GSparsity: Unifying Network Pruning and Neural Architecture Search by Group Sparsity

Avraam Chatzimichailidis[1,2,3] **Arber Zela**[4] **Janis Keuper**[1,5] **Yang Yang**[1]

[1]Department of High Performance Computing, Fraunhofer ITWM, Germany
[2]Department for Scientific Computing, TU Kaiserslautern, Germany
[3]Fraunhofer Center Machine Learning, Germany
[4]Department of Computer Science, University of Freiburg, Germany
[5]Institute for Machine Learning and Analytics, Offenburg University, Germany

Abstract In this paper, we propose a unified approach for network pruning and one-shot neural architecture search (NAS) via group sparsity. We first show that group sparsity via the recent Proximal Stochastic Gradient Descent (ProxSGD) algorithm achieves new state-of-the-art results for filter pruning. Then, we extend this approach to *operation pruning*, directly yielding a gradient-based NAS method based on group sparsity. Compared to existing gradient-based algorithms such as DARTS, the advantages of this new group sparsity approach are threefold. Firstly, instead of a costly bilevel optimization problem, we formulate the NAS problem as a single-level optimization problem, which can be optimally and efficiently solved using ProxSGD with convergence guarantees. Secondly, due to the operation-level sparsity, discretizing the network architecture by pruning less important operations can be safely done without *any* performance degradation. Thirdly, the proposed approach finds architectures that are both stable and performant on a variety of search spaces and datasets.

1 Introduction

Network pruning (see, e.g., [3] for an overview) and neural architecture search (NAS; see, e.g., [12] for an overview) are important subfields of deep learning, but so far they have evolved largely separately (a few exceptions are [41; 9]). In this work, we propose to unify these two fields via the framework of group sparsity, allowing us to use the latest advances in network pruning directly for gradient-based one-shot NAS with provable guarantees.

Key to the proposed framework is that the one-shot NAS approach [4; 2; 28] uses a large supernet in which all candidate operations are assumed to be active. Identifying a small subnet as the final architecture for inference could be formulated as *pruning away the rest of the supernet*.

The group sparsity approach allows us to prune on the level of entire *filters* or even entire *operations*, as well as to use network pruning for NAS in our *Group Sparsity (GSparsity)* approach.
Contributions. Our overarching contribution is a unified approach, based on the concept of group sparsity, for both network pruning and NAS:

- Group sparsity via the recent convergent Proximal Stochastic Gradient Descent (ProxSGD; [36]) algorithm achieves better results for filter pruning (than previous heuristic proximal algorithms).
- We extend the group sparsity approach to allow pruning entire operations by grouping all trainable parameters of each operation.
- We show that this approach renders the architecture parameters typical of most one-shot methods superfluous, casting the NAS problem as a standard single-level optimization problem, which can be solved optimally by the convergent ProxSGD algorithm.
- We show that GSparsity converges to a group-sparse solution, where the weights of non-important groups are zero. As a result, while previous methods suffer from substantial performance degradation in the discretization step, our approach avoids *any* such performance degradation.

2 Background and Related Work

Network pruning. At a fundamental level, it is usually possible to prune some weights while the network's performance still remains the same, leading to a sparse neural network [13], though is that it is not known in advance which weights could be pruned. Kernel/channel pruning is meant to bring a more structured sparsity pattern. It amounts to pruning all weights in the same kernel/channel in a convolution layer simultaneously and popular approaches are based on the concept of group sparsity, see, e.g., [32; 26; 16; 19; 23].

As group sparsity is based on the nonsmooth $L_{2,1}$-norm regularization, the resulting training problem is nonsmooth and cannot be solved by standard stochastic gradient descent (SGD) algorithm. In this paper, we apply the ProxSGD algorithm to solve the nonsmooth optimization problem efficiently and optimally and the performance is better than previous methods, which either rely on stochastic subgradient descent algorithms that converge very slowly to a sparse solution (see [36]), or use heuristic proximal-type algorithms [37; 42; 1] which do not have a guaranteed convergence.

Neural architecture search. As an algorithmic solution to designing new architectures, NAS has been developed to automate the architecture search process. The one-shot approach of NAS [4; 2; 28] based on weight sharing enables us to search on a large supernet and find a promising small subnet.

The DARTS approach in [25] formulates the one-shot NAS problem as a differentiable optimization problem that can be solved by stochastic algorithms. However, several limitations are still prevalent, including performance degradation after discretizing the architecture parameters [39; 40; 31]. Secondly, by framing the problem as a bilevel optimization problem [7], several approximations have to be defined, without any convergence guarantees taken into account. Lastly, as shown in [39] the behaviour of DARTS is not robust across search spaces.

3 The Proposed Group Sparsity Approach

In this section, we present the unified group-sparsity approach for both network pruning and NAS.

Let the vector $\boldsymbol{w}$ consist of the trainable parameters of the neural network. It is decomposed into subvectors $\boldsymbol{w} = (\boldsymbol{w}_k)_{k=1}^K$ such that $\boldsymbol{w}_k$ represents a group of weights (such as all weights in the same filter or convolution layer), and K denotes the total number of (non-overlapping) groups. We formulate the network training as an optimization problem, which aims at minimizing the loss function f augmented by the group sparsity regularization on $\boldsymbol{w}$:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \frac{1}{|\mathbb{X}|} \sum_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{w}, \boldsymbol{x}) + \sum_{k=1}^{K} \mu_k \left\| \boldsymbol{w}_k \right\|_2 , \tag{1}$$

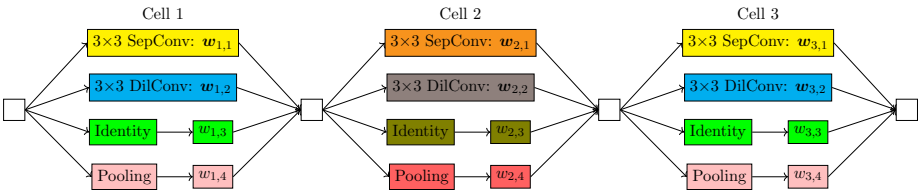
where $\boldsymbol{x}$ is a training example from the training dataset $\mathbb{X}$ (with $|\mathbb{X}|$ denoting the number of training examples). Besides, The L_2 norm defined as $\|\boldsymbol{w}\|_2 \triangleq \sqrt{\boldsymbol{w}^T \boldsymbol{w}}$ is a nonsmooth convex function.

The regularization $\sum_{k=1}^K \mu_k \|\boldsymbol{w}_k\|_2$ in (1) is usually referred to as the $L_{2,1}$ norm (or the mixed norm), and it is a natural generalization of L_1 norm when $\boldsymbol{w}_k$ is a vector. The advantage of the $L_{2,1}$-norm regularization is that it can promote a group-sparse neural network, in the sense that most groups will be zero and hence can be removed from the neural network without incurring any performance loss. Group sparsity is a structured sparsity that can be better exploited by hardware.

Since the loss function f is nonconvex and the $L_{2,1}$-norm regularization is nonsmooth, the training problem (1) is a nonsmooth nonconvex optimization problem. We adopt the ProxSGD algorithm proposed in [36], as it converges fast and provably to a stationary point. A description of the ProxSGD algorithm is given in Appendix A.

3.1 From Network Pruning to NAS

Depending on the specific task at hand, a group could be a kernel or filter inside a convolution layer. It could also be an operation (or a concatenation of several suboperations): an operation with



(a) Supernet in `Search`: Cell 1 and Cell 3 are of the same type and Cell 2 is of a different type (Operations in the same color shall be preserved or removed simultaneously.)

$$f(\boldsymbol{w}) + \mu \left(\left\| \begin{bmatrix} \boldsymbol{w}_{1,1} \\ \boldsymbol{w}_{3,1} \end{bmatrix} \right\|_2 + \left\| \begin{bmatrix} \boldsymbol{w}_{1,2} \\ \boldsymbol{w}_{3,2} \end{bmatrix} \right\|_2 + \left\| \begin{bmatrix} w_{1,3} \\ w_{3,3} \end{bmatrix} \right\|_2 + \left\| \begin{bmatrix} w_{1,4} \\ w_{3,4} \end{bmatrix} \right\|_2 + \|\boldsymbol{w}_{2,1}\|_2 + \|\boldsymbol{w}_{2,2}\|_2 + \|w_{2,3}\|_2 + \|w_{2,4}\|_2 \right)$$

(b) Definition of groups in `Search` (The regularization gains $\{\mu_k\}$ are assumed to be identical for all groups.)

Figure 1: Illustrative example of applying the group sparsity approach to NAS

trainable parameters can be removed if all of its parameters are 0. For operations without trainable parameters an additional scaling factor is added to the output of that operation For example, the 3×3 SepConv and Identity in Figure 1(a) can be removed if $\boldsymbol{w}_{1,1} = \boldsymbol{0}$ and $w_{1,3} = 0$, respectively.

Except for operations with no trainable parameters, the proposed formulation (1) does *not* need the architecture parameter. Its implications are twofold. Firstly, (1) is a single-level optimization problem and it is much easier to solve than the otherwise costly bilevel optimization problem. Secondly, it is no longer necessary to split the training dataset into two parts, one used to update the architecture parameters and the other for the network weights. The group sparsity regularization in (1) will also alleviate overfitting from which bilevel optimization may suffer.

4 Experiments

In this section, we perform extensive experiments to demonstrate the performance of the proposed algorithm compared to state-of-the-art algorithms. All experiments are performed on a single node using a GTX 1080 Ti GPU with 11 GB of memory. Our code is available at `https://github.com/cc-hpc-itwm/GSparsity`, along with the logs of all runs (of both our method and others).

4.1 Filter Pruning

We perform filter pruning for ResNet-50 [14] on ImageNet 2012. To this end, we put all parameters of the same filter into one group. This allows pruning of individual filters inside different convolutions. The experimental setup can be found in Appendix B.

algorithm	top-1 acc	MACs	code available?
baseline (reported)	76.13	100% (4.12G)	Y
SSS [17] (reported)	74.18	68.55%	Y (MXNet)
ThiNet-70 [27] (reported)	72.04	63.21%	Y (Caffe)
GSparsity (ours, $\mu = 0.02$)	75.21	63.11% (2.60G)	Y
GSparsity (ours, $\mu = 0.05$)	74.33	50.00% (2.06G)	Y
FPGM [15] (reported/reproduced)	74.83/69.69	46.50%/49.03% (2.02G)	Y
Hinge [22](reported)	74.70	46.55%	N
RRBP [43] (reported)	73.00	45.45%	N
ResRep [8] (reproduced)	0.10	45.15% (1.86G)	Y
GAL [21] (reported)	72.80	44.98%	N
GSparsity (ours, $\mu = 0.07$)	74.00	44.42% (1.83G)	Y
GSparsity (ours, $\mu = 0.1$)	73.34	42.23% (1.74G)	Y

Table 1: Filter pruning of ResNet-50 on ImageNet 2012.

Results. The performance of the baseline (unpruned) ResNet-50 and various filter pruning methods is summarized in Table 1. ResNet50 has 25.56M parameters and 4.12GMACs, and the pretrained network achieves an accuracy of 76.13%.

4.2 Operation Pruning

In this experiment, we conduct operation pruning by the proposed GSparsity algorithm, where each group consists of all parameters of the same operation. The experimental setup used for operation pruning is detailed in Appendix D.

Results. We see from Table 2 that when operations are pruned away such that 38.40% of weights are pruned, the network's retrained accuracy (97.45%) is almost identical to the unpruned baseline (97.50%). When 60.46% of weights are pruned, the retrained accuracy is 97.09%, i.e., 0.41% worse than the unpruned baseline. We also show in Table 2 the accuracy before and after the operations (with an L_2 norm smaller than a given threshold, namely, 1e-6, 1e-3 or 0.5) are pruned. We readily see that the proposed GSparsity approach does not incur any discretization error, even when the pruning threshold is only modestly small (such as 1e-3). Comparing the accuracies before and after retraining, we see that retraining can further enhance the performance of the pruned network.

μ	accuracy before pruning	accuracy after pruning			accuracy after retraining		inference time
		1e-6	1e-3	0.5	accuracy	parameters	
0	97.50	-	-	-	-	100%	7.13s
0.0001	96.50	96.50	96.50	92.08	97.45	78.25%	6.73s
0.0002	96.46	96.46	96.46	75.50	97.44	61.60%	6.40s
0.0005	96.36	96.36	96.36	13.34	97.32	52.09%	6.24s
0.002	96.47	96.47	96.47	10	97.09	39.54%	4.96s
0.004	96.48	96.48	96.48	10	96.84	32.80%	4.43s

Table 2: Operation pruning: The accuracy before/after operation pruning (but before retraining, with pruning thresholds 1e-6, 1e-3 and 0.5) and after retraining (with pruning threshold 1e-6).

4.3 Neural Architecture Search

In this subsection, we conduct NAS by GSparsity and compare it with state-of-the-art differentiable algorithms. We follow the NAS best practice checklist [24] and we refer the reader to the appendix for further information on the hyperparameters used during the experiments. To study the robustness of our method and various baselines, we run each method 3 times; we then evaluate each of the 3 resulting architectures 3 times and report means and standard deviations over the 9 results.

4.3.1 DARTS Search Space.

Search and evaluation settings. The `Search` is carried out on a small supernet with 16 initial channels and consists of 8 stacked cells, which is trained for 50 epochs on the full training set using ProxSGD. We use similar settings as DARTS [25] for `Evaluation`: 36 initial channels and SGD with momentum (now *without* the $L_{2,1}$-norm regularization) for 600 epochs. We stack 14 cells so that the network size is comparable to other methods.

Results on CIFAR-10 and CIFAR-100. The comparison between the proposed GSparsity and recent NAS algorithms is outlined in Table 3.[1] We see that the GSparsity approach has the highest average accuracy with low standard deviation. We conclude that the performance of GSparsity is both good and stable. Similar observations are drawn from experiments on CIFAR-100 [18]. The GSparsity method achieves the highest average accuracy of all methods, see Appendix for more details.

We also compare to the approach where scaling factors are appended to the operations and then pruned (instead of weights). It turns out that directly pruning weights yields better results. We refer the reader to the appendix for details, as well as for results on ImageNet-2012 and NAS-Bench-201.

[1] It would be interesting to compare with HAPG [33], but the authors' implementation is not available yet.

	CIFAR-10		CIFAR-100	
	accuracy	search cost	accuracy	search cost
DARTS (2nd)	96.98 ± 0.13	1.46 days	73.40 ± 7.79	1.33 days
P-DARTS	97.05 ± 0.20	0.25 day	**83.46 ± 0.24**	0.34 day
PC-DARTS	**97.13 ± 0.16**	0.13 day	82.57 ± 0.71	0.15 day
DrNAS	96.95 ± 0.08	0.83 day	83.15 ± 0.23	0.90 day
GDAS [10]	96.63 ± 0.12	0.18 day	80.99 ± 0.34	0.36 day
ISTA-NAS [35]	96.64 ± 0.15	0.03 day	82.25 ± 0.77	0.03 day
GAEA [20]	96.12 ± 0.29	0.22 day	79.10 ± 0.89	0.22 day
GSparsity (ours)	**97.17 ± 0.11**	0.42 day	**83.56 ± 0.34**	0.78 day

Table 3: NAS on DARTS search space for CIFAR-10/-100. All results are reproduced from their authors' implementations.

4.3.2 Robustness of GSparsity. Experiments in [39] show that DARTS performs poorly on different search spaces that only allow a subset of operations from the original DARTS search space. In this subsection, we test GSparsity on the **S1**, **S2** and **S4** [2] spaces from [39].

The search and evaluation settings are the same as for the DARTS search space, except that ScheduledDropPath has a maximum drop probability of 0.2 in `Evaluation`. Note that [39] proposed several methods to robustify DARTS, which require computing the Hessian of the validation loss w.r.t. the architecture parameters, thus imposing an additional overhead to `Search`. GSparsity does not rely on such heuristics and therefore has much lower runtime and memory requirements.

Table 4 evaluates the performance of GSparsity on these search spaces, comparing it against DARTS and DARTS-ES (DARTS with early stopping from [39]). GSparsity performs amongst the best on all three search spaces, and substantially better than DARTS-ES on **S2** and **S4**, with up to 1.53% absolute test accuracy improvement on **S4**. DARTS-ES in turn clearly outperforms DARTS on all spaces. This verifies the robustness of the proposed GSparsity algorithm.

Search Space	DARTS*	DARTS-ES*	GSparsity (ours)
S1	95.34 ± 0.71	**96.95 ± 0.07**	**96.94 ± 0.14**
S2	95.58 ± 0.40	96.59 ± 0.14	**97.40 ± 0.11**
S4	93.05 ± 0.18	95.83 ± 0.21	**97.36 ± 0.12**

Table 4: Performance of DARTS, DARTS-ES and the proposed GSparsity on CIFAR-10 (*Results taken from Table 1 of [39]).

5 Conclusion

In this paper, we proposed to use group sparsity in order to bridge the gap between network pruning and differential one-shot NAS. The proposed formulation is flexible and can be tailored for kernel/filter pruning, operation pruning and NAS. We use ProxSGD to solve the nonsmooth nonconvex optimization problem optimally, and achieve new state-of-the-art results for filter pruning. By tackling NAS from a pruning perspective, we are able to formulate it as a single-level optimization problem that can be solved optimally by ProxSGD. Experiments show that the proposed GSparsity method reaches superior and robust performance on filter pruning, can address operation pruning, and yields state-of-the-art and robust results in NAS on various datasets and search spaces, without suffering from overfitting or from performance degradation after discretizing the architecture. In future work, we would like to further enhance the performance of GSparsity by directly searching on the full supernet, possibly employing techniques from PC-Darts [34] or the progressive learning approach from DrNAS [5].

[2] The search space **S3** in [39] is {3×3 SepConv, Identity, Zero}. We do not consider it as we would implicitly get the operation Zero from **S2** when none of {3×3 SepConv, Identity} is selected.

6 Reproducibility Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes]

   (c) Did you discuss any potential negative societal impacts of your work? [No]

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [Yes]

   (b) Did you include complete proofs of all theoretical results? [No]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] The code for the training pipeline is publicly available at `https://github.com/cc-hpc-itwm/GSparsity`.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] The logs of all our runs during the search and evaluation phase, complete with hyperparameters for our NAS method as well as random seeds are publicly available.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] See appendix and logs for more details

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See the appendix for an ablation study that tests the performance of the GSparsity with respect to μ

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [No]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes]

   (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

   (l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] NAS-Bench-201

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] See appendix for an in-depth explanation of how the regularization parameter was chosen.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [No]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No data from other people is used.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No data from other people is used.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] Alvarez, J. M. and Salzmann, M. (2016). Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278.

[2] Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*.

[3] Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning?

[4] Brock, A., Lim, T., Ritchie, J., and Weston, N. (2018). SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*.

[5] Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C.-J. (2021). DrNAS: Dirichlet neural architecture search. In *International Conference on Learning Representations*.

[6] Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303.

[7] Colson, B., Marcotte, P., and Savard, G. (2007). An overview of bilevel optimization.

[8] Ding, X., Hao, T., Tan, J., Liu, J., Han, J., Guo, Y., and Ding, G. (2021). Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4510–4520.

[9] Dong, X. and Yang, Y. (2019a). Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, volume 32, pages 1–12.

[10] Dong, X. and Yang, Y. (2019b). Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[11] Dong, X. and Yang, Y. (2020). NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*.

[12] Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

[13] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, pages 1135–1143.

[14] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[15] He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[16] Huang, G., Liu, S., Maaten, L. V. D., and Weinberger, K. Q. (2018). CondenseNet: An Efficient DenseNet Using Learned Group Convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2752–2761.

[17] Huang, Z. and Wang, N. (2018). Data-driven sparse strcture selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision*, pages 304–320.

[18] Krizhevsky, A. (2012). Learning multiple layers of features from tiny images. *University of Toronto*.

[19] Li, J., Qi, Q., Wang, J., Ge, C., Li, Y., Yue, Z., and Sun, H. (2019a). OICSR: Out-in-channel sparsity regularization for compact deep neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7039–7048.

[20] Li, L., Khodak, M., Balcan, M.-F., and Talwalkar, A. (2020a). Geometry-aware gradient algorithms for neural architecture search. *arXiv preprint arXiv:2004.07802*.

[21] Li, Y., Gu, S., Mayer, C., Gool, L. V., and Timofte, R. (2019b). Towards optimal structured CNN pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2790–2799.

[22] Li, Y., Gu, S., Mayer, C., Gool, L. V., and Timofte, R. (2020b). Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8018–8027.

[23] Li, Y., Gu, S., Mayer, C., Van Gool, L., and Timofte, R. (2020). Group sparsity: The hinge between filter pruning and decomposition for network compression. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8015–8024.

[24] Lindauer, M. and Hutter, F. (2020). Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18.

[25] Liu, H., Simonyan, K., and Yang, Y. (2019). DARTS: Differentiable architecture search. In *International Conference on Learning Representations*.

[26] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning efficient CNN through network slimming. In *International Conference on Machine Learning (ICML)*, pages 2736–2744.

[27] Luo, J.-H., Wu, J., and Lin, W. (2017). ThiNet: A filter level pruning method for deep neural network compression. In *Proceedings of the International Conference on Computer Vision*, pages 5058–5066.

[28] Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*.

[Sovrasov] Sovrasov, V. Flops counter for convolutional networks in pytorch framework.

[30] Su, X., You, S., Wang, F., Qian, C., Zhang, C., and Xu, C. (2021). BCNet: Searching for network width with bilaterally coupled network. In *Proceedings of CVPR 2021*.

[31] Wang, R., Cheng, M., Chen, X., Tang, X., and Hsieh, C.-J. (2021). Rethinking architecture selection in differentiable NAS. In *International Conference on Learning Representations*.

[32] Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning Structured Sparsity in Deep Neural Networks.

[33] Wu, Y., Liu, A., Huang, Z., Zhang, S., and Van Gool, L. (2021). Neural architecture search as sparse supernet. In *2021 AAAI Conference on Artificial Intelligence*.

[34] Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G. J., Tian, Q., and Xiong, H. (2020). PC-DARTS: Partial channel connections for memory-efficient differentiable architecture search. In *International Conference on Learning Representations*, volume 1, pages 1–13.

[35] Yang, Y., Li, H., You, S., Wang, F., Qian, C., and Lin, Z. (2020a). Ista-nas: Efficient and consistent neural architecture search by sparse coding. *Advances in Neural Information Processing Systems*, 33.

[36] Yang, Y., Yuan, Y., Chatzimichailidis, A., van Sloun, R. J. G., Lei, L., and Chatzinotas, S. (2020b). ProxSGD: Training structured neural networks under regularization and constraints. In *International Conference on Learning Representations*.

[37] Yoon, J. and Hwang, S. J. (2017). Combined group and exclusive sparsity for deep neural networks. In *34th International Conference on Machine Learning, ICML 2017*, volume 8, pages 6031–6039.

[38] Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 68(1):49–67.

[39] Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. (2020a). Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*.

[40] Zela, A., Siems, J., and Hutter, F. (2020b). NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations*.

[41] Zhang, X., Huang, Z., Wang, N., Xiang, S., and Pan, C. (2021). You Only Search Once: Single Shot Neural Architecture Search via Direct Sparse Optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2891–2904.

[42] Zhou, H., Alvarez, J. M., and Porikli, F. (2016). Less is more: Towards compact CNNs. In *Proceedings of European Conference on Computer Vision*, pages 662–677.

[43] Zhou, Y., Zhang, Y., Wang, Y., and Tian, Q. (2019). Accelerate CNN via recursive Bayesian pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3306–3315.

[44] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Algorithm 1 The ProxSGD Algorithm for Problem (1)

Initialization: $\boldsymbol{w}(0), \boldsymbol{v}(-1) = \boldsymbol{0}$, $t = 0, T$, $\{\rho(t), \epsilon(t)\}_{t=0}^{T}$
for $t = 0 : 1 : T$ **do**

1. Compute the gradient $\boldsymbol{g}(t)$ based on the minibatch $\mathbb{M}(t)$: $\boldsymbol{g}(t) = \frac{1}{|\mathbb{M}(t)|} \sum_{\boldsymbol{x} \in \mathbb{M}(t)} \nabla_{\boldsymbol{w}} f(\boldsymbol{w}(t), \boldsymbol{x})$.
2. Update the momentum: $\boldsymbol{v}(t) = (1 - \rho(t))\boldsymbol{v}(t-1) + \rho(t)\boldsymbol{g}(t)$.
3. Compute the proximal mapping: for $k = 1, \ldots, K$,
$$\text{Prox}_{\widetilde{\mu}_k(t)\|\boldsymbol{w}_k\|_2}(\widetilde{\boldsymbol{w}}_k(t)) = \left(1 - \frac{\widetilde{\mu}_k(t)}{\|\widetilde{\boldsymbol{w}}_k(t)\|_2}\right)^{+} \widetilde{\boldsymbol{w}}_k(t), \tag{2}$$
where $\widetilde{\boldsymbol{w}}_k(t) \triangleq \boldsymbol{w}_k(t) - \boldsymbol{v}_k(t)/\tau_k(t)$ and $\widetilde{\mu}_k(t) \triangleq \mu_k/\tau_k(t)$.
4. Update the weight: $\boldsymbol{w}_k(t+1) = (1 - \epsilon(t))\boldsymbol{w}_k(t) + \epsilon(t)\,\text{Prox}_{\widetilde{\mu}_k(t)\|\boldsymbol{w}_k\|_2}(\widetilde{\boldsymbol{w}}_k(t))$, $k = 1, \ldots, K$.

end for

A The ProxSGD Algorithm

The details of ProxSGD are summarized in Algorithm 1. In Step 1, the instantaneous gradient $\boldsymbol{g}$ based on the minibatch is computed. In Step 2, the momentum $\boldsymbol{v}$ is updated, where $\rho(t)$ is the learning rate for the momentum. In Step 3, the proximal mapping is computed. In Step 4, the weight vector $\boldsymbol{w}$ is updated, where $\epsilon(t)$ is the learning rate for the weight. Note that $\tau_k(t)$ in Step 3 is a positive scalar that can make the learning rate $\epsilon(t)$ adaptive to each group k. This is best seen when $\mu_k = 0$, as the weight update in Step 4 becomes $\boldsymbol{w}_k(t+1) = \boldsymbol{w}_k(t) - \frac{\epsilon(t)}{\tau_k(t)}\boldsymbol{v}_k(t)$.

In contrast to the proximal algorithm used in [23] and [41], ProxSGD has a guaranteed convergence to a stationary point of (1). It converges much faster to a group sparse solution than stochastic subgradient algorithms, hence eliminating the performance degradation due to the discretization step in DARTS (i.e., setting operations with a small norm to 0).

There are different ways to choose μ_k, and the choice differs across different experiments. A straightforward choice for μ_k is some μ that is identical for all groups: $\mu_k = \mu,\ \forall k$. Sometimes it might be beneficial to normalize the regularization gain μ by the size of the group:
$$\mu_k = \frac{\mu}{\sqrt{|\boldsymbol{w}_k|}},\ \forall k, \tag{3}$$
or $\mu_k = \mu \cdot \sqrt{|\boldsymbol{w}_k|},\ \forall k$. There is no theoretical justification that favors one over another (see [38]), and we empirically find that $\mu_k = \mu, \forall k$ works well when the sizes of the groups are not very different, as in operation pruning. This is, however, not the case for NAS, and we find that the normalization specified in (3) yields the best result.

B Filter Pruning

Experiment setup. Firstly we determine which filters to prune away by training ResNet-50-*with* the $L_{2,1}$-norm regularization ($\mu_k = \mu/\sqrt{|\boldsymbol{w}_k|}$)-by ProxSGD for 90 epochs, with the following hyperparameters: initial learning rate 0.001 (which is linearly decayed by 10 every 30 epochs) and momentum 0.9. Furthermore, $\tau_k(t)$ in (2) is defined as $\tau_k(t) = \text{mean}\left(\sqrt{\boldsymbol{r}(t)/(1-\beta^t)}\right) + \delta$ ($\beta = 0.999$, $\delta = 10^{-8}$), and $\boldsymbol{r}(t)$ is the aggregate squared gradient updated iteratively as $\boldsymbol{r}(t) = \boldsymbol{r}(t-1)\beta + (1-\beta)\boldsymbol{g}(t)^2$.

ResNet-50 consists of 4 bottleneck layers, and bottleneck layer 1/2/3/4 consists of 3/4/6/3 blocks, and a block consists of three conv layers. We prune the filters of the first two layers only in each block so that the output of the last conv layer would have the same dimension as the residual layer.

Note that our objective is to reduce the multiple-accumulate operations (MACs). The second and third bottleneck layers constitute 60.83% of the total MACs, although they have only 35.55% of the total parameters. Therefore, to push more filters in the these layers to be 0, we set the regularization gain to be $\mu_k = \mu/\sqrt{|\boldsymbol{w}_k|}$, which is further doubled if it is in the second/third bottleneck layers.

After training with ProxSGD is completed, we prune (rather than mask) the zero filters from the model. As the convergence of iterative algorithms to an optimal solution $\boldsymbol{w}^\star$ is in the sense that $\|\boldsymbol{w}(t) - \boldsymbol{w}^\star\| \leq c$ for an arbitrarily small but strictly positive c, we prune the filters whose L_2 norm is smaller than 10^{-6}. Then we retrain the pruned network -*without* the $L_{2,1}$-norm regularization- for 90 epochs by SGD with momentum with an initial learning rate 0.1 (which is linearly decayed by 10 every 30 epochs), momentum 0.9 and batch size 256.

We remark that for many pruning methods, the code for ResNet-50 and ImageNet 2012 is not available in the repository (including TAS [9] and BCNet [30] which are not listed in Table 1). It is thus impossible to verify their results. Besides, it is not clear how the MACs are measured, making a fair comparison difficult. As an example, FPGM in Table 1 reported 46.50% of the original MACs, but it is 49.03% according to our calculator [Sovrasov].

B.1 Filter Pruning Ablation Study

In the experiment of filter pruning in Sec. 4.1, we test as an ablation study the performance of the GSparsity with respect to μ, and the results are summarized in Table 5. We can clearly see that the MACs and top-1 acc is a monotonic function of μ, so the appropriate value of μ achieving a target sparsity level can be found efficiently by the bisection search.

μ	top-1 acc after search	top-1 acc after retrain	MACs	Params
0.01	73.99	76.01	3.22G	22.70M
0.02	73.81	75.21	2.60G	17.08M
0.03	73.49	74.93	2.36G	15.59M
0.04	72.23	74.35	2.17G	14.67M
0.05	73.00	74.33	2.06G	14.03M
0.06	72.83	74.10	1.97G	13.56M
0.07	73.01	74.00	1.92G	13.22M
0.08	72.73	73.62	1.83G	12.86M
0.09	72.47	73.41	1.77G	12.51M
0.10	72.45	73.34	1.74G	12.36M

Table 5: GSparsity and filter pruning: Ablation study.

C ResNet-50 Structure after Pruning

In this section the resulting network structure of ResNet-50 after filter pruning is summarized. The structure of the unpruned ResNet-50 network is shown in Table 6. It consists of 4 layers, where each layer contains a certain number of blocks. Each block contains three different convolutions. In the experiments from Section 4.1, the ResNet-50 model is trained on ImageNet-2012 using GSparsity. Table 7 depicts the resulting networks, which have been trained using GSparsity with $\mu \in \{0.02, 0.05, 0.07, 0.10\}$. Note that we removed the filter height and width from each cell in Table 7 in order to save space.

D Operation pruning - Experiment setup

As a base network to be pruned, we choose one of the networks found in DARTS [25]: DARTS-V2, which has 3.3M parameters. To perform operation pruning, a group should consist of all trainable parameters of the same operation. For example, the dilated convolution operation consists of

Layers	Blocks	Conv1	Conv2	Conv3
Layer 1	Block 1	$1 \times 1, 64$	$3 \times 3, 64$	$1 \times 1, 256$
	Block 2	$1 \times 1, 64$	$3 \times 3, 64$	$1 \times 1, 256$
	Block 3	$1 \times 1, 64$	$3 \times 3, 64$	$1 \times 1, 256$
Layer 2	Block 1	$1 \times 1, 128$	$3 \times 3, 128$	$1 \times 1, 512$
	Block 2	$1 \times 1, 128$	$3 \times 3, 128$	$1 \times 1, 512$
	Block 3	$1 \times 1, 128$	$3 \times 3, 128$	$1 \times 1, 512$
	Block 4	$1 \times 1, 128$	$3 \times 3, 128$	$1 \times 1, 512$
Layer 3	Block 1	$1 \times 1, 256$	$3 \times 3, 256$	$1 \times 1, 1024$
	Block 2	$1 \times 1, 256$	$3 \times 3, 256$	$1 \times 1, 1024$
	Block 3	$1 \times 1, 256$	$3 \times 3, 256$	$1 \times 1, 1024$
	Block 4	$1 \times 1, 256$	$3 \times 3, 256$	$1 \times 1, 1024$
	Block 5	$1 \times 1, 256$	$3 \times 3, 256$	$1 \times 1, 1024$
	Block 6	$1 \times 1, 256$	$3 \times 3, 256$	$1 \times 1, 1024$
Layer 4	Block 1	$1 \times 1, 512$	$3 \times 3, 512$	$1 \times 1, 2048$
	Block 2	$1 \times 1, 512$	$3 \times 3, 512$	$1 \times 1, 2048$
	Block 3	$1 \times 1, 512$	$3 \times 3, 512$	$1 \times 1, 2048$

Table 6: The structure of the unpruned ResNet-50 network that is used for training on ImageNet-2012. Each cell contains the filter height × filter height as well as the number of output channels.

		$\mu = 0.02$			$\mu = 0.05$			$\mu = 0.07$			$\mu = 0.10$		
Layers	Blocks	Conv1	Conv2	Conv3	Conv1	Conv2	Conv3	Conv1	Conv2	Conv3	Conv1	Conv2	Conv3
Layer 1	Block 1	14	32	256	7	33	256	7	32	256	5	31	256
	Block 2	54	63	256	44	56	256	43	57	256	36	50	256
	Block 3	56	64	256	44	64	256	40	64	256	33	63	256
Layer 2	Block 1	72	127	512	42	110	512	32	102	512	24	88	512
	Block 2	14	51	512	9	45	512	8	50	512	6	40	512
	Block 3	60	93	512	34	72	512	32	58	512	30	54	512
	Block 4	97	124	512	59	121	512	51	118	512	41	116	512
Layer 3	Block 1	221	253	1024	161	238	1024	138	236	1024	124	222	1024
	Block 2	81	160	1024	51	130	1024	42	118	1024	39	105	1024
	Block 3	101	202	1024	64	176	1024	57	170	1024	52	161	1024
	Block 4	108	102	1024	61	160	1024	52	160	1024	40	143	1024
	Block 5	91	158	1024	54	130	1024	42	117	1024	34	100	1024
	Block 6	137	204	1024	92	178	1024	80	160	1024	72	150	1024
Layer 4	Block 1	512	512	2048	512	512	2048	500	512	2048	468	512	2048
	Block 2	461	510	2048	224	491	2048	172	464	2048	133	429	2048
	Block 3	44	70	2048	11	5	2048	10	4	2048	9	3	2048

Table 7: Final structure of ResNet-50 after training with GSparsity on the ImageNet-2012 dataset. This Table depicts the resulting output channels of the different convolutions for four different values of μ. Please refer to Table 6 for the structure of the unpruned network. The filter height and width have been omitted in this Table to save space.

four suboperations: `ReLU`, `Conv2d(C_in,C_in)`, `Conv2d(C_in, C_out)`, `BatchNorm2d(C_out)`. The group should consist of the trainable parameters of all suboperations.

We first use ProxSGD to train DARTS-V2 -*with* the $L_{2,1}$-norm regularization- on CIFAR-10, where the regularization gain is identical for all groups. We consider various values of $\mu \in \{0.0001, 0.0002, 0.0005, 0.002, 0.004\}$ to get different sparsity levels. After training with ProxSGD is finished, we prune the operations whose L_2 norm is smaller than 10^{-6}. The pruned network will be retrained -*without* the $L_{2,1}$-norm regularization- by SGD with momentum, and the retrained accuracy will be reported in the following.

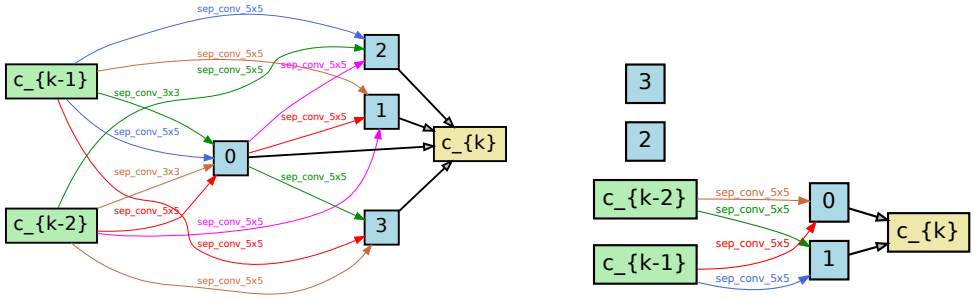
E CIFAR-10 and CIFAR-100 on DARTS Search Space

In the experiment of neural architecture search in Sec. 4.3, the DARTS search space consists of the following operations: 3×3 MaxPooling, 3×3 AvgPooling, Identity, 3×3 SepConv, 5×5 SepConv, 3×3 DilConv, and 5×5 DilConv.

We train the supernet during the search phase using the ProxSGD optimizer (with $L_{2,1}$-norm regularization) with a learning rate of 0.001 (without learning rate scheduler), momentum 0.8 and $\tau_k(t) = 1$. During evaluation, we train the network using SGD with learning rate 0.025, momentum parameter 0.9, weight decay 3e-4, an auxiliary tower with weight 0.4, cutout regularization with length 16 and ScheduledDropPath [44] with the maximum drop probability 0.3.

We tune the regularization gain μ in a similar way as the bisection method. Firstly, we try values of μ spanning a big range (such as μ =0.1, 1, 10, 100) to determine a small range in which the desirable μ (i.e., the desired sparsity level) lies. Then the bisection method is repeated in the small range, for example $[50, 100]$. We can typically find an appropriate μ within 10 trials. We remark that to reduce the effort to tune μ, a seemingly obvious way is to use a small μ and only keep the top k operations with the largest L_2 norm. However, this discretization step would incur notable performance degradation. Therefore, searching for the appropriate μ will reduce the performance degradation due to disretization and it is not an extra burden compared to other methods. The regularization gain μ_k is chosen according to (3) with $\mu = 60$.

One of the three architectures we found with GSparsity on CIFAR-10 is shown in Figure 2.



(a) Normal cell found by GSparsity for the CIFAR-10 dataset.

(b) Reduction cell found by GSparsity for the CIFAR-10 dataset.

Figure 2: Example of one architecture on CIFAR-10 found by GSparsity. We remove the restriction on the number of operations per node. In order to match the network size to other methods we stack less cells when evaluating the architecture.

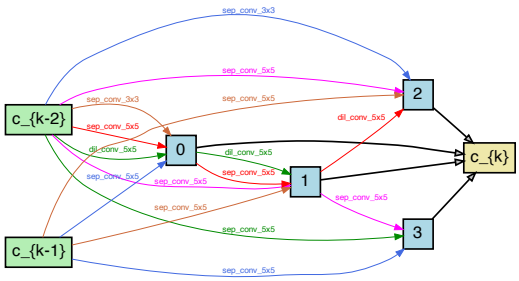
CIFAR-100 uses the same search-space as CIFAR-10, the only difference being the output of the network is set to 100 in order to deal with the increased number of classes. During the search-phase, the 8-cell supernet is trained for 100 epochs using the ProxSGD optimizer with $L_{2,1}$-norm regularization. We use a learning rate of 0.001 (without learning rate scheduler), momentum 0.8 and $\tau_k(t) = 1$. The regularization gain μ_k is chosen according to (3) with $\mu = 120$.

Figure 3 shows the normal and reduction cell of one of the three architectures that we found with the GSparsity method.

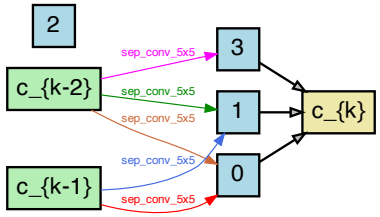
We note that the accuracies of the baselines, albeit reproduced by using the authors' original implementations, are generally worse than reported in the respective papers. One reason is that we consider the average performance based on all architectures (instead of the best architecture w.r.t. validation performance, as done by many recent papers)[3].

Accuracy of the best performing architecture. As previously mentioned, this paper deviates from the commonly used practice of only reporting the average accuracy of the model with the best performance. In Table 8 we used the results obtained from Table 3 to find the best performing

[3]However, the performance of the best architecture we could reproduce is still notably worse than originally reported.



(a) Normal cell found by GSparsity for the CIFAR-100 dataset.



(b) Reduction cell found by GSparsity for the CIFAR-100 dataset.

Figure 3: Example of one architecture on CIFAR-100 found by GSparsity. We remove the restriction on the number of operations per node. In order to match the network size to other methods, we stack less cells when evaluating the architecture.

architecture for each method on CIFAR-10 and CIFAR-100. On CIFAR-10, we observe that PC-DARTS is able to reach the highest accuracy with 97.38 ± 0.08, followed closely by P-DARTS with 97.25 ± 0.07 and our proposed method GSparsity with 97.24 ± 0.03. On CIFAR-100, our proposed method is able to find the best performing architecture, which reaches an average accuracy of 84.04 ± 0.28, followed by P-DARTS with 83.62 ± 0.19.

	CIFAR-10 Accuracy	CIFAR-100 Accuracy
DARTS (2nd)	97.09 ± 0.09	81.05 ± 0.23
P-DARTS	97.25 ± 0.07	83.62 ± 0.19
PC-DARTS	97.38 ± 0.08	83.38 ± 0.20
DrNAS	96.98 ± 0.07	83.41 ± 0.18
GDAS	96.77 ± 0.11	81.41 ± 0.40
ISTA-NAS	96.86 ± 0.02	83.17 ± 0.17
GAEA	96.57 ± 0.04	80.34 ± 0.05
GSparsity (ours)	97.24 ± 0.03	84.04 ± 0.28

Table 8: Accuracy of the best architecture found by different NAS methods for the DARTS space. Each method has been run three times, and each of those three architectures that have been found have been evaluated three times. This table summarizes the performance of the best architecture out of the three that have been search by each method, contrary to Table 3, which shows the average accuracy of all three architectures.

Pruning weights vs. pruning switches. We have also compared to the approach where scaling factors (which act as a switch) are appended to the operations and then pruned (instead of weights). It turns out that the magnitudes of the scaling factors are very sensitive to the value of μ and they are either all active or all zero. For example,

- when $\mu = 3.66$, all scaling factors are active, see the plot of operations at epoch 50, at log-search-switch/*-mu_3.66_div_0.5_time_20211002-145316/
- when $\mu = 3.67$, all scaling factors are zero, see the plot of operations at epoch 50, at log-search-switch/*-mu_3.67_div_0.5_time_20211002-145443/
- when $\mu = 3.69$, all scaling factors are active, see the plot of operations at epoch 50, at log-search-switch/*-mu_3.69_div_0.5_time_20211002-145456/

Therefore, it is more beneficial to directly prune the weights.

F ImageNet-2012 on DARTS Search Space

In the experiment of neural architecture search on ImageNet 2012 in Sec. 4.3, the DARTS search space consists the same operations as for CIFAR-10 and CIFAR-100.

The search and evaluation network on ImageNet 2012 differs from the network used to search and evaluate CIFAR-10/-100 and is similar to the network used in [5], [6] and [34]. We use three convolutions with stride 2 in order to downscale the spatial resolution of the ImageNet samples to 28×28.

During search, we train on 10% of the ImageNet 2012 samples. We use ProxSGD with $L_{2,1}$-norm regularization and train the network for 50 epochs. Similar to the CIFAR dataset, we used the bisection method to tune the regularization gain μ. We used μ=43 for the network trained directly on ImageNet-2012. The model was trained in parallel on 4 Titan GPUs.

We changed the optimizer during the searching process compared to prior NAS experiments. We use $\tau_k(t) = \text{mean}\left(\sqrt{\boldsymbol{r}(t)/(1-\beta^t)}\right) + \delta$. Here, we take the mean over all the elements in the group. The aggregate squared gradient $\boldsymbol{r}(t)$ is updated iteratively for each element inside the group as $\boldsymbol{r}(t) = \boldsymbol{r}(t-1)\beta + (1-\beta)\boldsymbol{g}(t)^2$. In our experiments we used β=0.1.

During evaluation, we scale the network to 14 cells with 48 initial channels, following previous works ([5], [6], [34]). The network is trained for 250 epochs using the SGD optimizer with momentum. We use an initial learning rate of 0.5, which we decay down to 0 using a cosine annealing scheduler, momentum 0.9, a batch-size of 512 and a weight decay value of 3e-5. We use label smoothing and an auxiliary tower with auxiliary weight of 0.4.

Note that because ImageNet 2012 is computationally demanding, we were only able to run each method once. The results are summarized in Table 9. We observe that PC-DARTS is able to achieve the highest top-1 accuracy of 75.69%, followed by our approach, which reached 75.51%. We reproduce DrNAS using the authors' implementation, but the accuracy we could reproduce is 65.25%, and it is notably below the one reported in [5]. It takes PC-DARTS and GSparsity roughly the same time to search for an architecture: 3.1 and 3.3 GPU days, respectively. DrNAS takes the longest with 7.3 GPU days.

	top-1 acc	top-5 acc	search cost	params
DARTS*	71.09	89.83	2.9 days	5.54M
P-DARTS*	74.11	91.73	0.3 days	3.67M
GSparsity* (ours)	75.29	92.42	0.5 days	6.29M
PC-DARTS	75.71	92.68	3.1 days	5.57M
DrNAS	65.25	86.23	7.3 days	3.33M
GSparsity (ours)	75.51	92.60	3.3 days	6.22M

Table 9: NAS on DARTS search space for ImageNet 2012 (*Architecture has been searched on CIFAR-10 or CIFAR-100). All results are reproduced from their authors' implementations.

F.0.1 NAS-Bench-201 Search Space.

We now evaluate our GSparsity method on the tabular NAS benchmark NAS-Bench-201 [11].

Architecture space and search settings. NAS-Bench-201 employs a fixed cell-based structure similar to DARTS, and we refer to [11] for more details about the network.

To search for a single cell structure, the operation of the same type in different cells are put into the same group (cf. Figure 1). Then we train the network with ProxSGD for 100 epochs using the full training set. The regularization gain μ_k follows (3) where $\mu = 200$.

Evaluation settings. The architectures found in the `Search` phase are simply queried from the NAS-Bench-201 database to obtain validation/test accuracy. We queried the mean performance for

	CIFAR-10		CIFAR-100		ImageNet-16-120	
	validation	test	validation	test	validation	test
DARTS (1st)	49.27 ± 13.4	59.84 ± 7.84	38.57 ± 0.00	38.97 ± 0.00	18.87 ± 0.00	18.41 ± 0.00
DARTS (2nd)	58.78 ± 13.4	65.38 ± 7.84	38.57 ± 0.00	38.97 ± 0.00	18.87 ± 0.00	18.41 ± 0.00
P-DARTS	64.72 ± 19.1	71.43 ± 14.2	38.57 ± 0.00	38.97 ± 0.00	28.03 ± 13.0	27.72 ± 13.2
GAEA	83.31 ± 1.31	83.18 ± 1.20	54.94 ± 0.26	54.88 ± 0.17	29.31 ± 3.19	28.42 ± 3.31
PC-DARTS	89.46 ± 1.05	93.06 ± 0.99	67.19 ± 1.36	67.76 ± 1.00	40.57 ± 0.77	40.84 ± 0.85
DrNAS	90.20 ± 0.00	**93.76 ± 0.00**	67.84 ± 1.74	67.62 ± 1.69	40.78 ± 0.00	**41.44 ± 0.00**
GSparsity (ours)	90.20 ± 0.00	**93.76 ± 0.00**	70.71 ± 0.00	**71.11 ± 0.00**	40.78 ± 0.00	**41.44 ± 0.00**

Table 10: NAS on NAS-Bench-201 search space (reproduced from their authors' implementations).

each architecture and in the table report mean/standard deviation across this mean performance for the architectures resulting from 3 `Search` runs.

Results. Table 10 shows that GSparsity performs amongst the best for all three datasets on the NAS-Bench-201 search space. DrNAS performs similar to GSparsity, finding the same architectures on CIFAR-10 and ImageNet-16-120.

G Convergence of ProxSGD vs. SGD

We compare the convergence of ProxSGD and SGD with $L_{2,1}$-regularization in the NAS setting. For this experiment we set the weight decay to a fixed value of $\mu = 50$ and we search for a network architecture using our GSparsity method. For SGD we search for a variety of learning rates and choose the best performing hyperparameters. The momentum stays fixed at $m = 0.9$. The results are summarized in Figure 4.

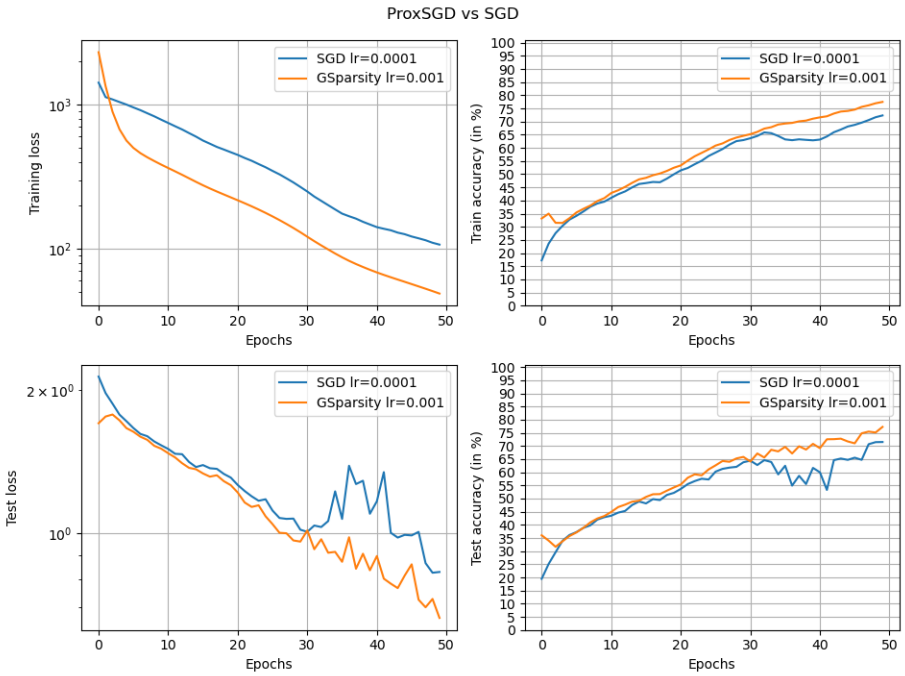
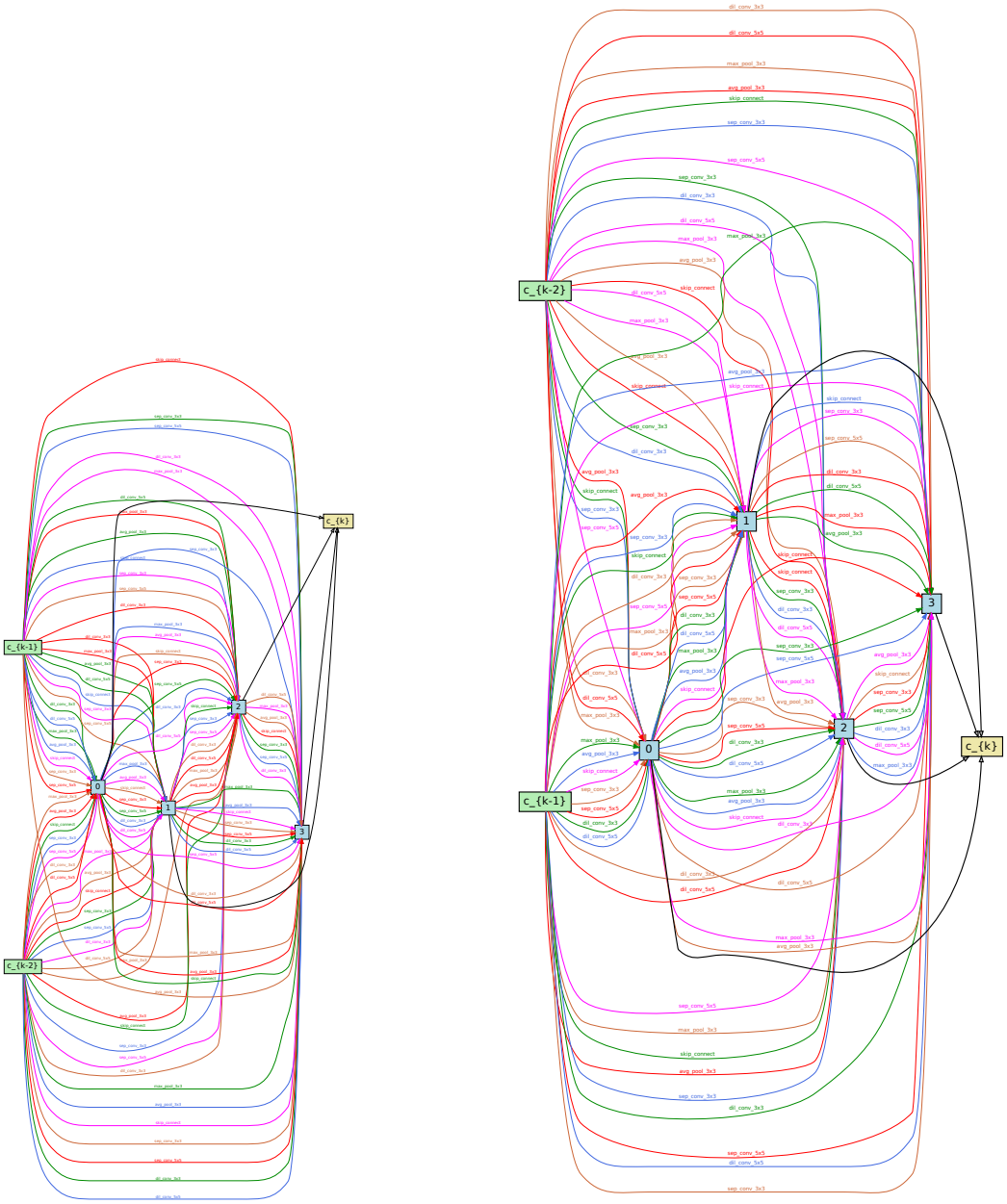


Figure 4: Neural architecture search using SGD and ProxSGD (denoted by GSparsity).

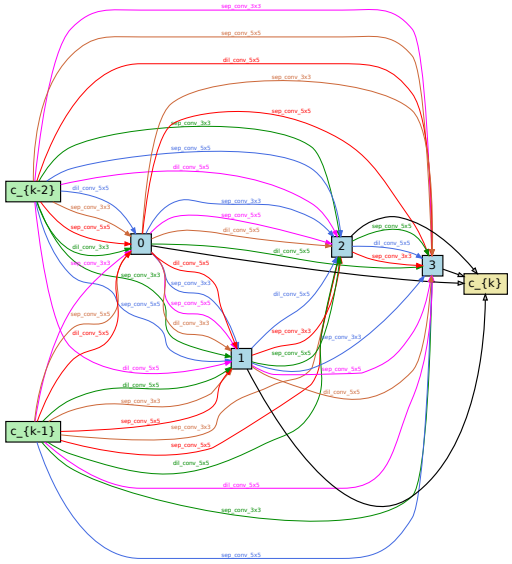
We observe that GSparsity outperforms SGD with a lower training objective and a higher validation accuracy. The biggest difference is observed in the cell architecture, which is depicted in Figure 5 for SGD and Figure 6 for ProxSGD.



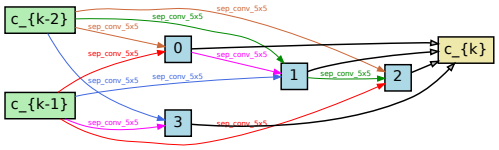
(a) Normal cell with 96 non-zero operations.

(b) Reduction cell with 95 non-zero operations.

Figure 5: Normal and reduction cell for $\mu = 50$ trained with SGD.



(a) Normal cell with 44 non-zero operations.



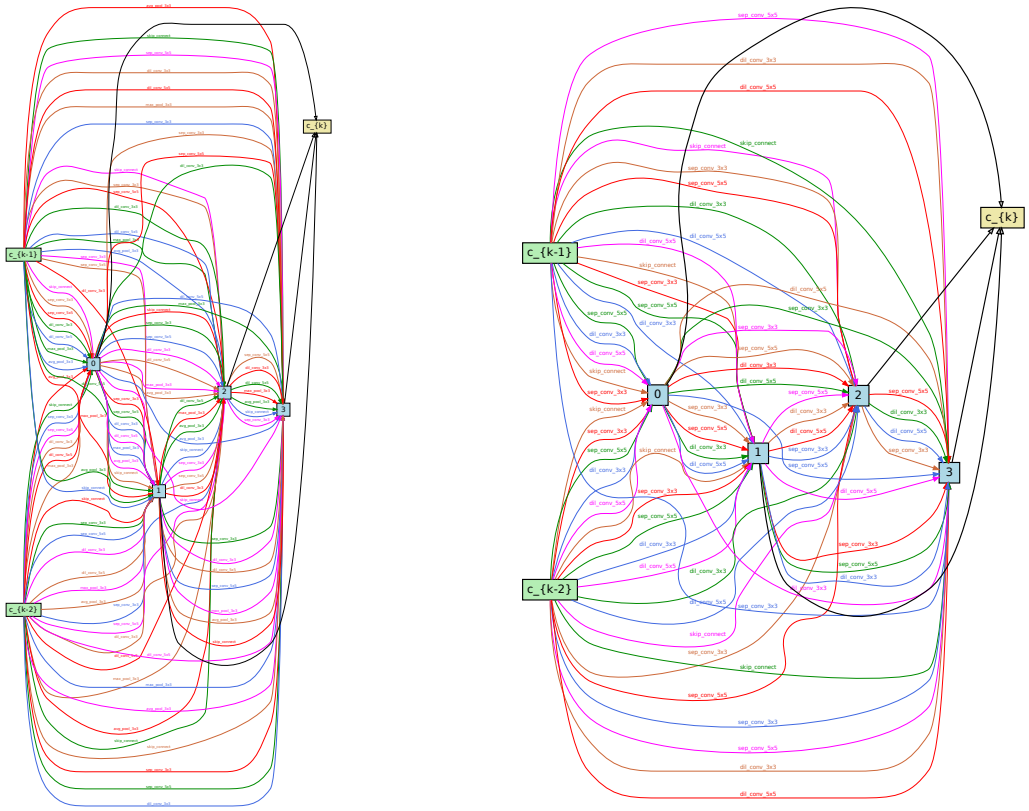
(b) Reduction cell with 11 non-zero operations.

Figure 6: Normal and reduction cell for $\mu = 50$ trained with ProxSGD.

We see that while GSparsity is able to converge to a sparse solution, with 44 non-zero operations in the normal cell and 11 operations in the reduction cell, SGD does not converge to a sparse solution. There are 96 remaining operations in the normal cell and 95 remaining operations in the reduction cell.

H GSparsity and NAS: Ablation Study

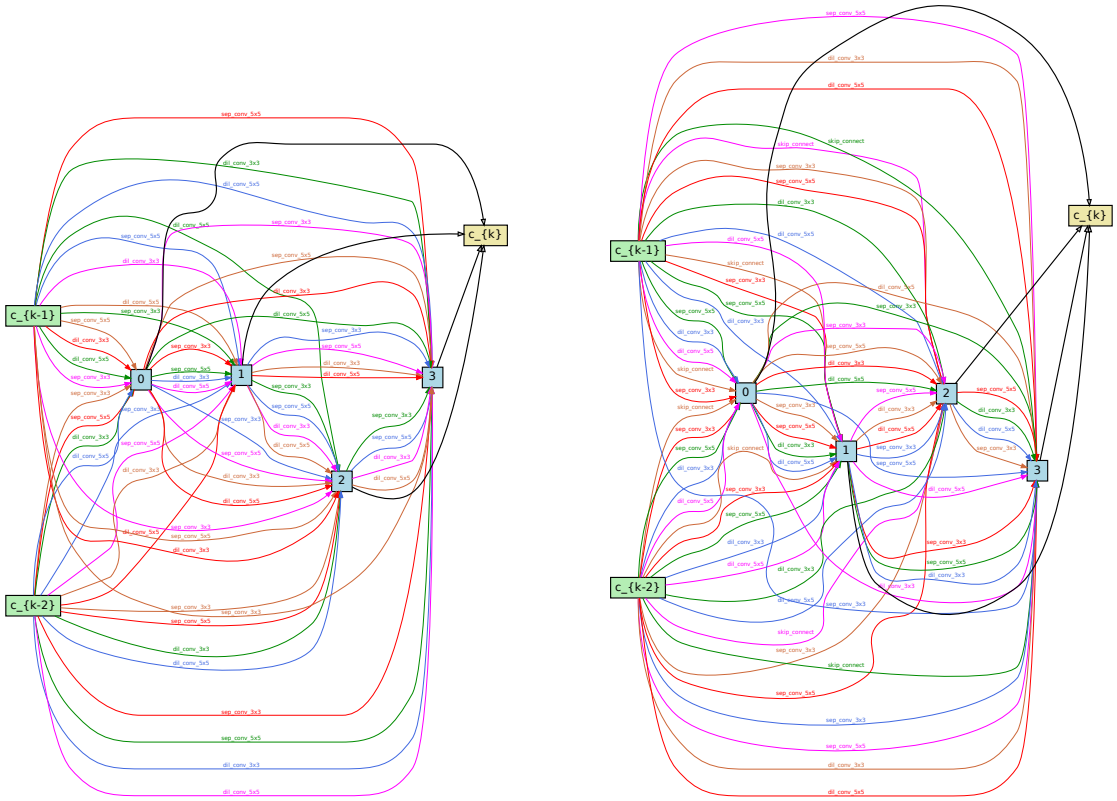
In order to see the effect of the regularization parameter μ on the structure of the final network, we depict the found architecture for $\mu \in [0.1, 1, 10, 50, 100, 200, 500, 1000]$ in Figures (7)-(14). During training of each model with GSparsity, the learning rate has been kept at a fixed value of $lr = 0.001$. One can observe that the number of non-zero operations decreases monotonically with the value of μ. For $\mu = 0.1$, the number of non-zero operations in the normal cell is 98, and in the reduction cell there are 64 operations. On the other hand, for a very large value of μ the network prunes all of the operations in both cells, as can be seen in Figure 14 for $\mu = 1000$. For small values of μ, the resulting number of operations is more sensitive for changes in its value. For example, going from $\mu = 0.1$ to $\mu = 1$, a change of only $\Delta\mu = 0.9$, the total number of non-zero operations is reduced by 42. But going from $\mu = 100$ to $\mu = 200$ only reduces the number of non-zero operations by 10.



(a) Normal cell with $\mu = 0.1$. There are 98 non-zero operations in this cell after pruning.

(b) Reduction cell with $\mu = 0.1$. There are 64 non-zero operations in this cell after pruning.

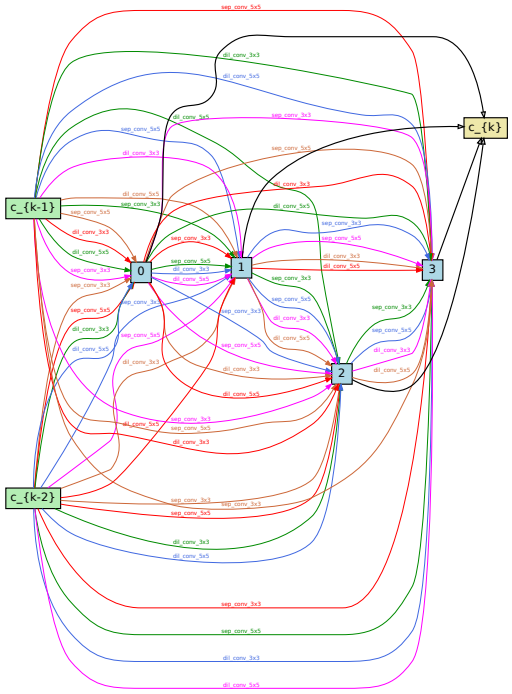
Figure 7: Resulting network structure after training with GSparsity with $\mu = 0.1$ and fixed $lr = 0.001$.



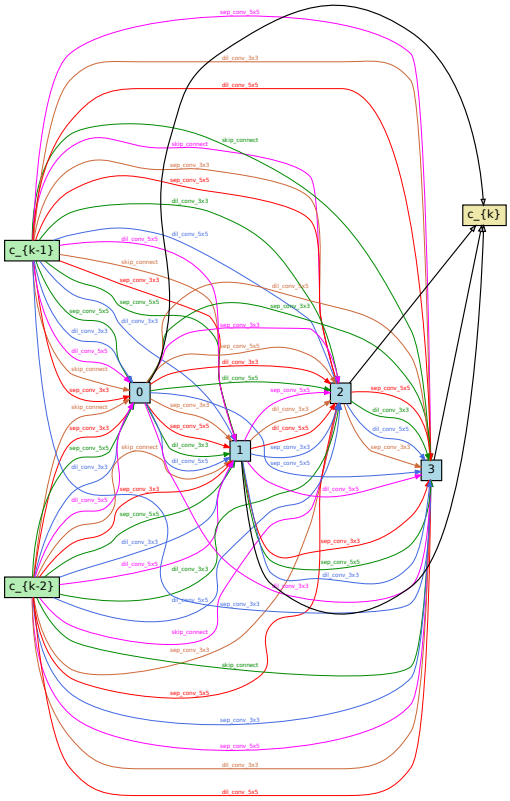
(a) Normal cell with $\mu = 1$. There are 56 non-zero operations in this cell after pruning.

(b) Reduction cell with $\mu = 1$. There are 64 non-zero operations in this cell after pruning.

Figure 8: Resulting network structure after training with GSparsity with $\mu = 1$ and fixed $lr = 0.001$.

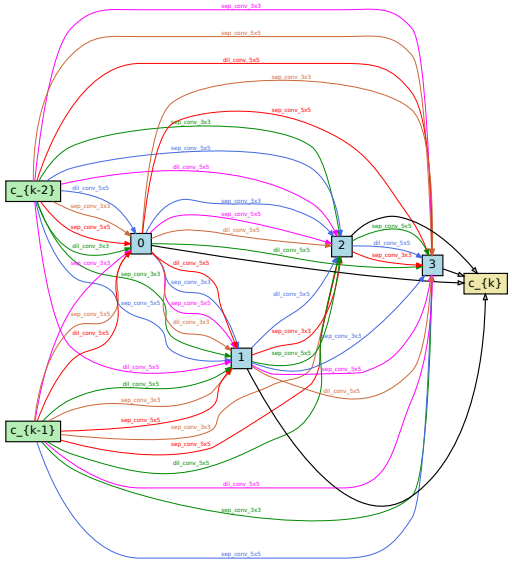


(a) Normal cell with $\mu = 10$. There are 56 non-zero operations in this cell after pruning.

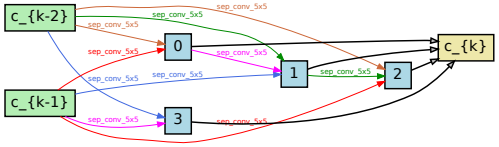


(b) Reduction cell with $\mu = 10$. There are 64 non-zero operations in this cell after pruning.

Figure 9: Resulting network structure after training with GSparsity with $\mu = 10$ and fixed $lr = 0.001$.

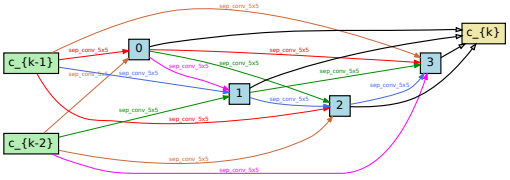


(a) Normal cell with $\mu = 50$. There are 44 non-zero operations in this cell after pruning.

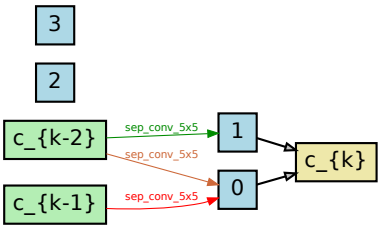


(b) Reduction cell with $\mu = 50$. There are 11 non-zero operations in this cell after pruning.

Figure 10: Resulting network structure after training with GSparsity with $\mu = 50$ and fixed $lr = 0.001$.

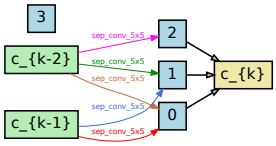


(a) Normal cell with $\mu = 100$. There are 14 non-zero operations in this cell after pruning.

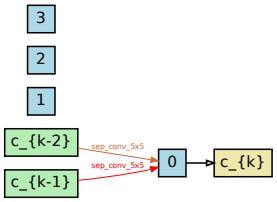


(b) Reduction cell with $\mu = 100$. There are 3 non-zero operations in this cell after pruning.

Figure 11: Resulting network structure after training with GSparsity with $\mu = 100$ and fixed $lr = 0.001$.

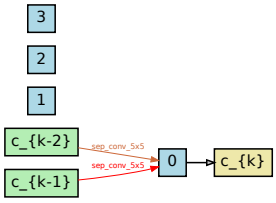


(a) Normal cell with $\mu = 200$. There are 5 non-zero operations in this cell after pruning.

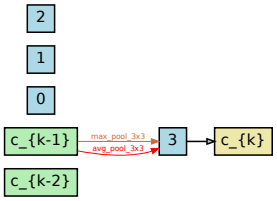


(b) Reduction cell with $\mu = 200$. There are 2 non-zero operations in this cell after pruning.

Figure 12: Resulting network structure after training with GSparsity with $\mu = 200$ and fixed $lr = 0.001$.

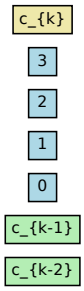


(a) Normal cell with $\mu = 500$. There are 2 non-zero operations in this cell after pruning.

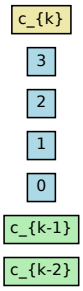


(b) Reduction cell with $\mu = 500$. There are 2 non-zero operations in this cell after pruning.

Figure 13: Resulting network structure after training with GSparsity with $\mu = 500$ and fixed $lr = 0.001$.



(a) Normal cell with $\mu = 1000$. There are 0 non-zero operations in this cell after pruning.



(b) Reduction cell with $\mu = 1000$. There are 0 non-zero operations in this cell after pruning.

Figure 14: Resulting network structure after training with GSparsity with $\mu = 1000$ and fixed $lr = 0.001$.