DARTS for Inverse Problems: a Study on Stability

Anonymous Author(s)
Affiliation
Address
email

Abstract

Differentiable architecture search (DARTS) is a widely researched tool for neural architecture search, due to its promising results for image classification. The main benefit of DARTS is the effectiveness achieved through the weight-sharing one-shot paradigm, which allows efficient architecture search. In this work, we investigate DARTS in a systematic case study of inverse problems, which allows us to analyze these potential benefits in a controlled manner. Although we demonstrate that the success of DARTS can be extended from classification to reconstruction, our experiments yield a fundamental difficulty in the evaluation of DARTS-based methods: The results show a large variance in all test cases and the weight-sharing performance of the architecture found during training does not always reflect its final performance. We conclude the necessity to 1) report the results of any DARTS-based methods from several runs along with its underlying performance statistics and 2) show the correlation between the training and final architecture performance.

1 Introduction

Recent progress in computer vision and related fields has illustrated the importance of suitable neural architecture designs and training schemes [4, 5, 1]. Neural architecture search (NAS) is the task of optimizing the architecture of a neural network automatically without resorting to human selection, scaling to larger search spaces with the promise of novel, improved architectures. To avoid the computational costs of training candidate architectures for black-box optimization models [16, 17, 6, 13, 11, 10], differentiable architecture search (DARTS) [9] proposes a continuous relaxation of the search problem, i.e. all candidate architectures within a given search space are jointly optimized using shared network parameters while the network also learns to weigh these operations. The final architecture can then be deduced by selecting the highest weighted operations. This is appealing as practically good architectures are proposed within a single optimization run. However, previous works such as [14] also indicate that the proposed results are often sub-optimal, especially when the search space is not well chosen (e.g. contains sub-optimal operations). Specifically, since network weights are randomly initialized, promising operations can have poor initial weights such that the architecture optimization tends to entirely discard them. As a result, the practical relevance of DARTS-proposed architectures depends heavily on network initialization.

Here, we apply DARTS to inverse problems, which have hardly been addressed in NAS, and allow us the choice of parameters and search space like a practitioner would do. Our main focus is the analysis of DARTS w.r.t. the stability of results. We investigate one-dimensional inverse problems which allow to conduct several evaluation runs to analyze the robustness of DARTS. We show that DARTS can automatically find well performing architectures in our setup, if the search space is well preconditioned. Yet, importantly, we find that the estimated network performance using jointly optimized, shared weights is often not well correlated with the reconstruction ability of the final model after operation selection and re-training, i.e. the relaxation in DARTS seems to be quite loose.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

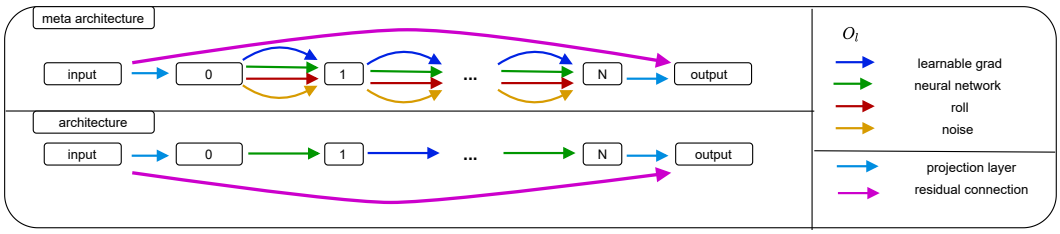

Figure 1: Proposed meta-architecture which can represent DnCNN [15]-like architectures.

2 DARTS

We introduce DARTS [9] in a more detailed way in the following, as it is the technical basis of our analysis. While the originally proposed method optimizes so-called *cells*, which are stacked in order to define the overall neural network architecture, and defines each cell in the form of a directed acyclic graph (DAG), we conduct our study of the behavior of DARTS on special, sequential and easy-to-interpret meta-architectures to be described below. Our sequential architecture merely consists of N nodes $x^{(i)}$, where $x^{(0)}$ represents the input data and the result $x^{(j+1)}$ of any layer is computed by applying some operation $o^{(j)}$ to the predecessor node $x^{(j)}$, i.e.,

$$x^{(j+1)} = o^{(j)}(x^{(j)}, \theta^{(j)}), \tag{1}$$

where $\theta^{(j)}$ are the (learnable) parameters of operation $o^{(j)}$. To determine which operation $o^{(j)}$ is most suitable to be applied to the feature $x^{(j)}$, one defines a set of candidate operations $o_t \in \mathcal{O}, t \in \{1, \dots, \mid \mathcal{O} \mid =: T\}$ and searches over the continuous relaxation of Eq. (1) for

$$o^{(j)} = \sum_{t=1}^{T} \beta_{o_t}^{(j)} o_t, \qquad \beta_{o_t}^{(j)} = \frac{\exp(\alpha_{o_t}^{(j)})}{\sum_{t'=1}^{T} \exp(\alpha_{o_{t'}}^{(j)})} \tag{2}$$

where $\alpha = (\alpha_{o_t}^{(j)})$ are *architecture parameters* that determine the selection of exactly one candidate operation in the limit of β becoming binary. Instead of looking for binary parameters directly, the optimization is relaxed to the soft-max of continuous parameters α. DARTS formulates this search as a bi-level optimization problem in which both, the network parameters $\theta = \{\theta^{(j)}\}_{j=1}^{T}$ and the architecture parameters α, are jointly optimized on the training and validation set, respectively, via

$$\min_{\alpha} \mathcal{L}_{val}(\theta(\alpha), \alpha) \tag{3}$$

$$\text{s.t. } \theta(\alpha) \in \arg\min_{\theta} \mathcal{L}_{train}(\theta, \alpha), \tag{4}$$

where $\mathcal{L}_{val}$ and $\mathcal{L}_{train}$ denote suitable loss functions for the validation and training data. The optimization is done by approximating (4) by one (or zero) iterations of gradient descent. After optimization, the discrete architecture is obtained by choosing $\hat{o}^{(j)} = \text{argmax}_{o_t} \alpha_{o_t}^{(j)}$ for each node. Subsequently, the final network given by the architecture eq. (1), but using $\hat{o}^{(j)}$ instead of $o^{(j)}$, is retrained from scratch. Thus, the fundamental assumption of DARTS is that the performance of the final network architecture on the validation set (the *architecture validation*) is highly correlated with the performance of the relaxed DARTS approach obtained in (3) (the *one-shot validation*).

3 Experimental Setup

We investigate the meta-architecture shown in fig. 1, which should be specifically well-suited for examples of signal recovery from known data formation processes such as blurring and subsampling with noise. We choose one operation out of several sequentially before adding the output to a residual branch. Image recovery networks such as DnCNN [15] are contained in this meta-architecture.

We search for the optimal architecture that can be defined as a sequence of operations selected from the set $\mathcal{O}_l$ which contains four operation types. Two of the four operation types in $\mathcal{O}_l$ are by design potentially beneficial operations: The first is a plain 2-layer convolutional network. The second is motivated from rolled-out architectures (see e.g. [3, 12, 7]) and tries to embed model-based knowledge about the recovery problems into the networks architecture. As all considered

Method	Data Formation	Architecture Validation (PSNR)		
		Max.	Mean	Med.
DARTS (good ops.)	Blur	23.46	21.56	21.60
DARTS (all ops.)	Blur	22.86	15.64	18.57
Random (all ops.)	Blur	20.86	9.45	8.10
Learnable Grad. only	Blur	17.45	16.36	16.49
Nets only	Blur	21.63	19.45	20.71
DARTS (good ops.)	Downsampling	18.03	16.36	16.66
DARTS (all ops.)	Downsampling	18.01	15.39	16.12
Random (all ops.)	Downsampling	13.78	5.08	4.31
Learnable Grad. only	Downsampling	14.35	13.24	13.55
Nets only	Downsampling	16.92	13.13	14.05

Table 1: Architecture validation PSNR values found for 1D inverse problems. Shown is the maximal, mean and median PSNR over 100 trials.

problems can be phrased as linear inverse problems in which a quantity x ought to be recovered from data $y = Ax +$ noise for a linear operator A, gradient descent on a quadratic data fidelity term yields updates $x^{k+1} = x^k - \tau A^T(Ax^k - y)$, which we augment by an additive 2-layer convolutional network to obtain a *learnable gradient descent layer*. To complement these beneficial layers we include two harmful operation types in $\mathcal{O}_l$, a layer that merely adds white Gaussian noise and a roll layer, which rolls the inputs in all dimensions, both of which the optimization should discard. We provide additional details for these operations in the supp. material. In total we set $\mathcal{O}_l = \{\text{learnable gradient descent}, \text{ 2-layer-CNN}, \text{ roll}, \text{ noise}\}$, and search for 10 successive layers. This allows an evaluation of the effectiveness of DARTS in two different cases: Training on all operations versus training only on beneficial operations. *A good differentiable architecture search algorithm should reliably find the optimal operations, even when presented with sub-optimal choices.*

For a fast test, we generate one-dimensional data sampling cosine waves of varying magnitude, amplitude and offset, and search for models to recover these samples from distorted measurements. We consider two distortion processes with varying difficulty: First, Gaussian noise and blurring and second, in addition to these, a subsampling by a factor of 4. For more details, we refer to the appendix. We evaluate the performance of these models on a validation set of previously unseen examples by computing their average peak signal-to-noise ratio (PSNR).

4 Evaluating DARTS

Plainly, DARTS *does* work for inverse problems as shown by results in table 1 in the *Max.* column, which shows the validation PSNR of the best architecture found over 100 trials: DARTS is able to propose good architectures in two different applications with a noticeably higher PSNR than either a random selection of operations or the choice of just one of the good operations. When comparing DARTS with all operations versus DARTS with only beneficial operations, then the best architectures in both settings are close. In particular, the best architecture DARTS found when searching over all operations did not contain *noise* or *roll*, which implies DARTS found one of the top 0.1% out of the 4^{10} many architectures in 100 trials.

Evaluating Robustness While DARTS is able to propose good architectures according to table 1, the assertion that it works well in this case study is predicated on the evaluation criterion of maximal PSNR over all trials. If we turn towards the second column in table 1 and evaluate mean PSNR scores the picture changes considerably. While DARTS with only good operations is stable, we see that the default of DARTS with all operations suffers considerably, dropping 7 PSNR values for the blur example and 3 PSNR for downsampling. Especially of note here is that DARTS performs *on average* worse than the maximum over 100 random architectures.

This prompts a conundrum. On the one hand, best benchmark results are achieved by running DARTS for as many trials as possible and returning the best found architecture, but on the other hand this setup treats the DARTS algorithm, originally a replacement of 0-th order NAS methods, itself as a component of an overarching NAS algorithm that sometimes proposes good architectures. While this strategy of repeatedly using DARTS to propose trials for NAS is clearly a feasible architecture

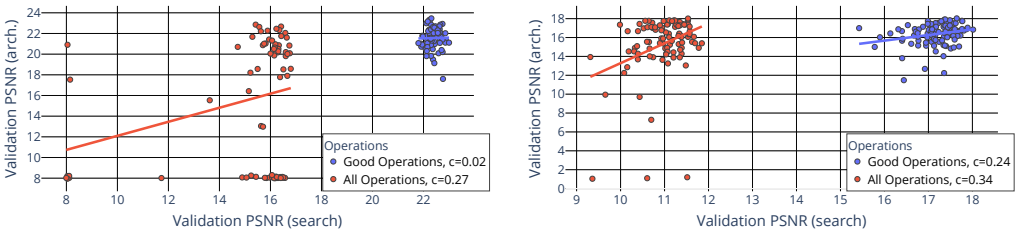

Figure 2: Scatter plot showing architecture PSNR (y-axis) plotted against 1-shot validation PSNR (i.e. the validation performance on the DARTS objective). Left: Blur. Right: Downsampling.

search tactic, it does fall short of the original goal, which is to find a good architecture within a single optimization of the weight-shared DARTS problem.

Correlation of Architecture and DARTS Performance Figure 2 takes a closer look at the trials considered in table 1, and plots for all trials separately the direct validation performance of the one-shot architecture (x-axis) versus the "true" architecture performance after retraining (y-axis). We also plot a regression line over all trials and report the correlation of all trials in the legend, showing the linear fit has limited expressiveness. As discussed above, the correlation of these quantities is a fundamental assumption of DARTS. Yet these plots show that DARTS' behavior is highly problem-dependent: The downsampling dataset (right) shows that, although the mean value of DARTS can be non-optimal, search performance and architecture performance are weakly correlated, even if the best architecture only has average search performance. The closely related blur dataset (left) shows a different behavior with 3 observable "failure" cases: Although the trials are nominally well correlated, we find a cluster of trials for which both search and architecture validation PSNR are low, a cluster where search performance is good, but architecture performance is low, and a cluster where architecture performance is high (close to DARTS with only good operations) but seemingly uncorrelated to search performance. In summary, the three problems we observe are that either DARTS fails directly, or that DARTS works but does not predict a useful architecture, or that DARTS does predict a useful architecture, but is unrelated to its search performance.

Discussion on DARTS From our result, we find two schools of thought for DARTS: For maximal performance, DARTS can be used as as a component in a larger search that proposes trial architectures. For average performance, and immediate performance with a single DARTS run, we should maximize its correlation with architecture performance. Positioning future DARTS papers in this respect, and understanding between the relation between maximal performance and maximal correlation, are important task in future research.

Discussion on optimal network architectures From the applied inverse problems perspective, it is interesting to see that the optimal architectures found for both problems, deblurring and super resolution, are hybrid versions that mix learnable grad- and net-operations. The optimal mixed architectures have a remarkable advantage of more than 1dB over the best plain (only net or only grad) architecture, possibly suggesting that the best way to approach inverse problems with deep learning are neither plain (convolutional) networks nor pure unrolling schemes.

5 Conclusions

In this paper we analyzed DARTS in a systematic study on one-dimensional inverse reconstruction problems, and made the following findings: While it is possible to find very well performing architectures using DARTS, multiple runs of the same setting yield a high variance. Unfortunately, judging the success of any DARTS-based model right after the one-shot training is difficult, since a strong correlation to the actual architecture performance is missing. Therefore, we advertise to look at a full statistical evaluation of DARTS performances over multiple trials, in all applications where this is feasible, and to show a reasonable correlation between the search and final architecture performances for any method that reports improved results based on a more faithful minimization of the one-shot DARTS objective.

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, December 2020.

[2] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, July 2018.

[3] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, page 399–406, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015.

[5] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of Tricks for Image Classification with Convolutional Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.

[6] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NeurIPS*, 2018.

[7] Erich Kobler, Teresa Klatzer, Kerstin Hammernik, and Thomas Pock. Variational networks: Connecting variational methods and deep learning. In *Pattern Recognition*, Lecture Notes in Computer Science, pages 281–293. Springer, 2017.

[8] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[9] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. *arXiv:1806.09055 [cs, stat]*, April 2019.

[10] Jovita Lukasik, David Friede, Arber Zela, Frank Hutter, and Margret Keuper. Smooth variational graph embeddings for efficient neural architecture search, 2021.

[11] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel. *ArXiv*, abs/2006.07556, 2020.

[12] Uwe Schmidt and Stefan Roth. Shrinkage fields for effective image restoration. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2774–2781, 2014.

[13] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. *arXiv preprint arXiv:1910.11858*, 2019.

[14] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and Robustifying Differentiable Architecture Search. *arXiv:1909.09656 [cs, stat]*, January 2020.

[15] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, July 2017.

[16] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[17] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

A Operations

While the precise type of algorithm is typically dictated by (smoothness) properties of the regularization, a partially parameterized network-based approach has a lot of freedom to choose from template layers based on the inverse problem $y = Ax + \text{noise}$

$$\arg\min_x D(Au, f), \tag{5}$$

where D is a data formation term arising from the distribution of noise present, i.e. $D(v, f) = \frac{1}{2}||v - f||^2$ for Gaussian noise. This optimization objective yields templates such as a **gradient descent layer** (operation set $\mathcal{O}_u$):

$$u^{k+1} = u^k - \tau A^T \nabla_x D(Au^k, f), \tag{6}$$

for input u^k and output u^{k+1} of a new layer. For suitable chosen τ, the application of this layer is guaranteed to reduce the objective (5). The gradient layer can be turned into a learnable operation by introducing a learnable mapping $n(u, \theta)$ after the gradient step,

$$u^{k+1} = u^k - \tau A^T \nabla_x D(Au^k, f) - \tau n(u^k, \theta), \tag{7}$$

as a **learnable gradient descent layer** in our operation set $\mathcal{O}_l$. We also consider a fully-learned **neural network layer** in our operation set $\mathcal{O}_l$, that learns an appropriate mapping $n(u, \theta)$ without knowledge of the operator A:

$$u^{k+1} = n\left(u^k, \theta\right), \tag{8}$$

The unlearnable counterpart to the neural network layer in $\mathcal{O}_u$ is a **skip layer**. For both layers, the learnable mapping $n(u, \theta)$ is parametrized by a small convolutional network, consisting of a convolution layer, followed by batch normalization, ReLU and a second convolution layer. These four layers are by design beneficial operations. In contrast to these beneficial layers we also include two negative operations to each operation set; a gradient layer with white Gaussian noise **noise layer** and a **roll layer**, which rolls the inputs.

B Hyperparameters

In table 2 we show the hyperparameters used for our experiments in the main paper.

C Data Generation

We generate synthethic one-dimensional cosine data from $N = 50$ equally spaced points ω_i on the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ with the model

$$x_i = \cos(f\omega_i + O_x) + O_y$$

for a random frequency f drawn uniformly from the interval $[0, 2\pi]$ and offsets O_x and O_y drawn from a normal Gaussian distribution. Such random drawn waves comprises our ground truth training data. We then generate measured data via the linear operation A and addition of noise,

$$y = Ax + n, \qquad n \in \mathcal{N}(0, \sigma_n).$$

These pairs (y, x) represent our training data. We sample new examples on-the-fly during both training and validation, so that no confounding effects of dataset size exist. All validation and training loss evaluations are each based on 2432 randomly drawn samples. For all experiments we chose $\sigma_n = 0.01$. For the *blur* experiments, the linear operator A is a Gaussian blur with kernel size 7 and $\sigma_b = 0.2$. For the *downsampling* experiments, this Gaussian blur is follows by a subsampling by a factor 4.

D Computational Setup

All experiments in the main body were run on a single `Nvidia GTX 2080ti` graphics card of which two were utilized.

Hyperparameter	
Param. learning rate	0.001
Param. weight decay	$1e-8$
Param. warm up	False
Alpha learning rate	0.001
Alpha weight decay	0.001
Alpha warm up	True
Alpha scheduler	Linear
Alpha optimizer	Gradient Descent

Table 2: Hyperparameters for the training runs in Table 1.

Hyperparameter	Default Value
Epochs	50
Batch size	128
Noise Level	0.10

Table 3: General Hyperparameters

E Hyperparameter Optimization

As we have seen in section 4 the DARTS search entails a discrepancy between being a feasible architecture search tactic and indeed finding a good architecture within a single optimization of the DARTS problem. This prompts us to also take a closer look at hyperparameter tuning.

Our one-dimensional case study allows us to optimize DARTS training hyperparameters with more granularity than it would be possible for image classification tasks, and we stress that we consider this mainly as an analysis tool - given that the neural architecture search itself is a hyperparameter optimization on which we stack another. To improve hyperparameters we apply BOHB [2], a Bayesian optimization method with hyperbanding [8] and run BOHB for 128 hyperband iterations, which is an affordable budget in this one-dimensional data setting. This hyperparameter tuning was conducted on a single `Nvidia GTX 1080 Ti` graphics card. The BOHB optimized hyperparameters are listed in table 4; the search range for the BOHB search is given in the second column. It is important to note, that BOHB is not an exhaustive search and thus there are no guarantees for success within our budget or even in general. The usage of BOHB as such covers the problem of hyperparameter optimization partially, but in general there is no simple fix of DARTS via hyperparameter search.

When applying BOHB to target one-shot validation performance, the resulting plot over all trials in fig. 3 shows a rather surprising outcome: In the case of downsampling, the correlation between one-shot validation and architecture validation increases notably and the average performance of DARTS improves in the case of blur, but the best architecture PSNR over 100 trials decreases in both cases. Overall, the apparent stabilization via optimization of the search loss removes not only negative, but also positive outliers. Conversely, we can use BOHB to search for hyperparameters that maximize the final architecture performance. This is twice as expensive (on top of the already expensive hyperparameter) search, due to the need for retraining the architecture, but feasible in our synthetic case. Figure 4 again shows blur and downsampling with hyperparameters optimized for the final architecture performance for blur data. These hyperparameters successfully increase the max. architecture performance, but the overall distribution is similar to the three failure cases discussed for fig. 2 for blur.

Overall, hyperparameters optimized with BOHB on the one-shot validation have to be considered with caution: When regarding the search of architectures over all operations for the case of blur in table 5, the dedicated BOHB-Blur hyperparameters are not as stable as the BOHB-DS hyperparameters. When changing the domain to downsampling, the exact opposite holds: BOHB-Blur hyperparameters improve over BOHB-DS hyperparameters in every statistic. Note that this could be due to both, the missing correlation between one-shot and architecture validation as well as the missing guarantee of any Bayesian search to find the optimal hyperparameters. In addition, table 5 even demonstrates that using the manual hyperparameters H1 leads to a better average performance compared with BOHB-DS and BOHB-Blur in most cases.

Hyperparameter	Search Range	BOHB-Blur	BOHB-DS	BOHB-Arch-Blur
Param. learning rate	$[1e-05, 1]$	0.0014232405	0.0020448382	0.0020882283
Param. weight decay	$[1e-08, 0.1]$	$8.616e-07$	$5.04e-08$	$4.4e-08$
Param. warm up	[True,False]	False	True	False
Alpha learning rate	$[1e-05, 0.1]$	0.0836808765	0.0100063746	$8.43195e-05$
Alpha weight decay	$[1e-05, 0.1]$	$5.05099e-05$	0.0058022776	0.0127425783
Alpha warm up	[True, False]	False	True	True
Alpha scheduler	[None, Linear]	Linear	Linear	Linear
Alpha optimizer	[Adam, Gradient Descent]	Adam	Gradient Descent	Adam

Table 4: BOHB optimized hyperparameters for different data formations, objectives.

Data Formation	Hyperparameters	Architecture Validation (PSNR)					
		Good Ops.			All		
		Max.	Mean	Med.	Max.	Mean	Med.
Blur	H1	23.46	21.56	21.60	22.86	15.64	18.57
Blur	BOHB-Blur	22.83	20.86	20.75	22.47	15.57	18.04
Blur	BOHB-DS	22.33	20.65	20.96	22.41	14.43	14.41
Blur	BOHB-Arch-Blur	23.57	22.05	22.38	22.94	12.76	8.21
Downsampling	H1	18.03	16.36	16.66	18.01	15.39	16.12
Downsampling	BOHB-Blur	18.42	16.83	16.95	17.73	14.36	14.57
Downsampling	BOHB-DS	17.51	15.33	15.84	18.12	12.36	13.65
Downsampling	BOHB-Arch-Blur	18.26	14.63	15.93	17.91	15.04	15.44

Table 5: Architecture validation PSNR values found in the 1D inverse problems setting with cosine data. Shown is the maximal, mean and median PSNR over 100 trials.

In summary, even hyperparameter tuning for different data formations and objectives does not improve the stability of the DARTS search itself. Also in this experiment we still find that DARTS either fails directly, or that it works but does not predict a useful architecture, or that is does predict a useful architecture, but it is unrelated to its search performance.

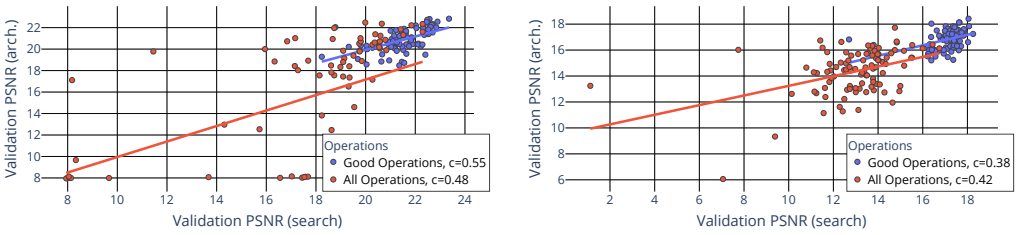

Figure 3: Scatter plot with hyperparameters searched for one-shot validation, showing architecture PSNR (y-axis) plotted against one-shot validation PSNR (x-axis). Left: Blur with hyperparameters *BOHB-Blur* . Right: Downsampling with hyperparameters *BOHB-DS*.

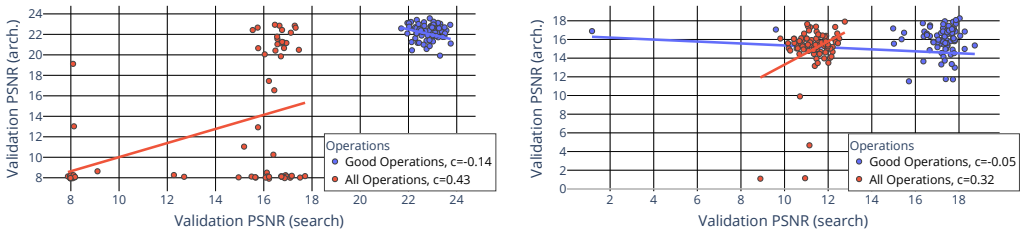

Figure 4: Scatter plot with hyperparameters searched for architecture validation on the blur dataset. We plot architecture PSNR (y-axis) against one-shot validation PSNR (x-axis). Left: Blur with hyperparameters *BOHB-Blur-Arch*. Right: Downsampling in the same setting.