolded, respectively.

Model	Orig.	WT			SiPP			FT			PFP		
	Err.	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR
ResNet18	30.24	+0.15	**85.79**	77.14	+0.15	81.58	78.36	-0.07	13.69	10.52	+0.22	30.42	**15.74**
ResNet101	22.63	-0.71	**81.56**	83.31	-0.59	**81.56**	81.85	+0.29	53.11	**53.28**	+0.26	50.33	44.64

B.1.5 Experimental Setup for Pascal VOC

In addition to CIFAR and ImageNet, we also consider the segmentation task from Pascal VOC 2011 (Everingham et al., 2015). We augment the nominal data training data using the extra labels as provided by Hariharan et al. (2011). As network architecture we consider a DeeplabV3 (Chen et al., 2017) with ResNet50 backbone pre-trained on ImageNet. During training we use the following data augmentation pipeline: (1) randomly resize (256x256 to 1024x1024) and crop to 513x513; (2) random horizontal flip; (3) channel-wise normalization. During inference, we resize to 513x513 exactly before the normalization (3) is applied. We report both intersection-over-union (IoU) and Top1 test error for each of the pruned and unpruned networks. The experimental hyperparameters are summarized in Table B.5.

B.1.6 Pruning Performance on VOC

The results of our pruning experiments are summarized in Figure B-3 and Table B.6. Specifically, we show the achievable test accuracy on nominal VOC data for various target prune ratios in Figure B-3. In Table B.6 we report the maximal prune ratio (PR) and ratio of removed flops (FR) for which the pruned network achieves commensurate accuracy (i.e. within 0.5% of the unpruned network's accuracy).

Table B.5: We report the hyperparameters used during training, pruning, and retraining for various architectures on Pascal VOC 2011. LR hereby denotes the learning rate and LR decay denotes the learning rate decay. Note that the learning rate is polynomially decayed after each step.

		DeeplabV3-ResNet50
Train	IoU test error (%)	34.78
	top-1 test error (%)	7.94
	Loss	cross-entropy
	Optimizer	SGD
	Epochs	45
	Warm-up	0
	Batch size	32
	LR	0.02
	LR decay	(1 - "step"/"total steps")$^{0.9}$
	Momentum	0.9
	Nesterov	✗
	Weight decay	1.0e-4
Prune	γ	1.0e-16
	α	0.80

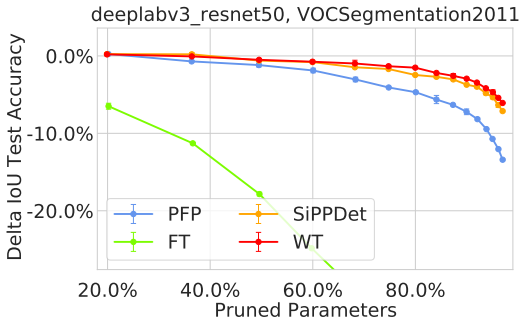


Figure B-3: The accuracy of the generated pruned models trained on VOC for the evaluated pruning schemes and various target prune ratios for a DeeplabV3-ResNet50 architecture.

B.2 Additional Results for Function Distance

In the following, we provide additional empirical evidence for the results presented in Section 7.3. We consider additional CIFAR networks for comparing informative input features and comparing matching predictions when injecting random noise as described in Section 7.3.

Table B.6: Overview of the pruning performance of each algorithm for DeeplabV3 trained and evaluated on Pascal VOC segmentation data. For each algorithm, the table reports the prune ratio (PR, %) and the ratio of flop reduction (FR, %) of pruned models when achieving IoU test accuracy within $\delta = 0.5\%$ of the original network's test accuracy (or the closest result when the desired test accuracy was not achieved for the range of tested PRs). The top values for the error and either PR (for weight-based) or FR (for filter-based algorithms) are bolded, respectively.

Model	Orig.	WT			SiPP			FT			PFP		
	Err.	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR	Err.	PR	FR
DeeplabV3	34.78	+0.47	**58.87**	58.65	+0.29	42.98	42.70	+0.00	0.00	0.00	-0.25	20.16	**19.14**

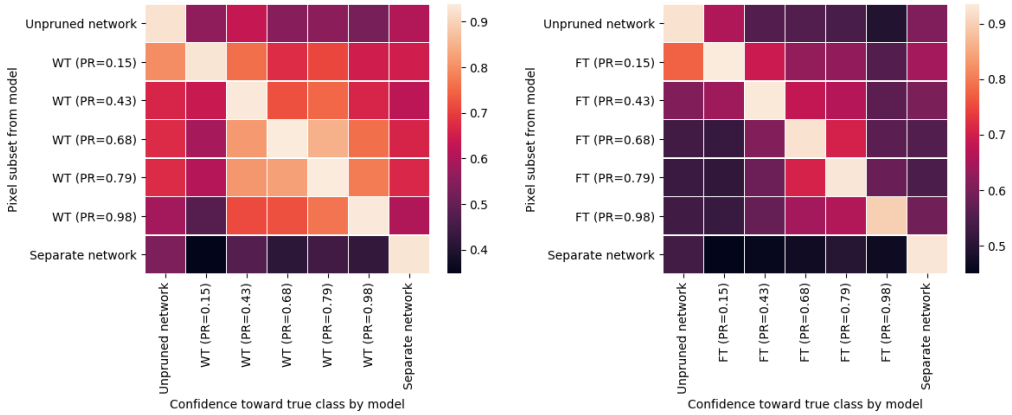
B.2.1 Comparison of Informative Features

Informative Features Based on Nominal Test Data

Figure B-4 shows results on comparison of informative features for pruned models on VGG16. The informative features were computed from a random subset of CIFAR-10 test data.

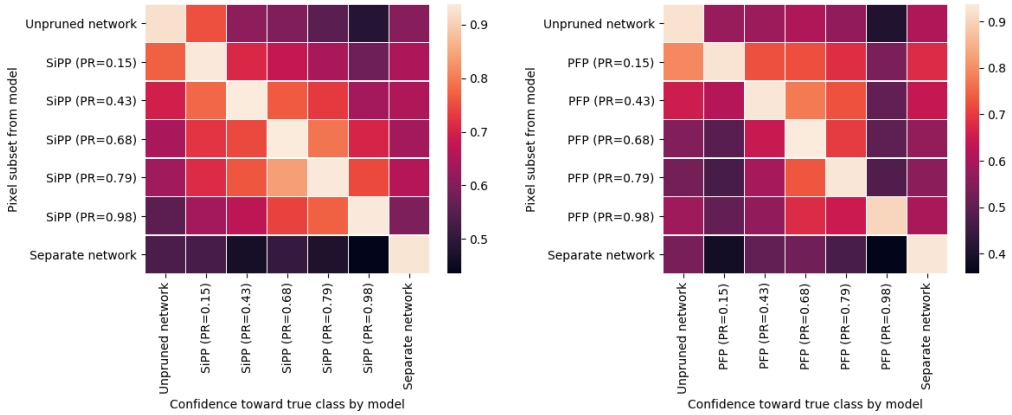
Informative Features Based on Out-of-distribution Test Data

We repeat the experiment with the informative features being computed from a random subset of CIFAR-10-C test data (any corruption). Figure B-5 includes results on comparison of informative features, c.f. Section 7.3.2, for models pruned by WT and FT on ResNet20. Figure B-6 shows results for VGG16. We note that even when tested with out-of-distribution test data we observe similar trends, i.e., pruned networks in general are more similar in the functional sense to their parent network than a separately trained, unpruned network.



(a) VGG16, Pruning by WT

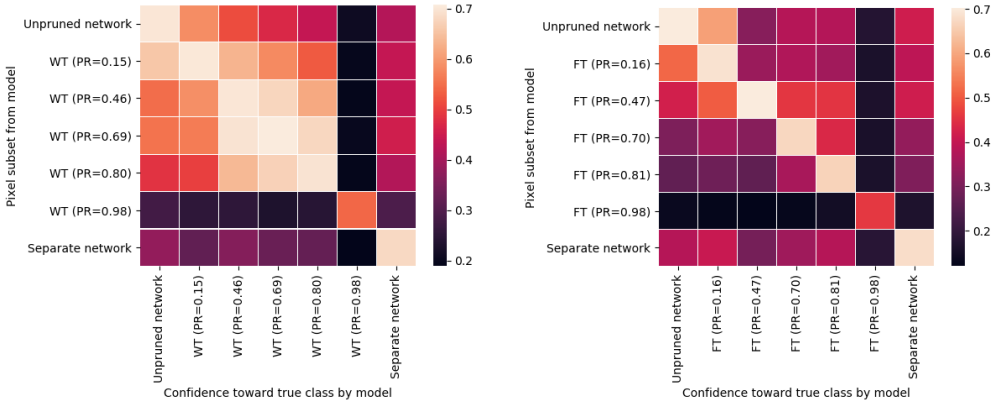
(b) VGG16, Pruning by FT



(c) VGG16, Pruning by SiPP

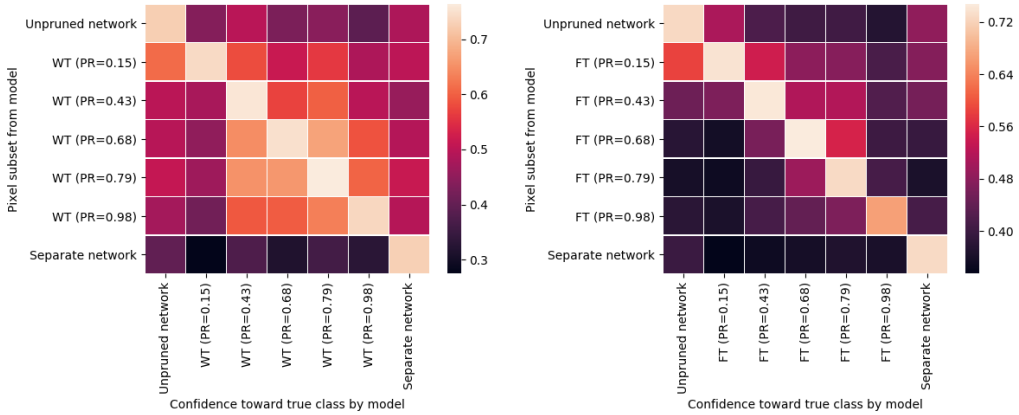
(d) VGG16, Pruning by PFP

Figure B-4: Heatmap of confidences on informative pixels from pruned VGG16 models. Y-axis is the model used to generate 10% pixel subsets of 2000 sampled CIFAR-10 test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a) Weight Thresholding (WT), (b) Filter Thresholding (FT), (c) SiPP, (d) Provable Filter Pruning (PFP).



(a) ResNet20, Pruning by WT (b) ResNet20, Pruning by FT

Figure B-5: Heatmap of confidences on informative pixels from pruned ResNet20 models. Y-axis is the model used to generate 10% pixel subsets of 2000 randomly sampled CIFAR-10-C corrupted test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a) Weight Thresholding (WT), (b) Filter Thresholding (FT).



(a) VGG16, Pruning by WT (b) VGG16, Pruning by FT

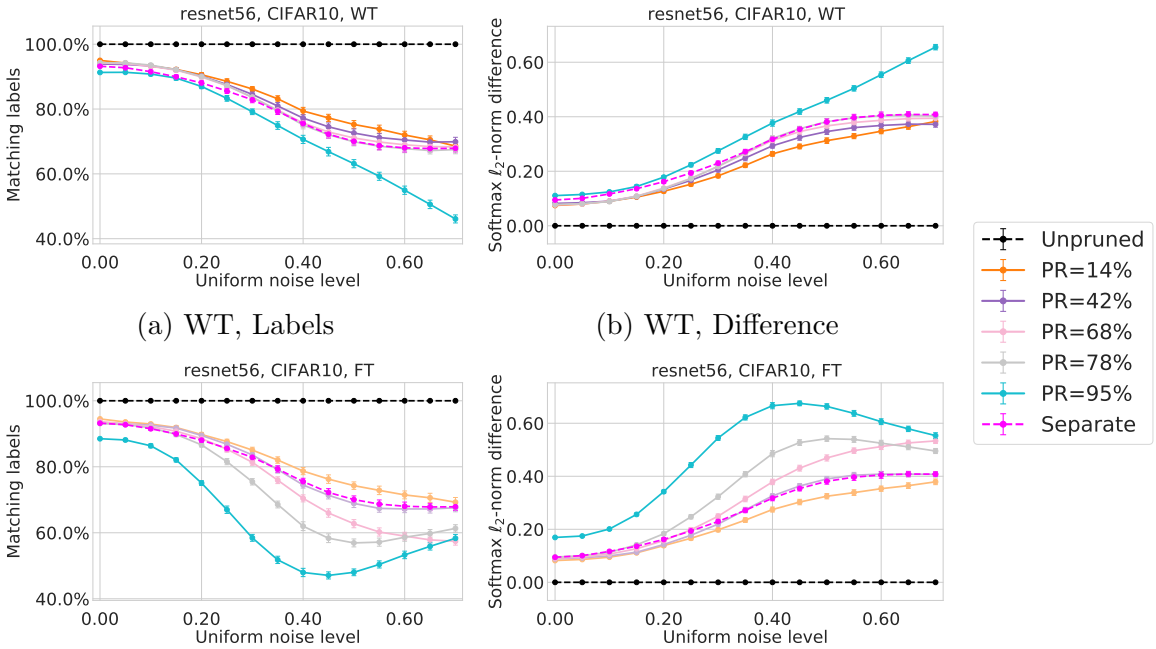
Figure B-6: Heatmap of confidences on informative pixels from pruned VGG16 models. Y-axis is the model used to generate 10% pixel subsets of 2000 randomly sampled CIFAR-10-C corrupted test images, x-axis describes the models evaluated with each 10% pixel subset, cells indicate mean confidence towards true class of the model from the x-axis on tested data from y-axis. Pruning by (a) Weight Thresholding (WT), (b) Filter Thresholding (FT).

B.2.2 Noise Similarities

We consider noise properties of networks when feeding perturbed data into the network. In particular, we are interested in understanding the similarities of the output of a pruned network and its unpruned parent as described in Section 7.3 of the main paper. To this end we consider two metrics: (i) percentage of matching predictions (labels) of pruned networks w.r.t. their unpruned parent for various target prune ratios and (ii) the norm-based difference between pruned networks and their unpruned parent. Each result also includes comparisons to a separately trained network of the same architecture with a different random initialization to highlight the functional similarities between unpruned and pruned networks. Overall, we find that pruned networks functionally approximate their pruned parent more closely than a separately trained network. We provide additional empirical evidence for this conclusion below.

Results for WT and FT on Additional Networks

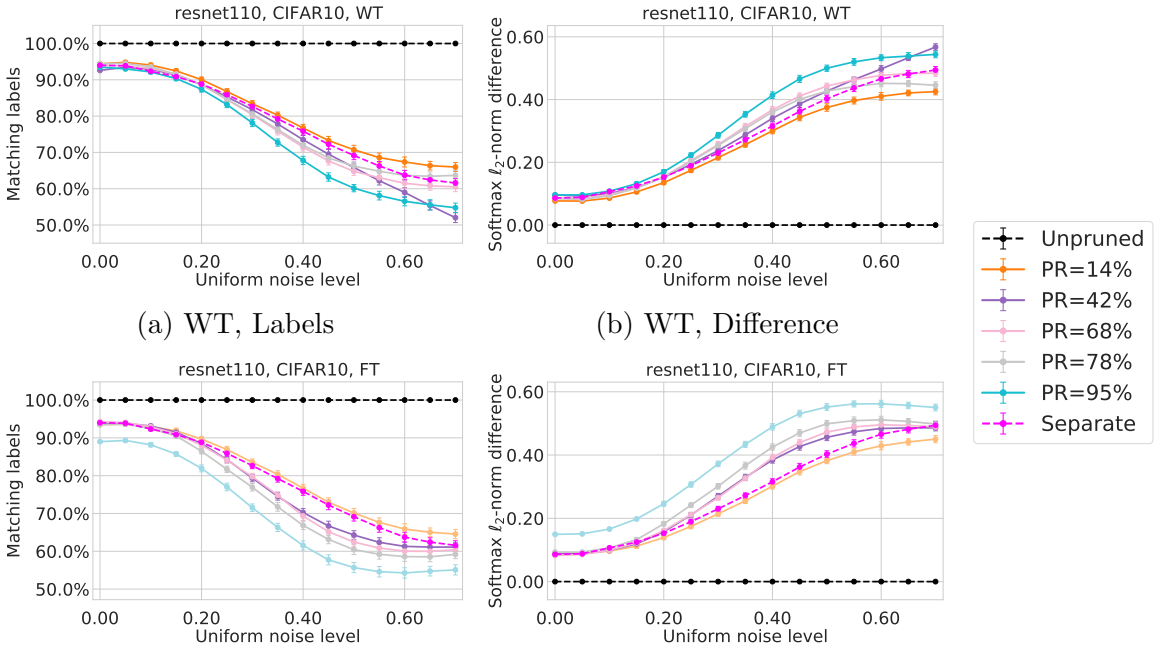
We consider the functional similarities between pruned networks and their unpruned parent for the additional neural network architectures ResNet56, ResNet110, VGG16, DenseNet22, and WideResNet16-8 trained on CIFAR-10 as shown in Figures B-7, B-8, B-9, B-10, and B-11, respectively. All networks shown here were retrained using the same iterative prune schedule. As apparent from the respective figures, the functional similarities are consistent across architectures for the same pruning strategies.



(a) WT, Labels (b) WT, Difference

(c) FT, Labels (d) FT, Difference

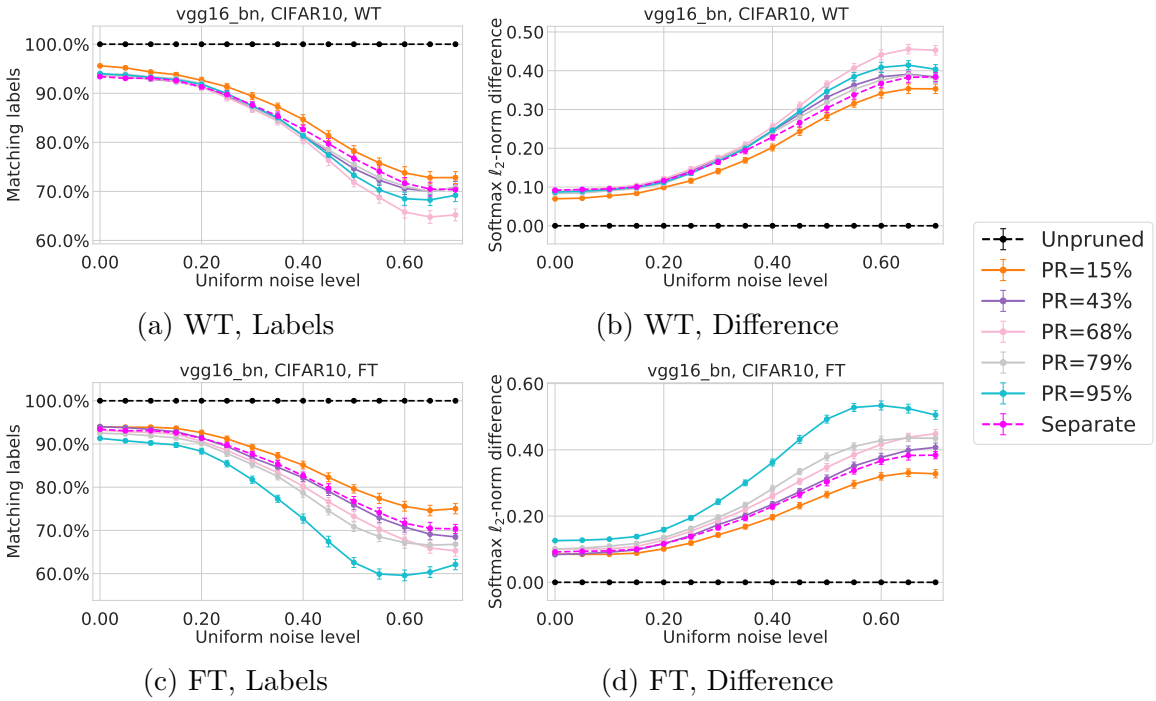
Figure B-7: The functional similarities between pruned ResNet56 models and their unpruned parent.



(a) WT, Labels (b) WT, Difference

(c) FT, Labels (d) FT, Difference

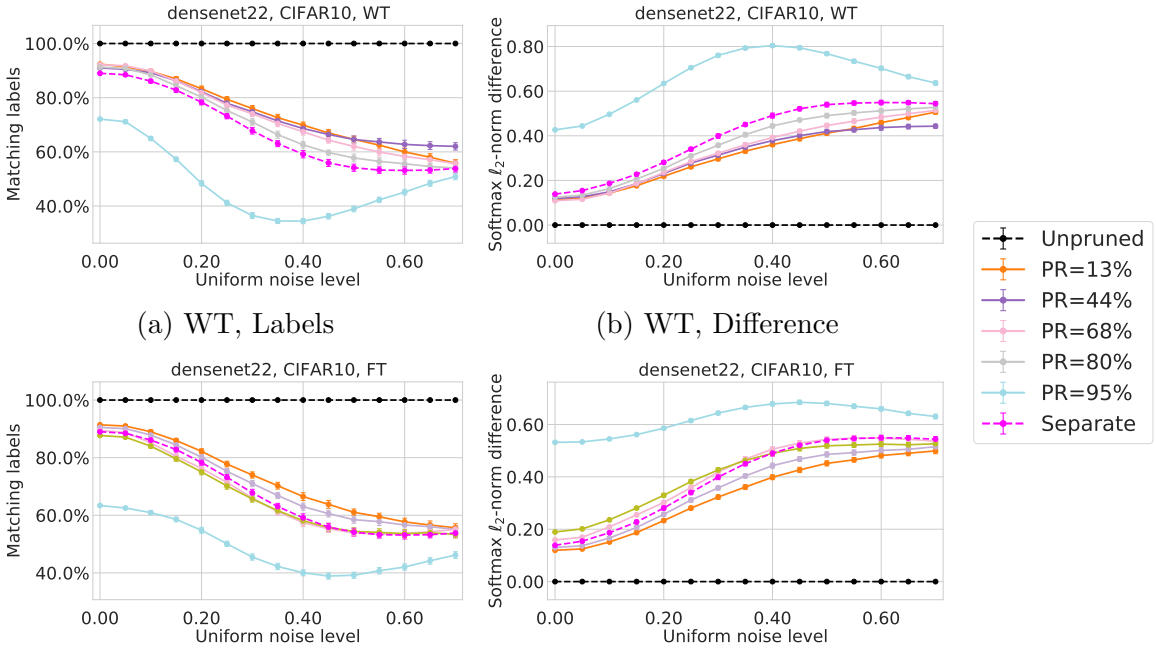
Figure B-8: The functional similarities between pruned ResNet110 models and their unpruned parent.



(a) WT, Labels (b) WT, Difference

(c) FT, Labels (d) FT, Difference

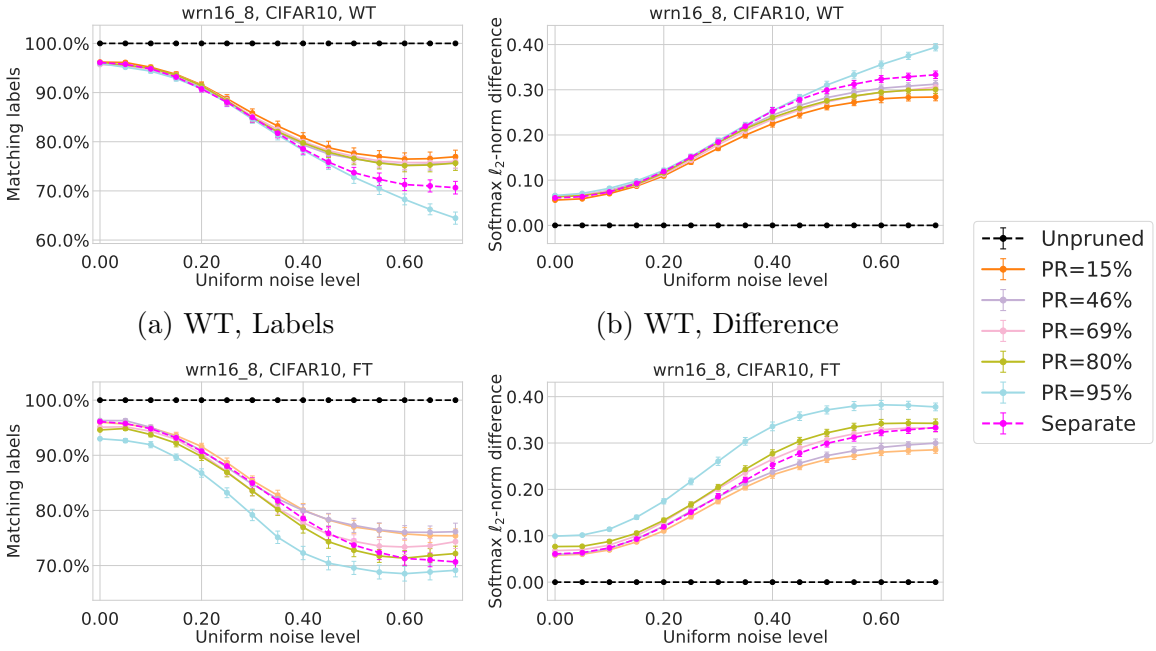
Figure B-9: The functional similarities between pruned VGG16 models and their unpruned parent.



(a) WT, Labels (b) WT, Difference

(c) FT, Labels (d) FT, Difference

Figure B-10: The functional similarities between pruned DenseNet22 models and their unpruned parent.



(a) WT, Labels

(b) WT, Difference

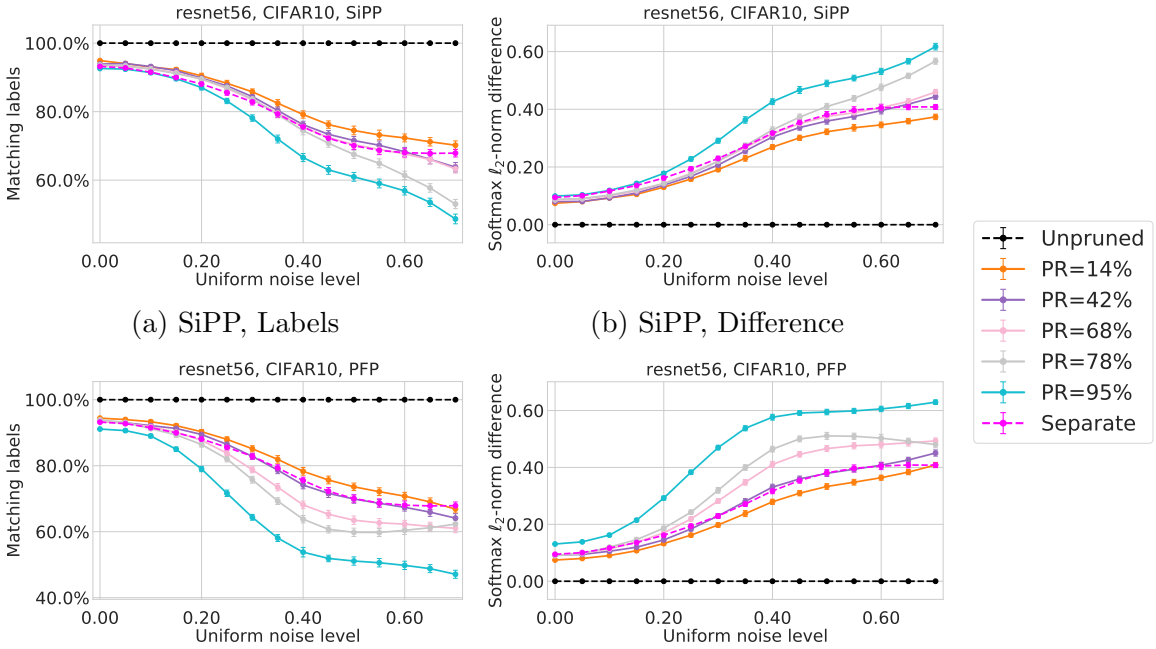
(c) FT, Labels

(d) FT, Difference

Figure B-11: The functional similarities between pruned WRN16-8 models and their unpruned parent.

Results for Additional Pruning Methods (SiPP and PFP)

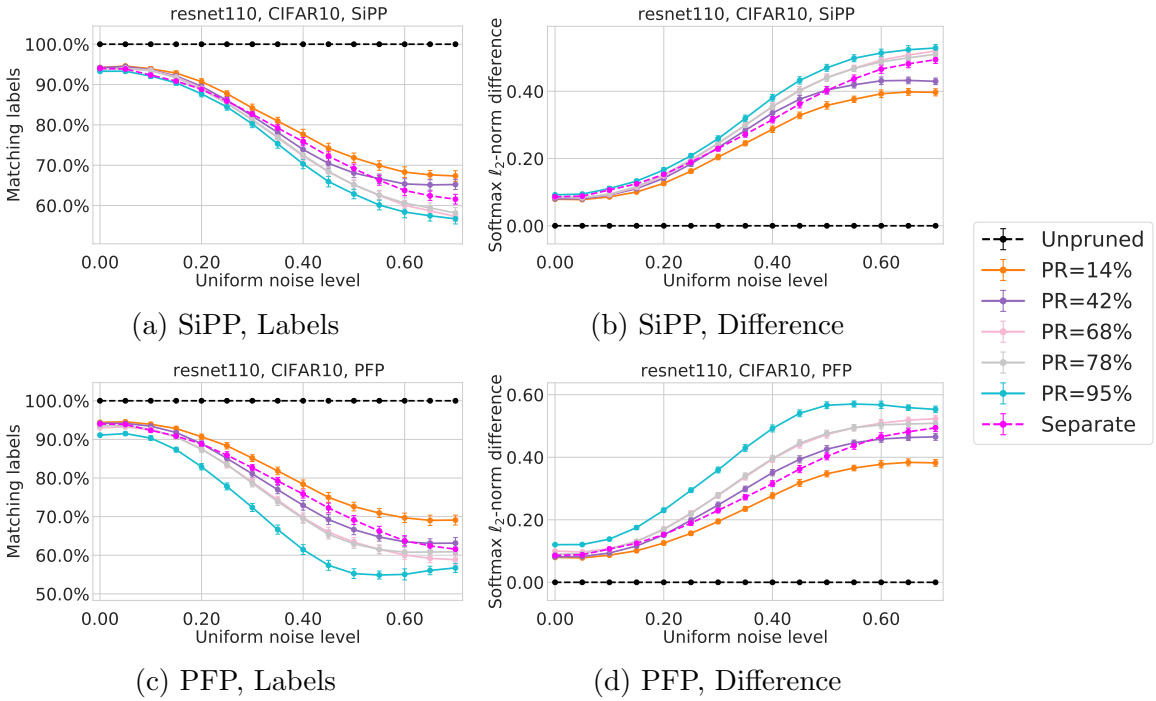
In the following we compare the functional similarities using the alternative weight and filter pruning methods SiPP (Baykal et al., 2021b) and PFP (Liebenwein et al., 2020), respectively. The methods are described in more detail in Section 7.2 of the main paper. Below we present results for ResNet56, ResNet110, VGG16, DenseNet22, and WRN16-8, see Figures B-12, B-13, B-14, B-15, and B-16 respectively. We note that the conclusions with regards to the functional similarities remain in essence unaltered for alternative pruning methods. Networks were trained with the same iterative prune-retrain schedule as in the previous subsection.



(a) SiPP, Labels (b) SiPP, Difference

(c) PFP, Labels (d) PFP, Difference

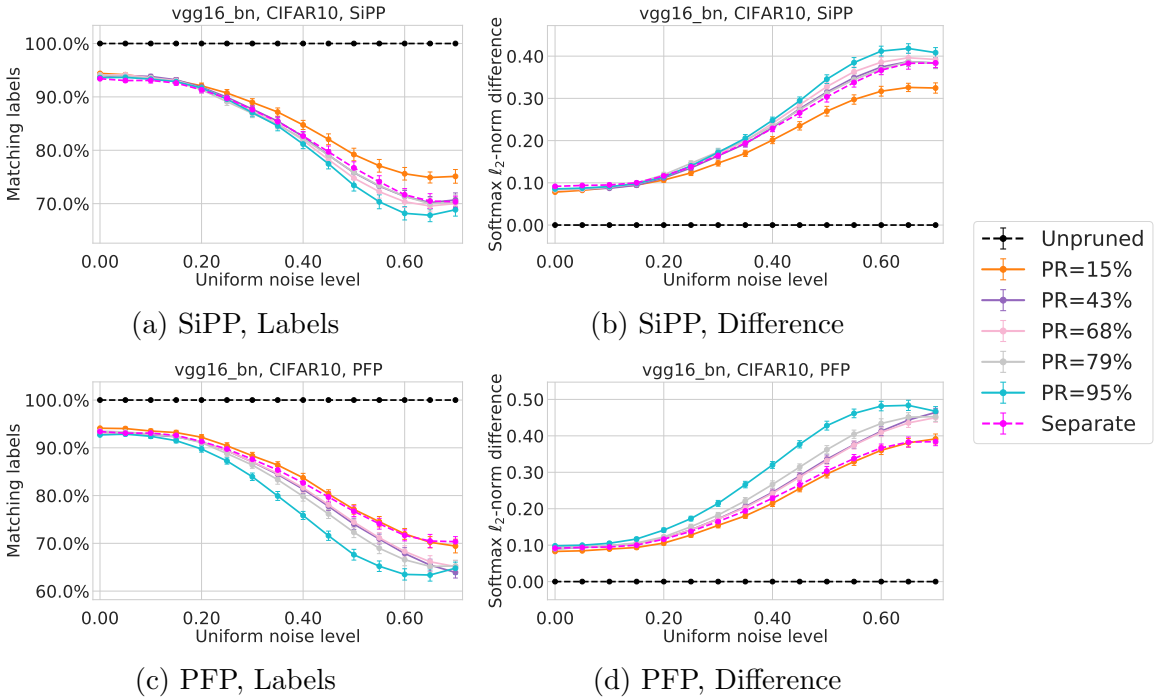
Figure B-12: The functional similarities between pruned ResNet56 models and their unpruned parent.



(a) SiPP, Labels (b) SiPP, Difference

(c) PFP, Labels (d) PFP, Difference

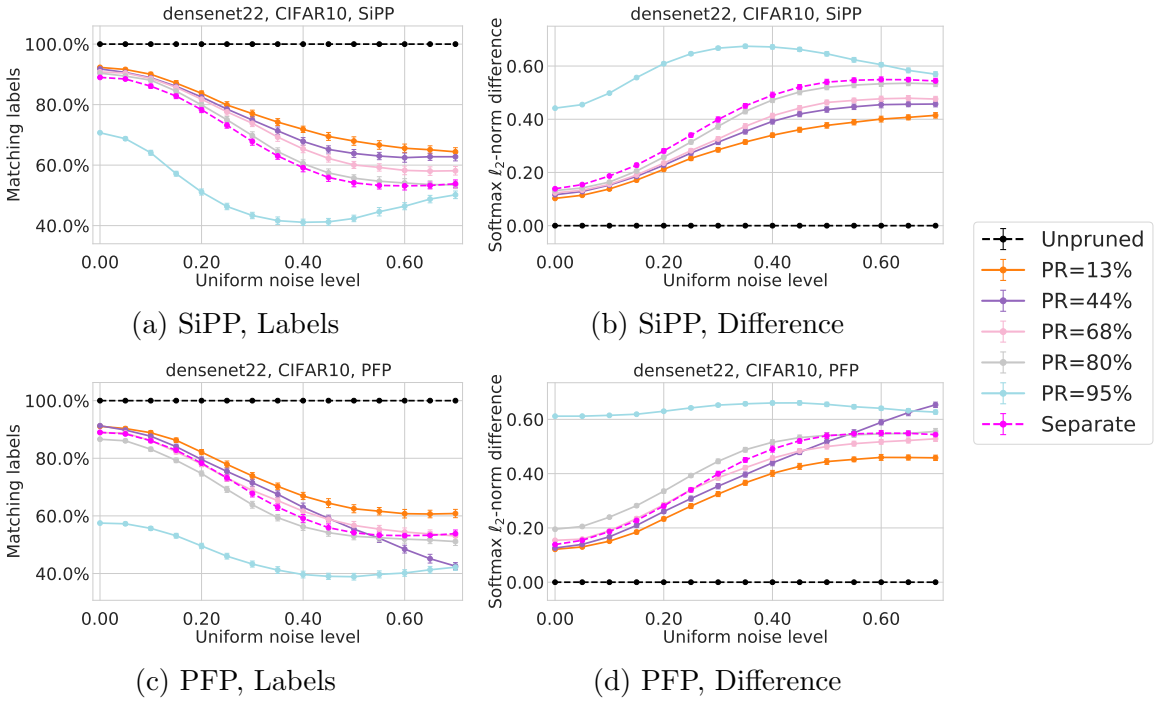
Figure B-13: The functional similarities between pruned ResNet110 models and their unpruned parent.



(a) SiPP, Labels (b) SiPP, Difference

(c) PFP, Labels (d) PFP, Difference

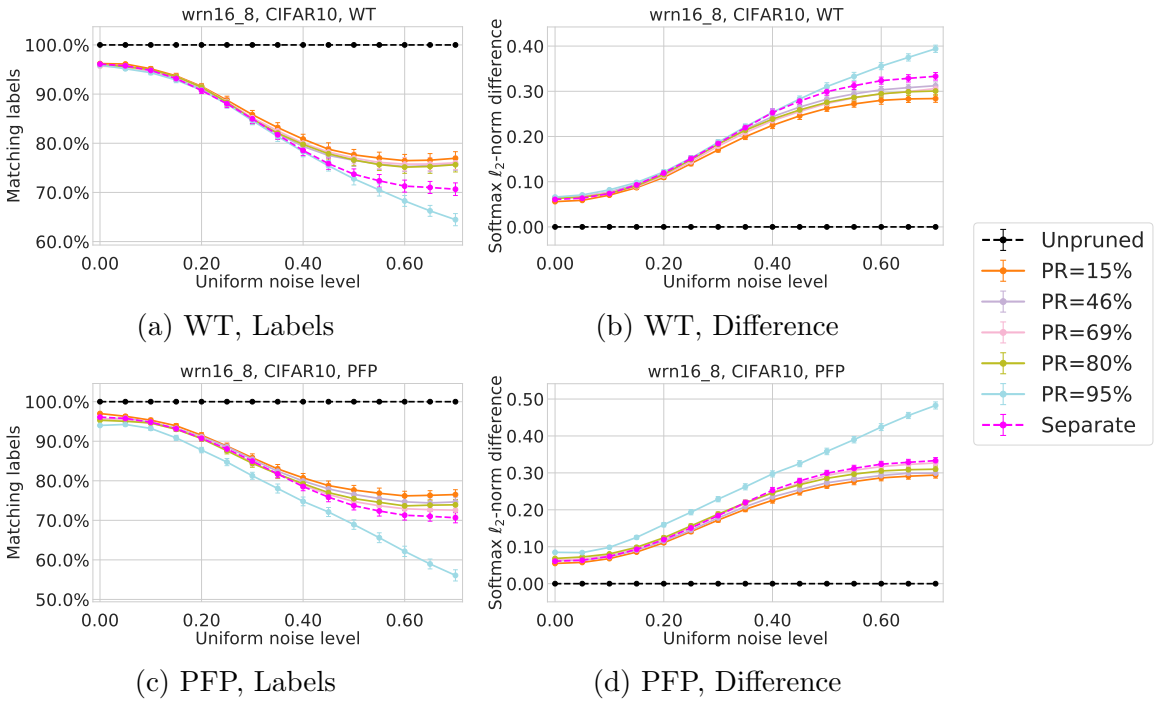
Figure B-14: The functional similarities between pruned VGG16 models and their unpruned parent.



(a) SiPP, Labels (b) SiPP, Difference

(c) PFP, Labels (d) PFP, Difference

Figure B-15: The functional similarities between pruned DenseNet22 models and their unpruned parent.



(a) WT, Labels (b) WT, Difference

(c) PFP, Labels (d) PFP, Difference

Figure B-16: The functional similarities between pruned WRN16-8 models and their unpruned parent.

B.3 Additional Results for Prune Potential

We present additional experimental evaluation for the results presented in Section 7.4. As in Section 7.4, we consider the *prune potential* for pruned networks when testing the network under noisy input and under corrupted images.

B.3.1 Detailed Results for Prune Potential with Corruptions

CIFAR-10

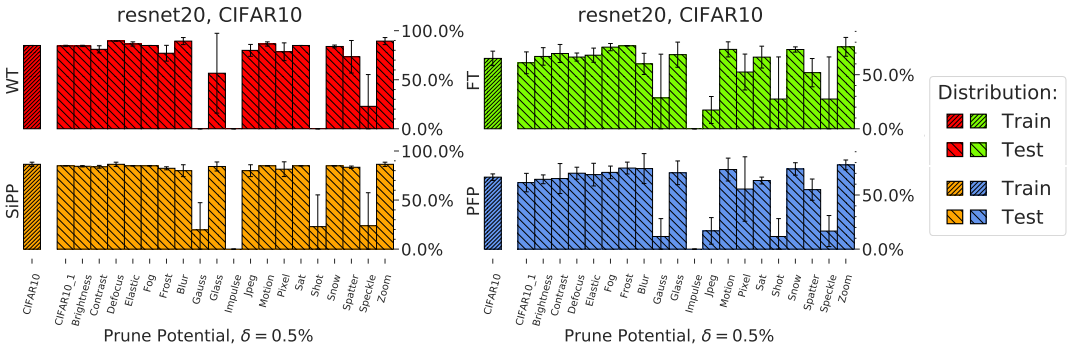
Following Section 7.4 we present additional results pertaining to the prune potential of networks when generalizing to out-of-distribution test data of various kinds. For CIFAR-10, we consider the o.o.d. data sets by Hendrycks and Dietterich (2019) (CIFAR-10-C) which are publicly available. Additionally, we also compare to CIFAR-10.1 by Recht et al. (2018), which is an alternative in-distribution test data set for CIFAR-10. For ImageNet, we consider the ImageNet versions of the CIFAR-10-C data sets, denoted by ImageNet-C (Hendrycks and Dietterich, 2019). Additionally, we compare to ObjectNet (Barbu et al., 2019), an o.o.d. data set that exhibits large variations over the context and the pose of an object instead of image corruptions. For VOC, we consider the same set of corruptions, denoted by VOC-C, based on the generalization of the CIFAR-10-C corruptions to any image data set by Michaelis et al. (2019). For CIFAR-10-C, ImageNet-C, and VOC-C, we evaluate the prune potential for severity level 3 out of 5 (Hendrycks and Dietterich, 2019; Michaelis et al., 2019). In accordance with the results presented in Section 7.4 we find that the prune potential can vary substantially depending on the task.

We consider the out-of-distribution prune potential for different CIFAR-10-C data sets (severity level 3) for the network architectures ResNet20, ResNet56, ResNet110, VGG16, DenseNet22, and WideResNet16-8, see Figures B-17, B-18, B-19, B-20, B-21, and B-22, respectively. Each network was weight-pruned and filter-pruned with WT and SiPP, and FT and PFP, respectively. We note that in general the prune potential varies across prune methods, network architectures, and task. Moreover, some of the networks seem to cope better with out-of-distribution data than other networks

(e.g. WideResNet16-8 as seen from Figure B-22). However, across all experiments the prune potential varies significantly and it seems difficult to predict clear trends highlighting the sensitivity of the prune potential w.r.t. out-of-distribution test data.

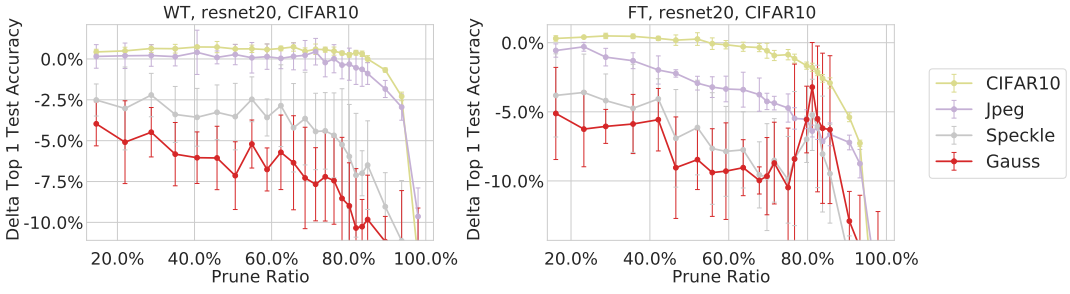
ImageNet and VOC

Finally, we consider the prune potential for out-of-distribution test data on a ResNet18 (ImageNet), ResNet101 (ImageNet), and DeeplabV3 (VOC), see Figures B-23, B-24, and B-25, respectively. We note that the prune potential in this case is equally sensitive to the test task emphasizing that our observations scale to larger networks and data sets as well instead of being confined to small-scale data sets such as CIFAR-10. Moreover, considering the expansive nature of ImageNet experiments we did not prune the network to very extreme prune ratios but instead stopped at around 80%-90%. In light of these observations, we conjecture that the nominal prune ratio is even higher, which would result in an even larger overall gap in prune potential between in-distribution and out-of-distribution test data.



(a) WT, SiPP

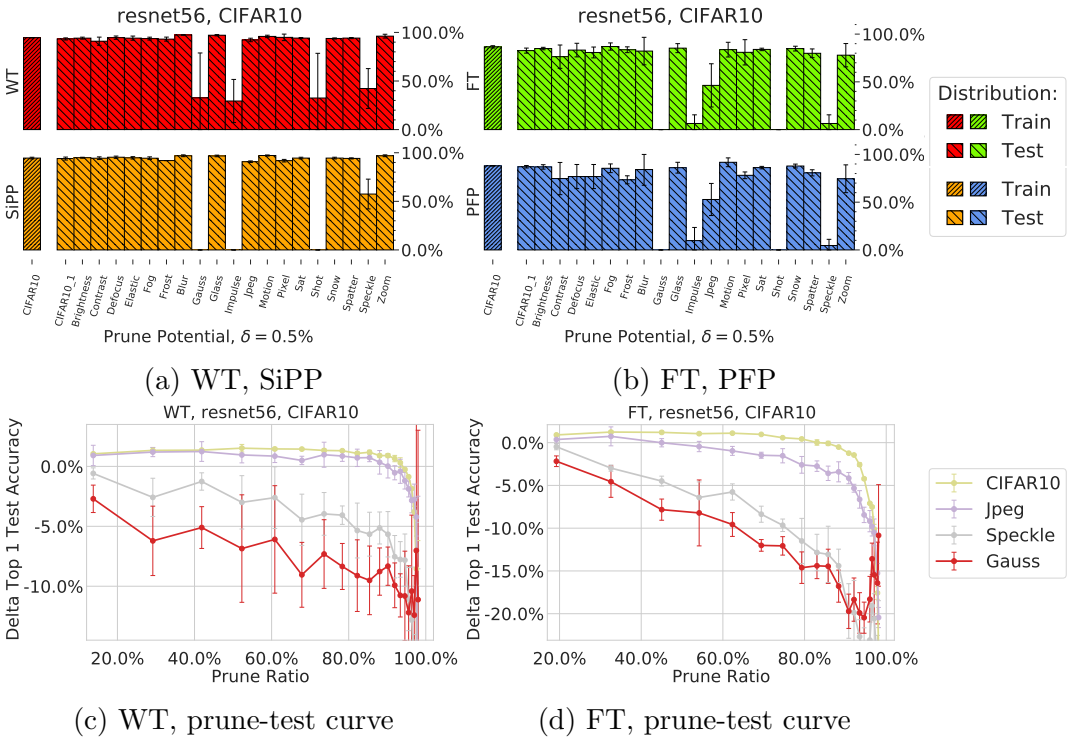
(b) FT, PFP



(c) WT, prune-test curve

(d) FT, prune-test curve

Figure B-17: The prune potential of a ResNet20 achievable for CIFAR-10 out-of-distribution data sets.



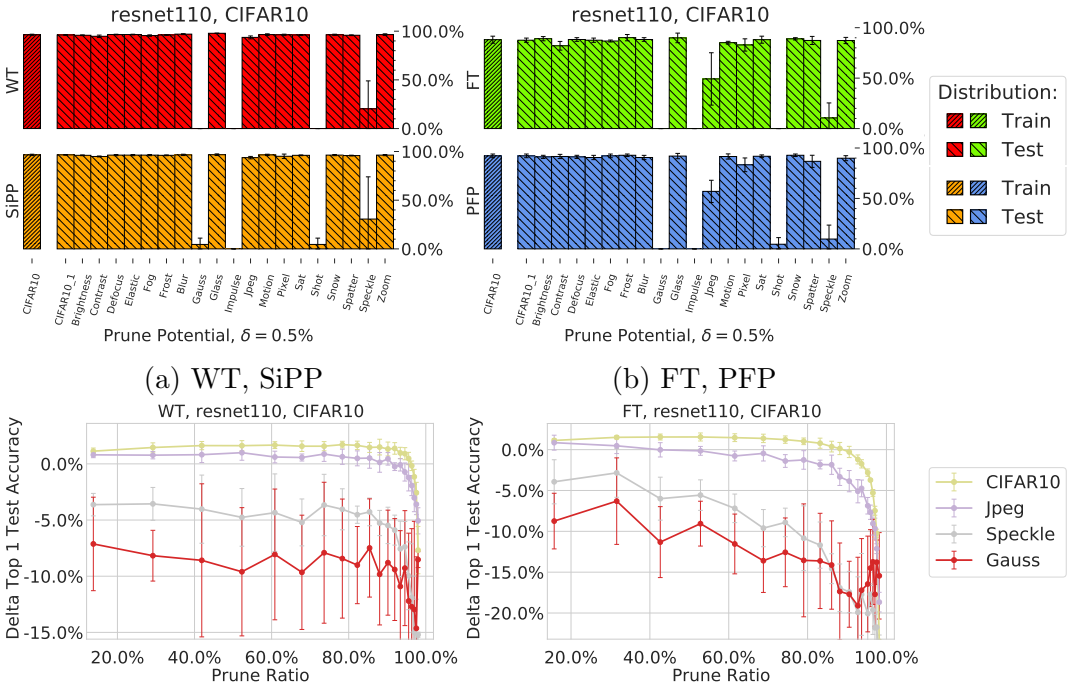
(a) WT, SiPP

(b) FT, PFP

(c) WT, prune-test curve

(d) FT, prune-test curve

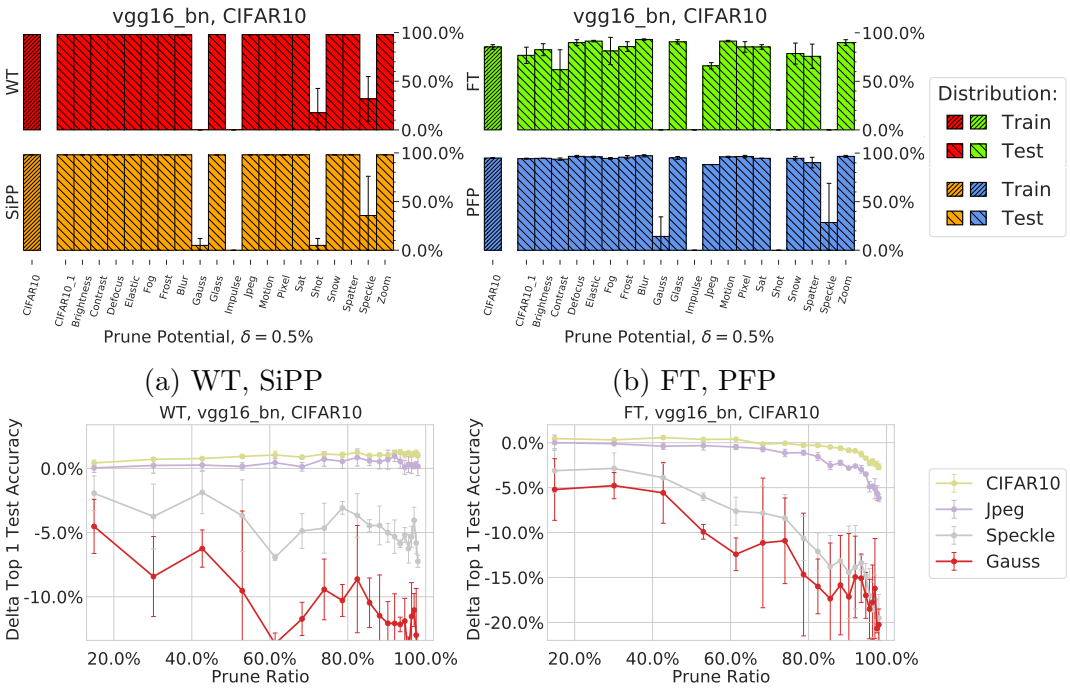
Figure B-18: The prune potential of a ResNet56 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP (b) FT, PFP

(c) WT, prune-test curve (d) FT, prune-test curve

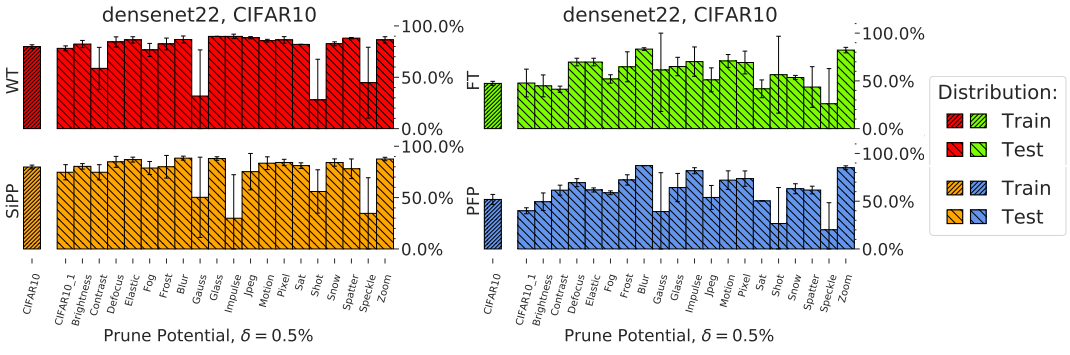
Figure B-19: The prune potential of a ResNet110 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP (b) FT, PFP

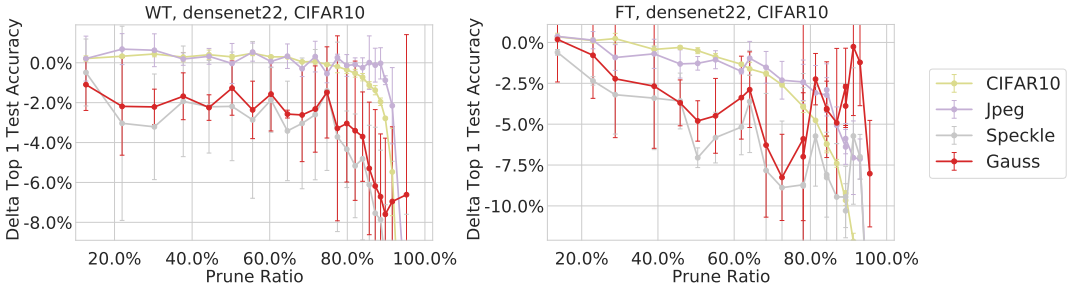
(c) WT, prune-test curve (d) FT, prune-test curve

Figure B-20: The prune potential of a VGG16 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP

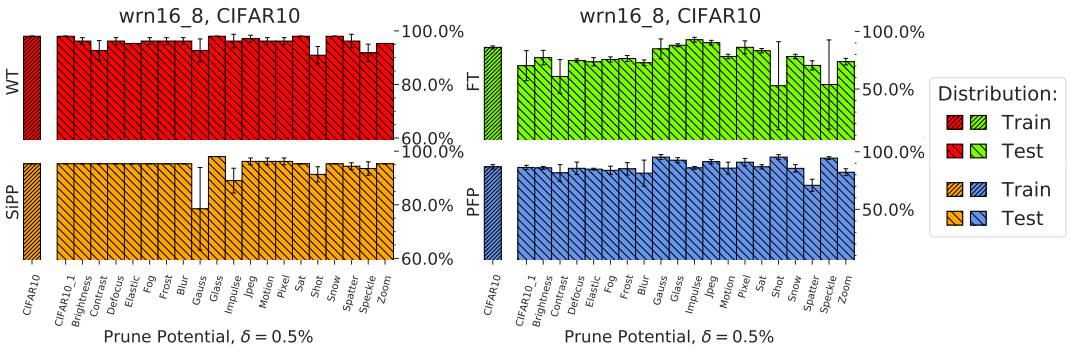
(b) FT, PFP



(c) WT, prune-test curve

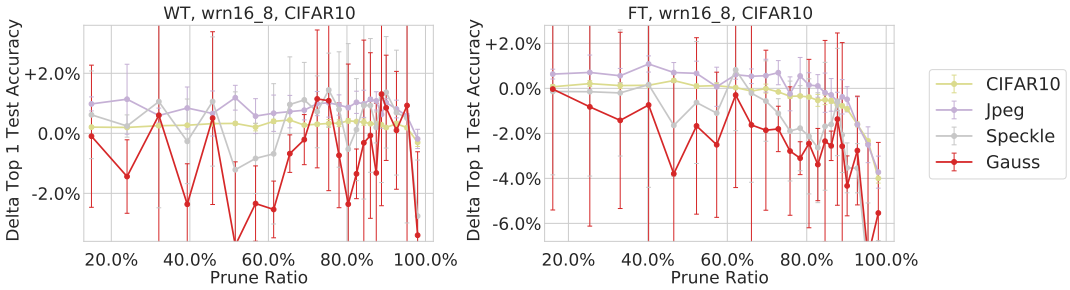
(d) FT, prune-test curve

Figure B-21: The prune potential of a DenseNet22 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP

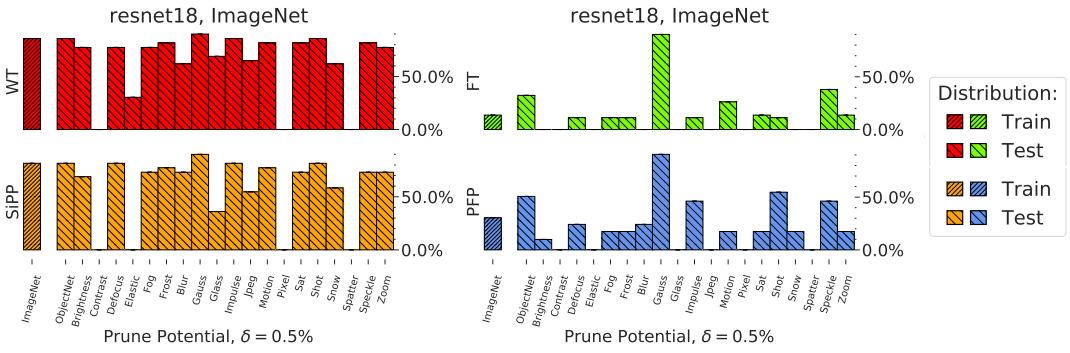
(b) FT, PFP



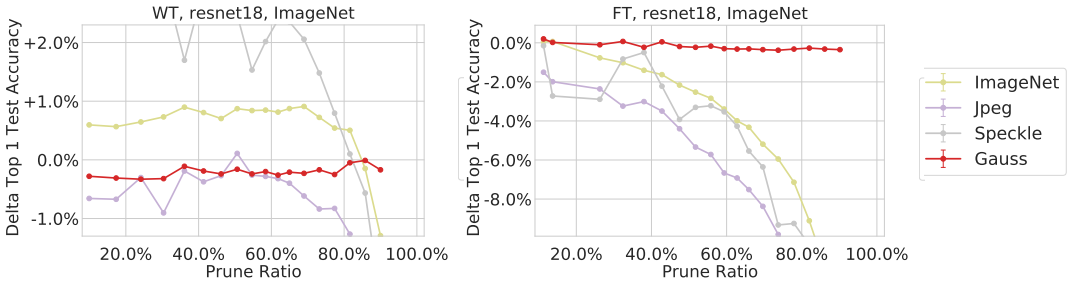
(c) WT, prune-test curve

(d) FT, prune-test curve

Figure B-22: The prune potential of a WRN16-8 achievable for CIFAR-10 out-of-distribution data sets.

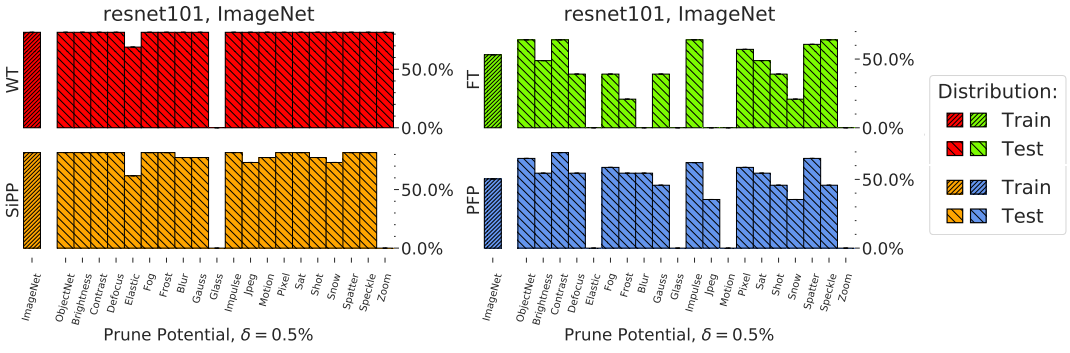


(a) WT, SiPP (b) FT, PFP

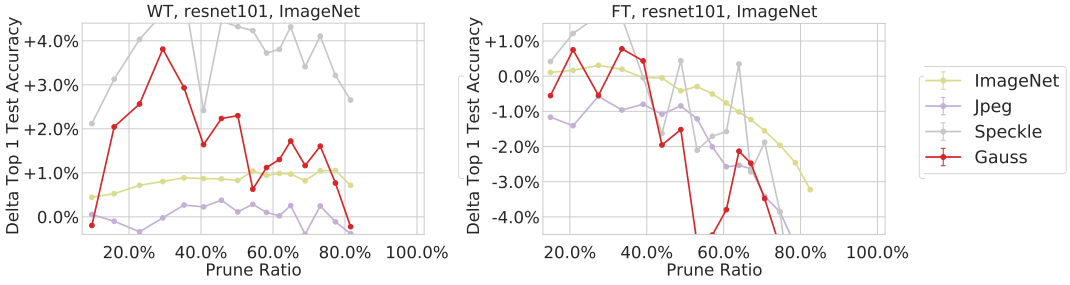


(c) WT, prune-test curve (d) FT, prune-test curve

Figure B-23: The prune potential of a ResNet18 achievable for ImageNet out-of-distribution data sets.

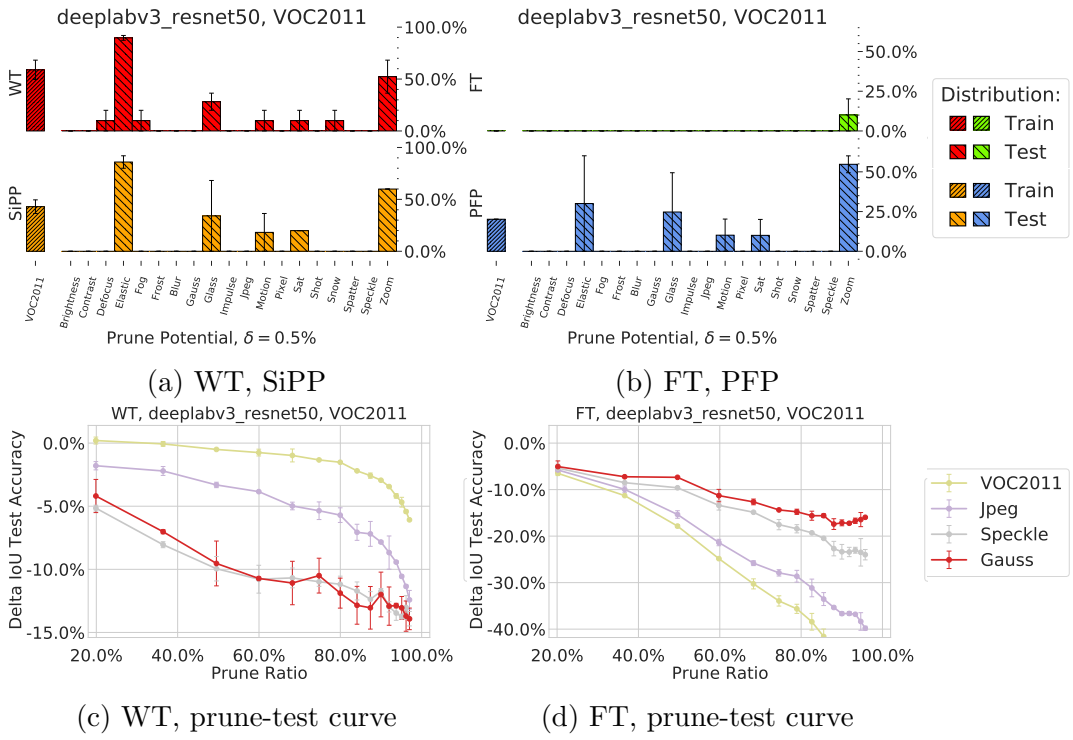


(a) WT, SiPP (b) FT, PFP



(c) WT, prune-test curve (d) FT, prune-test curve

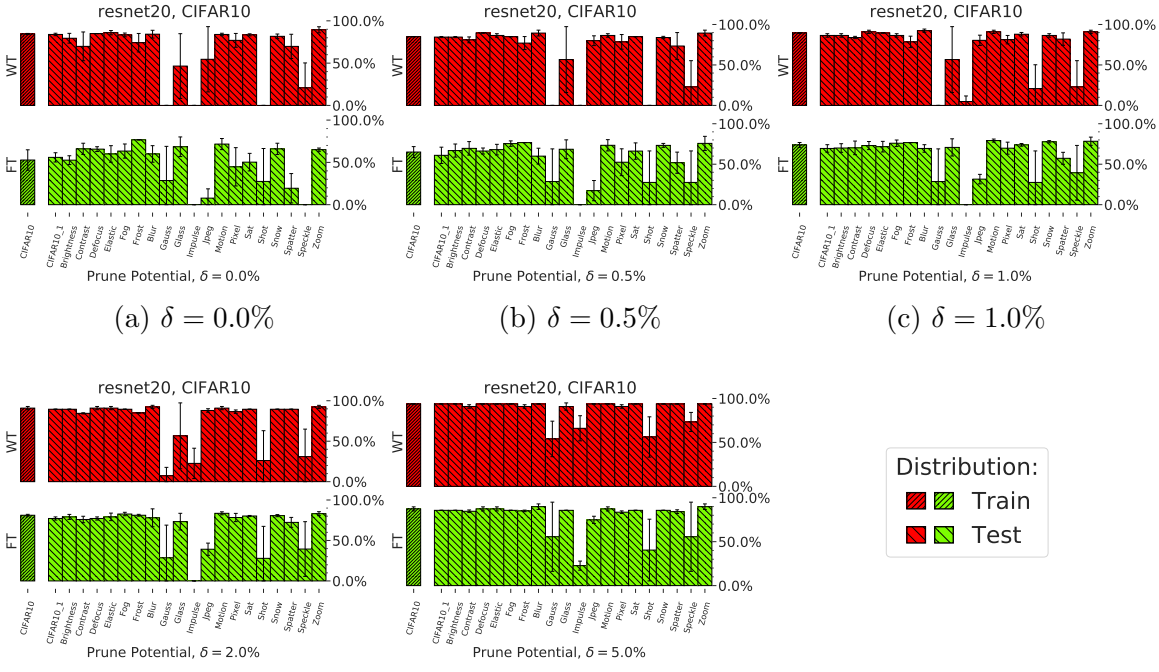
Figure B-24: The prune potential of a ResNet101 achievable for ImageNet out-of-distribution data sets.



(a) WT, SiPP (b) FT, PFP

(c) WT, prune-test curve (d) FT, prune-test curve

Figure B-25: The prune potential of a DeeplabV3 achievable for Pascal VOC out-of-distribution data sets.



(a) $\delta = 0.0\%$ (b) $\delta = 0.5\%$ (c) $\delta = 1.0\%$

(d) $\delta = 2.0\%$ (e) $\delta = 5.0\%$

Figure B-26: The prune potential of a ResNet20 trained on CIFAR-10 for WT and FT, respectively. In each figure the same experiment is repeated with different values of δ ranging from 0% to 5%.

B.3.2 Choice of Commensurate Accuracy

We evaluated the prune potential for a ResNet20 trained on CIFAR-10 for a range of possible values for δ to ensure that our observations hold independent of the specific choice of δ. Recall that δ denotes the amount of "slack" when evaluating the prune potential, i.e., the difference in test accuracy between the pruned and unpruned network for which the pruned network's performance is considered commensurate. Specifically, as shown in Figure B-26, we consider a range of δ between 0% and 5%. Naturally, the prune potential is higher overall for larger values of δ. However, we can see that our main observations essentially remain unchanged, that is the prune potential still significantly varies across different tasks and distributions. Overall, these results confirm our previous findings.

B.3.3 Detailed Results for Excess Error with Corruptions

Recall that the excess error is defined as the additional error incurred on the test distribution compared to the error on the train distribution. The difference in excess error between pruned and unpruned networks thus quantifies the *additional error* incurred by the pruned network on the test distribution *on top of* the error increase incurred by the pruned network compared to the unpruned network according to the prune-accuracy curve on the train distribution.

We used ordinary least squares (linear regression) to compute the prediction of the relationship between prune ratio and difference in excess error. The y-intercept is set to 0 since by the definition the difference in excess error is 0% for a prune ratio of 0%. The shaded regions describe the 95% confidence intervals, which were computed based on bootstrapping.

The results for the CIFAR-10 network architectures ResNet20, ResNet56, ResNet110, VGG16, DenseNet22, and WRN16-8 are shown in Figures B-27, B-28, B-29, B-30, B-31, and B-32, respectively. The results for ImageNet network architectures ResNet18 and ResNet101 are shown in Figures B-33 and B-34, respectively. The results for the VOC network architecture DeeplabV3 is shown in Figure B-35. Note that ideally the

slope would be zero indicating that the prune-accuracy curve on nominal data is predictive of o.o.d. data. However, as shown most pruned networks exhibit a significant increase in excess error that increases with higher prune ratios. These results further corroborate our understanding that networks cannot be pruned to full extent when faced with o.o.d. data.

Notable exceptions include WRN16-8 (CIFAR-10) and ResNet101 (ImageNet) with little correlation between the prune ratio and the difference in excess error indicating that those networks may be genuinely overparameterized in a robust sense confirming our findings from previous sections.

The negative delta in excess error for FT on DeeplabV3 (Figure B-35), on the other hand, is a spurious consequence of the prune potential of FT for nominal data already being zero rather than a consequence of the amount of overparameterization in the network.

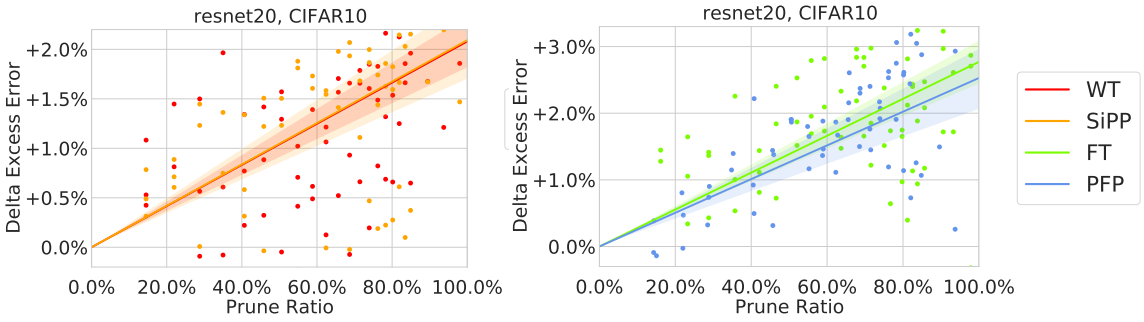


Figure B-27: The difference in excess error for a ResNet20 trained on CIFAR-10.

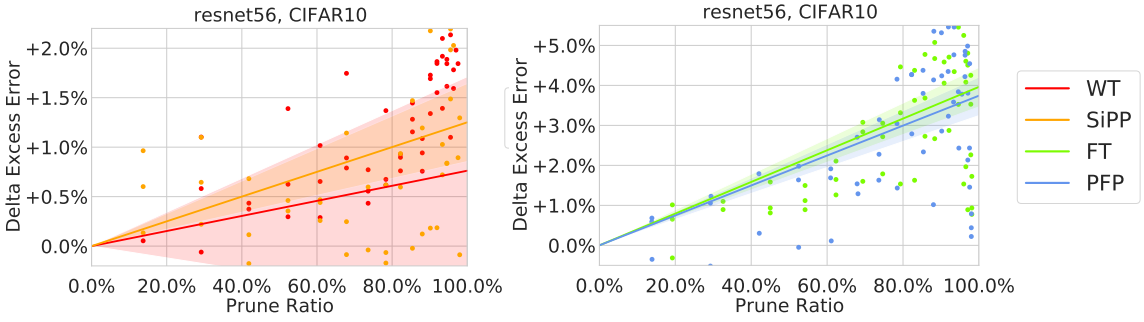


Figure B-28: The difference in excess error for a ResNet56 trained on CIFAR-10.

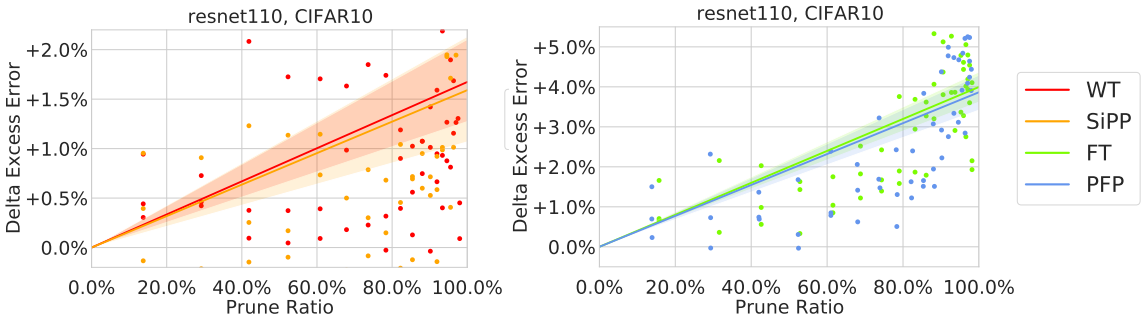


Figure B-29: The difference in excess error for a ResNet110 trained on CIFAR-10.

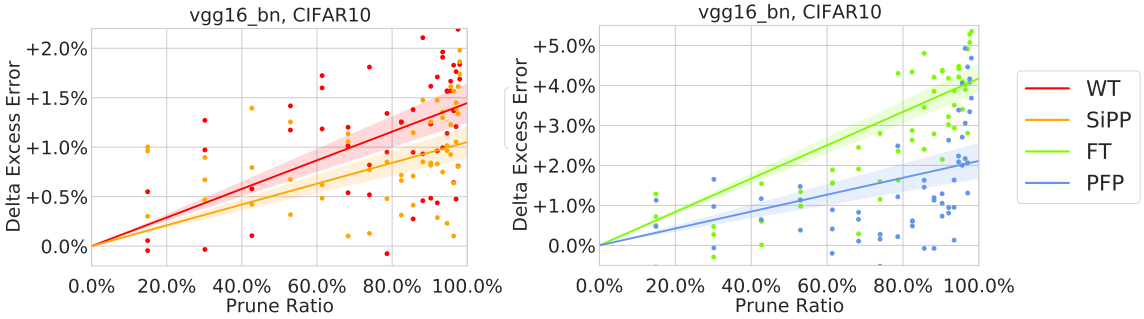


Figure B-30: The difference in excess error for a VGG16 trained on CIFAR-10.

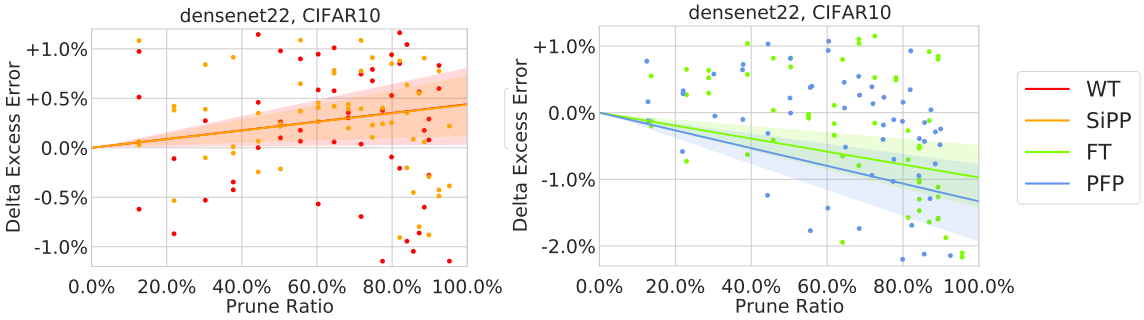


Figure B-31: The difference in excess error for a DenseNet22 trained on CIFAR-10.

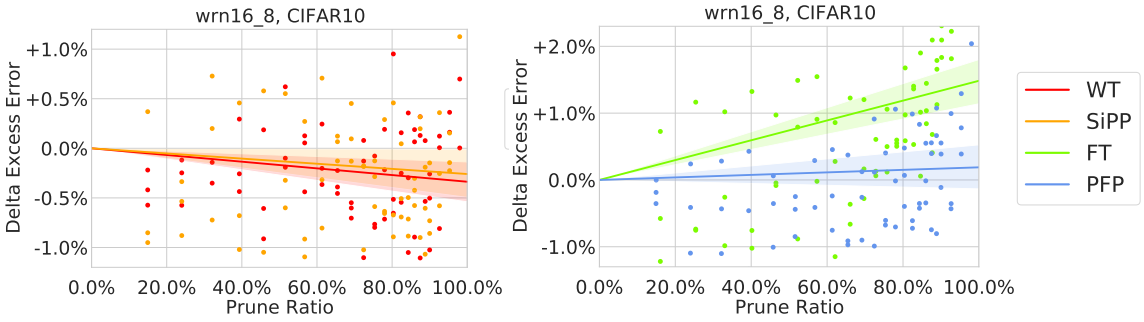


Figure B-32: The difference in excess error for a WRN16-8 trained on CIFAR-10.

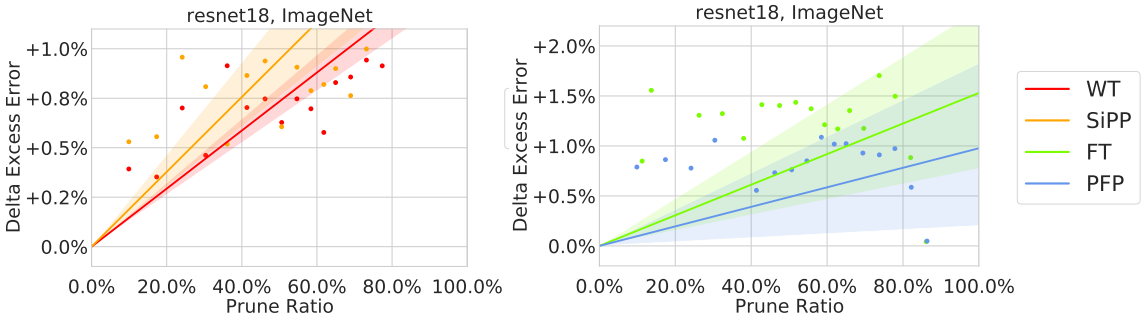


Figure B-33: The difference in excess error for a ResNet18 trained on ImageNet.

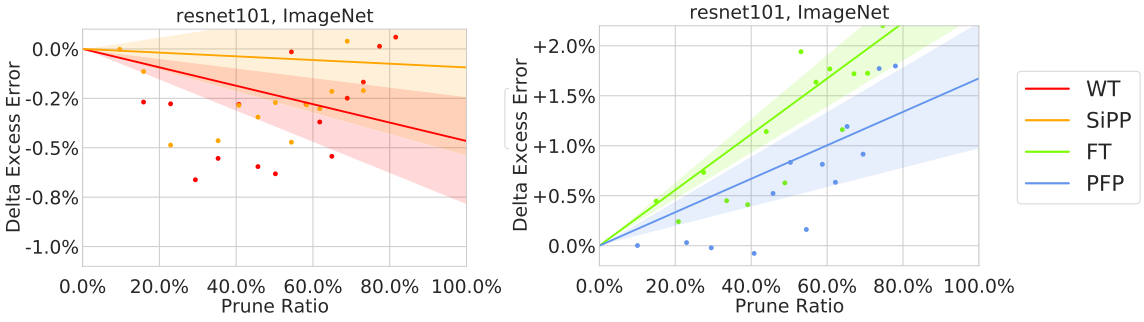


Figure B-34: The difference in excess error for a ResNet101 trained on ImageNet.

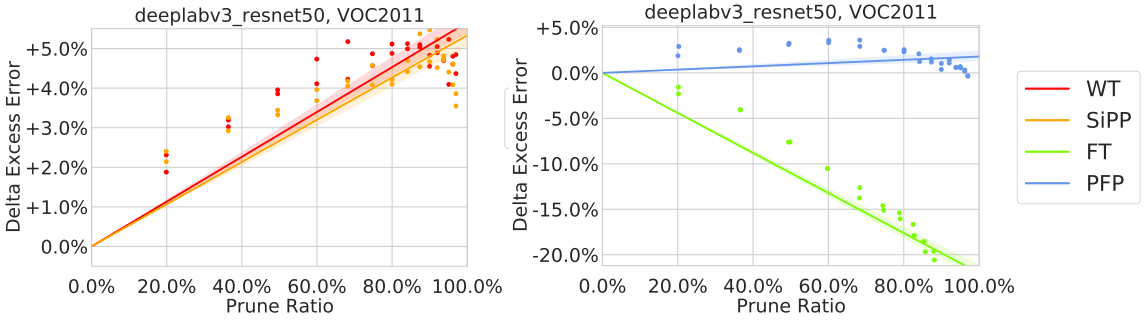


Figure B-35: The difference in excess error for a DeeplabV3 trained on Pascal VOC.

B.3.4 Results for Overparameterization

We summarize our results pertaining to using the prune potential as a way to gauge the amount of overparameterization. Specifically, for each network and prune method, we evaluate the *average* and *minimum* prune potential for both the train and test distribution. The average and minimum are hereby computed over the corruptions/variations that are included in each distribution. Note that for these experiments the train distribution only contains the nominal data; thus the average and minimum coincides. For the test distribution we take the average and minimum over all the respective corruptions. The mean and standard deviation reported are computed over three repetitions of the same experiment.

The resulting prune potentials are listed in Tables B.7 and B.8 for weight pruning (WT, SiPP) and filter pruning (FT, PFP), respectively. Note that for most networks we can observe around 20% drop in average prune potential between train and test distribution while most networks have 0% (!) minimum prune potential for data from the test distribution. As previously observed some networks may be considered *genuinely* overparameterized in the robust sense including WRN16-8, ResNet101, which manifests itself with a very stable prune potential across both train and test distribution.

Table B.7: The average and minimum prune potential computed on the train and test distribution, respectively, for weight prune methods (WT, SiPP). The train distribution hereby consists of nomimal data, while the test distribution consists of the CIFAR-10-C, ImageNet-C, VOC-C corruptions.

Model	WT - Prune Potential (%)				SiPP - Prune Potential (%)			
	Average		Minimum		Average		Minimum	
	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.
ResNet20	84.9 ± 0.0	**66.7 ± 3.3**	84.9 ± 0.0	**0.0 ± 0.0**	86.4 ± 2.2	**70.4 ± 4.3**	86.4 ± 2.2	**0.0 ± 0.0**
Resnet56	94.6 ± 0.0	**82.3 ± 4.2**	94.6 ± 0.0	**4.6 ± 6.5**	94.5 ± 0.9	**78.5 ± 1.0**	94.5 ± 0.9	**0.0 ± 0.0**
ResNet110	96.3 ± 0.6	**77.9 ± 1.4**	96.3 ± 0.6	**0.0 ± 0.0**	96.5 ± 0.7	**78.8 ± 2.8**	96.5 ± 0.7	**0.0 ± 0.0**
VGG16	98.0 ± 0.0	**80.9 ± 2.2**	98.0 ± 0.0	**0.0 ± 0.0**	98.0 ± 0.0	**80.7 ± 1.9**	98.0 ± 0.0	**0.0 ± 0.0**
DenseNet22	79.8 ± 1.9	**76.0 ± 6.7**	79.8 ± 1.9	**24.9 ± 35.2**	79.8 ± 1.9	**74.1 ± 9.1**	79.8 ± 1.9	**21.5 ± 30.4**
WRN16-8	98.0 ± 0.0	**95.7 ± 0.7**	98.0 ± 0.0	**90.0 ± 2.1**	95.3 ± 0.0	**94.1 ± 0.8**	95.3 ± 0.0	**78.1 ± 15.1**
ResNet18	85.8 ± 0.0	**63.6 ± 0.0**	85.8 ± 0.0	**0.0 ± 0.0**	81.6 ± 0.0	**57.8 ± 0.0**	81.6 ± 0.0	**0.0 ± 0.0**
ResNet101	81.6 ± 0.0	**76.8 ± 0.0**	81.6 ± 0.0	**0.0 ± 0.0**	81.6 ± 0.0	**70.7 ± 0.0**	81.6 ± 0.0	**0.0 ± 0.0**
DeeplabV3	58.9 ± 9.3	**11.6 ± 2.7**	58.9 ± 9.3	**0.0 ± 0.0**	43.0 ± 6.6	**11.5 ± 3.1**	43.0 ± 6.6	**0.0 ± 0.0**

Table B.8: The average and minimum prune potential computed on the train and test distribution, respectively, for filter prune methods (FT, PFP). The train distribution hereby consists of nomimal data, while the test distribution consists of the CIFAR-10-C, ImageNet-C, VOC-C corruptions.

Model	FT - Prune Potential (%)				PFP - Prune Potential (%)			
	Average		Minimum		Average		Minimum	
	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.
ResNet20	65.0 ± 6.7	**55.3 ± 4.8**	65.0 ± 6.7	**0.0 ± 0.0**	66.5 ± 3.0	**53.9 ± 4.4**	66.5 ± 3.0	**0.0 ± 0.0**
ResNet56	86.6 ± 1.2	**64.8 ± 3.7**	86.6 ± 1.2	**0.0 ± 0.0**	88.1 ± 0.0	**64.9 ± 4.0**	88.1 ± 0.0	**0.0 ± 0.0**
ResNet110	88.1 ± 3.5	**68.5 ± 3.2**	88.1 ± 3.5	**0.0 ± 0.0**	92.2 ± 1.8	**71.6 ± 1.8**	92.2 ± 1.8	**0.0 ± 0.0**
VGG16	85.4 ± 2.4	**66.3 ± 0.5**	85.4 ± 2.4	**0.0 ± 0.0**	95.0 ± 0.5	**77.9 ± 2.1**	95.0 ± 0.5	**0.0 ± 0.0**
DenseNet22	**47.4 ± 2.2**	58.2 ± 5.3	47.4 ± 2.2	**9.6 ± 13.6**	**51.8 ± 5.3**	59.6 ± 6.0	51.8 ± 5.3	**12.6 ± 17.8**
WRN16-8	86.2 ± 1.3	**75.7 ± 4.1**	86.2 ± 1.3	**36.6 ± 28.6**	86.9 ± 1.9	**86.6 ± 1.0**	86.9 ± 1.9	**66.7 ± 1.7**
ResNet18	13.7 ± 0.0	**13.5 ± 0.0**	13.7 ± 0.0	**0.0 ± 0.0**	30.4 ± 0.0	**22.5 ± 0.0**	30.4 ± 0.0	**0.0 ± 0.0**
ResNet101	53.1 ± 0.0	**33.5 ± 0.0**	53.1 ± 0.0	**0.0 ± 0.0**	50.3 ± 0.0	**43.0 ± 0.0**	50.3 ± 0.0	**0.0 ± 0.0**
DeeplabV3	**0.0 ± 0.0**	0.5 ± 0.5	**0.0 ± 0.0**	**0.0 ± 0.0**	20.2 ± 0.1	**6.8 ± 2.6**	20.2 ± 0.1	**0.0 ± 0.0**

B.4 Detailed Results for Robust Pruning

In this section, we consider whether including additional data augmentation techniques derived from the corruptions of CIFAR-10-C can boost and/or stabilize the prune potential of a network. Specifically, we incorporate a subset of the corruptions into the training pipeline to train and retrain the pruned network in a robust manner. Below, we list details pertaining to the experimental setup as well as report the results on the conducted experiments.

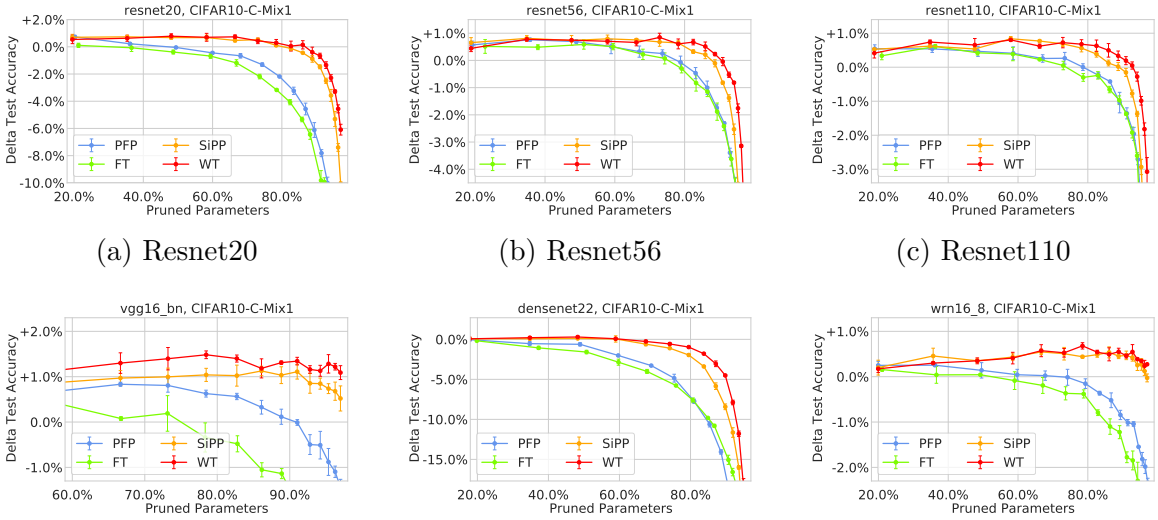
B.4.1 Experimental Setup and Prune Results

To train, prune, and retrain networks we consider the same prune pipeline and experimental setting as described in Section B.3.1. In addition, we incorporate a subset of the CIFAR-10-C corruptions into the training and retraining pipeline by corrupting the training data with the respective corruption technique. That is, when sampling a batch of training data each training image is corrupted with a CIFAR-10-C corruption (or no corruption) uniformly at random. The subset of corruptions used as part of the train and test distribution are listed in Table B.9. Note that the train and test distribution are mutually exclusive, i.e., they do not share any of the corruptions. However, as shown in Table B.9 each category of corruption is used in both the train and test distribution. For each corruption, we choose severity level 3 out of 5 just as

Table B.9: The list of corruptions used for the train and test distribution, respectively, categorized according to type.

	Train Distribution	Test Distribution
Nominal	CIFAR-10 (no corruption)	CIFAR-10.1
Noise	Impulse, Shot	Gauss
Blur	Motion, Zoom	Defocus, Glass
Weather	Snow	Brightness, Fog, Frost
Digital	Contrast, Elastic, Pixel	Jpeg

before. The nomimal prune-accuracy curves (CIFAR-10) for each of the trained and pruned networks are shown in Figure B-36.



(a) Resnet20 (b) Resnet56 (c) Resnet110

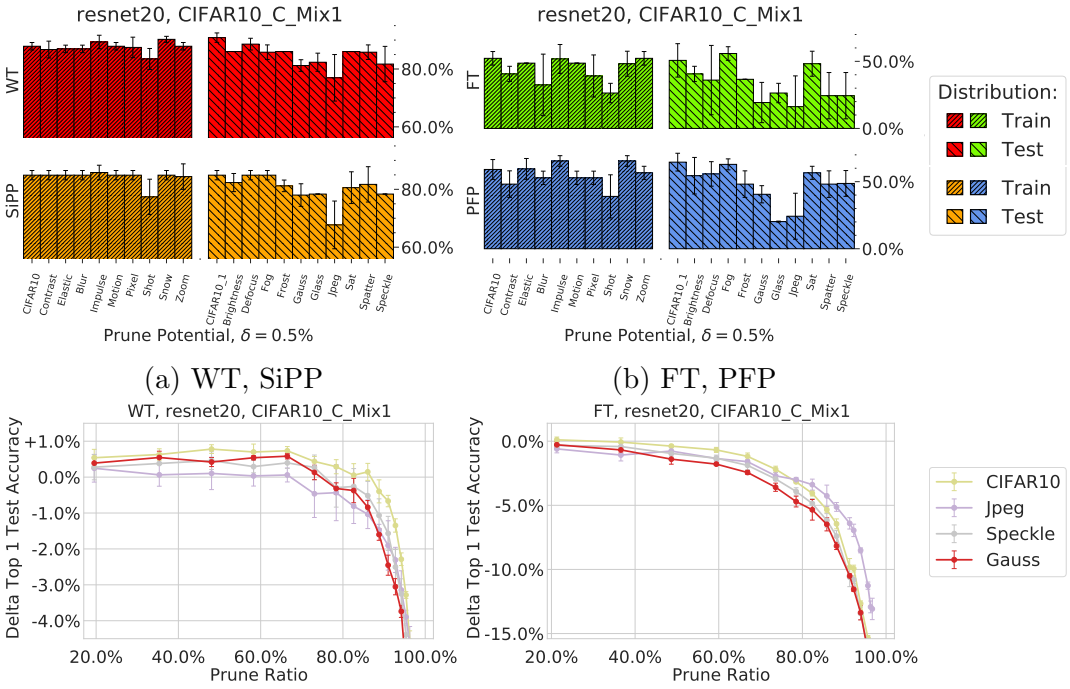
(d) VGG16 (e) DenseNet22 (f) WRN16-8

Figure B-36: The difference in test accuracy (nominal CIFAR-10) to the uncompressed network.

B.4.2 Results for Prune Potential

Our results for the prune potential of the network architectures ResNet20, ResNet56, ResNet110, VGG16, DenseNet22, and WRN16-8 are shown in Figures B-37, B-38, B-39, B-40, B-41, and B-42, respectively. We note that overall the prune potential for corruptions from the train distribution can be well preserved since we already included the respective corruptions during training and we can predict the prune potential accurately. However, we can also observe that the prune potential improves for some of the corruptions that were not included during retraining. Despite training

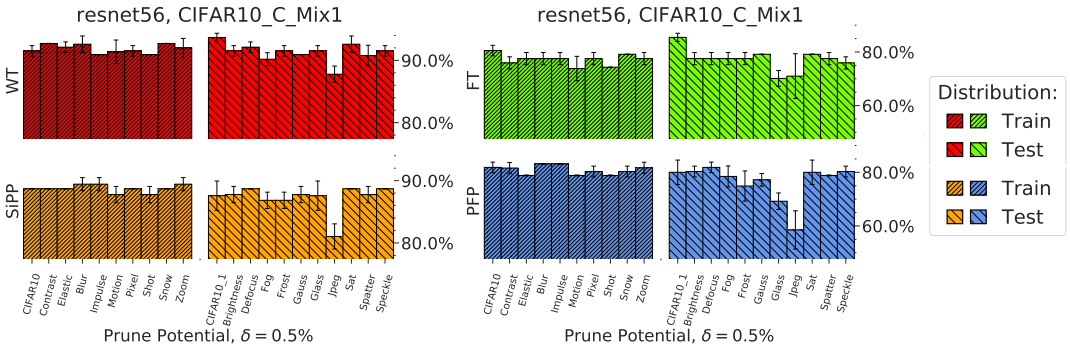
in a robust manner, however, the prune potential can still be significantly lower for corruptions from test distribution and/or exhibit high variance (low predictability).



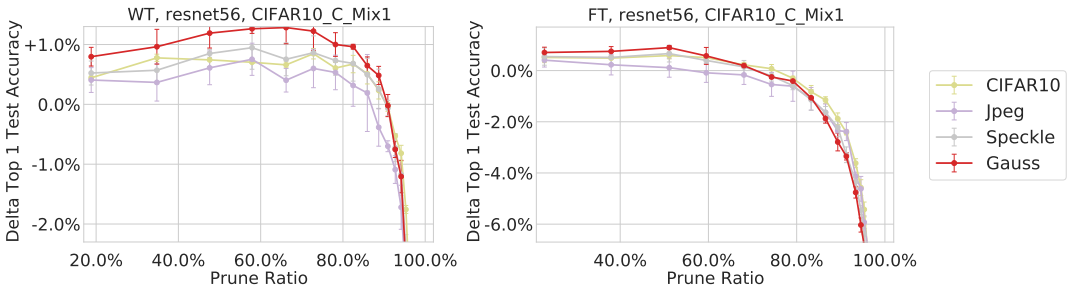
(a) WT, SiPP (b) FT, PFP

(c) WT, prune-test curve (d) FT, prune-test curve

Figure B-37: The prune potential of a robustly pruned ResNet20 achievable for CIFAR-10 out-of-distribution data sets.

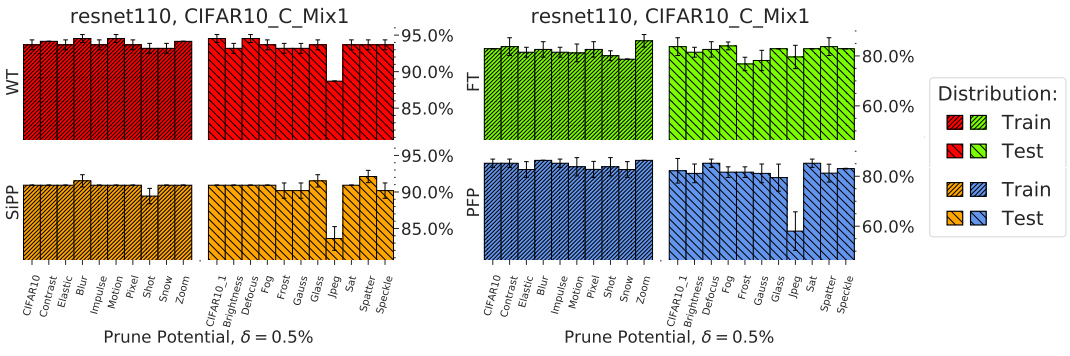


(a) WT, SiPP (b) FT, PFP

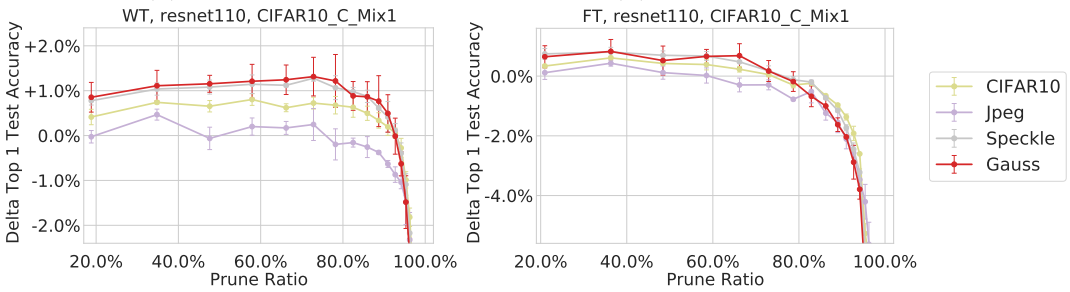


(c) WT, prune-test curve (d) FT, prune-test curve

Figure B-38: The prune potential of a robustly pruned ResNet56 achievable for CIFAR-10 out-of-distribution data sets.

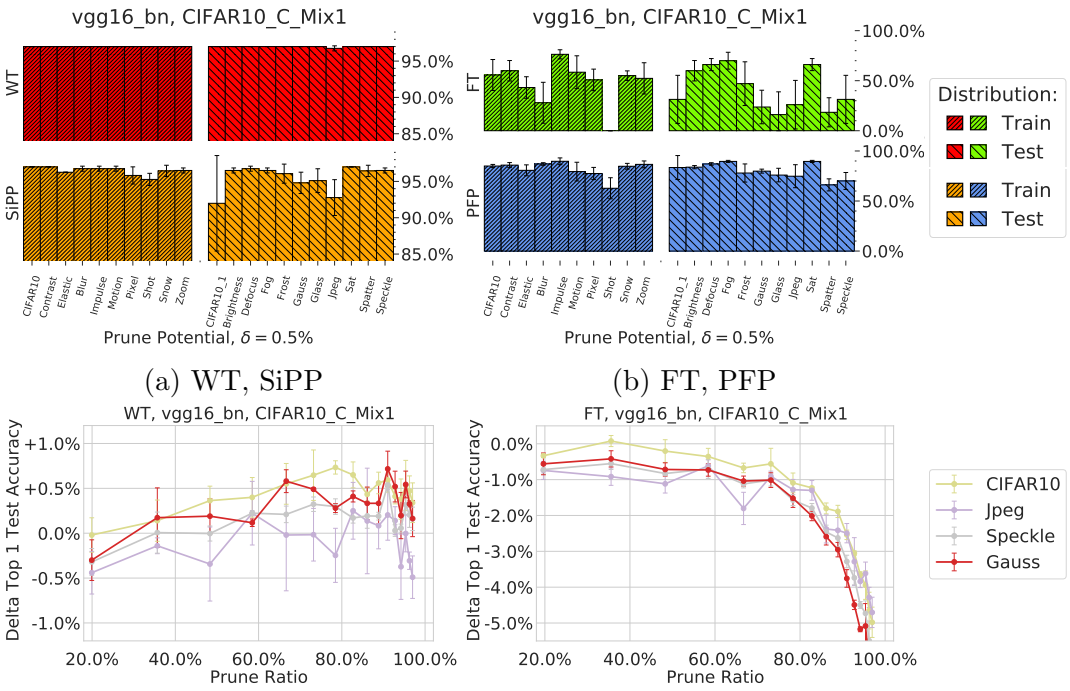


(a) WT, SiPP (b) FT, PFP



(c) WT, prune-test curve (d) FT, prune-test curve

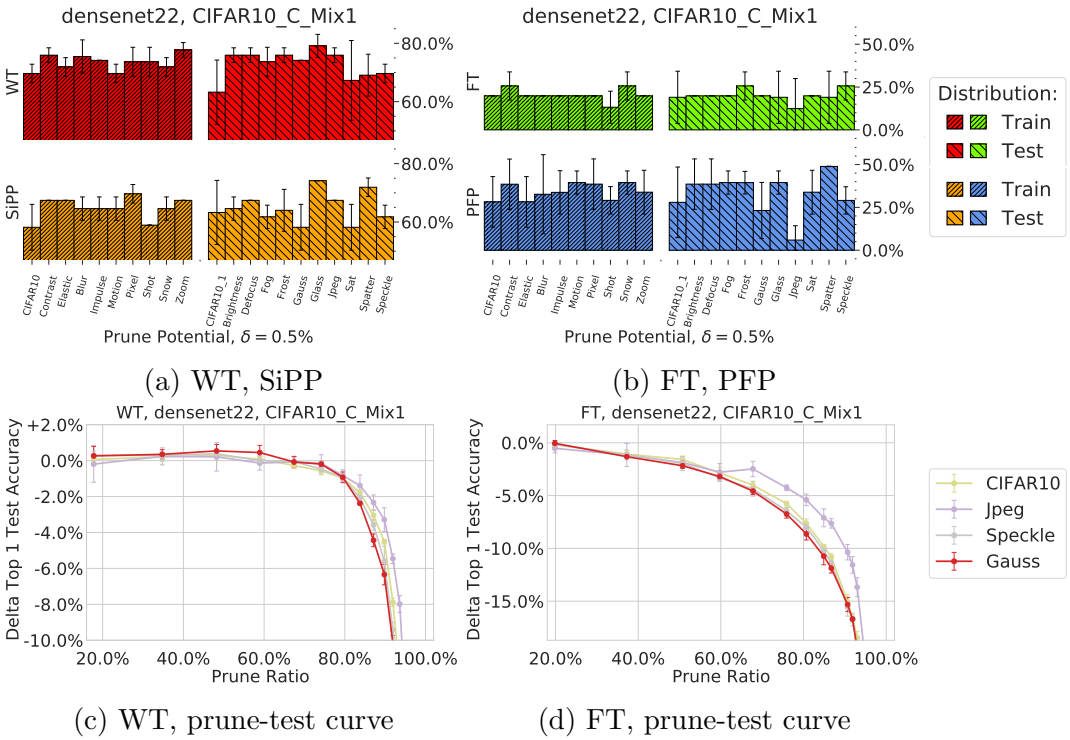
Figure B-39: The prune potential of a robustly pruned ResNet110 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP (b) FT, PFP

(c) WT, prune-test curve (d) FT, prune-test curve

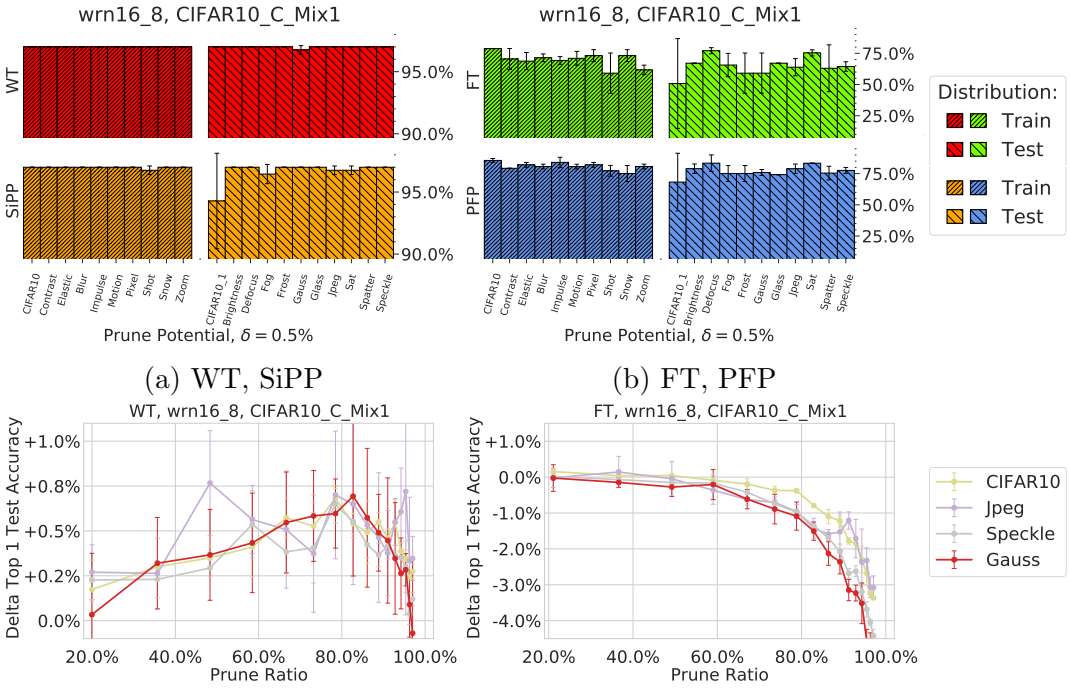
Figure B-40: The prune potential of a robustly pruned VGG16 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP (b) FT, PFP

(c) WT, prune-test curve (d) FT, prune-test curve

Figure B-41: The prune potential of a robustly pruned DenseNet22 achievable for CIFAR-10 out-of-distribution data sets.



(a) WT, SiPP

(b) FT, PFP

(c) WT, prune-test curve

(d) FT, prune-test curve

Figure B-42: The prune potential of a robustly pruned WRN16-8 achievable for CIFAR-10 out-of-distribution data sets.

B.4.3 Results for Excess Error

Following the approach described in Section B.3.3 we also evaluated the resulting difference in excess error between pruned and unpruned networks. Our results are shown in Figures B-43, B-44, B-45, B-46, B-47, and B-48 for ResNet20, ResNet56, ResNet110, VGG16, DenseNet22, and WRN16-8, respectively, all of which have been trained in a robust manner. We note that for most networks, except for smaller ones like ResNet20, the correlation between prune ratio and difference in excess error almost disappears. These results encourage the use of robust pruning techniques in order to ensure that pruned networks perform reliably. However, we note that the excess error is computed as an *average* over all corruptions included in the train and test distribution, respectively. Thus, it is not an appropriate measure to estimate whether particular corruptions could still impact the prune potential more significantly than others.

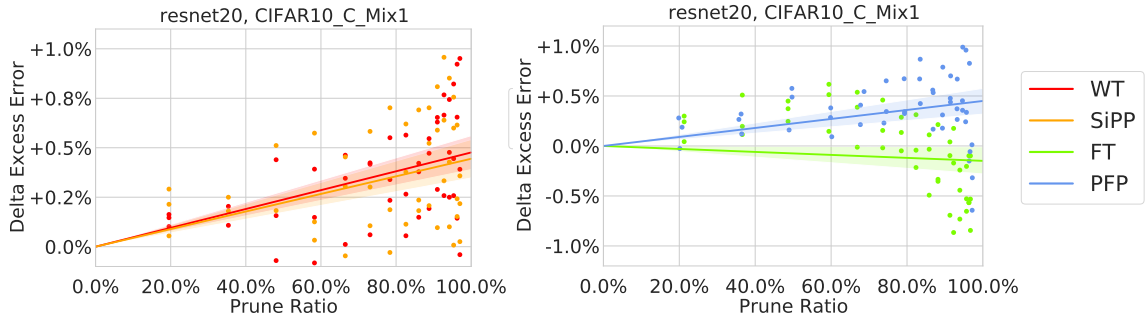


Figure B-43: The difference in excess error for a robustly pruned ResNet20 trained on CIFAR-10.

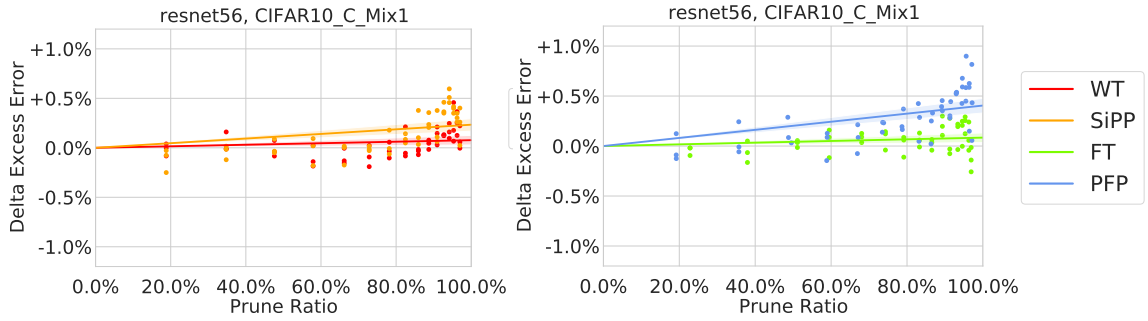


Figure B-44: The difference in excess error for a robustly pruned ResNet56 trained on CIFAR-10.

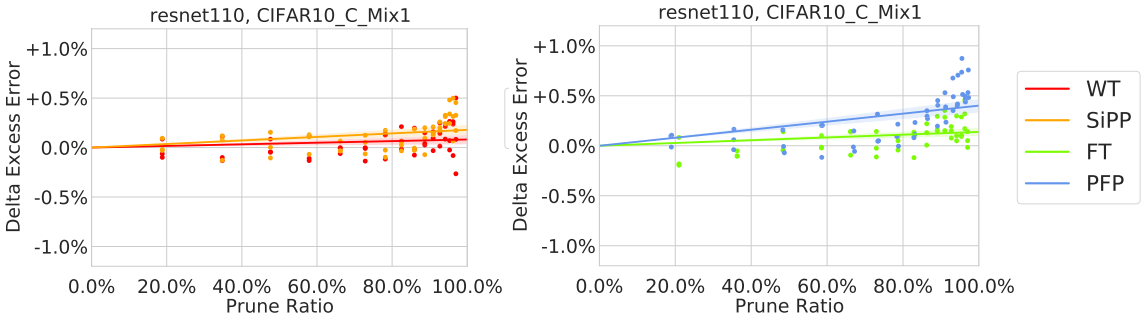


Figure B-45: The difference in excess error for a robustly pruned ResNet110 trained on CIFAR-10.

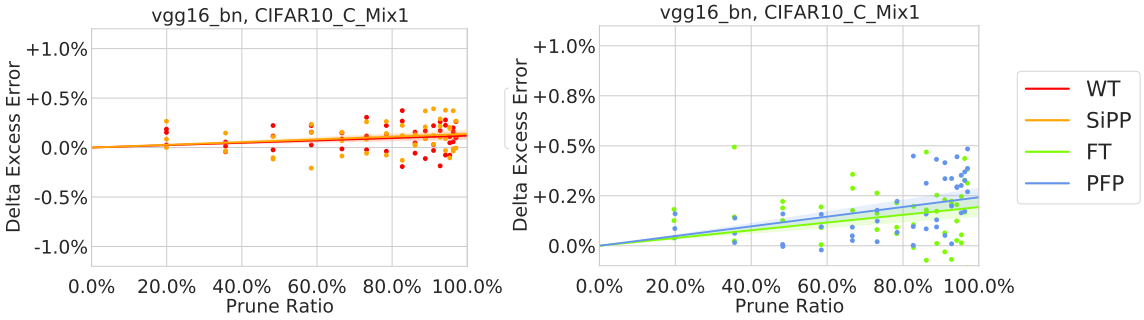


Figure B-46: The difference in excess error for a robustly pruned VGG16 trained on CIFAR-10.

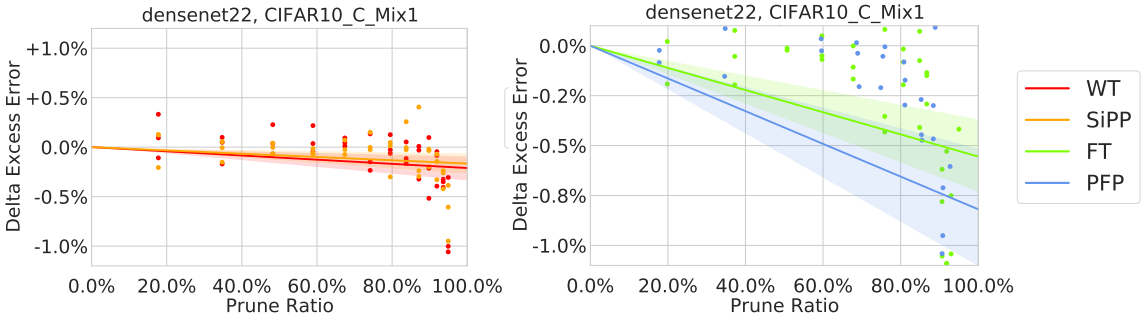


Figure B-47: The difference in excess error for a robustly pruned DenseNet22 trained on CIFAR-10.

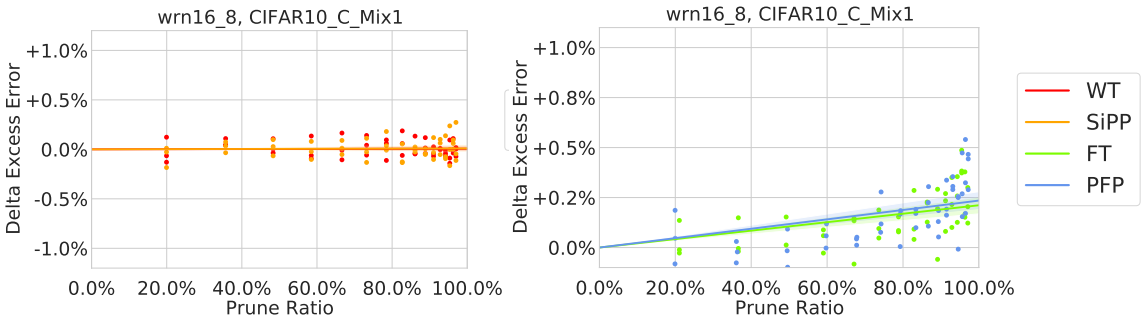


Figure B-48: The difference in excess error for a robustly pruned WRN16-8 trained on CIFAR-10.

B.4.4 Results for Overparameterization

Finally, we report the average and minimum prune potential across all networks and prune methods for the train and test distribution, respectively. As highlighted in Section B.3.4, the prune potential is used to gauge the amount of overparameterization in the network.

The resulting prune potentials are listed in Tables B.10 and B.11 for weight prune methods (WT, SiPP) and filter prune methods (FT, PFP), respectively. In contrast to Section B.3.4 the minimum and average prune potential on the train distribution differ since here the train distribution contains multiple corruptions. We note that with robust training the average prune potential remains almost unaffected by changes in the distribution as also apparent from the results in Section B.4.3. In addition, for most networks even the minimum prune potential on the test distribution is nonzero. These results further encourage the use of robust training techniques when pruning neural networks.

As elaborated upon in Section 7.5, we observe that we can regain much of the prune potential by *explicitly regularizing* the pruned network during retraining. In other words, the amount of overparameterization is not only a function of the data set and network architecture, but of the training procedure as well.

However, we note that these observations hinge upon the particular choice of the train and test distribution, which share certain commonalities in this case. Potentially, it might be possible to construct test distributions that differ significantly from the train distribution, in which case pruned networks might suffer disproportionally more from the distribution change compared to unpruned networks. These results would then be analogous to the ones without robust training presented in Section B.3.

Table B.10: The average and minimum prune potential computed on the train and test distribution, respectively, for weight prune methods (WT, SiPP). The train and test distribution hereby each consist of a mutually exclusive subset of corruptions as listed in Table B.9.

Model	WT - Prune Potential (%)				SiPP - Prune Potential (%)			
	Average		Minimum		Average		Minimum	
	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.
ResNet20	87.5 ± 1.9	**84.7 ± 0.9**	83.5 ± 3.6	**72.6 ± 4.9**	84.2 ± 1.7	**80.3 ± 1.6**	77.4 ± 6.1	**65.9 ± 6.1**
ResNet56	91.8 ± 0.5	**91.3 ± 0.4**	90.2 ± 1.0	**87.8 ± 1.3**	88.7 ± 0.5	**87.2 ± 0.1**	87.8 ± 1.3	**81.0 ± 2.0**
ResNet110	93.8 ± 0.3	**93.2 ± 0.4**	92.7 ± 0.0	**88.7 ± 0.0**	90.8 ± 0.1	**90.2 ± 0.1**	89.4 ± 1.0	**83.6 ± 1.6**
VGG16	97.0 ± 0.0	**97.0 ± 0.0**	97.0 ± 0.0	**96.8 ± 0.3**	96.5 ± 0.1	**95.5 ± 0.5**	94.6 ± 0.5	**88.3 ± 3.9**
DenseNet22	73.4 ± 1.1	**72.7 ± 1.9**	67.4 ± 0.0	**51.8 ± 5.0**	**64.7 ± 1.3**	64.8 ± 0.6	55.4 ± 5.0	**51.8 ± 5.0**
WRN16-8	97.0 ± 0.0	**97.0 ± 0.0**	97.0 ± 0.0	**96.8 ± 0.3**	97.0 ± 0.0	**96.7 ± 0.5**	96.8 ± 0.3	**94.3 ± 3.8**

Table B.11: The average and minimum prune potential computed on the train and test distribution, respectively, for filter prune methods (FT, PFP). The train and test distribution hereby each consist of a mutually exclusive subset of corruptions as listed in Table B.9.

Model	FT - Prune Potential (%)				PFP - Prune Potential (%)			
	Average		Minimum		Average		Minimum	
	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.	Train Dist.	Test Dist.
ResNet20	44.0 ± 6.3	**34.4 ± 6.1**	19.3 ± 15.0	**7.1 ± 10.0**	55.2 ± 6.2	**47.7 ± 6.4**	39.0 ± 16.1	**13.3 ± 9.4**
ResNet56	77.1 ± 0.4	**77.1 ± 0.9**	72.2 ± 2.9	**67.3 ± 6.1**	80.8 ± 0.4	**76.3 ± 0.5**	78.8 ± 0.2	**58.5 ± 7.1**
ResNet110	82.1 ± 1.8	**81.7 ± 0.8**	78.7 ± 0.0	**74.9 ± 2.7**	84.4 ± 2.1	**80.0 ± 2.2**	81.6 ± 2.1	**58.0 ± 7.7**
VGG16	48.0 ± 3.3	**41.5 ± 5.2**	**0.0 ± 0.0**	**0.0 ± 0.0**	81.9 ± 1.8	**79.8 ± 2.3**	**62.7 ± 10.5**	63.9 ± 3.9
DenseNet22	20.3 ± 0.7	**20.0 ± 5.7**	13.2 ± 9.4	**6.6 ± 9.4**	34.2 ± 8.8	**33.1 ± 6.5**	17.5 ± 14.2	**0.0 ± 0.0**
WRN16-8	69.7 ± 4.1	**64.8 ± 2.2**	54.2 ± 12.9	**24.4 ± 17.3**	80.7 ± 0.7	**76.9 ± 2.9**	72.1 ± 3.1	**57.2 ± 14.9**

Bibliography

Dimitris Achlioptas, Zohar Karnin, and Edo Liberty. Matrix entry-wise sampling: Simple is best. *Submitted to KDD*, 2013(1.1):1–4, 2013.

Ryan P Adams, Jeffrey Pennington, Matthew J Johnson, Jamie Smith, Yaniv Ovadia, Brian Patton, and James Saunderson. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

Alireza Aghasi, Afshin Abdi, Nam Nguyen, and Justin Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3180–3189, 2017.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in Neural Information Processing Systems 32*, pages 6158–6169. Curran Associates, Inc., 2019.

Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

Alexander Amini, Liam Paull, Thomas Balch, Sertac Karaman, and Daniela Rus. Learning steering bounds for parallel autonomous systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

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*, pages 254–263, 2018.

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332, 2019.

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015. URL http://arxiv.org/abs/1511.00561.

Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In *Advances in Neural Information Processing Systems*, pages 9448–9458, 2019.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6241–6250, 2017.

Cenk Baykal. *Sampling-based Algorithms for Fast and Deployable AI*. PhD thesis, Massachusetts Institute of Technology, 2021.

Cenk Baykal, Lucas Liebenwein, and Wilko Schwarting. Training support vector machines using coresets. *arXiv preprint arXiv:1708.03835*, 2017.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

Cenk Baykal, Lucas Liebenwein, Dan Feldman, and Daniela Rus. Low-regret active learning. *arXiv preprint arXiv:2104.02822*, 2021a.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *SIAM Journal on Mathematics of Data Science (Under Review; arXiv preprint arXiv:1910.05422)*, 2021b.

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.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2020*, pages 129–146, 2020.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf.

Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*, 2016.

Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Brandon Carter, Jonas Mueller, Siddhartha Jain, and David Gifford. What made you do this? understanding black-box decisions with sufficient input subsets. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 567–576, 2019.

Brandon Carter, Siddhartha Jain, Jonas Mueller, and David Gifford. Overinterpretation reveals image classification model pathologies. *arXiv preprint arXiv:2003.08907*, 2020.

Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Patrick H. Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-jui Hsieh. GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking. *Advances in Neural Information Processing Systems*, 2018-December:10988–10998, jun 2018a. URL http://arxiv.org/abs/1806.06950.

Ricky T. Q. Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual flows for invertible generative modeling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5d0d5594d24f0f955548f0fc0ff83d10-Paper.pdf.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018b.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294, 2015a.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing convolutional neural networks. *CoRR*, abs/1506.04449, 2015b. URL http://arxiv.org/abs/1506.04449.

Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020.

François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

Anna Choromanska, Krzysztof Choromanski, Mariusz Bojarski, Tony Jebara, Sanjiv Kumar, and Yann LeCun. Binary embeddings with structured hashed projections. In *International Conference on Machine Learning*, pages 344–353, 2016.

Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 163–172. ACM, 2015.

Michael B Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1758–1777. SIAM, 2017.

Jonathan A DeCastro, Lucas Liebenwein, Cristian-Ioan Vasile, Russ Tedrake, Sertac Karaman, and Daniela Rus. Counterexample-guided safety contracts for autonomous driving. In *International Workshop on the Algorithmic Foundations of Robotics*, 2018.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156, 2013.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1uR4GZRZ.

Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2019.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. *arXiv preprint arXiv:1801.06733*, 2018.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017a.

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017b.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Petros Drineas and Anastasios Zouzias. A note on element-wise matrix sparsification via a matrix-valued bernstein inequality. *Information Processing Letters*, 111(8): 385–389, 2011.

Petros Drineas, Michael W Mahoney, and Shan Muthukrishnan. Relative-error cur matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2): 844–881, 2008.

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eK3i09YQ.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

N. Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W. Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-N7PBXqOUJZ.

Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

Dan Feldman. Core-sets: Updated survey. *Sampling Techniques for Supervised or Unsupervised Tasks*, pages 23–44, 2020.

Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578. ACM, 2011.

Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A ptas for k-means clustering based on weak coresets. In *Proceedings of the twenty-third annual symposium on Computational geometry*, pages 11–18. ACM, 2007.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode. *arXiv preprint arXiv:2002.02798*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

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

Neil Gallagher, Kyle R Ulrich, Austin Talbot, Kafui Dzirasa, Lawrence Carin, and David E Carlson. Cross-spectral factor analysis. In *Advances in Neural Information Processing Systems*, pages 6842–6852, 2017.

Noah Gamboa, Kais Kudrolli, Anand Dhoot, and Ardavan Pedram. Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators. *arXiv preprint arXiv:2001.03253*, 2020.

Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/germain15.html.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2013.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

Shupeng Gui, Haotao N Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. In *Advances in Neural Information Processing Systems*, pages 1285–1296, 2019.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

Yiwen Guo, Chao Zhang, Changshui Zhang, and Yurong Chen. Sparse dnns with improved adversarial robustness. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 242–251. Curran Associates, Inc., 2018.

Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015a. URL http://arxiv.org/abs/1510.00149.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 1135–1143, 2015b.

Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *2011 International Conference on Computer Vision*, pages 991–998. IEEE, 2011.

Ramin Hasani, Alexander Amini, Mathias Lechner, Felix Naser, Radu Grosu, and Daniela Rus. Response characterization for auditing cell dynamics in long short-term memory networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019a.

Ramin Hasani, Guodong Wang, and Radu Grosu. A machine learning suite for machine components' health-monitoring. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9472–9477, 2019b.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits. In *International Conference on Machine Learning*, pages 4082–4093. PMLR, 2020.

Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Max Tschaikowski, Gerald Teschl, and Daniela Rus. Closed-form continuous-depth models. *arXiv preprint arXiv:2106.13898*, 2021a.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):7657–7666, May 2021b.

Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2234–2240. AAAI Press, 2018.

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

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

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

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

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. *arXiv preprint arXiv:2102.00554*, 2021.

Sara Hooker, Aaron Courville, Yann Dauphin, and Andrea Frome. Selective brain damage: Measuring the disparate impact of model pruning. *arXiv preprint arXiv:1911.05248*, 2019.

Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. *arXiv preprint arXiv:1805.12549*, 2018.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2078–2087. PMLR, 10–15 Jul 2018a. URL http://proceedings.mlr.press/v80/huang18d.html.

Chin-Wei Huang, Ricky TQ Chen, Christos Tsirigotis, and Aaron Courville. Convex potential flows: Universal probability distributions with optimal transport and convex optimization. *arXiv preprint arXiv:2012.05942*, 2020.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018b.

Jonathan H Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. *arXiv preprint arXiv:1605.06423*, 2016.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Yerlan Idelbayev and Miguel A Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020.

Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1231–1240, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

Priyank Jaini, Kira A Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In *International Conference on Machine Learning*, pages 3009–3018. PMLR, 2019.

William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, pages 5122–5131. PMLR, 2020.

Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12569–12577, 2019.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, nov 2015a. URL http://arxiv.org/abs/1511.06530.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015b.

Diederik P Kingma and Prafulla Dhariwal. Glow: generative flow with invertible 1× 1 convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 10236–10245, 2018.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*, 2016.

Zhifeng Kong and Kamalika Chaudhuri. The expressive power of a class of normalizing flow models. In *International Conference on Artificial Intelligence and Statistics*, pages 3599–3609. PMLR, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Abhisek Kundu and Petros Drineas. A note on randomized element-wise matrix sparsification. *arXiv preprint arXiv:1404.0320*, 2014.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. *arXiv preprint arXiv:2002.03231*, 2020.

Michael Langberg and Leonard J Schulman. Universal ε-approximators for integrals. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 598–607. SIAM, 2010.

Valero Laparra, Jesús Malo, and Gustau Camps-Valls. Dimensionality reduction via regression in hyperspectral imagery. *IEEE Journal of Selected Topics in Signal Processing*, 9(6):1026–1036, 2015.

Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2554–2564. IEEE, 2016.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6553.

Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10):642–652, 2020a.

Mathias Lechner, Ramin Hasani, Daniela Rus, and Radu Grosu. Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5446–5452. IEEE, 2020b.

Mathias Lechner, Ramin Hasani, Radu Grosu, Daniela Rus, and Thomas A Henzinger. Adversarial training is not ready for robot learning. *arXiv preprint arXiv:2103.08187*, 2021.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

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.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1): 1334–1373, 2016.

Chong Li and CJ Shi. Constrained optimization based low-rank approximation of deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 732–747, 2018.

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

Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. Oicsr: Out-in-channel sparsity regularization for compact deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7046–7055, 2019a.

Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 3870–3882. PMLR, 2020.

Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019b.

Lucas Liebenwein. Torchprune: A research library for pytorch-based neural network pruning, compression, and more. https://github.com/lucaslie/torchprune, 2021.

Lucas Liebenwein, Cenk Baykal, Igor Gilitschenski, Sertac Karaman, and Daniela Rus. Sampling-based approximation algorithms for reachability analysis with provable guarantees. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.014.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 93–138, 2021a.

Lucas Liebenwein, Ramin Hasani, Alexander Amini, and Daniela Rus. Sparse flows: Pruning continuous-depth models. In *Advances in Neural Information Processing Systems (Under Review; arXiv preprint arXiv:2106.12718)*, 2021b.

Lucas Liebenwein, Alaa Maalouf, Oren Gal, Dan Feldman, and Daniela Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. In *Advances in Neural Information Processing Systems (Under Review; arXiv preprint arXiv:2107.11442)*, 2021c.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188, 2017.

Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJem8lSFwB.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=S1eYHoC5FX.

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

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

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019c. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

Björn Lütjens, Lucas Liebenwein, and Katharina Kramer. Machine learning-based estimation of forest carbon stocks to increase transparency of forest preservation efforts. *arXiv preprint arXiv:1912.07850*, 2019.

Alaa Maalouf, Ibrahim Jubran, and Dan Feldman. Fast and accurate least-mean-squares solvers. In *Advances in Neural Information Processing Systems*, pages 8305–8316, 2019.

Alaa Maalouf, Adiel Statman, and Dan Feldman. Tight sensitivity bounds for smaller coresets. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2051–2061, 2020.

Alaa Maalouf, Harry Lang, Daniela Rus, and Dan Feldman. Deep learning meets projective clustering. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=EQfpYwF3-b.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asma. Dissecting neural odes. In *34th Conference on Neural Information Processing Systems, NeurIPS 2020*. The Neural Information Processing Systems, 2020.

Shannon McCurdy. Ridge regression and provable deterministic ridge leverage score sampling. In *Advances in Neural Information Processing Systems*, pages 2463–2472, 2018.

Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S. Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming. *arXiv preprint arXiv:1907.07484*, 2019.

Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

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

Alejandro Molina, Alexander Munteanu, and Kristian Kersting. Core dependency networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI). AAAI Press Google Scholar*, 2018.

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019.

Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz*, 32(1):37–53, 2018.

Vaishnavh Nagarajan and Zico Kolter. Deterministic PAC-bayesian generalization bounds for deep networks via generalizing noise-resilience. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Hygn2o0qKX.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1g5sA4twr.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *ICLR (Workshop)*, 2015. URL http://arxiv.org/abs/1412.6614.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564*, 2017a.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5949–5958, 2017b.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BygfghAcYX.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 442–450, 2015.

Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3898–3907. PMLR, 2018a.

Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3898–3907. PMLR, 10–15 Jul 2018b. URL http://proceedings.mlr.press/v80/oliva18a.html.

Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. *arXiv preprint arXiv:2006.00104*, 2020.

Dimitris Papailiopoulos, Anastasios Kyrillidis, and Christos Boutsidis. Provable deterministic leverage score sampling. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 997–1006. ACM, 2014.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3967–3976, 2019.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Xi Peng, Zhang Yi, and Huajin Tang. Robust subspace clustering via thresholding ridge regression. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. Ac/dc: Alternating compressed/decompressed training of deep neural networks. *arXiv preprint arXiv:2106.12379*, 2021.

Konstantinos Pitas, Mike Davies, and Pierre Vandergheynst. Revisiting hard thresholding for dnn pruning. *arXiv preprint arXiv:1905.08793*, 2019.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Torchdyn: A neural differential equations library. *arXiv preprint arXiv:2009.09346*, 2020a.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, Jinkyoo Park, et al. Hypersolvers: Toward fast continuous-depth models. *Advances in Neural Information Processing Systems*, 33, 2020b.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400, 2019.

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

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

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

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

Wilko Schwarting, Tim Seyde, Igor Gilitschenski, Lucas Liebenwein, Ryan Sander, Sertac Karaman, and Daniela Rus. Deep latent competition: Learning to race using visual control policies in latent space. In *Conference on Robot Learning*, Proceedings of Machine Learning Research. PMLR, 2020.

Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Towards compact and robust deep neural networks. *arXiv preprint arXiv:1906.06110*, 2019.

Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. Hydra: Pruning adversarially robust neural networks. *arXiv preprint arXiv:2002.10509*, 2020.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10(Nov):2615–2637, 2009.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096, 2015.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. *arXiv preprint arXiv:2004.14340*, 2020.

Akash Srivastava, Lazar Valkov, Chris Russell, Michael U. Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/44a2e0804995faf8d2e3b084a1e2db1d-Paper.pdf.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

Takeshi Teshima, Isao Ishikawa, Koichi Tojo, Kenta Oono, Masahiro Ikeda, and Masashi Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. *arXiv preprint arXiv:2006.11469*, 2020.

Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xCxXwTzx4L1.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

Joel A Tropp et al. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

Murad Tukan, Cenk Baykal, Dan Feldman, and Daniela Rus. On coresets for support vector machines. In *International Conference on Theory and Applications of Models of Computation*, pages 287–299. Springer, 2020a.

Murad Tukan, Alaa Maalouf, Matan Weksler, and Dan Feldman. Compressed deep networks: Goodbye svd, hello robust low-rank approximation. *arXiv preprint arXiv:2009.05647*, 2020b.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Ramon van Handel. Probability in high dimension. Technical report, PRINCETON UNIV NJ, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

Roman Vershynin. High-dimensional probability. *An Introduction with Applications*, 2016.

Charles Vorbach, Ramin Hasani, Alexander Amini, Mathias Lechner, and Daniela Rus. Causal navigation by continuous-time neural networks. *arXiv preprint arXiv:2106.08314*, 2021.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgsACVKPH.

Luyu Wang, Gavin Weiguang Ding, Ruitong Huang, Yanshuai Cao, and Yik Chau Lui. Adversarial robustness of pruned neural networks. *ICLR Workshop submission*, 2018.

Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. *arXiv preprint arXiv:1908.05164*, 2019.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating Filters for Faster Deep Neural Networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:658–666, mar 2017. URL http://arxiv.org/abs/1703.09746.

Kyle Wiggers. Openai's massive gpt-3 model is impressive, but size isn't everything, May 2021. URL https://venturebeat.com/2020/06/01/ai-machine-learning-openai-gpt-3-size-isnt-everything/.

Arie Wahyu Wijayanto, Jun Jin Choong, Kaushalya Madhawa, and Tsuyoshi Murata. Towards robust compressed convolutional neural networks. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–8. IEEE, 2019.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. TRP: Trained Rank Pruning for Efficient Deep Neural Networks. *IJCAI International Joint Conference on Artificial Intelligence*, 2021-Janua:977–983, apr 2020. URL http://arxiv.org/abs/2004.14566.

Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369, 2013.

Liu Yang and George Em Karniadakis. Potential flow generator with l2 optimal transport regularity for generative models. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 581–590. IEEE, 2020.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJ94fqApW.

Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*, pages 10820–10830. PMLR, 2020.

Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 2, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018a.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018b.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2016. URL https://openreview.net/forum?id=Sy8gdB9xx&.

Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating Very Deep Convolutional Networks for Classification and Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, may 2015a. URL http://arxiv.org/abs/1505.06798.

Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015b.

Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*, pages 1984–1992, 2015c.

Yilin Zhang and Karl Rohe. Understanding regularized spectral clustering via graph conductance. In *Advances in Neural Information Processing Systems*, pages 10631–10640, 2018.

Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4082–4090. JMLR. org, 2017.

Yiren Zhao, Ilia Shumailov, Robert Mullins, and Ross Anderson. To compress or not to compress: Understanding the interactions between adversarial attacks and neural network compression. *arXiv preprint arXiv:1810.00208*, 2018.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In *Proceedings of the International Conference on Learning Representations (ICLR) 2017*, feb 2017.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Compressibility and generalization in large-scale deep learning. *arXiv preprint arXiv:1804.05862*, 2018a.

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. In *International Conference on Learning Representations*, 2018b.

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.

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.