Transfer NAS with Meta-learned Bayesian Surrogates

Gresa Shala[1], **Thomas Elsken**[2], **Frank Hutter**[1,2] **& Josif Grabocka**[1]
[1]University of Freiburg
{shalag,fh,grabocka}@cs.uni-freiburg.de
[2]Bosch Center for Artificial Intelligence
thomas.elsken@de.bosch.com

Abstract

While neural architecture search (NAS) is an intensely-researched area, approaches typically still suffer from either (i) high computational costs or (ii) lack of robustness across datasets and experiments. Furthermore, most methods start searching for an optimal architecture *from scratch*, ignoring prior knowledge. This is in contrast to the manual design process by researchers and engineers that leverage previous deep learning experiences by, e.g., transferring architectures from previously solved, related problems. We propose to adopt this human design strategy and introduce a novel surrogate for NAS, that is meta-learned across prior architecture evaluations across different datasets. We utilize Bayesian Optimization (BO) with deep-kernel Gaussian Processes, graph neural networks for the architecture embeddings and a transformer-based set encoder of datasets. As a result, our method consistently achieves state-of-the-art results on six computer vision datasets, while being as fast as one-shot NAS methods.

1 Introduction

While deep learning has removed the need for manual feature engineering, it has shifted this manual work to the meta-level, introducing the need for manual architecture engineering. The natural next step is to also remove the need to manually define the architecture. This is the problem tackled by the field of neural architecture search (NAS).

However, there is no NAS method yet that is both generally robust and efficient. Blackbox optimization methods, such as reinforcement learning [1], evolutionary algorithms [2], and Bayesian optimization [3, 4] reliably minimize the validation error but are slow. On the other hand, one-shot methods [5, 6] often have problems with robustness [7], and the newest trend of zero-cost proxies often does not provide more information about an architecture's performance than simple statistics, such as the architecture's number of parameters [8].

An understudied path towards efficiency in NAS is to transfer information across datasets. While a few NAS approaches in this direction exist [9–13, 3], an effective approach to obtain efficient NAS methods that has been overlooked in the literature is to exploit the common formulation of NAS as a hyperparameter optimization (HPO) problem [14–17] and draw on the extensive literature on transfer HPO [18–22][1]. This approach is naturally motivated by how researchers and engineers tackle new deep learning problems: they leverage the knowledge they obtained from previous experimentation and, e.g., re-use architectures designed for one task and apply or adapt them to a novel task.

In this work, we present an efficient Bayesian Optimization (BO) method with a novel deep-kernel surrogate that yields a new NAS method which combines the best of both worlds: the reliability of

[1] For a more in-depth discussion of related work we refer to Section 2

6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans.

blackbox optimization at a computational cost in the same order of magnitude as one-shot approaches. Concretely, we propose a BO method for NAS that leverages dataset-contextualized surrogates for transfer learning. Following Lee et al. [13], we use a graph encoder [23] to encode neural architectures and an attention-based dataset encoder to obtain context features. We then use *deep kernel learning* [24] to obtain *meta-learned kernels* for the *joint space of architectures and datasets*, allowing us to use the full power of BO for efficient NAS. In contrast to standard transfer HPO methods that meta-learn parametric surrogates from a pool of source datasets [18, 19, 22], in this work we explore the direction of meta learning surrogates by contextualizing them on the dataset characteristics (a.k.a. meta-features) [25–27]. Exploiting these dataset characteristics is particularly important in NAS to find a strong architecture quickly. Instead of using manually chosen ad-hoc meta-features, we represent meta-features as the latent embeddings of a set-transformer network for image datasets [28], that is jointly meta-learned as part of our deep surrogate in an end-to-end manner. As a result, our surrogates are optimized for efficiently transferring architectures for a new target dataset based on its meta-features. To sum up, our contributions are as follows:

- Inspired by manual architecture design, we treat NAS as a transfer or few-shot learning problem. We leverage ideas from transfer HPO to meta-learn a kernel for Bayesian Optimization, which encodes both architecture and dataset information.
- We are the first to combine deep-kernel Gaussian Processes (GPs) with a graph neural network encoder, a transformer-based dataset encoder, the first to apply BO with deep GPs to NAS, and the first to do all of this in a transfer NAS setting.
- Our resulting method outperforms both state-of-the-art blackbox NAS methods as well as state-of-the-art one-shot methods across six computer vision benchmarks.

To foster reproducibility, we make our code available at `https://github.com/TNAS-DCS/TNAS-DCS.git`. We address the points in the "NAS Best Practices Checklist" in Appendix H.

2 Related Work

NAS is an intensely-researched field, with over 1000 papers published in the last two years alone[2]. We therefore limit our discussion of NAS to the most related fields of Bayesian optimization for NAS and meta learning approaches for NAS. For a full discussion of the NAS literature, we refer the interested readers to a series of recent surveys by Elsken et al. [29], Wistuba et al. [30], Xie et al. [31], Ren et al. [32], Benmeziane et al. [33] and Dong et al. [34], and for an introduction to BO to Shahriari et al. [35], Hutter et al. [36].

Bayesian optimization (BO) for NAS. As BO is commonly used in hyperparameter optimization (HPO), one can simply treat architectural choices as categorical hyperparameters and re-use, e.g., tree-based HPO methods that can natively handle categorical choices well [14, 15, 37]. While Gaussian Processes (GPs) are more typically applied to continuous hyperparameters, they can also be used for NAS by creating an appropriate kernel; such kernels for GP-based BO were manually engineered by Swersky et al. [38], Kandasamy et al. [39] and Ru et al. [3]. A recent alternative is to exploit (Bayesian) neural networks for BO [40, 41, 4]. However, while these neural networks are very expressive and parallelize well, they require more data to fit well than GPs and thus are outperformed by GP-based approaches when only a few function evaluations can be afforded. In this work, we combine the sample efficiency of GPs and the expressive power of neural networks, by using deep GPs combined with a graph neural network encoder.

Meta learning for NAS. To mitigate the computational infeasibility of starting NAS methods from scratch for each new task, several approaches have been proposed along the lines of meta and transfer learning [9–13, 3]. Most of these warm-start the weights of architectures in a target task [9–12]. Ru et al. [3] extracts architectural motifs that can be reused on other datasets. Most related to our work is MetaD2A [13], where the authors propose to generate candidate architectures and rank them conditioned directly on a task, utilizing a meta-feature extractor [28]. However, there are two key differences in our work: (i) the performance predictor of MetaD2A is not probabilistic and thus can not naturally *trade-off exploration vs. exploitation* but rather *only exploits* what has been observed during meta-training. (ii) During the meta testing phase, MetaD2A simply proposes N architectures

[2] See list at: `https://www.automl.org/automl/literature-on-neural-architecture-search`

for a new task, trains the top-K (as estimated by the performance predictor) out of N architectures and returns the best. In particular, the performance of the K evaluated architectures is *not* used as feedback for MetaD2A and thus the method *never* adapts to function evaluations on the test task, blindly following what has been observed during meta training. This can cause problems for new tasks which are poorly correlated with the meta training data, and indeed, MetaD2A stagnates on several of the datasets in our experimental analysis. In contrast, TNAS uses the function evaluations from meta testing to update the surrogate employed within our BO framework, thus allowing to adapt to the meta testing scenario.

Meta & transfer learning for HPO There are many approaches to achieve meta or transfer learning in HPO, see, e.g., the survey by Vanschoren [42] or the nine different classes of methods discussed by Feurer et al. [43, Section 7]. One particularly promising approach is to employ Deep Kernel Learning. While existing kernels for BO for NAS are manually defined [38, 39, 3], *Deep Kernel* Learning [24] is an approach that strives to learn the kernel function by using a neural network to transform the input to a latent representation, which is then used in a kernel function. Wistuba and Grabocka [22] and Jomaa et al. [44] utilized a deep kernel for transfer learning in HPO. While numerical hyperparameters can be encoded using an MLP, we adapt this Deep Kernel Learning approach to NAS by using a graph neural network to encode architectures as inputs; we also extend it by encoding datasets into a latent embedding and learning a deep kernel that spans the combined space of architectures and datasets.

3 Proposed Method

3.1 Bayesian optimization with deep kernel Gaussian processes

We start by introducing Gaussian Processes, which represents the surrogate of our method within Bayesian optimization (BO). In a typical hyperparameter optimization (HPO) setup, the inputs $x \in \mathcal{X}$ represent hyperparameter configurations, and the target $y \in \mathcal{Y}$ denotes the performance of a machine learning method when trained with the hyperparameter configuration x. Consider the training $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ and testing $\mathcal{D}^* = \{(x_i^*, y_i^*)\}_{i=1}^{n^*}$ splits of a dataset of evaluated hyperparameters. In that context, GPs are non-parametric models that assume a prior over functions, and approximate the target $y \in \mathcal{Y} \subseteq \mathbb{R}_+$ given the features $x \in \mathcal{X} \subseteq \mathbb{R}^L$. The estimation of the target variable y^* for the test instances x^* is also jointly Gaussian as $\begin{bmatrix} y \\ y^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K(x,x) & K(x,x^*) \\ K(x,x^*)^T & K(x^*,x^*) \end{pmatrix}\right)$. Each respective block of the covariance matrix is the result of applying a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ on pairs of instances, e.g., $K(x, x^*)_{i,j} := k(x_i, x_j^*)$. The estimated posterior mean and the posterior covariance of GPs [45] for the target y^* of the test instances x^* is given as follows:

$$\mathbb{E}[y^* \mid x^*, x, y] = K(x^*, x)K(x, x)^{-1}y \tag{1}$$

$$\text{cov}[y^* \mid x^*, x] = K(x^*, x^*) - K(x, x^*)^T K(x, x)^{-1} K(x, x^*). \tag{2}$$

We refer to, e.g., Murphy [46] for the derivation. GPs are *lazy* models that rely on the similarity of the test instances to the training instances via kernel functions k, such as the Matérn kernel. Unfortunately, typical kernels used with GPs are designed manually and rely on sub-optimal assumptions [47], which deteriorates the GP's performance. A promising direction for designing powerful and efficient kernel functions that adapt to a learning task is Deep Kernel Learning [24], where kernels are represented as trainable neural networks. A mapping $\xi : \mathcal{X} \to \mathbb{R}^M$ projects the features to a latent representation, where similar instances are co-located.

The embedding ξ for the deep kernel of our GPs is a fully-connected neural network, that takes as input the encoding of the architecture ψ as well as the dataset encoding ϕ. In detail, the L-dimensional architecture encoding ψ is fused with the K-dimensional dataset encoding ϕ and processed through a fully connected neural network $\xi : \mathbb{R}^{K+L} \to \mathbb{R}^M$, where the last layer has M neurons.

The architecture encoding ψ consists of a directed acyclic *graph encoder* [23] to obtain the encoding for the architectures. By using one GRU cell to traverse the topological order of the DAG in the direction from the input to the output, and another GRU cell to pass through the DAG in the backward direction, we obtain latent representations of the graph which are then put through a fully-connected neural network layer to obtain the encoding for the architecture.

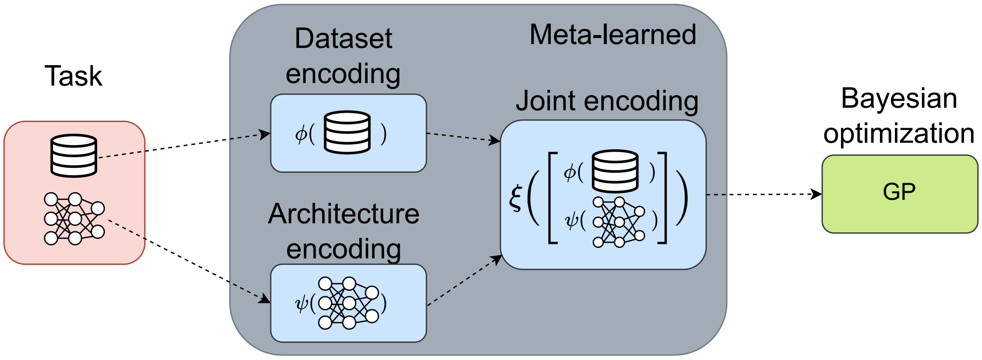


Figure 1: Illustration of TNAS . We employ a GNN ψ to encode architectures, a transformer ϕ to encode a dataset, and an MLP ξ to merge the encodings. This joint encoding is then fed into a GP surrogate used within BO. All encodings are meta-learned.

The dataset encoding ϕ is based on the encoder proposed by Lee et al. [13]. It consists of two stacked *Set-Transformer* [28] architectures. The first Set-Transformer layer captures the interaction between randomly sampled data points of the same class, whereas the second one captures the interactions between the different classes of the dataset. The resulting output of the second Set-Transformer layer represents the dataset encoding. We tuned the dimensionality of the embedding of the dataset encoder and graph encoder, as well as the architecture of the feed-forward neural network of our method using the multi-fidelity Bayesian optimization method BOHB [37] on the meta-training dataset; please refer to Appendix E for details.

Putting it all together, the kernel/similarity between two architectures, specifically $\mathbf{x}$ evaluated on dataset $\mathcal{D}$, and $\mathbf{x}'$ evaluated on dataset $\mathcal{D}'$, is:

$$k\left(\mathbf{x}, \mathcal{D}, \mathbf{x}', \mathcal{D}'; w\right) = k\Bigg(\xi\Big(\Big[\psi(\mathbf{x}; w^{(\psi)}), \phi\left(\mathcal{D}; w^{(\phi)}\right)\Big]; w^{(\xi)}\Big), \\ \xi\Big(\Big[\psi(\mathbf{x}'; w^{(\psi)}), \phi\left(\mathcal{D}'; w^{(\phi)}\right)\Big]; w^{(\xi)}\Big); w^{(k)}\Bigg)$$

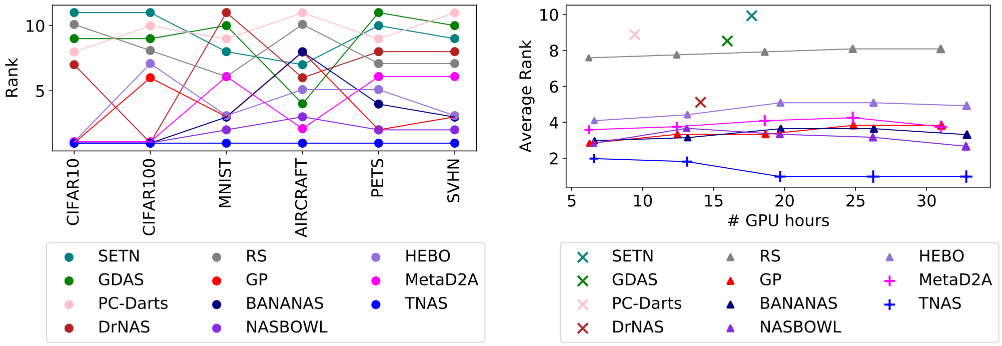
with $w^{(\xi)}$ being the parameters of the neural network ξ, $w^{(\psi)}$ the parameters of the architecture encoding ψ, $w^{(\phi)}$ the parameters of the dataset encoding ϕ, and $w^{(k)}$ additional parameters of the kernel function. We denote the cumulative parameters as $w := \left(w^{(\xi)}, w^{(\psi)}, w^{(\phi)}, w^{(k)}\right)$. We refer to Figure 1 for an overview of our framework. All parameters are jointly meta-learned to maximize the marginal likelihood [22], which we discuss in detail in Appendix B.

4 Experimental Setup

We follow the experimental setup and use the same meta datasets as Lee et al. [13]. It consists of 4230 meta-training datasets derived from ImageNet. For each of these datasets, the accuracy of one (different) architecture from the NAS-Bench-201 [48] search space is given. For the evaluation of our method and the baselines, we use six popular computer vision datasets: CIFAR-10, CIFAR-100, SVHN, Aircraft, Oxford IIT Pets, and MNIST. For CIFAR-10 and CIFAR-100, we query the performances of architectures from the NAS-Bench-201 benchmark, whereas for the other four datasets we train the suggested architectures from scratch using the NAS-Bench-201 pipeline (as these are not available in the benchmark). For all experiments, we use the NAS-Bench-201 search space. We ran three trials for each experiment and report the mean and standard deviations.

5 Research Hypotheses and Experimental Results

Our experiments are designed to validate the following research hypotheses for our approach, dubbed TNAS :



(a) Ranking of TNAS across benchmarks.

(b) Average Ranking over the course of runtime.

Figure 2: Consistency of TNAS compared to baselines.

Hypothesis 1: *TNAS is more efficient than classical HPO methods applied to NAS as well as HPO methods specifically adapted to NAS and outperforms them in terms of anytime-performance, while achieving strong final performance.*

Hypothesis 2: *TNAS is competitive with one-shot approaches in terms of runtime.*

Hypothesis 3: *TNAS achieves the consistency of blackbox optimization algorithms (such as classical HPO methods) while being as efficient as one-shot methods.*

For all classical HPO methods except for RS, we use the 5 top-performing architectures from the meta-training dataset as a starting point. In that sense, all these baselines are "transfer learning" by being initialized with the best architectures on the meta-training dataset.

In Figure 2, we show that TNAS consistently achieves strong results, while the existing state-of-the-art NAS baselines have much higher variance - their ranking changes across benchmarks. Furthermore, the figure shows how the ranking evolves over the course of running of the methods (by means of runtime in GPU hours). The analysis indicates that TNAS consistently achieves the best performance, for both small and large computational budgets, in particular also when compared to MetaD2A. We show more detailed results on these six datasets in Appendix D.1. We show the performance comparison of TNAS to classical HPO methods in Figure 3, whereas Figure 4 shows the comparison to one-shot NAS methods. Furthermore, in Appendix D.2 we empirically analyse our design choices, and demonstrate a lift in performance compared to *ceteris paribus* ablations that do not employ these designs.

6 Conclusions and Future Work

NAS is an intensely researched task and in essence is an instance of the hyperparameter optimization (HPO) problem. In this work, we exploited this relationship and, motivated by state-of-the-art transfer HPO methods, adapted deep Gaussian Process (GP) surrogates to capture architecture representations. Furthermore, we proposed a novel conditioning of the deep GP on dataset meta-features to enable transferring well-performing architectures from source datasets with similar meta-features. In addition, we showed that our novel deep GPs with dataset meta-features and architecture encodings achieve the highest accuracy on six computer vision datasets compared to a broad range of HPO methods, BO methods for NAS, and one-shot NAS methods. Lastly, we demonstrated that proxy architecture evaluations allow our method to discover more accurate architectures within the same search time one-shot NAS methods require.

References

[1] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*, 2017.

[2] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4780–4789, 2019.

[3] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations*, 2021.

[4] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. 2021.

[5] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

[6] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019.

[7] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: iclr.cc.

[8] Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, Sébastien Bubeck, and Debadeepta Dey. A deeper look at zero-cost proxies for lightweight nas. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/. https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/.

[9] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/bdb3c278f45e6734c35733d24299d3f4-Paper.pdf.

[10] Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1eowANFvr.

[11] Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *CVPR*, pages 12362–12372, 2020. URL https://doi.org/10.1109/CVPR42600.2020.01238.

[12] Martin Wistuba. Xfernas: Transfer neural architecture search. In Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 247–262, Cham, 2021. Springer International Publishing. ISBN 978-3-030-67664-3.

[13] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=rkQuFUmUOg3.

[14] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, 2013.

[15] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 2015.

[16] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 2902–2911, 2017.

[17] Noor Awad, Neeratyoy Mallik, and Frank Hutter. Dehb: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI'21)*, 2021.

[18] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I*, pages 199–214, 2016.

[19] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization. *CoRR*, abs/1802.02219, 2018.

[20] Valerio Perrone and Huibin Shen. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12751–12761, 2019.

[21] David Salinas, Huibin Shen, and Valerio Perrone. A quantile-based approach for hyperparameter transfer learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 8438–8448, 2020.

[22] Martin Wistuba and Josif Grabocka. Few-shot bayesian optimization with deep kernel surrogates. In *International Conference on Learning Representations*, 2021.

[23] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[24] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, 2016.

[25] Joaquin Vanschoren. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018.

[26] Hadi S. Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2vec: learning dataset meta-features. *Data Min. Knowl. Discov.*, 35(3):964–985, 2021.

[27] Adriano Rivolli, Luís P.F. Garcia, Carlos Soares, Joaquin Vanschoren, and André C.P.L.F. de Carvalho. Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101, 2022. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2021.108101. URL `https://www.sciencedirect.com/science/article/pii/S0950705121011631`.

[28] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 3744–3753, 2019.

[29] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[30] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

[31] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Zhengsu Chen, Lanfei Wang, An Xiao, Jianlong Chang, Xiaopeng Zhang, et al. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *arXiv preprint arXiv:2008.01475*, 2020.

[32] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

[33] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.

[34] Xuanyi Dong, David Jacob Kedziora, Katarzyna Musial, and Bogdan Gabrys. Automated deep learning: Neural architecture search is not the end. *CoRR*, abs/2112.09245, 2021. URL `https://arxiv.org/abs/2112.09245`.

[35] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[36] F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*, volume 5. 2019. Available for free at http://automl.org/book.

[37] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pages 1437–1446. Proceedings of Machine Learning Research, 2018.

[38] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. Osborne. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In *NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13)*, 2013.

[39] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, 2018.

[40] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2171–2180, 2015.

[41] J. Springenberg, A. Klein, S. Falkner, and F. Hutter. Bayesian optimization with robust Bayesian neural networks. In D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*, 2016.

[42] J. Vanschoren. Meta-learning. In Hutter et al. [36], chapter 2, pages 35–61. Available for free at http://automl.org/book.

[43] Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical transfer learning for bayesian optimization, 2018. URL `https://arxiv.org/abs/1802.02219`.

[44] Hadi Samer Jomaa, Sebastian Pineda Arango, Lars Schmidt-Thieme, and Josif Grabocka. Transfer learning for bayesian hpo with end-to-end landmark meta-features. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

[45] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

[46] K. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, Cambridge, MA, 2012.

[47] Alexander I Cowen-Rivers, Wenlong Lyu, Zhi Wang, Rasul Tutunov, Hao Jianye, Jun Wang, and Haitham Bou Ammar. Hebo: Heteroscedastic evolutionary bayesian optimisation. *arXiv preprint arXiv:2012.03826*, 2020. winning submission to the NeurIPS 2020 Black Box Optimisation Challenge.

[48] Xuanyi Dong and Yi Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=HJxyZkBKDr`.

[49] Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20309–20319. Curran Associates, Inc., 2020.

[50] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *NeurIPS 4th Workshop on Meta-Learning*, 2020.

[51] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 660–676, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58526-6.

[52] Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10480–10490. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/768e78024aa8fdb9b8fe87be86f64745-Paper.pdf`.

[53] Massimiliano Patacchiola, Jack Turner, Elliot J. Crowley, Michael F. P. O'Boyle, and Amos J. Storkey. Bayesian meta-learning for the few-shot setting via deep kernels. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[54] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018.

[55] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1126–1135, 2017.

[56] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[57] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

[58] Jacob R. Gardner, Geoff Pleiss, Kilian Q. Weinberger, David Bindel, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7587–7597, 2018.

[59] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *arXiv:2104.10201 [cs.LG]*, 2021.

[60] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77): 2539–2561, 2011.

[61] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[62] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJlS634tPr`.

[63] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Dr{nas}: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=9FWas6YbmB3`.

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

A Method Preliminaries

We consider the following problem: given a history of Q datasets, where for each dataset $\mathcal{D}^{(q)}$ we have already evaluated a set of neural network architectures $x_1^{(q)}, \ldots, x_n^{(q)}$ with corresponding performance (e.g., accuracy) $y_1^{(q)}, \ldots, y_n^{(q)}$. For a new dataset $\mathcal{D}^{(\text{new})}$, we want to quickly discover an optimal architecture by leveraging information from the history of datasets. We build upon the state-of-the-art few-shot Bayesian optimization (BO) framework by Wistuba and Grabocka [22], which was proposed to address a similar problem: transferring optimal hyperparameter configurations across datasets. The authors propose to learn a deep kernel across tasks, which is then used for a Gaussian process (GP) surrogate in the typical BO setup.

However, while hyperparameter configurations can typically be presented by an N-dimensional vector, it is less clear how to represent neural network architectures. Simply representing architectures as vectors and plugging them into an off-the-shelf GP kernel is likely sub-optimal. In fact, White et al. [49] have shown that the type of architecture representation substantially impacts the performance of a downstream NAS algorithm. To address this issue, we employ graph neural networks (GNNs) to obtain a learnable representation of neural networks. GNNs are a common choice in NAS as neural networks architectures can be naturally represented as graphs [50, 49, 51, 52].

Furthermore, while we could directly feed this architecture encoding into the GP's kernel, we argue that the kernel should also be conditioned on the characteristics of a dataset in order to meaningfully asses (dis-)similarities between architectures. To motivate this, consider the question "What makes two architectures similar?". We argue that two architectures are not similar only because they share some similar sub-graph components (which will be represented by the GNN encoding), but also because they achieve similar performance on the target dataset. Following this line of reasoning, we condition the deep kernel on the characteristics (meta-features) of a dataset [25]. In a similar fashion as for the architecture encoding, we again use a learnable representation of datasets via employing a set transformer [28].

The architecture and dataset encoding are then processed by a fully-connected neural network, whose output serves as the input for an off-the-shelf kernel function, e.g., a Matérn kernel, which is finally used to compute the distance of two (architecture, dataset) datapoints. This results in an end-to-end learnable encoding of the problem, and the parameters of the GNN, transformer and fully-connected neural network are meta-learned in a similar fashion as in Wistuba and Grabocka [22].

B Meta-learning deep-kernel GP Surrogates

Recall that we assume we are given a set of Q datasets, where on each dataset $\mathcal{D}_q$ we have $N_q \in \mathbb{N}_+$ evaluated architectures. We denote the n-th architecture evaluated on the q-th dataset as $x_{q,n}$ and its validation accuracy as $y_{q,n}$. Overall, the meta-dataset of all the evaluations on all the datasets is defined as $\mathcal{M} := \bigcup_{q=1}^{Q} \bigcup_{n=1}^{N_q} \{(x_{q,n}, y_{q,n}, \mathcal{D}_q)\}$. By $\mathbf{x} := (x_{1,1}, \ldots, x_{Q,N_q}), \mathbf{y} := (y_{1,1}, \ldots, y_{Q,N_q})$ and $\mathcal{D} := (\mathcal{D}_1, \ldots, \mathcal{D}_Q)$ we denote the vectors containing all the architectures, accuracies and datasets, respectively.

The parameters w of the deep kernel are optimized jointly by maximizing the log marginal likelihood of the GP surrogate on the meta training dataset:

$$\begin{aligned} &\arg\max_{w} \quad \log p\left(\mathbf{y} \mid \mathbf{x}, \mathcal{D} \,;\, w\right) \\ &\propto \arg\min_{w} \quad \mathbf{y}^T K^{-1}(\mathbf{x}, \mathcal{D}; w)\, \mathbf{y} + \log |K(\mathbf{x}, \mathcal{D}; w)|. \end{aligned} \tag{3}$$

In practice, we resort to sampling mini-batches and employ stochastic gradient descent, following established practices on Deep Kernel Learning for GPs [24, 22, 53]. As all components of our method are differentiable, our approach is end-to-end differentiable.

Algorithm 1 Meta-learning our deep-kernel GPs

```
Require: meta-dataset $\mathcal{M}$; learning rates $\eta_{SGD}, \eta_{REP}$;
  inner update steps $v$.
while not converged do
  Sample mini-batch from $\mathcal{M}$:
    $\mathbf{x} = [x_1, \ldots, x_k], \mathbf{y} = [y_1, \ldots, y_k], \mathcal{D} = [\mathcal{D}_1, \ldots, \mathcal{D}_k]$
  $\mathcal{L}(w) = \mathbf{y}^T K^{-1}(\mathbf{x}, \mathcal{D}; w)\mathbf{y} + \log |K(\mathbf{x}, \mathcal{D}; w)|$
  $w' \leftarrow w$
  for $j = 1$ to $v$ do
    $w' \leftarrow w' - \eta_{SGD} \nabla_{w'} \mathcal{L}(w')$
  Update $w \leftarrow w - \eta_{REP}\,(w - w')$
```

For updating the meta-parameters w, we use the meta-learning algorithm REPTILE [54], due to its simplicity and faster convergence time compared to, e.g., MAML [55]. Our method's pseudocode is given in Algorithm 1. The procedure samples a mini-batch of (architecture, dataset) pairs with corresponding validation accuracy (line 3). We simply sample datasets and architectures uniformly at random from the meta-training dataset. Then we fit the surrogate to estimate the accuracies y of architectures x on a data set $\mathcal{D}$ using SGD by minimizing Equation 3 (lines 4-7). Finally, we use REPTILE [54] to update w (line 8). Once the optimal w are found, we plug in the meta-learned kernel for a GP, with the posterior and use vanilla Bayesian Optimization (BO) to quickly identify the optimal configuration in the new response surface.

C Baselines

Classic HPO. A simple and standard baseline is Random Search (RS) [56]. RS samples architectures uniformly at random from the search space and returns the top preforming one. Another simple yet powerful baseline is Bayesian Optimization with a vanilla GP surrogate [57]. We use the Matérn $5/2$ kernel and rely on the GPytorch [58] implementation. We tried both Expected Improvement (EI) and Upper Confidence Bound (UCB) as acquisition functions, with UCB performing better in our experiments. Additionally, we compare to HEBO [47], a black-box HPO method that performs input and output warping to mitigate the effects of heteroscedasticity and non-stationarity on HPO problems. HEBO won the 2020 NeurIPS blackbox optimization challenge [59]. We use the implementation provided by the authors.

HPO for NAS. White et al. [4] proposed a BO method for NAS that uses an ensemble of fully-connected neural networks as a surrogate, named BANANAS. Moreover, BANANAS uses a path encoding for the neural architectures, which serves as an input to the ensemble. When applied to a new test task, BANANAS starts the neural architecture search from scratch. NASBOWL [3] is a GP-based BO method for NAS and utilizes the Weisfeiler-Lehman kernel [60]. We use the implementations provided by the authors for BANANAS and NASBOWL.

State-of-the-art in NAS. One-shot methods have recently shown strong empirical performance for NAS. We compare to GDAS [6], SETN [61], PC-DARTS [62] and DrNAS [63]. For these methods, we compare to published results from the literature for six computer vision datasets.

Transfer NAS. We compare to the most related transfer NAS method, MetaD2A [13], which is a transfer learning NAS method with a dataset-contextualized neural network generator and performance predictor. The neural network generator and performance predictor are meta-trained on the same source datasets that we also use. When applied to a test task, MetaD2A generates 500 candidate architectures conditioned on the test dataset and then selects the top architectures based on its performance predictor. We use the implementation provided by MetaD2A's authors.

D Additional Experimental Results

D.1 Comparison of TNAS to baselines

Results for Hypothesis 1. In Figure 3, we compare the performance of TNAS, with several HPO baselines. For all methods, we use the 5 top-performing architectures from the meta-training dataset as a starting point. In that sense, all these baselines are "transfer learning" by being initialized with the best architectures on the meta-training dataset. On all of the datasets except CIFAR100, TNAS finds top-performing architectures faster than all the baselines and achieves stronger anytime performance. Furthermore, on all benchmarks, TNAS eventually performs best.

Results for Hypothesis 2. We demonstrate that with our meta-learned deep kernel within Bayesian Optimization, the search time can be significantly reduced, to the same order as one-shot approaches. Figure 4 shows the performance of both TNAS and MetaD2A compared to state-of-the-art NAS methods. Except for CIFAR10 and CIFAR100, TNAS clearly outperforms the NAS baselines by the time the one-shot approaches finish the search. On CIFAR10, TNAS achieves similar performance as

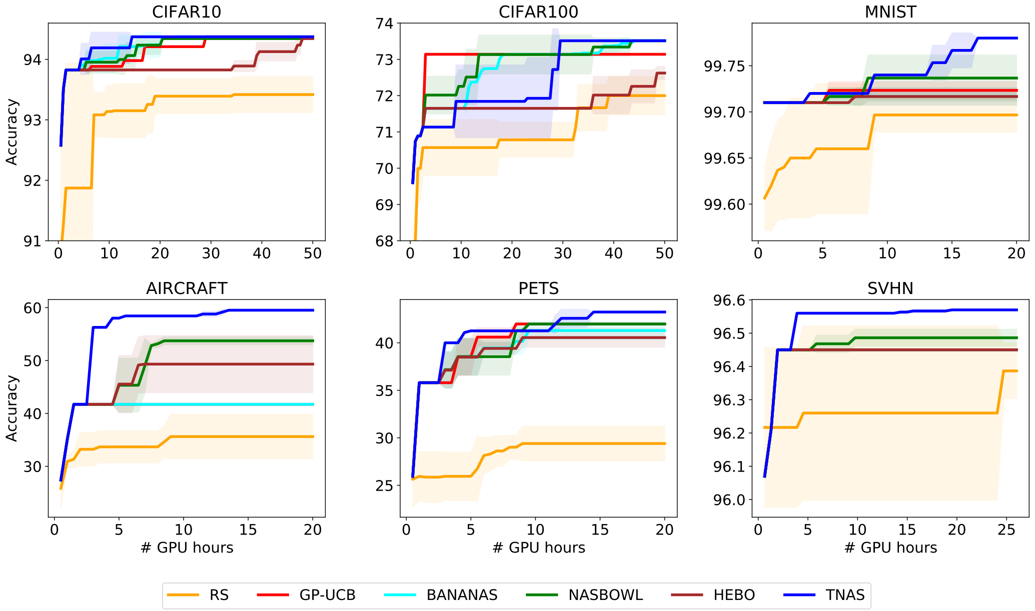


Figure 3: Comparing TNAS to random search (RS) and the four different Bayesian optimization methods on six image datasets.

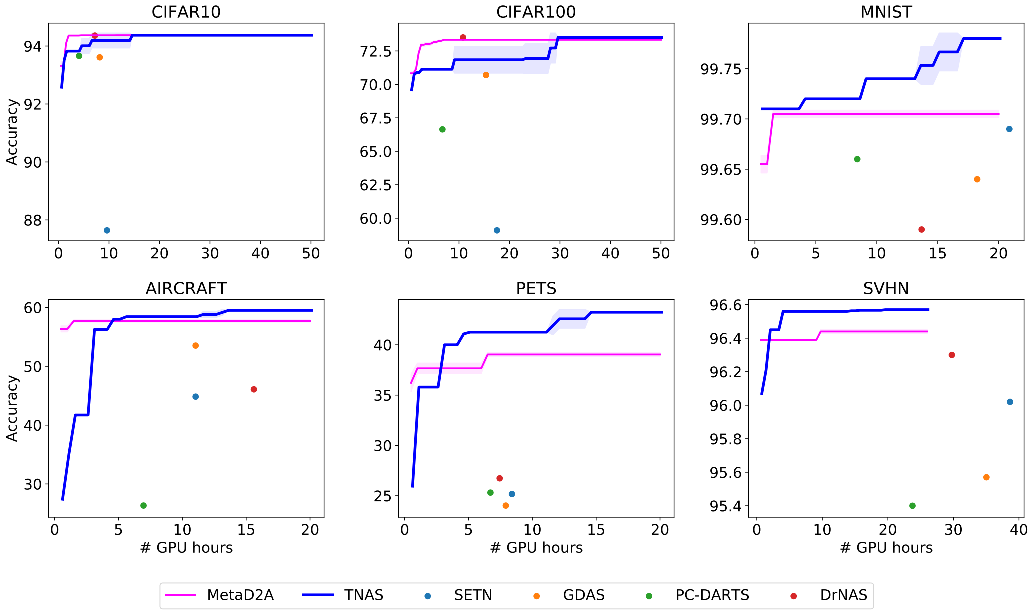


Figure 4: Comparison of TNAS to state-of-the-art NAS methods.

the baselines, while it is slightly inferior on CIFAR100 for some baselines and only overtakes them given more time. We refer to Table 1 in the appendix for concrete numbers. We can furthermore observe the drawbacks of MetaD2A discussed earlier: (i) MetaD2A does not *trade-off exploration vs. exploitation* but rather *only exploits* what has been observed during meta-training and (ii) MetaD2A does not use evaluations on the new target dataset as feedback and *never* adapts. As a result, MetaD2A stagnates on several of the datasets.

D.2 Ablating our Design Choices

We empirically analyse our design choices, namely the graph and dataset encoder and demonstrate a lift in performance compared to *ceteris paribus* ablations that do not employ these designs.

Table 1: Accuracy of our method (TNAS) in terms of time and accuracy compared to state-of-the-art NAS methods.

Data	Method	GPU days	Accuracy
CIFAR-10	SETN	0.40	87.64±0.00
	GDAS	0.34	93.61±0.09
	PC-DARTS	0.17	93.66±0.17
	DrNAS	0.30	94.36±0.00
	MetaD2A	2.08	**94.37**±0.00
	TNAS	2.18	**94.37**±0.00
CIFAR-100	SETN	0.73	59.09±0.24
	GDAS	0.64	70.70±0.30
	PC-DARTS	0.28	66.64±2.34
	DrNAS	0.45	**73.51**±0.00
	MetaD2A	2.08	**73.51**±0.15
	TNAS	2.18	**73.51**±0.00
MNIST	SETN	0.87	99.69±0.04
	GDAS	0.76	99.64±0.04
	PC-DARTS	0.35	99.66±0.04
	DrNAS	0.57	99.59±0.02
	MetaD2A	0.83	99.71±0.02
	TNAS	0.89	99.78±0.00
Aircraft	SETN	0.46	44.84 ±3.96
	GDAS	0.46	53.52±0.48
	PC-DARTS	0.29	26.33±3.40
	DrNAS	0.65	46.08±7.00
	MetaD2A	0.83	57.71±0.72
	TNAS	0.89	**59.51**±0.0
Pets	SETN	0.35	25.17±1.68
	GDAS	0.33	24.02±2.75
	PC-DARTS	0.28	25.31±1.38
	DrNAS	0.31	26.73±2.61
	MetaD2A	0.83	39.04±0.72
	TNAS	0.89	43.24±0.0
SVHN	SETN	1.61	96.02±0.04
	GDAS	1.46	95.57±0.04
	PC-DARTS	0.99	95.40±0.04
	DrNAS	1.24	96.30±0.02
	MetaD2A	1.08	96.44±0.05
	TNAS	1.18	**96.57**±0.00

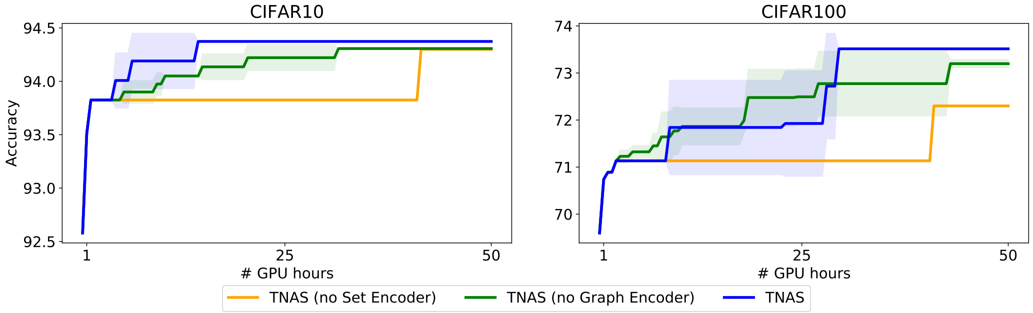


Figure 5: Ablation of the components of TNAS .

Concretely, we ablate our method's components and test them on CIFAR-10 and CIFAR-100, reporting the result in Figure 5. *TNAS (no Set Encoder)* shows the performance of our method using a graph encoder, but no set-encoder (i.e., without using any dataset meta-features). *TNAS (no Graph Encoder)* shows the performance of our method using the dataset encoding in combination with a matrix encoding for the architectures (i.e., no graph encoder). *TNAS* outperforms the other variations; thus, we conclude that *using both the learnable dataset meta-features and a graph neural network* encoding is the most robust surrogate design. This finding validates the design choices of our method.

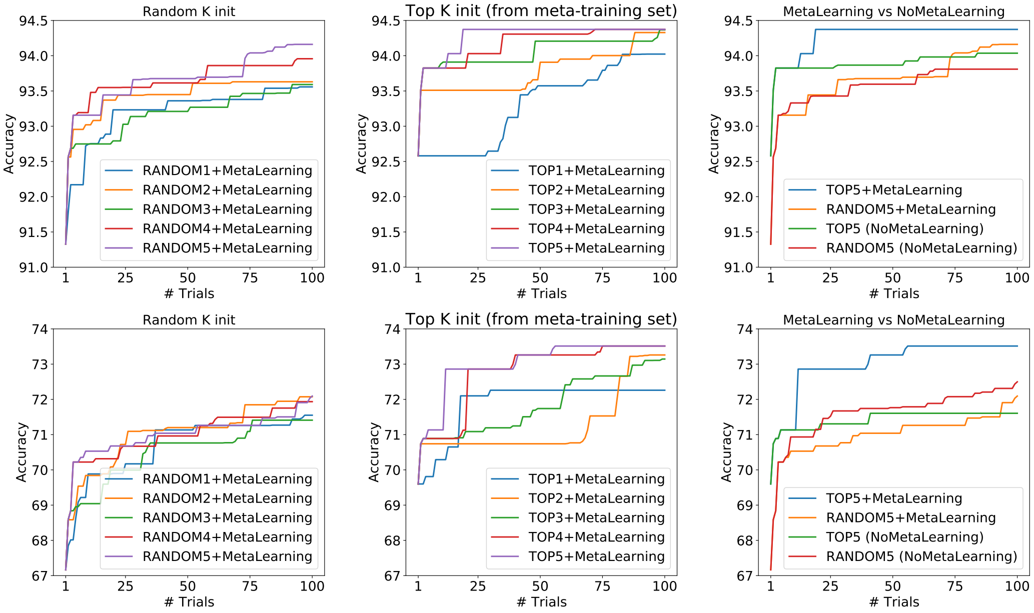


Figure 6: Ablation of the initial design and the initialization of our method on CIFAR-10 (top row) and CIFAR-100 (bottom row). Left: using up to 5 random architectures; middle: using up to 5 top architectures from the meta-training set; right: ablating the comparison of (i) a meta-learned surrogate and a randomly initialized one with (ii) a random initial design and an initial design from the top 5 architectures from the meta-training set.

We also empirically evaluated (i) whether there is actually a benefit in meta-learning the surrogate, and (ii) our method's performance with different initial architectures. Remember that as an initial design for our method, we used the top-5 performing architectures from the meta-training dataset. As alternatives, we consider values other than 5 and also start from randomly sampled architectures. We also turn-off meta-learning. The plots for these experiments are shown in Figure 6, evaluated on CIFAR-10 and CIFAR-100. The results suggest that both design choices are beneficial.

E Hyperparameter optimization for GNN and set transformer architectures

We tuned the dimensionality of the embedding of the dataset encoder and graph encoder (*Embedding dims.*), the architecture of the feed-forward neural network of our method (*Num. layers*, *Num. units in layer 1*, *Num. units in layer 2*, *Num. units in layer 3*, *Num. units in layer 4*), and the learning rate of the joint meta-training using BOHB [37] on the meta-training dataset.

F Societal Implications

Better optimization of neural network architectures allows the enhancement of the performance of many crucial applications of deep learning, such as climate science, medicine, and recommender systems. Furthermore, through the use of transfer learning to speed up the optimization of neural architectures, we reduce the time costs, and therefore the energy consumption of our method. However, there are also potential disadvantages in the neural network components of our method because they

Hyperparameter	**Range**
Embedding dims.	$\{8, 16, 20, 28, 32, 48, 56\}$
Num. layers	$[1, 4]$ (integer)
Num. units in layer 1	$[16, 512]$ (log space)
Num. units in layer 2	$[16, 512]$ (log space)
Num. units in layer 3	$[16, 512]$ (log space)
Num. units in layer 4	$[16, 512]$ (log space)
Learning rate	$[10^{-6}, 10^{-1}]$ (log space)

Table 2: The hyperparameter space for tuning our method with BOHB [37].

are more difficult to interpret compared to simpler, traditional ML methods; we therefore encourage research on interpretable deep learning.

G Future work

A deep surrogate in principle allows us to capture the interaction between architectures and hyperparameter configurations. The NAS literature at the moment underexplores the impact of hyperparameters on the performance of an architecture. In fact, it is commonly known that the same architecture would perform differently if the training pipeline is altered, for example by changing the learning rate, the number of epochs, or the degree of regularization. Our previously-defined surrogate can be trivially extended to model the interaction of architecture embeddings, the dataset meta-features and hyperparameter configurations. We have not explored this direction empirically due to the lack of available NAS meta-datasets that vary both architectures and hyperparameters but would like to do in the future.

H NAS Best Practice Checklist

We now describe how we addressed the individual points of the NAS best practice checklist [64].

1. **Best Practices for Releasing Code**

   For all experiments you report:

   (a) Did you release code for the training pipeline used to evaluate the final architectures? The code for the training pipeline for the architectures can be found in the repo we provide.

   (b) Did you release code for the search space? We used the NAS-Bench-201 search space in our experiments, the description and code for which is publicly available.

   (c) Did you release the hyperparameters used for the final evaluation pipeline, as well as random seeds? We the NAS-Bench-201 pipeline and hyperparameters as our final evaluation pipeline. We release it as well as the random seeds in the repo we provide.

   (d) Did you release code for your NAS method? The code for our NAS method can be found in `https://anonymous.4open.science/r/TNAS-DCS-CC08`.

   (e) Did you release hyperparameters for your NAS method, as well as random seeds? The hyperparameters for our NAS method, as well as random seeds for the experiments can be found in the repo we provide.

2. **Best practices for comparing NAS methods**

   (a) For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?? Yes, for fair comparison we made sure to use the same evaluation pipeline for all NAS methods we compare.

   (b) Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)? We ran all the methods on the same hardware and the same environment to control for confounding factors.

(c) Did you run ablation studies? We ran extensive ablation studies on the components of our method.

(d) Did you use the same evaluation protocol for the methods being compared? Yes.

(e) Did you compare performance over time? Yes.

(f) Did you compare to random search? Performance comparison to random search as well as other baselines can be found in Figure 3.

(g) Did you perform multiple runs of your experiments and report seeds? For each of the experiments we performed three runs with different seeds $(333, 444, 555)$.

(h) Did you use tabular or surrogate benchmarks for in-depth evaluations? We used NAS-Bench-201 as a tabular benchmark.

3. **Best practices for reporting important details**

(a) Did you report how you tuned hyperparameters, and what time and resources this required? We use BOHB[37] to tune the hyperparameters of our method. We ran BOHB with three different random seeds for 24 hours.

(b) Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)? Yes.

(c) Did you report all the details of your experimental setup? The details of our experimental setup can be found in Section 4.