

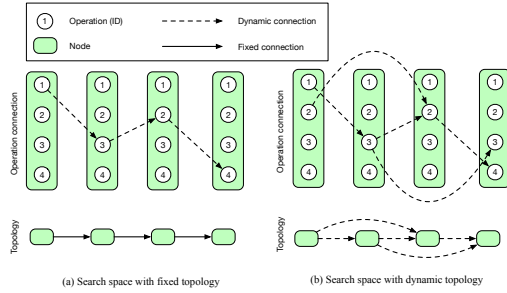
HOW TO TRAIN YOUR SUPER-NET: AN ANALYSIS OF TRAINING HEURISTICS IN WEIGHT-SHARING NAS

SUPPLEMENTARY MATERIAL

Anonymous authors

Paper under double-blind review

A METHODOLOGY DETAILS



	NASBench-101	NASBench-201	DARTS-NDS
# Arch.	423,624	15,625	5,000
# Op.	3	5	8
Channel	Dynamic	Fix	Fix
Optimal	Global	Global	Sample
Nodes= (n)	5	4	4
Param.	$O(n)$	$O(n) - O(n^2)$	$O(n) - O(n^2)$
Edges	$O(n^2)$	$O(n^2)$	$O(n)$
Merge	Concat.	Sum	Sum

Table 1: Search Spaces

Figure 1: Comparison between fixed and dynamic topology search spaces.

In this section, we provide some additional details about our methodology.

A.1 ADAPTATION OF FAIRNAS

Originally, FairNAS (Chu et al., 2019) was proposed in a search space with a fixed sequential topology, as depicted by Figure 1 (a), where every node is sequentially connected to the previous one, and only the operations on the edges are subject to change. However, our benchmark search spaces exploit a more complex dynamic topology, as illustrated in Figure 1 (b), where one node can connect to one or more previous nodes.

Before generalizing to a dynamic topology search space, we simplify the original approach into a 2-node scenario: for each input batch, FairNAS will first randomly generate a sequence of all o possible operations. It then samples one operation at a time, computes gradients for the fixed input batch, and accumulates the gradients across the operations. Once all operations have been sampled, the super-net parameters are updated with the average gradients. This ensures that all possible paths are equally exploited. With this simplification, FairNAS can be applied regardless of the topology. For a sequential-topology search space, we repeat the 2-node policy for every consecutive node pair. Naturally, for a dynamic topology space, FairNAS can be adopted in a similar manner, i.e., one first samples a topology, then applies the 2-node strategy for all connected node pairs. Note that adapting FairNAS increases the training time by a factor o .

A.2 SEARCH SPACES

Here we provide additional details of the search space used in this paper. A summary of these search spaces and their properties is shown in Table 1. The search spaces differ in the number of architectures that have known stand-alone accuracy (# Arch.), the number of possible operations (# Op.), how the channels are handled in the convolution operations (Channel), where dynamic means that the number of super-net channels might change based on the sampled architecture, and the type of optimum that is known for the search space (Optimal). We further provide the maximum number of nodes (n), excluding the input and output nodes, in each cell, as well as a bound on the number of

shared weights (Param.) and edge connections (Edges). Finally, the search spaces differ in how the nodes aggregate their inputs if they have multiple incoming edges (Merge).

A.3 KENDALL TAU V.S. SPEARMAN RANKING CORRELATION (SPR)

Kendall-tau is not the only metric to evaluate the ranking correlation. Spearman ranking correlation is also widely adopted in this field (Guo et al., 2019; Dong & Yang, 2020). Note that our idea of sparsity also applies to SpR. In Table 2, we compare the performance of KdT, SpR and their sparse variants, in the same setting as Figure 4 in the main paper. Note that SpR and KdT performs similarly but that their sparse variants effectively improve the correlation on all search spaces.

Corr. of Perf.	KdT.	S-KdT	SpR	S-SpR
NASBench-101	0.29	0.45	0.23	0.41
NASBench-201	0.42	0.55	0.38	0.57
DARTS-NDS	0.08	0.19	0.09	0.20

Table 2: Comparison of Kendall Tau (KdT) and Spearman ranking (SpR) with their sparse variants.

A.4 SPARSE KENDALL-TAU IMPLEMENTATION DETAILS

To compute the sparse Kendall-Tau we need access to two quantities: 1) the performance of the sampled architectures based on the trained super-net; and 2) the associated ground-truth performances. For each architecture in 1), we compute the average top-1 accuracy over $n = 3$ super-nets (that were trained with different random initialization) to improve the stability of the evaluation. We round the ground-truth top-1 accuracy to a precision of 0.1% for each sampled architecture to obtain the ground-truth performance 2). We then rank the architectures in 1) and 2) and compute the Kendall-Tau rank coefficient Kendall (1938) between the two ranked lists.

Sparse Kendall-Tau threshold. This value should be chosen according to what is considered a significant improvement for a given task. For CIFAR-10, where accuracy is larger than 90%, we consider a 0.1% performance gap to be sufficient. For tasks with smaller state-of-the-art performance, larger values might be better suited.

Number of architectures. In practice, we observed that the sparse Kendall-Tau metric became stable and reliable when using at least $n = 150$ architectures. We used $n = 200$ in our experiments to guarantee stability and fairness of the comparison of the different factors.

A.5 LIMITATION OF SPARSE KENDALL-TAU

We nonetheless acknowledge that our sparse Kendall-Tau has some limitations. For example, a failure case of using sparse Kendall-Tau for super-net evaluation may occur when the top 10% architectures are perfectly ordered, while the bottom 90% architectures are purely randomly distributed. In this case, the Kendall Tau will be close to 0. However, the search algorithm will always return the best model, as desired.

Nevertheless, while this corner case would indeed be problematic for the standard Kendall Tau, it can be circumvented by tuning the threshold of our sKdT. A large threshold value will lead to a small number of groups, whose ranking might be more meaningful. For instance in some randomly-picked NASBench-101 search processes, setting the threshold to 0.1% merges the top 3000 models into 9 ranks, but still yields an sKdT of only 0.2. Increasing the threshold to 10% clusters the 423K models into 3 ranks, but still yields an sKdT of only 0.3. This indicates the stability of our metric.

In Figure 2, we randomly picked 12 settings and show the corresponding bipartite graphs relating the super-net and ground-truth rankings to investigate where disorder occurs. In practice, the corner case discussed above virtually never occurs; the ranking disorder is typically spread uniformly across the architectures.

B EXPERIMENTAL DETAILS

Below, we provide additional details about our implementations and settings. We also report the best settings in Table 3.

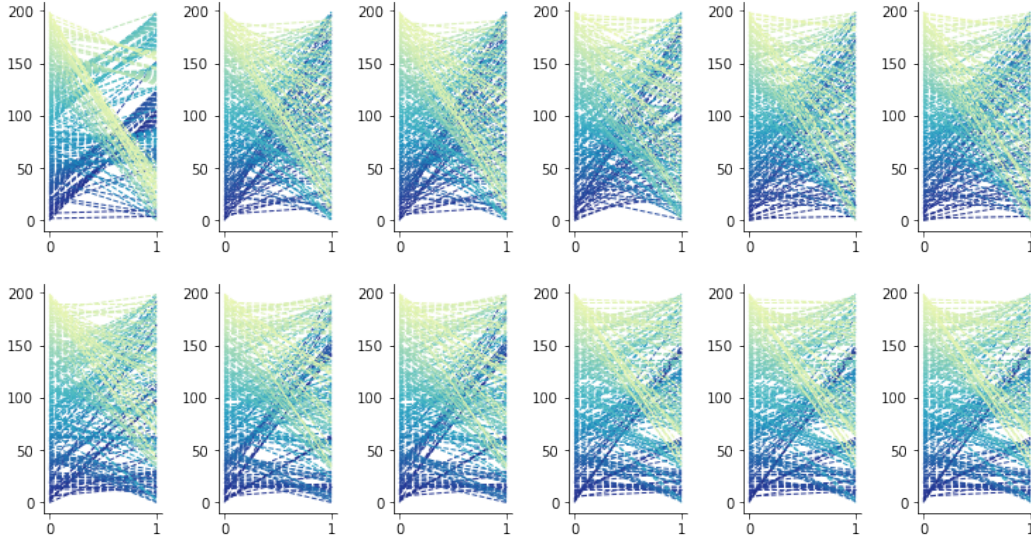


Figure 2: **Ranking disorder examples.** We randomly select 12 runs from our experiments. For each sub-plot, 0 indicates the architecture ground-truth rank, and 1 indicates the ranking according to their super-net accuracy. We can clearly see that the ranking disorder happens uniformly across the search space and does not follow a particular pattern.

Table 3: Parameter settings that obtained the best searched results.

Search Space	implementation					low fidelity				hyperparam.				sampling
	Dynamic Conv	OFA Conv	WSBN	Dropout	Op map	# layer	portion	batch-size	# channels	batch-norm	learning rate	epochs	weight decay	
NASBench-101	Interpolation	N	N	0.	Node	9	0.75	256	128	Tr=F A=T	0.025	400	1e-3	FairNAS
NASBench-201	Fix	N	N	0.	Edge	5	0.9	128	16	Tr=F A=T	0.025	1000	3e-3	FairNAS
DARTS-NDS	Fix	N	Y	0.	Edge	12	0.9	256	36	Tr=F A=F	0.025	400	0	FairNAS

For batch-norm, we report Track statistics (Tr) and Affine (A) setting with True (T) or False (F).
For other notation, Y = Yes, N = No.

B.1 TRAINING DETAILS

We use PyTorch (Paszke et al., 2019) for our experiments. Since NASBench-101 was constructed in TensorFlow we implement a mapper that translates TensorFlow parameters into our PyTorch model. We exploit two large-scale experiment management tools, SLURM (Slurm, 2020) and Kubernetes (Kubernetes, 2020), to deploy our experiments. We use various GPUs throughout our project, including NVIDIA Tesla V100, RTX 2080 Ti, GTX 1080 Ti and Quadro 6000 with CUDA 10.1. Depending on the number of training epochs, parameter sizes and batch-size, most of the super-net training finishes within 12 to 24 hours, with the exception of FairNAS, whose training time is longer, as discussed earlier. We split the data into training/validation using a 90/10 ratio for all experiments, except those involving validation on the training portion. Please consult our submitted code for more details.

B.2 COST OF COMPUTING THE STAND-ALONE MODEL PERFORMANCE

Computing the final accuracy is more expensive than training the super-net. Despite the low-fidelity heuristics reducing the weight-sharing costs, training a stand-alone network to convergence has higher cost, e.g., DARTS searches for 50 epochs but trains from scratch for 600 epochs (Liu et al., 2019). Furthermore, debugging and hyper-parameter tuning typically require training thousands of stand-alone models. Note that, as one typically evaluates a random subset of architectures to understand the design space (Radosavovic et al., 2019), sparse Kendall-Tau can be computed without additional costs. In any event, the budget for sparse Kendall-Tau is bounded with n .

B.3 STAND-ALONE ACCURACY V.S. SPARSE KENDALL-TAU

A common misconception is that the super-net quality is well reflected by stand-alone accuracy of the final selected architecture. Neither sparse Kendall-Tau (sKdT) nor stand-alone accuracy (SAA)

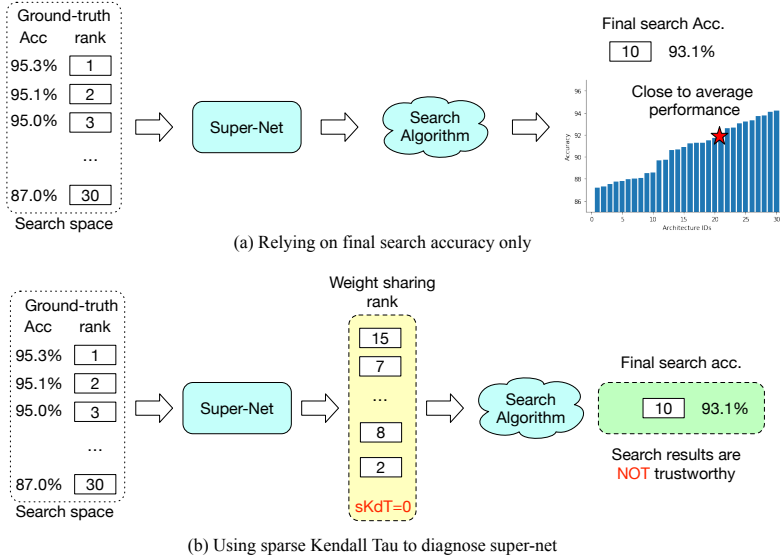


Figure 3: **Comparing sparse Kendall-Tau and final search accuracy.** Here, we provide a toy example to illustrate why one cannot rely on the final search accuracy to evaluate the quality of the super-net. Let us consider a search space with only 30 architectures, whose accuracy ranges from 95.3% to 87% on the CIFAR-10 dataset, and we run a search algorithm on top. (a) describes a common scenario: we run the search for multiple times, yielding a best architecture with 93.1% accuracy. While this may seem good, it does not give any information about the quality of the search or the super-net. If we had full knowledge about the performance of every architecture in this space, we would see that this architecture is close to the average performance and hence no better than random. In (b), the sparse Kendall-Tau allows us to diagnose this pathological case. A small sparse Kendall-Tau implies that there is a problem with super-net training.

are perfect. Both are tools to measure different aspects of a super-net. Below, we discuss this in more detail.

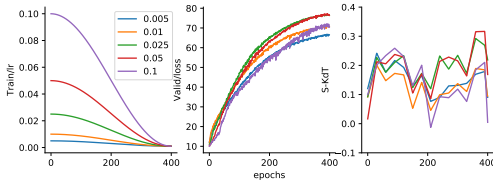
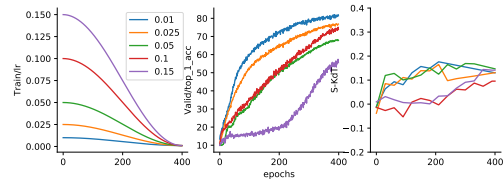
Let us consider a completely new search space in which we have no prior knowledge about performance. As depicted by Figure 3, if we only rely on the SAA, the following situation might happen: Due to the lack of knowledge, the ranking of the super-net is purely random, and the search space accuracy is uniformly distributed. When trying different settings, there will be 1 configuration that ‘outperforms’ the others in terms of SAA. However, this configuration will be selected by pure chance. By only measuring SAA, it is technically impossible to realize that the ranking is random. By contrast, if one measures the sKdT (which is close to 0 in this example), an ill-conditioned super-net can easily be identified. In other words, purely relying on SAA could lead to pathological outcomes that can be avoided using sKdT.

Additionally, SAA is related to both the super-net and the search algorithm. sKdT allows us to judge super-net accuracy independently from the search algorithm. As an example, consider the use of a reinforcement learning algorithm, instead of random sampling, on top of the super-net. When observing a poor SAA, one cannot conclude if the problem is due to a poor super-net or to a poor performance of the RL algorithm. Prior to our work, people relied on the super-net accuracy to analyze the super-net quality. This is not a reliable metric, as shown in Fig. 9 in the main paper. We believe that sKdT is a better alternative.

C ADDITIONAL RESULTS

C.1 WEIGHT-SHARING PROTOCOL P_{ws} – OTHER FACTORS

We report results of additional factors such as the number of training epochs, weight decay and path sampling here. In summary, our experiments show that more training epochs positively influence

Figure 4: **Learning rate** on NASBench-101.Figure 5: **Learning rate** on DARTS-NDS.

super-net quality. The behavior of weight decay varies across datasets, and one cannot simply disable it as suggested by Nayman et al. (2019).

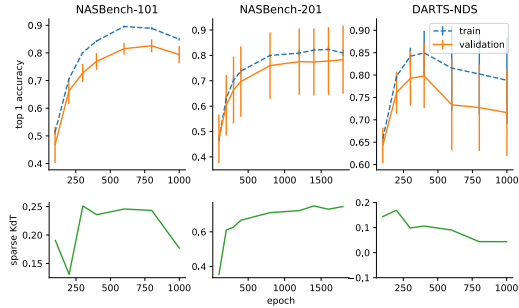
Learning rate.

We report the learning rate validation results for NASBench-101 in Figure 4 and for DARTS-NDS in Figure 5. For NASBench-101, we can see that learning rates of 0.025 and 0.05 clearly outperform

other learning rates in terms of sparse Kendall-Tau and validation accuracy. For DARTS-NDS, although the best validation accuracy is obtained with a learning rate of 0.01, the sparse Kendall-Tau suggests that there is no significant difference once the learning rate is below 0.025, which is the stand-alone training learning rate. We pick 0.025 to be consistent with the other search spaces.

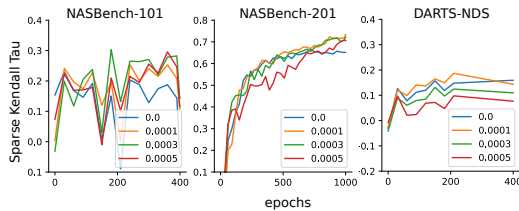
Number of epochs.

Since the cosine learning rate schedule decays the learning rate to zero towards the end of training, we evaluate the impact of the number of training epochs. In stand-alone training, the number of epochs was set to 108 for NASBench-101, 200 for NASBench-201, and 100 for DARTS-NDS. Figure 6 shows that increasing the number of epochs significantly improves the accuracy in the beginning, but eventually decreases the accuracy for NASBench-101 and DARTS-NDS. Interestingly, the number of epochs impacts neither the correlation of the ranking nor the final selected model performance after 400 epochs. We thus use 400 epochs for the remaining experiments.

Figure 6: **Validating the number of epochs.** Each data point summarizes 3 individual runs.

Weight decay.

Weight decay is used to reduce overfitting. For WS-NAS, however, overfitting does not occur because there are billions of architectures sharing the same set of parameters, which in fact rather causes underfitting. Based on this observation, Nayman et al. (2019) propose to disable weight decay during super-net training. Figure 7, however, shows that the behavior of weight decay varies across datasets. While on DARTS-NDS weight decay is indeed harmful, it improves the results on NASBench 101 and 201. We conjecture that this is due to the much larger number of architectures in DARTS-NDS (243 billion) than in the NASBench series (less than 500,000).

Figure 7: **Weight decay validation.**

C.2 WEIGHT-SHARING PROTOCOL P_{ws} – PATH SAMPLING

Aside from the Random-NAS described in the main paper, we additionally include two variants of Random-NAS: 1) As pointed out by Ying et al. (2019), two super-net architectures might be topologically equivalent in the stand-alone network by simply swapping operations. We thus include architecture-aware random sampling that ensures equal probability for unique architectures (Yu

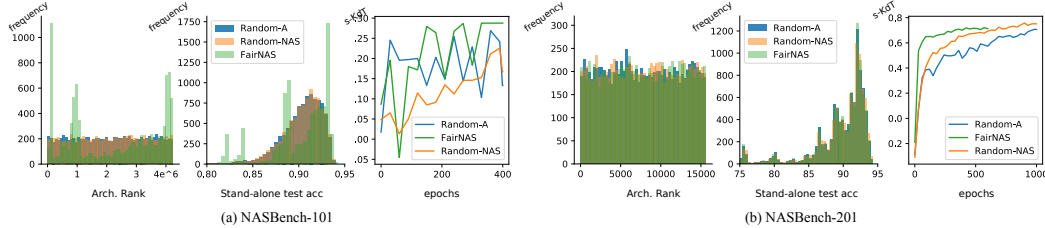


Figure 8: **Path sampling comparison on NASBench-101 (a) and NASBench-201 (b).** We sampled 10,000 architectures using different samplers and plot histograms of the architecture rank and the stand-alone test accuracy. We plot the s-KdT across the epochs. Results averaged across 3 runs.

et al., 2020). We name this variant Random-A; 2) We evaluate a variant called FairNAS (Chu et al., 2019), which ensures that each operation is selected with equal probability during super-net training. Although FairNAS was designed for a search space where only operations are searched but not the topology, we adapt it to our setting (see Appendix A.1 for details).

With the hyper-parameters fixed, we now compare three path-sampling techniques. Since DARTS-NDS does not contain enough samples trained in a stand-alone manner, we only report results on NASBench-101 and 201. In Figure 8, we show the sampling distributions of different approaches and the impact on the super-net in terms of sparse Kendall-Tau. These experiments reveal that, on NASBench-101, uniformly randomly sampling one architecture, as in Li & Talwalkar (2019); Yu et al. (2020), is strongly biased in terms of accuracy and ranking. This can be observed from the peaks around rank 0, 100,000, and 400,000. The reason is that a single architecture can have multiple encodings, and uniform sampling thus oversamples such architectures with equivalent encodings. FairNAS samples architectures more evenly and yields consistently better sparse Kendall-Tau values, albeit by a small margin.

On NASBench-201, the three sampling policies have a similar coverage. This is because, in NASBench-201, topologically-equivalent encodings were not pruned. In this case, Random-NAS performs better than in NASBench-101, and FairNAS yields good early performance but quickly saturates. In short, using different sampling strategies might in general be beneficial, but we advocate for FairNAS in the presence of a limited training budget.

C.3 MAPPING f_{ws} – DYNAMIC CHANNELING

Other dynamic channeling methods. In NASBench-101, the output cell concatenates the feature maps from previous nodes. However, the concatenation has a fixed target size, which requires the number of output channels in the intermediate nodes to be dynamically adapted during super-net training. To model this, we initialize the super-net convolution weights so as to accommodate the largest possible number of channels c_{max} , and reduce it dynamically to c output channels using one of the following heuristics: 1) Use a fixed chunk of weights, $[0 : c]$ (Guo et al., 2019); 2) Shuffle the channels before applying 1) (Zhang et al., 2018); 3) Linearly interpolate the c_{max} channels into c channels via a moving average across the neighboring channels. The strategies are compared in Table 2 in the main paper. Shuffling the channels drastically degrades all metrics. Interpolation yields a lower super-net accuracy than using a fixed chunk, but improves the other metrics. Altogether, interpolation comes out as a more robust solution.

Random subspace baseline. Since each sub-space now encompasses fewer architectures, it is not fair to perform a comparison with the full NASBench 101 search space. To this end, for each sub-space, we construct a baseline space where we drop architectures uniformly at random until the number of remaining architectures matches the size of the sub-space. We repeat this process with 3 different initializations, while keeping all other factors unchanged when training the super-net. We refer to this as ‘Baseline’ in Table 2 in the main paper. Table 6 indicates the number of architectures in each such sub-spaces. Here we provide additional result in Table 4 for each individual sub-space and shows the sparse Kendall-Tau remains similar as the baseline using the full search space, which clearly evidence the effectiveness of our approach to disable the dynamic channeling.

Table 6: NASBench-101 sub-spaces obtained by fixing the number of channels.

# Incoming Edge	# Arch.
1	120933
2	201441
3	90782
4	10467

Table 4: **Disabling dynamic channels**

Edges	Accuracy	S-KdT	P > R	Final searched model
<i>Baseline: random sampling sub-spaces with dynamic channeling.</i>				
1	70.04± 8.15	0.173	0.797	91.19±2.01
2	78.29±10.51	0.206	0.734	82.03±1.50
3	79.92± 9.42	0.242	0.576	92.20±1.19
4	79.37± 17.34	0.270	0.793	92.32±1.10
Average	76.905 ± 10.05	0.223	0.865	89.435 ± 4.30
<i>Disable dynamic channels by fixing the edges to the output node.</i>				
1	76.92± 7.87	0.435	0.991	93.94±0.22
2	74.32± 8.21	0.426	0.925	93.34±0.01
3	77.24± 9.18	0.487	0.901	93.66±0.07
4	79.31± 7.04	0.493	0.978	93.65±0.07
Average	76.95 ± 8.29	0.460	0.949	93.65 ± 0.73

Table 5: **Comparison of different mappings f_{ws}** . We report s-KdT / final search performance.

	NASBench-101	NASBench-201	DARTS-NDS
Baseline	0.236 / 92.32	0.740 / 92.92	0.159 / 93.59
WSBN	0.056 / 91.33	0.675 / 92.04	0.331 / 92.95
Global-Dropout	0.179 / 90.95	0.676 / 91.76	0.102 / 92.30
Path-Dropout	0.128 / 91.19	0.431 / 91.42	0.090 / 91.90
OFA Kernel	0.132 / 92.01	0.574 / 91.83	0.112 / 92.83

C.4 MAPPING f_{ws} – OTHER FACTORS

We evaluate the weight-sharing batch normalization (WSBN) of Luo et al. (2018b), which keeps an independent set of parameters for each incoming edge. Furthermore, we test the two commonly-used dropout strategies: right before global pooling (global dropout); and at all edge connections between the nodes (path dropout). Note that path dropout has been widely used in WS-NAS Luo et al. (2018a); Liu et al. (2019); Pham et al. (2018). For both dropout strategies, we set the dropout rate to 0.2. Finally, we evaluate the super convolution layer of Cai et al. (2020), referred to as OFA kernel, which accounts for the fact that, in CNN search spaces, convolution operations appear as groups, and thus merges the convolutions within the same group, keeping only the largest kernel parameters and performing a parametric projection to obtain the other kernels. The results in Table 5 show that all these factors negatively impact the search performances and the super-net quality.

C.5 WS ON EDGES OR NODES?

Most existing works build f_{ws} to define the shared operations on the graph nodes rather than on the edges. This is because, if f_{ws} maps to the edges, the parameter size increases from $O(n)$ to $O(n^2)$, where n is the number of intermediate nodes. We provide a concrete example in Figure 9. However, the high sparse Kendall-Tau on NASBench-201 in the top part of Table 7, which is obtained by mapping to the edges, may suggest that sharing on the edges is beneficial. Here we investigate if this is truly the case.

On NASBench-101, by design, each node merges the previous nodes’ outputs and then applies parametric operations. This makes it impossible to build an equivalent sharing on the edges. We therefore construct sharing on the edges for DARTS-NDS and sharing on the nodes for NASBench-201. As shown in Table 5, for both spaces, sharing on the edges yields a marginally better super-net than sharing on the nodes. Such small differences might be due to the fact that, in both spaces, the number of nodes is 4, while the number of edges is 6, thus mapping to edges will not drastically affect the number of parameters. Nevertheless, this indicates that one should consider having a larger number of shared weights when the resources are not a bottleneck.

Table 7: **Comparison of operations on the nodes or on the edges**. We report sKT / final search performance.

	NASBench-101	NASBench-201	DARTS-NDS
Baseline	0.236 / 92.32	0.740 / 92.92	0.159 / 93.59
Op-Edge	N/A	as Baseline	0.189 / 93.97
Op-Node	as Baseline	0.738 / 92.36	as Baseline

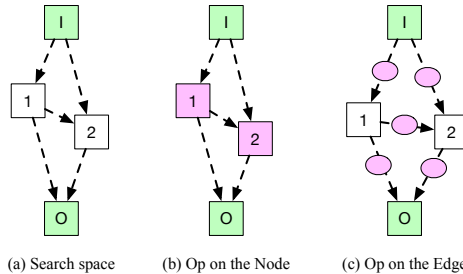


Figure 9: **(a)** Consider a search space with 2 intermediate nodes, 1, 2, with one input (I) and output (O) node. This yields 5 edges. Let us assume that we have 4 possible operations to choose from, as indicated as the purple color code. **(b)** When the operations are on the nodes, there are 2×4 ops to share, i.e., $I \rightarrow 2$ and $1 \rightarrow 2$ share weights on node 2. **(c)** If the operations are on the edges, then we have 5×4 ops to share.

C.6 RESULTS FOR ALL FACTORS

We report the numerical results for all hyper-parameter factors in Table 8, low-fidelity factors in Table 9 and implementation factors in Table 10. These results were computed from the last epochs of 3 different runs, as those reported in the main text.

REFERENCES

- Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HylxE1HKwS>.
- Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *arXiv:1907.01845*, 2019. URL <http://arxiv.org/abs/1907.01845>.
- Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HJxyZkBKDr>.
- Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single Path One-Shot Neural Architecture Search with Uniform Sampling. *arXiv:1904.00420*, 2019. URL <http://arxiv.org/abs/1904.00420>.
- Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.
- Kubernetes. kubernetes.io, 2020. URL <https://kubernetes.io/docs/reference/>.
- Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv:1902.07638*, 2019.
- Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. *International Conference on Learning Representations*, 2019.
- Renqian Luo, Fei Tian, Tao Qin, En-Hong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, 2018a.
- Renqian Luo, Fei Tian, Tao Qin, En-Hong Chen, and Tie-Yan Liu. Weight sharing batch normalization code, 2018b. URL https://www.github.com/renqianluo/NAO_pytorch/NAO_V2/operations.py#L144.
- Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik. Xnas: Neural architecture search with expert advice. In *Advances in Neural Information Processing Systems*, 2019.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8026–8037. Curran Associates, Inc., 2019.
- Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *International Conference on Machine Learning*, 2018.
- Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On Network Design Spaces for Visual Recognition. In *International Conference on Computer Vision*, 2019.
- Slurm. Slurm workload manager, 2020. URL <https://slurm.schedmd.com/documentation.html>.
- Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. *arXiv:1902.09635*, 2019.

Table 8: Results for all WS Protocol P_{ws} factors on the three search spaces.

Factor and settings	NASBench-101				NASBench-201				DARTS-NDS			
	Super-net Accuracy	S-KdT	P > R	Final Performance	Super-net Accuracy	S-KdT	P > R	Final Performance	Super-net Accuracy	S-KdT	P > R	Final Performance
Batch-norm.												
affine F track F	0.651±0.05	0.161	0.996	0.916±0.13	0.660±0.13	0.783	0.997	92.67±1.21	0.735±0.18	0.056	0.224	93.14±0.28
affine T track F	0.710±0.04	0.240	0.996	0.924±0.01	0.713±0.14	0.718	0.707	91.71±1.05	0.265±0.21	-0.071	0.213	91.89±2.01
affine F track T	0.144±0.09	0.084	0.112	0.882±0.02	0.182±0.15	-0.171	0.583	86.41±4.84	0.359±0.25	-0.078	0.023	90.33±0.76
affine T track T	0.153±0.10	-0.008	0.229	0.905±0.01	0.134±0.09	-0.417	0.274	90.77±0.40	0.216±0.18	-0.050	0.109	90.49±0.32
Learning rate.												
0.005	0.627±0.07	0.091	0.326	0.908±0.01	0.658±0.11	0.668	0.141	90.14±0.55	0.792±0.08	0.130	0.033	91.81±0.68
0.01	0.668±0.06	0.095	0.546	0.919±0.00	0.713±0.12	0.670	0.711	91.21±1.18	0.727±0.05	0.131	0.258	92.86±0.64
0.025	0.715±0.05	0.220	0.910	0.917±0.01	0.659±0.13	0.665	0.844	92.42±0.58	0.656±0.14	0.218	0.299	93.42±0.20
0.05	0.727±0.05	0.143	0.905	0.911±0.02	0.631±0.14	0.594	0.730	92.02±0.70	0.623±0.04	0.147	0.489	91.70±0.33
0.1	0.690±0.07	0.005	0.905	0.909±0.02	0.609±0.28	0.571	0.618	91.82±0.81	0.735±0.06	0.096	0.099	92.73±0.24
0.15	0.000±0.00	-0.274	N/A	N/A	0.551±0.14	0.506	0.553	91.22±1.20	0.371±0.27	0.027	0.218	91.20±0.72
0.2	-	-	-	-	0.519±0.12	0.557	0.035	88.74±0.11	0.102±0.48	-0.366	N/A	N/A
Epochs.												
100	0.468±0.07	0.190	0.759	0.920±0.01	0.472±0.09	0.355	0.997	92.11±1.67	0.643±0.04	0.144	0.901	93.90±0.49
200	0.662±0.05	0.131	0.685	0.914±0.01	0.604±0.12	0.610	0.881	91.88±2.01	0.761±0.05	0.169	0.778	94.08±0.21
300	0.727±0.03	0.251	0.739	0.920±0.01	0.664±0.13	0.627	0.840	91.42±1.91	0.793±0.06	0.098	0.870	93.22±0.95
400	0.769±0.03	0.236	0.932	0.921±0.01	0.697±0.14	0.667	0.158	89.83±0.97	0.798±0.07	0.106	0.036	92.34±0.22
600	0.815±0.02	0.246	0.556	0.911±0.01	0.720±0.13	0.682	0.285	90.28±0.82	0.734±0.10	0.090	0.209	93.23±0.19
800	0.826±0.02	0.243	0.177	0.907±0.00	0.760±0.13	0.711	0.378	91.53±0.53	0.728±0.10	0.044	0.853	93.29±0.81
1000	0.794±0.03	0.177	0.831	0.920±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.717±0.09	0.044	0.997	93.92±0.90
1200	-	-	-	-	0.775±0.13	0.723	0.198	90.81±0.56	-	-	-	-
1400	-	-	-	-	0.774±0.13	0.750	0.604	92.26±0.33	-	-	-	-
1600	-	-	-	-	0.778±0.13	0.731	0.882	91.85±1.20	-	-	-	-
1800	-	-	-	-	0.783±0.13	0.746	0.266	90.64±0.82	-	-	-	-
Weight decay.												
0.0	0.645±0.05	-0.037	0.179	0.899±0.01	0.713±0.13	0.652	0.266	90.58±0.99	0.670±0.03	0.159	0.629	93.09±0.73
0.0001	0.719±0.03	0.109	0.659	0.912±0.01	0.756±0.13	0.734	0.612	91.88±0.59	0.751±0.05	0.143	0.396	93.37±0.44
0.0003	0.771±0.03	0.144	0.648	0.915±0.01	0.772±0.13	0.721	0.726	92.34±0.57	0.759±0.06	0.110	0.890	93.82±0.51
0.0005	0.782±0.03	0.117	0.910	0.911±0.02	0.764±0.13	0.705	0.882	92.61±0.59	0.739±0.07	0.077	0.051	91.61±1.01
Sampling.												
Random-A	0.717±0.04	0.133	0.862	0.919±0.02	0.764±0.13	0.705	0.882	92.61±0.59	-	-	-	-
Random-NAS	0.638±0.20	0.167	0.949	0.913±0.02	0.765±0.14	0.750	0.897	92.17±1.01	-	-	-	-
FairNAS	0.789±0.03	0.288	0.382	0.908±0.01	0.774±0.14	0.713	0.917	93.06±0.31	-	-	-	-

Table 9: Results for all low-fidelity factors on the three search spaces.

Factor and settings	NASBench-101				NASBench-201				DARTS-NDS			
	Super-net Accuracy	S-KdT	P > R	Final Performance	Super-net Accuracy	S-KdT	P > R	Final Performance	Super-net Accuracy	S-KdT	P > R	Final Performance
Number of Layer (-X indicates the baseline minus X)												
Baseline	0.769±0.03	0.236	0.932	0.921±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.670±0.03	0.159	0.629	93.09±0.73
-1	0.759±0.03	0.214	0.222	0.901±0.01	0.749±0.13	0.710	0.796	91.85±0.92	0.843±0.04	0.178	0.299	92.35±1.25
-2	0.817±0.03	0.228	0.713	0.910±0.02	0.777±0.13	0.700	0.822	92.68±0.37	0.852±0.03	0.205	0.609	92.65±1.89
Train portion												
0.25	0.433±0.07	0.216	0.281	0.901±0.01	0.660±0.11	0.668	0.979	92.30±1.14	0.597±0.14	0.132	0.359	92.27±1.84
0.5	0.612±0.06	0.251	0.424	0.896±0.02	0.740±0.12	0.669	0.979	93.17±0.47	0.666±0.17	0.083	0.551	92.22±1.36
0.75	0.688±0.05	0.222	0.857	0.920±0.01	0.758±0.13	0.725	0.618	92.46±0.19	0.715±0.18	0.096	0.081	92.29±0.47
0.9	0.722±0.05	0.186	0.996	0.931±0.01	0.772±0.13	0.721	0.726	92.34±0.57	0.703±0.18	0.042	0.065	92.78±0.10
Batch size (/ X indicates the baseline divide by X)												
Baseline	0.769±0.03	0.236	0.932	0.921±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.670±0.03	0.159	0.629	93.09±0.73
/2	0.670±0.05	0.246	0.807	0.920±0.01	0.728±0.16	0.719	0.842	92.37±0.61	0.698±0.20	0.037	0.209	93.24±0.13
/4	0.686±0.07	0.155	0.913	0.921±0.01	0.703±0.16	0.679	0.672	92.35±0.34	0.633±0.20	0.033	0.690	93.68±0.62
# channel (/ X indicates the baseline divide by X)												
Baseline	0.769±0.03	0.236	0.932	0.921±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.670±0.03	0.159	0.629	93.09±0.73
/2	0.658±0.05	0.156	0.704	0.898±0.02	0.697±0.14	0.667	0.158	89.83±0.97	0.776±0.05	0.190	0.993	93.90±0.71
/4	0.604±0.06	0.093	0.907	0.922±0.01	0.606±0.13	0.616	0.878	92.86±0.34	0.707±0.05	0.202	0.359	92.93±0.58

Table 10: Results for all implementation factors on the three search spaces.

Factor and settings	NASBench-101				NASBench-201				DARTS-NDS			
	Super-net Accuracy	S-KdT	P > R	Final Performance	Super-net Accuracy	S-KdT	P > R	Final Performance	Super-net Accuracy	S-KdT	P > R	Final Performance
Other factors												
Baseline	0.769±0.03	0.236	0.932	0.921±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.670±0.03	0.159	0.629	93.09±0.73
OFA Kernel	0.708±0.08	0.132	0.203	0.921±0.19	0.672±0.18	0.574	0.605	91.83±0.86	0.782±0.05	0.112	0.399	93.22±0.43
WSBN	0.155±0.07	0.085	0.504	0.809±0.13	0.703±0.14	0.676	0.585	92.06±0.48	0.744±0.16	0.033	0.682	92.88±1.22
Path dropout rate												
Baseline	0.769±0.03	0.236	0.932	0.921±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.670±0.03	0.159	0.629	93.09±0.73
0.05	0.750±0.02	0.206	0.819	0.915±0.07	0.490±0.09	0.712	0.881	92.25±0.89	0.184±0.06	0.006	0.359	92.93±0.60
0.15	0.726±0.02	0.186	0.482	0.910±0.01	0.250±0.03	0.640	0.526	91.44±1.25	0.366±0.05	0.059	0.570	92.61±1.28
0.2	0.669±0.01	0.110	0.282	0.901±0.01	0.185±0.02	0.431	0.809	92.15±0.85	0.518±0.06	0.090	0.009	91.45±0.58
Global dropout												
Baseline	0.769±0.03	0.236	0.932	0.921±0.01	0.782±0.13	0.740	0.589	92.92±0.48	0.670±0.03	0.159	0.629	93.09±0.73
0.2	0.739±0.05	0.233	0.221	0.910±0.00	0.712±0.13	0.702	0.950	91.76±1.36	0.557±0.19	0.018	0.451	93.51±0.27

Please refer to Appendix C.5 for mapping on the node or edge and Appendix C.3 for dynamic channel factor results.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=H1loF2NFwr>.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Conference on Computer Vision and Pattern Recognition*, 2018.