Adaptive Sharpness-Aware Pruning for Robust Sparse Networks

Anna Bair[1] **Hongxu Yin**[2] **Maying Shen**[2] **Pavlo Molchanov**[2] **Jose M. Alvarez**[2]
[1]Carnegie Mellon University [2]NVIDIA
abair@cmu.edu, {dannyy, mshen, pmolchanov, josea}@nvidia.com

Abstract

Robustness and compactness are two essential attributes of deep learning models that are deployed in the real world. The goals of robustness and compactness may seem to be at odds, since robustness requires generalization across domains, while the process of compression exploits specificity in one domain. We introduce *Adaptive Sharpness-Aware Pruning (AdaSAP)*, which unifies these goals through the lens of network sharpness. The AdaSAP method produces sparse networks that are robust to input variations which are *unseen at training time*. We achieve this by strategically incorporating weight perturbations in order to optimize the loss landscape. This allows the model to be both primed for pruning and regularized for improved robustness. AdaSAP improves the robust accuracy of pruned models on classification and detection over recent methods by up to +6% on OOD datasets, over a wide range of compression ratios, pruning criteria, and architectures.

1 Introduction

Robustness to out of distribution (OOD) data and efficiency are two important requirements for modern deep learning models. In this work, we focus on efficiency via network compression, in particular network pruning. Pruning retains neurons deemed important over the training distribution, so pruned networks can have reduced OOD generalization, as observed in [29, 40].

Our goal is to produce compact and robust neural networks. We emphasize robustness to input variations that are unseen at training time. Our method leverages a new flatness-based optimization procedure that both primes the network for pruning and improves network robustness. Flatness can mitigate the loss in performance during the pruning procedure and regularize the network towards improved robustness. AdaSAP significantly improves the relative robustness over prior pruning art, seen in Figure 1.[1] Our contributions are:

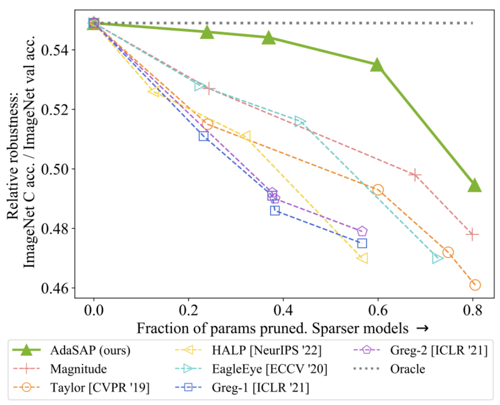

Figure 1: Robustness of pruned models trained on ImageNet degrades on ImageNet-C at high sparsities for many SOTA pruning methods. AdaSAP reduces the degradation in robust performance relative to validation performance. We approach the dashed line, which indicates equal rates of degradation of robust and standard performance.

[1]Relative robustness in Fig. 1 refers to the ratio between the accuracy on corrupted images and the standard validation accuracy at a given prune ratio. A dense model in this setup retains 55% of its validation performance when given corrupted images (location of grey dashed line).

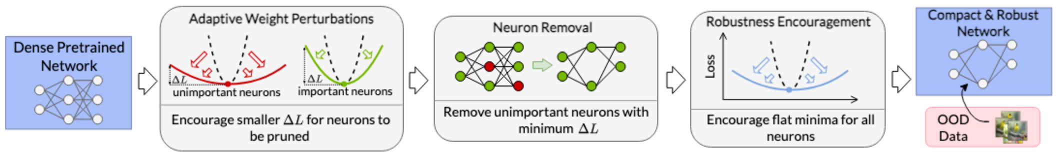
Dense Pretrained Network
Adaptive Weight Perturbations
ΔL
unimportant neurons
ΔL
important neurons
Encourage smaller ΔL for neurons to be pruned
Neuron Removal
Remove unimportant neurons with minimum ΔL
Robustness Encouragement
Loss
Encourage flat minima for all neurons
Compact & Robust Network
OOD Data

Figure 2: AdaSAP is a three step process that takes as input a dense pretrained model and outputs a sparse robust model. The process can be used with any pruning method.

- We introduce Adaptive Sharpness-Aware Pruning (AdaSAP), a method to jointly optimize for sparsity and robustness in deep networks.
- We propose novel *adaptive weight perturbations* that prepare a network for both pruning and robustness to OOD inputs. This strategy actively manipulates the flatness of the loss surface so as to minimize the impact of removing neurons with lower importance scores.
- We demonstrate state-of-the-art performance compared to prior art by noticeable margins across a wide range of setups, covering (i) two tasks (classification and detection), (ii) two OOD types (image corruption and distribution shift), (iii) four networks (ResNet50 and MobileNet V1/V2 for classification, SSD512 for detection), (iv) two pruning paradigms (parameter-based and latency-based), and (v) sparsity ratios spanning from around 20% to 80%.

2 Related Works

Robustness. In this work we primarily focus on robustness to corrupted images [24, 49] due to its application in real-world settings rather than other forms of robustness, such as adversarial robustness [2, 5, 6, 24, 25, 44, 54, 56]. Several prior works examine the robustness of sparse networks [11, 17, 51]. However, almost all current pruning methods still lead to sparse networks having worse performance on OOD data than dense networks, a result we replicate in Figure 1 [29, 40].

Efficiency. Efforts to improve network efficiency include pruning, distillation, quantization, and adaptation [3, 8, 16, 18, 20, 21, 26, 28, 37, 38, 45–47, 52, 57, 59, 63, 65, 66]. In the current work, we focus on structured channel-wise pruning due to the easily obtained speedups. By pruning larger structural elements within a network, such as kernels or filters, rather than individual weights, the pruned elements can be removed and more directly lead to speed gains [1, 46, 52].

Flat minima. The relationship between flat minima and generalization has been studied for many years [4, 7, 19, 27, 32–35, 55, 61, 62]. Flat minima are generally correlated with improved generalization, but flatness can also be metric-dependent [12]. Sharpness-Aware Minimization (SAM) [15] is a way to optimize for finding flatter minima and leads to better generalization. Recent work improves the efficiency, performance, and applicability of SAM [13, 36, 42, 48] The present work is inspired by SAM and its use in model compression [15, 48] but differs in its use of adaptive weight perturbations and its emphasis on the robustness of pruned models. In this work we use the formulation introduced in ASAM [36] due to its improved performance over that of SAM.

3 Method

Flat minima are minima within relatively large regions of the parameter space that have low loss. Optimizing for flatter minima improves generalization and adversarial robustness in deep networks and generalization in sparse networks [15, 48, 55]. Based on these observations, we hypothesize that optimizing for flatter minima during pruning can also enhance robustness.

(Step 1) Adaptive Weight Perturbations. Our procedure performs gradient updates which are informed by the local flatness in order to best prepare the network for pruning. We follow the intuition that important weights are worth adding some sharpness to our model, while unimportant weights can have stronger regularization [46]. Unlike previous flatness-based optimization methods [15, 32, 36, 50] which apply regularization uniformly, we adapt the size of allowable perturbations based on the likelihood of each weight to be pruned: weights with low importance scores have large perturbation balls which enforce that they lie within flatter regions and vice versa.

The complete derivation of our gradient update is in Appendix 7, but we briefly review it here: Consider a network whose parameters are grouped into I partitions, where each partition i includes the set of weights $\mathbf{w}_i$. Consider training loss $L_S(\mathbf{w})$ of the network over the training dataset S, and a set of perturbation ball radius sizes $\boldsymbol{\rho}$, with each ρ_i corresponding to the size of allowable perturbations for neuron $\mathbf{w}_i$. Our objective, which places unimportant neurons in flatter regions, is:

$$\min_{\mathbf{w}} \max_{\{\|T_{\mathbf{w}}^{-1}\epsilon_i\| \leq \rho_i\}_{i=1}^{I}} L_S(\mathbf{w} + \epsilon). \tag{1}$$

where $T_{\mathbf{w}}$ and its inverse $T_{\mathbf{w}}^{-1}$ are transformations that can reshape the perturbation region (*i.e.*, not necessarily a ball) [36]. The values ρ_i are a function of each neuron's importance score. This objective essentially finds the maximal (transformed) perturbation ϵ_i (transformed to $T_{\mathbf{w}}^{-1}\epsilon_i$) within the range of allowable sizes ρ_i for each neuron $\mathbf{w}_i$. We then perturb all weights $\mathbf{w}$ with the set of perturbations ϵ and minimize the loss over the perturbed weights.

Our enhanced gradient update optimizes for finding a minimum that also lies in a region of low loss. Beginning with our objective in Eq. 1, we perform a series of approximations in order to find the most adversarial perturbations $\boldsymbol{\epsilon} = \{\epsilon_i\}$ (*i.e.,* highest loss near the minimum) which we can apply to the neurons $\mathbf{w} = \{\mathbf{w}_i\}$ in our network. Our final flatness-informed gradient update is

$$\nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i) \approx \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i + \hat{\epsilon}_i}$$

where

$$\hat{\epsilon}_i = \rho_i \frac{T_{\mathbf{w}}^2 \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)}{\|\nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)\|_2}.$$

We use this gradient update in place of the standard gradient update during the warmup phase of our pruning procedure before neuron removal. This sets up our network so that when pruning begins, we have a model that is closer to a flat minima.

(Step 2) Neuron Removal. We use structured channel-wise pruning in order to obtain latency and size reductions. AdaSAP can work with a range of pruning methods. The method could also be applied to unstructured pruning, although we do not explore this in the present work.

(Step 3) Robustness Encouragement. We regularize weights uniformly since pruning is complete and we now enforce robustness for the entire network. This uniform (i.e. non-adaptive) weight regularization is equivalent to using the optimization procedure SAM [15].

3.1 Final Comments

AdaSAP *is not a pruning method* but rather an optimization paradigm that performs robustness-aware pruning. Our method can be used in conjunction with any existing pruning criteria. Additionally, we note that our method *differs from robust pruning methods* such as [51, 58, 67] since we consider a setting in which the model may be exposed to *novel types of corruptions at test time*. We demonstrate results on OOD datasets in an attempt to demonstrate the versatility of our method: despite only training on ImageNet data, we can still expect the models to be robust to unseen input variation.

4 Results

4.1 Experiment Details

Full details can be found in Appendix 6.

Pruning criteria. We evaluate our method on two types of pruning: *parameter-based*, which reduces the number of paramenters, and *latency-based*, which optimizes for faster inference. We refer to our two variants as AdaSAP$_P$ (parameter-specific) and AdaSAP$_L$ (latency-specific). We use ℓ_2 norm magnitude pruning for AdaSAP$_P$ and HALP for AdaSAP$_L$.

Metrics. We report the Top1 accuracy of the pruned models on ImageNet (validation), ImageNet C, and ImageNet V2. Additionally, we report two *robustness ratios*: R_C and R_{V2}, defined as the ratio in Top1 accuracy on the robustness dataset to the Top1 accuracy on the validation set. That is, $R_{\text{C}} = \text{acc}_{\text{C}}/\text{acc}_{val}$ and $R_{\text{V2}} = \text{acc}_{\text{V2}}/\text{acc}_{val}$.

Optimization. We use ASAM [36], a recent improvement upon SAM.

Baselines. We compare to Taylor pruning, [46] HALP [52], GR-eg [60], EagleEye [39], ABCPruner [41], SMCP [31], and magnitude pruning [21].

4.2 Classification

Parameter reduction.

Table 1 shows how AdaSAP compares favorably to other pruning methods at a variety of pruning ratios for ResNet50. Our method performs strongly across all accuracy and robustness ratio metrics. MobileNet V1 and V2 results show a similar trend and are included in Appendix 8.

Latency Reduction.

Table 2 shows that using AdaSAP with HALP outperforms top prior works across all metrics.

Table 1: **ResNet50 - Parameter Reduction.**

Method	Size ↓	Val	R_{V2}	R_{C}	IN-V2	IN-C
			ResNet50			
Dense	1	77.32	0.83	0.55	64.79	42.46
Magnitude	0.20	73.80	0.83	0.48	61.40	35.27
Taylor	0.20	73.56	0.82	0.46	60.56	33.93
EagleEye 1G	0.27	74.13	0.83	0.47	61.30	34.84
AdaSAP$_P$	**0.20**	**74.63**	**0.83**	**0.50**	**62.08**	**37.30**
Magnitude	0.44	76.83	**0.85**	0.51	**64.92**	39.27
Taylor	0.42	75.85	0.84	0.50	63.51	37.84
Greg-1	0.43	73.72	0.82	0.48	60.47	35.04
Greg-2	0.43	73.84	0.83	0.48	61.05	35.39
ABCPruner	0.44	73.52	0.82	0.50	60.46	36.64
AdaSAP$_P$	**0.40**	**77.27**	0.83	**0.53**	64.51	**41.23**
Greg-1	0.62	75.16	0.82	0.49	61.82	36.88
Greg-2	0.62	75.36	0.82	0.49	62.06	37.09
ABCPruner	0.71	74.84	0.83	0.51	61.73	38.35
AdaSAP$_P$	**0.62**	**77.99**	**0.84**	**0.52**	**65.49**	**42.68**
Magnitude	0.76	77.32	0.84	0.53	65.20	40.73
Taylor	0.76	77.05	0.84	0.52	64.53	39.68
Greg-1	0.77	76.25	0.83	0.51	63.61	38.96
EagleEye 3G	0.78	77.07	0.84	0.53	64.84	40.67
AdaSAP$_P$	0.77	**78.29**	**0.84**	**0.55**	**66.14**	**43.22**

Table 2: **ResNet50 - Latency Reduction.**

Method	Speedup ↑	Size ↓	Val	R_{V2}	R_{C}	IN-V2	IN-C
				ResNet50			
Dense	1	1	77.32	0.838	0.55	64.79	42.46
HALP	2.6	0.43	74.46	0.82	0.47	61.21	35.03
AdaSAP$_L$	2.6	0.41	**75.37**	**0.83**	**0.50**	**62.61**	**37.93**
HALP	1.6	0.68	76.55	0.83	0.51	63.62	39.13
SMCP	1.7	0.60	76.62	0.83	0.51	63.86	38.72
AdaSAP$_L$	1.6	0.65	**77.28**	**0.85**	**0.54**	**65.35**	**41.63**
HALP	1.2	0.87	77.45	0.84	0.53	64.88	40.77
SMCP	1.2	0.87	77.57	0.84	0.53	65.02	40.91
AdaSAP$_L$	1.1	0.82	**77.93**	0.84	**0.55**	**65.53**	**42.92**

4.3 Object Detection

Results and training details can be found in Appendix 6 and 8. Across two pruning levels, AdaSAP consistently outperforms the HALP baseline by noticeable margins.

4.4 Ablations

Sensitivity to importance metrics. Although ℓ_2 norm leads to the best performance in the parameter-based setting, AdaSAP can work with other saliency metrics, such as Taylor importance, and outperform baselines, as seen in Appendix 9.

Adaptive perturbations. Using adaptive weight perturbations in the first step leads to improved performance over using uniform weight perturbations (i.e. SAM) by helping the network achieve a more prunable state, as seen in Appendix 9.

4.5 Sharpness Analysis

We measure sharpness of the model directly before and after pruning. Full results are in Appendix 8. Using AdaSAP leads to flatter models both before and after pruning. This can help explain why our method leads to improved generalization.

5 Conclusion

We introduce the Adaptive Sharpness-Aware Pruning method (AdaSAP) which optimizes for both accuracy and robust generalization during the pruning procedure. The method consists of three steps: application of our novel adaptive weight perturbations, pruning, and flatness-based robustness encouragement. AdaSAP outperforms a variety of SOTA pruning techniques on clean and robust performance and relative robustness in both image classification and object detection.

References

[1] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

[2] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *arXiv preprint arXiv:1805.12177*, 2018.

[3] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018.

[4] Brian Bartoldson, Ari Morcos, Adrian Barbu, and Gordon Erlebacher. The generalization-stability tradeoff in neural network pruning. *Advances in Neural Information Processing Systems*, 33:20852–20864, 2020.

[5] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[6] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017.

[7] Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. *Advances in Neural Information Processing Systems*, 34:22405–22418, 2021.

[8] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019.

[9] Pau de Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081*, 2020.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] James Diffenderfer, Brian Bartoldson, Shreya Chaganti, Jize Zhang, and Bhavya Kailkhura. A winning hand: Compressing deep networks can improve out-of-distribution robustness. *Advances in Neural Information Processing Systems*, 34:664–676, 2021.

[12] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017.

[13] Jiawei Du, Daquan Zhou, Jiashi Feng, Vincent YF Tan, and Joey Tianyi Zhou. Sharpness-aware training for free. *arXiv preprint arXiv:2205.14083*, 2022.

[14] M. Everingham, L. Van Gool, C. KI Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88:303–308, 2009.

[15] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[17] Yonggan Fu, Qixuan Yu, Yang Zhang, Shang Wu, Xu Ouyang, David Cox, and Yingyan Lin. Drawing robust scratch tickets: Subnetworks with inborn robustness are found within randomly initialized networks. *Advances in Neural Information Processing Systems*, 34:13059–13072, 2021.

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

[19] Shuxuan Guo, Jose M Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. *Advances in Neural Information Processing Systems*, 33:1298–1310, 2020.

[20] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[21] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[22] 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.

[23] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[24] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[25] Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. Using trusted data to train deep networks on labels corrupted by severe noise. *Advances in neural information processing systems*, 31, 2018.

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

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

[28] J. Hoffmann, S. Agnihotri, T. Saikia, and T. Brox. Towards improving robustness of compressed cnns. In *ICML UDL Workshop*, 2021.

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

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

[31] Ryan Humble, Maying Shen, Jorge Albericio Latorre, Eric Darve, and Jose Alvarez. Soft masking for cost-constrained channel pruning. In *European Conference on Computer Vision*, pages 641–657. Springer, 2022.

[32] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[33] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

[34] Jean Kaddour, Linqing Liu, Ricardo Silva, and Matt Kusner. When do flat minima optimizers work? In *Advances in Neural Information Processing Systems*.

[35] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[36] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In *International Conference on Machine Learning*, pages 5905–5914. PMLR, 2021.

[37] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *NeurIPS*, 1989.

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

[39] Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European conference on computer vision*, pages 639–654. Springer, 2020.

[40] Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proceedings of Machine Learning and Systems*, 3:93–138, 2021.

[41] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565*, 2020.

[42] Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Towards efficient and scalable sharpness-aware minimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12360–12370, 2022.

[43] 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, 2019.

[44] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[45] Pavlo Molchanov, Jimmy Hall, Hongxu Yin, Jan Kautz, Nicolo Fusi, and Arash Vahdat. Lana: latency aware network acceleration. In *European Conference on Computer Vision*, pages 137–156. Springer, 2022.

[46] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[47] 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.

[48] Clara Na, Sanket Vaibhav Mehta, and Emma Strubell. Train flat, then compress: Sharpness-aware minimization learns more compressible models. *arXiv preprint arXiv:2205.12694*, 2022.

[49] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.

[50] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[51] V. Sehwag, S. Wang, P. Mittal, and S. Jana. Hydra: Pruning adversarially robust neural networks. *NeurIPS*, 33:19655–19666, 2020.

[52] Maying Shen, Pavlo Molchanov, Hongxu Yin, and Jose M Alvarez. When to prune? a policy towards early structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12247–12256, 2022.

[53] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose Alvarez. Structural pruning via latency-saliency knapsack. In *Advances in Neural Information Processing Systems*, 2022.

[54] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. *Advances in neural information processing systems*, 30, 2017.

[55] David Stutz, Matthias Hein, and Bernt Schiele. Relating adversarially robust generalization to flat minima. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7807–7817, 2021.

[56] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[57] E. Tartaglione, A. Bragagnolo, A. Fiandrotti, and M. Grangetto. Loss-based sensitivity regularization: towards deep sparse neural networks. *NN*, 2022.

[58] Manoj-Rohit Vemparala, Nael Fasfous, Alexander Frickenstein, Sreetama Sarkar, Qi Zhao, Sabine Kuhn, Lukas Frickenstein, Anmol Singh, Christian Unger, Naveen-Shankar Nagaraja, et al. Adversarial robust model compression using in-train pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 66–75, 2021.

[59] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[60] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020.

[61] Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. *Advances in Neural Information Processing Systems*, 33:2958–2969, 2020.

[62] Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

[63] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

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

[65] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019.

[66] S. Yu, Z. Yao, A. Gholami, Z. Dong, S. Kim, MW Mahoney, and K. Keutzer. Hessian-aware pruning and optimal neural implant. In *WACV*, 2022.

[67] Qi Zhao and Christian Wressnegger. Holistic adversarially robust pruning. In *The Eleventh International Conference on Learning Representations*, 2022.

[68] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Advances in neural information processing systems*, 33:9865–9877, 2020.

Supplementary Material

6 Experiment Details

6.1 Datasets

We conduct image classification experiments on the ImageNet dataset [10]. To evaluate the generalization ability of pruned models, we also evaluate models on ImageNet-C [24] and ImageNet-V2 [49] datasets. The former consists of the validation images from the ImageNet dataset, but with nineteen types of corruptions applied with five different levels of severity. The latter is a dataset created in the same manner as the original dataset, but consisting of different images. Therefore, it is intended to measure out of distribution performance of models. This ImageNet-V2 dataset includes three different sets, each one with a slightly different sampling strategy. We focus our experiments on the `MatchedFrequencies` dataset, but include evaluation all three datasets in Table 3 and see that our method consistently outperforms the baseline. Results do not use ASAM.

We conduct object detection experiments on the Pascal VOC dataset [14]. We additionally create a Pascal VOC-C dataset, in which we apply the ImageNet-C corruptions to the Pascal VOC test set. We only use one severity level. We evaluate the model on the Pascal VOC test set as well as our VOC-C dataset. We report mean average precision (mAP) and a robustness ratio $R_{\mathrm{C}} = \mathrm{mAP}_{\mathrm{C}}/\mathrm{mAP}_{val}$.

6.2 Additional Robustness Datasets

As described above, we choose the ImageNet V2 Matched Frequencies dataset to perform our main set of experiments, but there are two other ImageNet V2 datasets: Threshold 0.7 and Top Images. These three datasets vary in terms of selection criteria for included images. Matched Frequencies attempts to match the image selection frequency of MTurk workers in the original ImageNet validation set. Threshold 0.7 samples from among images with a selection frequency of at least 0.7. Top Images includes the images with the highest selection frequency within each class. We provide the pruned model robustness comparison on these the additional ImageNet V2 datasets in Table 3.

6.3 Network Architectures

In our experiments we prune three different networks for the classification task: ResNet50, MobileNet V1 and MobileNet V2 [22, 30]. We use a pretrained model trained for 90 epochs with a cosine learning rate as in HALP [53] and EagleEye [39]. For object detection, we follow the experimental setup in HALP [53] to prune an SSD512 model with ResNet50 as the backbone. We perform Distributed Data Parallel training across 8 V100 GPUs with batch size 128 for all experiments.

6.4 Pruning Schedule.

Given a pre-trained model, for any architecture, we run the warm up for 10 epochs, and then we follow the same pruning schedule as in [9]: we prune every 30 iterations and, in each iteration, we prune away a p_r fraction of neurons so that the final network is pruned by a fraction p (resulting in a network of size $1 - p$). To determine the p_r fraction, we follow an exponential decay schedule. Let $k = 1 - p$ and let k_r be the number of neurons remaining after r pruning iterations, where the total number of pruning iterations is R. Let m be the number of neurons in the dense network, and define $\alpha = \frac{r}{R}$. Then, $k_r = \exp\{\alpha \log k + (1 - \alpha \log m)\}$. We fine-tune the pruned model for another 79 epochs (to reach 90 epochs total).

6.5 Optimization Hyperparameters

The base optimizer is SGD with cosine annealing learning rate with a linear warmup over 8 epochs, a largest learning rate of 1.024, momentum of 0.875, and weight decay $3.05e - 05$. Unless otherwise stated we use $\rho_{min} = 0.01$ and $\rho_{max} = 2.0$ for all experiments – we observe these values scale well across networks and tasks. For robustness encouragement we use $\rho = 2.0$ in line with prior work [36]. For some ablation experiments, the original SAM [15] optimizer is sufficient and in these cases we reduce $\rho_{max} = 0.1$ and use constant $\rho = 0.05$ for finetuning.

Table 3: Our AdaSAP method outperforms the Taylor pruning baseline on the additional ImageNet V2 datasets. Additionally, the AdaSAP model pruned to 76% outperforms the dense model across all three datasets. Results without ASAM.

Method	Size $\downarrow$	Matched Frequences	Threshold 0.7	Top Images
Dense	1	64.80	73.79	79.00
Taylor	0.20	60.56	69.74	75.53
AdaSAP$_P$	0.20	**62.03**	**70.83**	**76.62**
Taylor	0.40	63.51	72.54	77.83
AdaSAP$_P$	0.40	**64.62**	**73.75**	**78.87**
Taylor	0.76	64.53	73.32	78.72
AdaSAP$_P$	0.76	**66.00**	**74.70**	**79.66**

Algorithm 1 AdaSAP Pruning Procedure

```
Input: Pretrained model weights $\mathbf{w}$ partitioned into neurons $w_i$, pruning importance criteria $\phi$, $\rho$ bounds ($\rho_{\min}$, $\rho_{\max}$)
  Let train_iter = Algorithm 2
  for epoch in warmup_epochs do                                   ▷ Adaptive Weight Perturbation
      for each iteration do
          Sample training batch $b$
          $\mathbf{w}$ = train_iter ($\mathbf{w}$, $b$, $\rho_{\min}$, $\rho_{\max}$)
      end for
  end for
  for epoch in pruning_epochs do                                  ▷ Pruning
      for each iteration do
          Sample training batch $b$
          $\mathbf{w}$ = train_iter ($\mathbf{w}$, $b$, $\rho_{\min}$, $\rho_{\max}$)
          if iteration % prune_frequency = 0 then
              $\mathbf{s} = \phi(\mathbf{w})$                     ▷ Score all neurons
              idxs = rank($s$)[: prune_num]                       ▷ Neurons with lowest score
              $\mathbf{w}_{\texttt{idxs}} = 0$                    ▷ Prune neurons
          end if
      end for
  end for
  for epoch in finetune_epochs do                                 ▷ Robustness encouragement
      for each iteration do
          Sample training batch $b$
          $\mathbf{w}$ = train_iter ($\mathbf{w}$, $b$, $\rho_{\min} = \rho_{\max} =$ constant)   ▷ SAM
      end for
  end for
  return $\mathbf{w}$
Output: Pruned and finetuned model
```

Algorithm 2 AdaSAP Optimization Iteration

```
Input: model weights $\mathbf{w}$ partitioned into neurons $\mathbf{w}_i$, training batch $b$, $\rho$ bounds ($\rho_{\min}$, $\rho_{\max}$), loss $L$, score function $\psi$, learning rate $\eta$
  for each neuron $\mathbf{w}_i$ do
      $s_i = \psi(\mathbf{w}_i)$                                  ▷ Score
      Compute $\rho_i$ as in Eq. 3                                ▷ Determine perturbation ball size
      Compute $\hat{\epsilon}_i$ as in Eq. 6                      ▷ Optimal perturbation
      $g_i \approx \nabla_{\mathbf{w}_i} L_{b,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i+\hat{\epsilon}_i}$   ▷ Gradient approximation
  end for
  $\mathbf{w} = \mathbf{w} - \eta\mathbf{g}$
  return $\mathbf{w}$
```

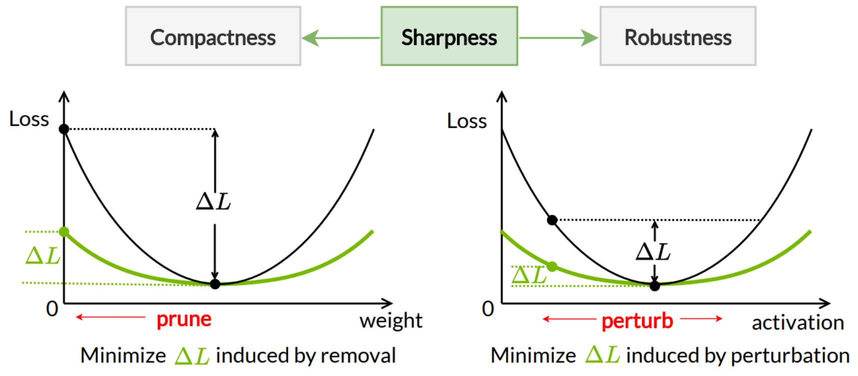

Figure 3: **(Left)** Before pruning, encourage neurons that will be pruned to lie within a flat minimum, since their removal will affect the loss less. **(Right)** After pruning, promote robustness by encouraging flatness across the network.

7 Full Derivation of Gradient Update

Setup. We derive the gradient update for training. Consider a network whose parameters are grouped into I partitions, where each partition i includes the set of weights $\mathbf{w}_i$. In our setting, since we consider channel-wise pruning, each channel constitutes a partition. For simplicity, throughout this paper we refer to a channel-wise group of weights as a *neuron*. We use $\mathbf{w}_i$ to refer to a single neuron and $\mathbf{w}$ to refer to the collection of all neurons $\{\mathbf{w}_i\}_{i=1}^{I}$.

Perturbation regions. Consider training loss $L_S(\mathbf{w})$ of the network over the training dataset S. In order to quantify our weight perturbation, we consider a set of perturbation ball radius sizes $\boldsymbol{\rho}$, with each ρ_i corresponding to the size of allowable perturbations for neuron $\mathbf{w}_i$. Given this notation, the goal of our warmup procedure is to optimize for a network in which unimportant neurons lie in flatter minima prior to pruning via the following objective

$$\min_{\mathbf{w}} \max_{\{\|T_{\mathbf{w}}^{-1}\epsilon_i\|\leq\rho_i\}_{i=1}^{I}} L_S(\mathbf{w}+\epsilon). \tag{2}$$

where $T_{\mathbf{w}}$ and its inverse $T_{\mathbf{w}}^{-1}$ are transformations that can be applied during optimization so as to reshape the perturbation region (*i.e.*, not necessarily a ball). This strategy was studied in [36] in order to allow for greater perturbation exploration and leads to improved performance. We determine the values ρ_i based on computing an importance score for each neuron, similar to a pruning criterion. Consider a neuron importance score function $\psi(\cdot)$, such as the ℓ_2 norm of the neuron weight, the Taylor importance score [46], or other scores that are some function of weights, gradients, and higher order terms [38, 52, 59, 66]. Given the score for each neuron as $s_i = \psi(\mathbf{w}_i)$, we can now compute the perturbation ball size ρ_i that maps the score for a given neuron to be within the desired range of perturbation sizes

$$\rho_i = \rho_{\max} - \frac{s_i - s_{\min}}{s_{\max} - s_{\min}}(\rho_{\max} - \rho_{\min}), \tag{3}$$

Scores $s_{\min}$ and $s_{\max}$ are obtained empirically at each gradient step, but $\rho_{\max}$ and $\rho_{\min}$ are global algorithm hyperparameters that are set ahead of time and we find them to scale across various experiment settings. Due to the progress in pruning literature, neuron importance estimates $\psi(\cdot)$ can be evaluated during training with little additional computational overhead.

Optimal gradient update. Our enhanced gradient update optimizes for finding a minimum that also lies in a region of low loss. Beginning with our objective in Eq. 2, we perform a series of approximations in order to find the most adversarial perturbations $\boldsymbol{\epsilon} = \{\epsilon_i\}$ (*i.e.*, highest loss near the minimum) which we can apply to the neurons $\mathbf{w} = \{\mathbf{w}_i\}_{i=1}^{I}$ in our network in order to obtain a flatness-informed gradient update.

We use a first order Taylor expansion around each neuron i with respect to $\boldsymbol{\epsilon}$ around 0 (in line with [15]) to approximate the inner maximization in Eq. 2. This gives us $\boldsymbol{\epsilon}^*$, the optimal $\boldsymbol{\epsilon}$ we can use to inform our gradient update. Define $\tilde{\epsilon}_i = T_{\mathbf{w}}^{-1}\epsilon_i$. For simplicity we denote the conditional loss due to weight perturbations as the function $L_{S,\mathbf{w}}(\cdot)$. Then we derive our optimal update as follows for each neuron:

$$\begin{aligned}\tilde{\epsilon}_i^* &=_{\|\tilde{\epsilon}_i\|_2 \leq \rho_i} \; L_{S,\mathbf{w}}(\mathbf{w}_i + T_{\mathbf{w}}\tilde{\epsilon}_i), \\ &\approx_{\|\tilde{\epsilon}_i\|_2 \leq \rho_i} \; L_{S,\mathbf{w}}(\mathbf{w}_i) + \tilde{\epsilon}_i^{\top} T_{\mathbf{w}} \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i), \\ &=_{\|\tilde{\epsilon}_i\|_2 \leq \rho_i} \; \tilde{\epsilon}_i^{\top} T_{\mathbf{w}} \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i),\end{aligned} \tag{4}$$

$\| \cdot \|_2$ being the ℓ_2 norm. We can then approximate $\tilde{\boldsymbol{\epsilon}}^*$ by rescaling the gradient associated with each neuron so that its norm is ρ_i through

$$\tilde{\epsilon}_i^* = \rho_i \frac{\mathrm{sign}(\nabla_{w_i} L_{S,\mathbf{w}}(\mathbf{w}_i))|T_{\mathbf{w}} \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|}{(\|T_{\mathbf{w}} \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)\|_2^2)^{1/2}}. \tag{5}$$

Since we were originally approximating $\tilde{\epsilon}_i = T_{\mathbf{w}}^{-1}\epsilon_i$, we recover our approximation of ϵ^* as

$$\hat{\epsilon}_i = \rho_i \frac{T_{\mathbf{w}}^2 \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)}{\|\nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)\|_2}. \tag{6}$$

Approximating the optimal gradient update. We then find an approximation to our desired gradient. Referring to Eq. 2, we can find the gradient of the inner maximization in order to derive a sharpness-aware gradient update. We use the inner maximization and plug in our approximation $\hat{\epsilon}$ for each individual neuron during adjacent pruning steps, defined as

$$\begin{aligned}\nabla_{\mathbf{w}_i} \max_{\|T_{\mathbf{w}}^{-1}\epsilon_i\|_2 \leq \rho_i} L_{S,\mathbf{w}}(\mathbf{w}_i + \epsilon_i) &\approx \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i + \hat{\epsilon}_i) \\ &= \frac{d(\mathbf{w}_i + \hat{\epsilon}_i)}{d\mathbf{w}_i} \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i + \hat{\epsilon}_i} \\ &= \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i + \hat{\epsilon}_i} + \frac{d\hat{\epsilon}_i}{d\mathbf{w}_i} \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i + \hat{\epsilon}_i} \\ &\approx \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i + \hat{\epsilon}_i},\end{aligned} \tag{7}$$

where we drop the second order terms for efficiency. This leads us to our final gradient update approximation between adjacent pruning steps as $\nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i) \approx \nabla_{\mathbf{w}_i} L_{S,\mathbf{w}}(\mathbf{w}_i)|_{\mathbf{w}_i + \hat{\epsilon}_i}$.

We use this gradient update in place of the standard gradient update during the warmup phase of our pruning procedure before neuron removal. This sets up our network so that when pruning begins, we have a model that is closer to a flat minima. Optimization details are in Algorithm 2.

8 Additional Results

8.1 MobileNet Results

We include results on MobileNet V1 and MobileNet V2 in Tables 4 and 5. These parallel our main results in the main paper, in which we evaluated AdaSAP$_P$ and AdaSAP$_L$ on ResNet50 and compared them against other pruning methods. Here, we perform a similar comparison, where Table 4 includes our results on AdaSAP$_P$ and Table 5 includes our results on AdaSAP$_L$. We compare against Taylor importance [46], EagleEye [39], and PolarReg [68] for AdaSAP$_P$ and against HALP [53], SMCP [31], MetaPruning [43], AutoSlim [64], AMC [23], and EagleEye [39] for AdaSAP$_L$. Results here do not use ASAM. We can observe that AdaSAP performs strongly compared to baselines, particularly in the parameter-based setting.

8.2 Object Detection Results

As summarized in the main paper, AdaSAP improves performance on object detection in addition to image classification. Results are in Table 6.

8.3 Margin of Improvement in Classification

Figure 4 shows the margin of performance on various corruption types for the classification task. AdaSAP outperforms a Taylor pruned model on all corruptions, across three different sparsities. We can see that it tends to particularly improve performance on several corruptions that may be important for the autonomous driving application, such as pixelated images, fog, and snow.

Table 4: **MobileNet-V1/V2 - Parameters.** Top1 Accuracy as a function of the pruning ratio.

Method	Size ↓	Val	R_{V2}	R_C	IN-V2	IN-C
			MobileNet-V1			
Dense	1	72.63	0.82	0.45	59.30	32.79
Taylor	0.40	69.61	0.80	0.42	55.90	29.51
AdaSAP$_P$	**0.39**	**71.05**	**0.82**	**0.44**	**58.08**	**31.06**
Taylor	0.52	71.21	0.81	0.43	57.65	30.92
AdaSAP$_P$	**0.52**	**71.58**	**0.82**	**0.44**	**58.75**	**31.77**
EagleEye	0.56	70.86	0.80	0.42	56.88	29.98
AdaSAP$_P$	**0.56**	**72.05**	**0.82**	**0.45**	**58.94**	**32.24**
			MobileNet-V2			
Dense	1	72.10	0.81	0.45	58.50	32.40
Taylor	0.72	70.49	0.81	0.43	56.87	30.22
AdaSAP$_P$	**0.72**	**72.06**	**0.82**	**0.45**	**58.73**	**32.63**
PolarReg	0.87	71.72	0.82	0.45	58.46	32.04
Taylor	0.88	71.93	0.82	0.45	58.75	32.21
AdaSAP$_P$	0.88	**72.34**	**0.82**	**0.45**	**59.28**	**32.80**

Table 5: **MobileNet-V1/V2 - Latency.** Top1 accuracy for latency constrained pruning for various speedup ratios. "–" indicates that we could not evaluate the model due to unavailable code or models.

Method	Speedup ↑	Val	R_{V2}	R_C	IN-V2	IN-C
			MobileNet-V1			
Dense	1	72.63	0.82	0.45	59.30	32.79
MetaPruning	2.06	66.1	–	–	–	–
AutoSlim	2.27	67.9	–	–	–	–
HALP	2.32	68.30	0.80	0.41	54.95	28.15
SMCP	**2.39**	68.34	0.80	**0.42**	54.38	**28.68**
AdaSAP$_L$	2.33	**68.45**	**0.81**	0.41	**55.42**	28.29
0.75 MobileNetV1	1.37	68.4	–	–	–	–
AMC	1.42	70.5	–	–	–	–
MetaPruning	1.42	70.9	–	–	–	–
EagleEye	1.47	70.86	0.80	0.42	56.88	29.98
HALP	1.68	71.31	0.81	0.43	57.38	30.77
SMCP	**1.72**	71.00	0.81	0.44	57.20	31.02
AdaSAP$_L$	1.70	**71.48**	**0.82**	**0.44**	**58.23**	**31.35**
			MobileNet-V2			
Dense	1	72.10	0.81	0.45	58.50	32.40
HALP	**1.84**	70.42	0.81	0.45	57.21	31.69
AdaSAP$_L$	1.81	**71.35**	0.81	**0.46**	**57.85**	**32.63**
HALP	1.33	72.16	0.81	0.46	58.53	**33.04**
AdaSAP$_L$	**1.39**	**72.19**	**0.82**	0.46	**59.36**	32.91

8.4 Margin of Improvement in Object Detection

Similarly to our result on classification, we include margins of performance improvement on various corruption types for the object detection task in Figure 5.

Table 6: **Object Detection.** mAP on validation images and ImageNetC style corruptions on the Pascal VOC dataset.

Method	Size ↓	Val	Corrupted	R_C
HALP	0.4	0.774	0.583	0.753
AdaSAP	0.4	**0.795**	**0.620**	**0.780**
HALP	0.2	0.770	0.580	0.753
AdaSAP	0.2	**0.793**	**0.616**	**0.776**

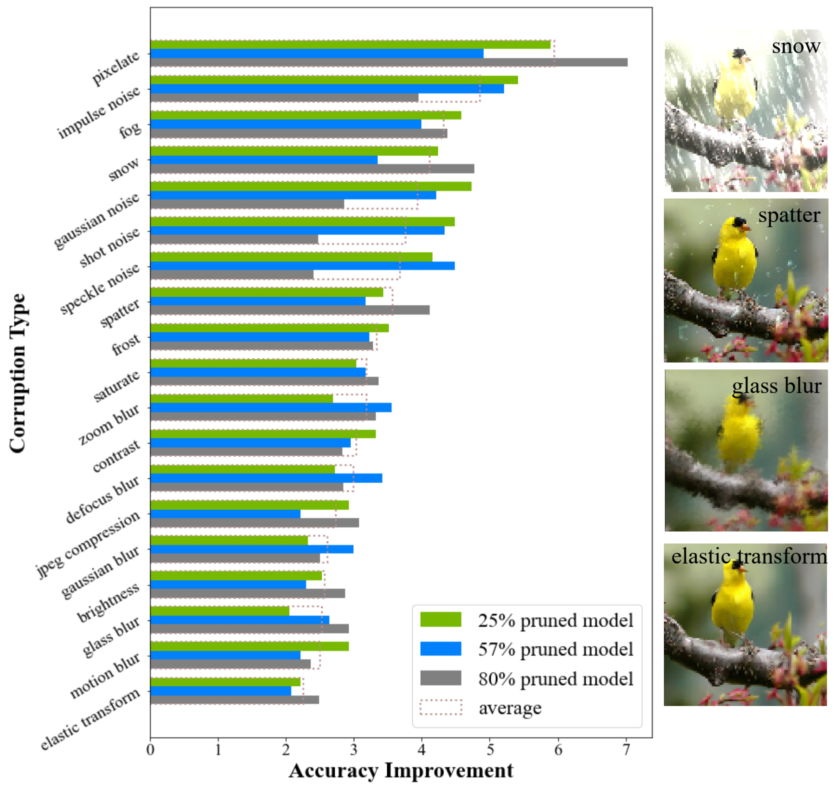

Figure 4: Performance difference on various ImageNet C corruption types on models of varying sparsity. Accuracy improvement is the Top1 accuracy on a model trained with AdaSAP minus that of a Taylor pruned model.

8.5 Relative Robustness

In Figure 6 we show that AdaSAP outperforms baselines on each of the constituent elements of the relative robustness metric. Recall that relative robustness is robust accuracy divided by standard validation accuracy. In addition to outperforming baselines on relative robustness, AdaSAP also outperforms on ImageNet validation accuracy and ImageNet C robust accuracy.

8.6 Sharpness

Consider the sharpness defined in [15] as $\max_{\|\epsilon\|_2 \leq \rho} L_S(\mathbf{w} + \epsilon) - L_S(\mathbf{w})$ This measures the maximum amount that the loss could change if the weights are perturbed within a ball of radius ρ. We measure sharpness directly before and after pruning, to evaluate how sharpness prepares the network to be optimally pruned, as well as how pruning affects the model sharpness. Table 7 shows that using AdaSAP leads to flatter models both before and after pruning. In line with previous results that found an association between flatness and generalization, this result can help explain why our models have

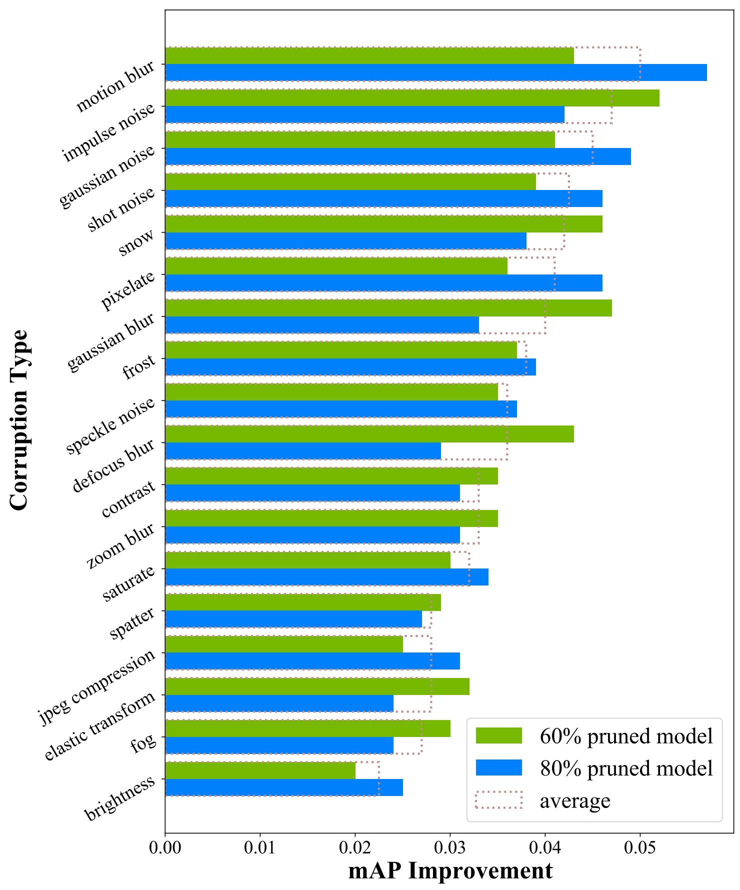

Figure 5: **Pascal VOC-C dataset.** Performance improvement per corruption as the difference between a model trained using AdaSAP minus that of a HALP pruned model.

Table 7: **Sharpness before and after pruning without finetuning**. Lower sharpness values indicates a flatter loss landscape. AdaSAP achieves flatter minima both directly before and after pruning, before any finetuning.

Method	Size ↓	Sharpness pre pruning	Sharpness post pruning
Taylor	0.423	0.039	0.044
AdaSAP$_P$	**0.403**	**0.037**	**0.039**
Taylor	0.760	0.039	0.041
AdaSAP$_P$	**0.759**	**0.037**	**0.038**

better generalization and robust generalization performance. See Figure 3 for a visual explanation of the benefits of sharpness in both pruning and robustness.

8.7 Confidence Intervals

We include confidence intervals over three repeats in Table 8. AdaSAP does lead to significant performance improvement over other methods.

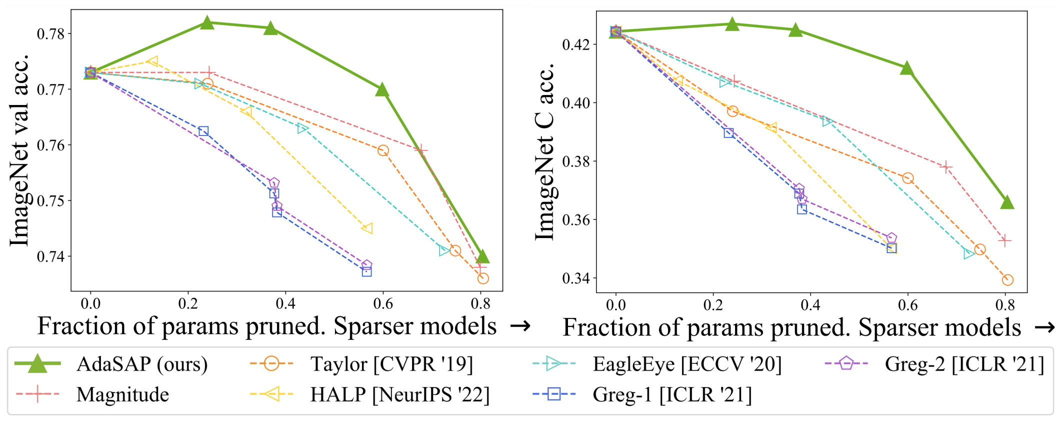

Figure 6: ImageNet Validation and ImageNet C performance on AdaSAP vs. baselines. Contains the same data as used to produce Figure 1 but demonstrates that AdaSAP additionally dominates baselines on both components of the relative robustness metric.

Table 8: **Confidence intervals over three repeats.** EagleEye checkpoints are obtained from the official repository with only one seed.

Method	Size ↓	Val	IN-V2	IN-C
Magnitude	0.20	73.71 ± 0.12	61.21 ± 0.17	35.33 ± 0.18
Taylor	0.20	73.42 ± 0.19	60.36 ± 0.19	33.97 ± 0.12
EagleEye	0.27	74.13	61.30	34.84
AdaSAP$_P$	0.20	$\mathbf{74.54 \pm 0.09}$	$\mathbf{62.21 \pm 0.13}$	$\mathbf{37.30 \pm 0.19}$
Magnitude	0.76	77.32 ± 0.06	65.18 ± 0.27	40.64 ± 0.20
Taylor	0.76	77.05	64.53	39.68
EagleEye	0.78	77.07	64.84	40.67
AdaSAP$_P$	0.77	$\mathbf{78.23 \pm 0.06}$	$\mathbf{65.98 \pm 0.32}$	$\mathbf{43.43 \pm 0.20}$

9 Additional Ablations

9.1 Performance change after pruning

In Table 9 we show how the loss and accuracy change over the course of pruning. Our hypothesis is that our method sets up the network for better pruning, so that the performance drop over the course of pruning is minimized. In most cases, our method has the best validation loss and accuracy both before and after pruning. This indicates that our method sets up the model to be pruned well, and also preserves performance well throughout the pruning process.

9.2 Sensitivity to Pruning Criteria

Table 10 shows that AdaSAP can also be used with other pruning criteria, such as Taylor importance. Although ℓ_2 norm magnitude (AdaSAP$_P$) performs the best, AdaSAP with Taylor pruning still outperforms Taylor pruning with SGD optimization.

9.3 Varying the length of the adaptive weight perturbation step.

Throughout the main set of experiments we set to 10 the number of epochs used for the adaptive weight perturbation step. This delineates how long we use the AdaSAP optimizer for, as well as how soon into the procedure we begin pruning. In this experiment, we analyze the sensitivity of the approach to this parameter. We report results for this experiment in Table 11. We can observe that as we make this period longer, validation accuracy and ImageNet C accuracy both drop slightly, while

Table 9: Validation Loss and Accuracy directly before and after pruning (before additional fine-tuning). Across several pruning levels, our method generally reaches the lowest validation loss and accuracy both before and after pruning. Results do not use ASAM.

Method	Size ↓	Val Loss Before	Val Loss After	Val Acc Before	Val Acc After
Taylor	0.2	**2.245**	3.315	**67.798**	42.915
Mag	0.2	2.416	3.288	63.843	43.21
SAM	0.2	2.255	3.2	67.577	45.802
AdaSAP$_P$	0.2	2.293	**3.146**	66.555	**46.313**
Taylor	0.63	2.284	2.69	66.67	57.046
Mag	0.63	2.408	2.587	63.844	59.275
SAM	0.63	2.327	2.676	65.541	57.449
AdaSAP$_P$	0.63	**2.251**	**2.574**	**67.473**	**59.962**
Taylor	0.76	2.441	2.42	63.389	63.711
Mag	0.76	2.292	2.406	66.714	64.087
SAM	0.76	2.401	2.379	64.019	64.589
AdaSAP$_P$	0.76	**2.213**	**2.347**	**68.463**	**65.393**

Table 10: **Sensitivity to pruning criteria.** AdaSAP performs best when combined with magnitude pruning, but is flexible enough to be used with other criteria, such as Taylor importance. Here we see that AdaSAP with Taylor pruning matches or outperforms SGD with Taylor pruning. Results do not use ASAM.

Method	Size ↓	Val	R_{V2}	R_{C}	IN-V2	IN-C
Taylor + SGD	0.42	75.85	0.84	0.50	63.51	37.84
AdaSAP$_{P,\text{Taylor}}$	0.43	76.26	0.84	0.50	63.77	38.07
AdaSAP$_P$	**0.41**	**76.93**	**0.84**	**0.52**	**64.49**	**39.64**
Taylor + SGD	0.76	77.05	0.84	0.52	64.53	39.68
AdaSAP$_{P,\text{Taylor}}$	0.76	77.42	0.84	0.52	65.24	40.27
AdaSAP$_P$	**0.76**	**77.86**	**0.85**	**0.53**	**66.00**	**41.30**

Table 11: **Effects of varying number of epochs of adaptive weight perturbation.** Increasing the number of epochs leads to smaller models but slightly worse validation and ImageNet C performance.

Num epochs	Size ↓	Val	R_{V2}	R_{C}	IN-V2	IN-C
5	0.48	73.89	0.83	0.48	61.29	35.65
10	0.46	73.82	0.83	0.48	60.91	35.59
20	0.44	73.63	0.84	0.48	61.48	35.25
30	0.43	73.47	0.83	0.48	60.72	35.04

Table 12: Comparison of various weight perturbation strategies during robustness encouragement.

Perturbation Type	Val	IN-C	IN-V2
No weight perturbations	73.98	35.84	62.00
Adaptive weight perturbations	74.10	35.72	61.94
Uniform weight perturbations	**74.39**	**35.86**	**62.03**

ImageNet V2 seems to have no discernible pattern. Ratios R_C and R_{V2} also stay consistent across the experiment.

Table 13: **Comparison of AdaSAP to SAM optimizer.** AdaSAP outperforms SAM and SGD on standard validation performance and ImageNet-C performance, but slightly trails SAM on ImageNet-V2 when both methods are augmented with ASAM.

Method	Size ↓	Val	R_{V2}	R_{C}	IN-V2	IN-C
		SGD				
Dense	1	77.32	0.84	0.54	64.79	42.46
Taylor + SGD	0.20	73.56	0.82	0.46	60.56	33.93
		AdaSAP vs. SAM (without ASAM)				
Taylor + SAM	0.20	73.62	0.83	0.47	61.37	34.49
AdaSAP$_P$	0.20	$\mathbf{74.38}$	$\mathbf{0.83}$	$\mathbf{0.48}$	$\mathbf{62.03}$	$\mathbf{35.86}$
Taylor + SGD	0.42	75.85	0.84	0.50	63.51	37.84
Taylor + SAM	0.43	76.27	0.84	0.50	63.81	37.72
AdaSAP$_P$	$\mathbf{0.40}$	$\mathbf{77.03}$	$\mathbf{0.84}$	$\mathbf{0.51}$	$\mathbf{64.62}$	$\mathbf{39.57}$
Taylor + SGD	0.76	77.05	0.84	0.52	64.53	39.68
Taylor + SAM	0.76	77.29	0.84	0.52	64.84	40.07
AdaSAP$_P$	0.76	$\mathbf{77.86}$	$\mathbf{0.85}$	$\mathbf{0.53}$	$\mathbf{66.00}$	$\mathbf{41.30}$
		AdaSAP vs. SAM (with ASAM)				
SAM + ASAM	0.19	73.93	0.84	0.50	61.76	36.66
AdaSAP$_P$ **+ ASAM**	0.19	$\mathbf{74.63}$	0.83	0.50	$\mathbf{62.08}$	$\mathbf{37.30}$
SAM + ASAM	0.41	77.10	0.84	0.53	64.94	40.90
AdaSAP$_P$ **+ ASAM**	$\mathbf{0.40}$	$\mathbf{77.27}$	0.83	0.53	64.51	$\mathbf{41.23}$

9.4 Robustness Encouragement

As mentioned in the main text, we consider various ablations to determine the necessity of various steps of the AdaSAP procedure. In the third step of our procedure, robustness encouragement, we choose to apply uniform perturbations across all weights in the network. This differs from the first step, in which we apply adaptive weight perturbations. In Table 12, we examine the effects of different weight perturbation strategies during the robustness encouragement phase. We can see that while all three strategies lead to relatively close final performance across the three datasets, uniform weight perturbations perform slightly better, suggesting that our choice of applying them in our procedure may be slightly benefitting the performance.

9.5 Importance of Adaptive Weight Perturbations

In Table 13 we extend an ablation from the main paper in which we compare AdaSAP to SAM, effectively evaluating the importance of warmup with adaptive weight perturbations. Here, we perform the comparison on a wider range of sparsities and observe that a similar pattern emerges.