Speeding up NAS with Adaptive Subset Selection

Vishak Prasad[1] **Colin White**[2] **Paarth Jain**[1] **Sibasis Nayak**[1] **Rishabh Iyer**[3]
Ganesh Ramakrishnan[1]

[1]Indian Institute of Technology, Bombay
[2]Abacus.AI
[3]The University of Texas at Dallas

Abstract The majority of recent developments in neural architecture search (NAS) have been aimed at decreasing the computational cost of various techniques without affecting their final performance. Towards this direction, many low-fidelity and performance prediction methods have been considered, including using subsets of the training data. In this work, we initiate the study of *adaptive* subset selection for NAS and present it as complementary to state-of-the-art NAS approaches. We uncover a natural connection between one-shot NAS algorithms and adaptive subset selection and devise an algorithm that makes use of state-of-the-art techniques from both areas. We use these techniques to substantially reduce the runtime of DARTS-PT, a leading one-shot NAS algorithm, without sacrificing accuracy. Our results are consistent across multiple datasets, and our code and all materials needed to reproduce our results are available at `https://anonymous.4open.science/r/SubsetSelection_NAS-2BE4`.

1 Introduction

Recent developments in neural architecture search (NAS) have focused on decreasing the cost of NAS without sacrificing performance. Towards this direction, "one-shot" methods improve the search efficiency by training just a single over-parameterized neural network (supernetwork) [18, 1]. For example, the popular DARTS [18] algorithm applies a continuous relaxation to the architecture parameters, allowing the architecture parameters and the weights to be simultaneously optimized via gradient descent. While many follow-up works have improved the performance of DARTS [28, 13], the algorithms are still slow and are expensive in terms of budget and CO2 emissions [27].

On the other hand, the field of subset selection for efficient machine learning model training has seen a flurry of activity. In this area of study, facility location [20], clustering [2], and other subset selection algorithms are used to select a representative subset of the training data, substantially reducing the runtime of model training. Recently, adaptive subset selection algorithms have been used to speed up model training even further [12, 11]. Adaptive subset selection is a powerful technique which regularly updates the current subset of the data as the search progresses, to ensure that the performance of the model is maintained.

In this work, we uncover a natural connection between one-shot NAS algorithms and adaptive subset selection to devise an algorithm that makes use of state-of-the-art techniques from both areas. Specifically, DARTS-PT [28] (a leading one-shot algorithm) and GLISTER [12] (a leading adaptive subset selection algorithm) are both cast as bi-level optimization problems on the training and validation sets, allowing us to formulate a combined approach, Adaptive-NAS, as a mixed discrete and continuous bi-level optimization problem (see Figure 1 for an overview). Across several search spaces, we show that the resulting algorithm achieves significantly improved runtime compared to DARTS-PT, without sacrificing performance. Specifically, due to the use of adaptive subset selection, the training data can be reduced to 10% of the full training set size, resulting in an order of magnitude decrease in runtime, without sacrificing accuracy. To validate this approach, we compare to baselines such as facility location, entropy-based subset selection [21], and random subset selection.

AutoML Conference 2022 Workshop Track

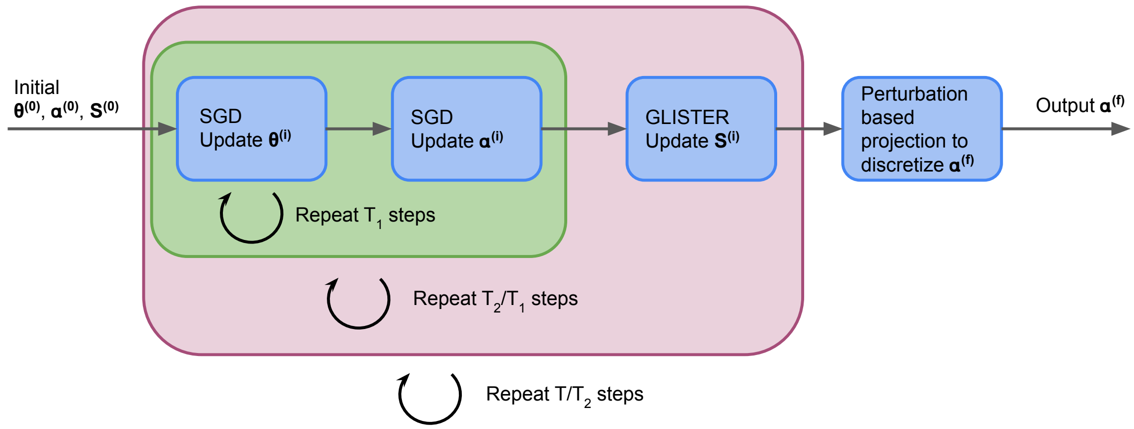

Figure 1: Overview of Adaptive-NAS. The algorithm starts with initial weights $\theta^{(0)}$, architecture hyperparameters $\alpha^{(0)}$, and subset of the training data $S^{(0)}$. Throughout the search, the weights $\theta^{(i)}$ and architecture weights $\alpha^{(i)}$ are updated with SGD, and the subset $S^{(i)}$ is updated with GLISTER, according to different time schedules. Then the final architecture $\alpha^{(f)}$ is discretized and returned.

Our contributions. We summarize our main contributions below.

- We introduce Adaptive-NAS, the first NAS algorithm to make use of adaptive subset selection. This technique is used to substantially reduce the training time needed for one-shot NAS algorithms. We also add facility location as a novel baseline for subset selection applied to NAS.
- Through extensive experiments, we show that Adaptive-NAS substantially reduces the runtime needed for DARTS-PT, with no decrease in final accuracy of the returned architecture. We release all of our code and materials needed to reproduce our results.

2 Related Work

Neural architecture search. NAS has been studied since the 1980s [5, 26, 19] and has been revitalized in the last few years [35, 18, 8, 23]. One line of work aims to predict the performance of neural architectures before they are fully trained, through low-fidelity estimates such as training for fewer epochs [34, 30] or learning curve extrapolation [3, 32]. Another line of work takes a *one-shot* approach by representing the entire space of neural architectures by a single "supernetwork", and then performing gradient descent to efficiently converge to a high-performing architecture [18]. Many follow-up works have improved its performance [17, 31, 13, 15, 33]. Recently, Wang et al. [28] introduced a more reliable perturbation-based operation scoring technique when returning the final architecture, achieving much stronger performance compared to DARTS.

Subset selection. Many tools have been developed in the field of subset selection for efficient machine learning model training. Popular fixed subset selection methods include coreset algorithms [6, 20], facility location [20], and entropy-based methods [21]. Recently, GLISTER was proposed as an adaptive subset selection method based on a greedy search [12]; an adaptive gradient-matching algorithm for subset selection was also proposed [11].

Subset selection in NAS. A few existing works have applied (offline) subset selection to the field of NAS. Na et al. [21] consider facility location, k-center, and entropy-based techniques, showing that the latter results in the best speedup for DARTS. Shim et al. [24] consider core-set sampling to speed up PC-DARTS by a factor of 8. Concurrent work [10] uses subset selection algorithms to speed up on multi-fidelity methods such as Hyperband [16] and ASHA [14]. Finally, other recent work uses a generative model to create a small set of *synthetic* training data, used to efficiently train architectures during NAS [25, 22].

3 Methodology

Preliminaries. In this section, we start by reviewing the ideas behind DARTS and DARTS-PT. The DARTS search space consists of a cell expressed as a directed acyclic graph, where each edge (i, j) can take on choices of operations $o^{(i,j)}$ such as `max_pool_3x3` or `sep_conv_5x5`. Denote the full set of possible operations by $\mathcal{O}$. Each choice of operation for (i, j) is given a continuous variable $\alpha^{(i,j)}$. Denote $\mathcal{U}$ and $\mathcal{V}$ to be the training and validation sets, respectively. Denote $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ as the training and validation loss, respectively. These losses are determined by the architecture weights, the architecture itself, and the dataset.

DARTS and DARTS-PT are both gradient-based optimization methods which train a supernetwork consisting of weights θ and architecture parameters α. DARTS and DARTS-PT both attempt to solve the following expression via alternating gradient updates:

$$\min_{\alpha} \mathcal{L}_{\text{val}} \left(\arg\min_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, \mathcal{U}\right), \alpha, \mathcal{V} \right). \tag{1}$$

GLISTER. GLISTER approximately solves the following expression by first approximating the bi-level optimization expression using a single gradient step, and then using a greedy data subset selection procedure [12].

$$\min_{S \subseteq \mathcal{U}, |S| \leq k} \mathcal{L}_{\text{val}} \left(\arg\min_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, S\right), \alpha, \mathcal{V} \right). \tag{2}$$

Adaptive-NAS. We present a formulation that organically combines Expressions (1) and (2) into a single mixed discrete and continuous bi-level optimization problem. The inner optimization is the minimization of training loss during architecture training, over a subset of the training data of size k. In the outer optimization, we minimize the validation loss by simultaneously optimizing for the neural architecture and for the subset of the training data. This allows us to efficiently find the best neural architecture:

$$\arg\min_{S \subseteq \mathcal{U}, |S| \leq k, \alpha} \mathcal{L}_{\text{val}} \left(\arg\min_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, S\right), \alpha, \mathcal{V} \right). \tag{3}$$

Evaluating this expression is computationally prohibitive because of the expensive inner optimization. Instead, we iteratively perform a joint optimization of the inner weights, and the outer training subset and architecture. In order to iteratively update the training subset and architecture, we compute meta-approximations of the inner optimization. For the architecture, we compute

$$\nabla_{\alpha} \mathcal{L}_{\text{val}} \left(\arg\min_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, S\right), \alpha, \mathcal{V} \right) \tag{4}$$

$$\approx \nabla_{\alpha} \mathcal{L}_{\text{val}} \left(\theta - \zeta \nabla_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, S\right), \alpha, \mathcal{V}\right). \tag{5}$$

For the subset, following Killamsetty et al. [12], we run a greedy procedure using a similar approximation to the inner optimization:

$$\mathcal{L}_{\text{val}} \left(\arg\min_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, S\right), \alpha, \mathcal{V} \right) \tag{6}$$

$$\approx \mathcal{L}_{\text{val}} \left(\theta - \zeta \nabla_{\theta} \mathcal{L}_{\text{train}} \left(\theta, \alpha, S\right), \alpha, \mathcal{V}\right). \tag{7}$$

Then we can iteratively update the outer parameters (architecture and subset), and the inner parameters (weights). Following prior work [12, 18], we only update the architecture and subset every t_1 and t_2 steps, respectively, for efficiency.

Table 1: Performance of one-shot NAS algorithms on NAS-Bench-201 CIFAR-10.

Performance on NAS-Bench-201 CIFAR-10			
Algorithm	Test accuracy	GPU hours	% Data used
DARTS-PT	88.21 (88.11)	7.50	100
DARTS-PT-ENTROPY	86.31 ± 4.66	**0.62**	10
DARTS-PT-RAND	86.94 ± 3.58	**0.62**	10
DARTS-PT-FL	89.27 ± 1.09	1.60	10
Adaptive-NAS	**92.22** ± 1.76	0.83	10

4 Experiments

In this section, we describe our experimental setup and results. We run experiments on NAS-Bench-201 with CIFAR-10, CIFAR-100, and ImageNet16-120, DARTS with CIFAR-10, and DARTS-S4 with CIFAR-10. See Appendix B for details.

We run experiments with DARTS-PT, Adaptive-NAS, and three other (non-adaptive) data subset selection methods added to DARTS-PT: DARTS-PT-RAND (random subset selection), DARTS-PT-FL (facility location), and DARTS-PT-ENTROPY (the entropy-based method taken by Na et al. [21]).

Experimental setup. Following Wang et al. [28], we use 50% of the full training dataset for supernet training and 50% for validation, and we report the accuracy of the final returned architecture on the held-out test set. In our main experiments, for all of the (adaptive and non-adaptive) subset selection methods, we set the subset size to 10% of the training dataset. We run the same experimental procedure for each method: select a size-10% subset of the full training dataset, train and discretize the supernet on the subset, and train the final architecture using the full training dataset. For DARTS-PT, we run the same procedure using the full training dataset at each step. We otherwise use the exact same training pipeline as in Wang et al. [28]: batch size of 64, learning rate of 0.025, momentum of 0.9, and cosine annealing.

We run all experiments on a NVIDIA Tesla V100 GPU. We run each algorithm with 5 random seeds, reporting the mean and standard deviation of each method, except DARTS-PT: due to the extreme runtime and availability of existing results, we run once and verify that the result is nearly identical to published results [28]. We also report the time it takes to output the final returned architecture.

Experimental results and discussion. In Tables 1, 2, and 3, we report the results on NAS-Bench-201. On CIFAR-10 and ImageNet16-120, Adaptive-NAS achieves significantly higher accuracy than all other algorithms tested. On CIFAR-100, Adaptive-NAS is essentially tied with DARTS-PT-FL for the highest accuracy. Furthermore, all NAS algorithms that use subset selection have significantly decreased runtime – Adaptive-NAS sees a factor of 9 speedup compared to DARTS-PT. Note that DARTS-PT-FL takes more time when the number of examples per class in the dataset is higher, so it sees comparatively higher runtimes on CIFAR-10.

In Tables 8 and 9, we report the results on S4 CIFAR-10 and DARTS CIFAR-10. Once again, the runtime of Adaptive-NAS is significantly faster than DARTS-PT – a factor of 7 speedup. On these search spaces, the performance of the subset-based methods are more similar when compared to NAS-Bench-201, and on the DARTS search space, Adaptive-NAS does not outperform DARTS-PT. A possible explanation is that S4 and DARTS are significantly larger search spaces than NAS-Bench-201 and require more training data to distinguish between architectures. To test this, we included one more experiment in Table 9, giving Adaptive-NAS 20% training data instead of 10%. We find that the accuracy significantly increases, moving within one standard deviation of the accuracy of DARTS-PT.

Table 2: Performance of one-shot NAS algorithms on NAS-Bench-201 CIFAR-100.

Performance on NAS-Bench-201 CIFAR-100			
Algorithm	Test accuracy	GPU hours	%Data used
DARTS-PT	61.650	8.00	100
DARTS-PT-ENTROPY	56.79 ± 7.63	**0.58**	10
DARTS-PT-RAND	56.95 ± 4.54	**0.58**	10
DARTS-PT-FL	**64.28** ± 3.10	0.67	10
Adaptive-NAS	64.27 ± 3.37	0.87	10

Table 3: Performance of one-shot NAS algorithms on NAS-Bench-201 ImageNet16-120.

Performance on NAS-Bench-201 Imagenet16-120			
Algorithm	Test accuracy	GPU hours	%Data used
DARTS-PT	35.00	33.50	100
DARTS-PT-ENTROPY	26.52 ± 3.73	**1.58**	10
DARTS-PT-RAND	27.04 ± 5.53	**1.58**	10
DARTS-PT-FL	29.30 ± 5.35	1.90	10
Adaptive-NAS	**36.10** ± 6.96	2.60	10

Overall, Adaptive-NAS achieves the highest average performance across all search spaces. Furthermore, Adaptive-NAS achieves no less than a seven-fold increase in runtime compared to DARTS-PT, on all search spaces.

In Appendix B, we give ablation studies on the percentage of data used.

5 Conclusions, Limitations, and Broader Impact

In this work, we used a connection between one-shot NAS algorithms and adaptive subset selection to devise an algorithm that makes use of state-of-the-art techniques from both areas. Specifically, DARTS-PT and GLISTER, leading techniques in one-shot NAS and adaptive subset selection, respectively, are both cast as bi-level optimization problems on the training and validation sets, which allowed us to formulate a combined approach, Adaptive-NAS, as a mixed discrete and continuous bi-level optimization problem. We showed that the resulting algorithm is able to train on an (adaptive) dataset that is 10% of the size of the full training set, without sacrificing accuracy, resulting in an order of magnitude decrease in runtime.

Limitations. While Adaptive-NAS uses a subset of the data when training and discretizing the supernetwork, the full dataset is used for training the final architecture. An interesting direction for future work is to use an adaptive subset of the data even when training the final architecture, which may lead to even faster runtime, perhaps at a small cost to performance. Another interesting direction for future work is to apply adaptive subset selection to other non supernet-based NAS algorithms such as regularized evolution [23] or BANANAS [29].

Broader impact. Our work combines techniques from two different areas: adaptive subset selection for machine learning, and neural architecture search. The goal of our work is to make it easier and quicker to develop high-performing architectures on new datasets. Our work also helps to unify two sub-fields of machine learning that had thus far been disjoint. Since the end product of our work is a NAS algorithm, it is not itself meant for one application but can be used in any end-application. For example, it may be used to more efficiently find deep learning architectures for applications that help to reduce CO2 emissions, or for creating large language models. Our hope is that future AI models discovered by our work will have a net positive impact, due to the push for the AI community to be more conscious about the societal impact of its work [7].

6 Reproducibility Checklist

1. For all authors...

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

   (b) Did you describe the limitations of your work? [Yes] [See Section 5.]

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] [See Section 5.]

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

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] [We did not include theoretical results.]

   (b) Did you include complete proofs of all theoretical results? [N/A] [We did not include theoretical results.]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] [We include the code, data, and instructions to reproduce the results here: `https://anonymous.4open.science/r/SubsetSelection_NAS-2BE4`.]

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] [We include our raw results; see `https://anonymous.4open.science/r/SubsetSelection_NAS-2BE4/README.md`.]

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] [We include scripts to generate our exact results. See `https://anonymous.4open.science/r/SubsetSelection_NAS-2BE4`.]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes] [We included multipe documentation files, and put in a reasonable effort to make our code as easy to use as possible.]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] [See Sections 3 and 4.]

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] [We compared to different methods in exactly the same setting. See Section 4.]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] [See Section 4.]

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes] [See Section 4.]

(i) Did you compare performance over time? [No] [Since we only compared one-shot NAS methods, we did not compare performance over time.]

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] [We ran 5 random seeds for each method.]

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] [We reported standard deviation. See Section 4.]

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

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

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A] [We did not tune any hyperparameters.]

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

   (a) If your work uses existing assets, did you cite the creators? [Yes] [We cited all assets we used.]

   (b) Did you mention the license of the assets? [N/A] [All assets were public.]

   (c) Did you include any new assets either in the supplemental material or as a URL? [No] [We did not include new assets.]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Our experiments were conducted only on publicly available datasets.

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Our experiments were conducted only on publicly available datasets.

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

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] [We did not conduct research with human subjects.]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] [We did not conduct research with human subjects.]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] [We did not conduct research with human subjects.]

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018.

[2] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[3] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

[4] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020.

[5] WB Dress. Darwinian optimization of synthetic neural systems. Technical report, Oak Ridge National Lab., TN (United States), 1987.

[6] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300, 2004.

[7] Brent Hecht, Lauren Wilcox, Jeffrey P Bigham, Johannes Schöning, Ehsan Hoque, Jason Ernst, Yonatan Bisk, Luigi De Russis, Lana Yarosh, Bushra Anjum, Danish Contractor, and Cathy Wu. It's time to do something: Mitigating the negative impacts of computing through a change to the peer review process. *ACM Future of Computing Blog*, 2018.

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

[9] Vishal Kaushal, Ganesh Ramakrishnan, and Rishabh Iyer. Submodlib: A submodular optimization library. *arXiv preprint arXiv:2202.10680*, 2022.

[10] Krishnateja Killamsetty, Guttu Sai Abhishek, Alexandre V Evfimievski, Lucian Popa, Ganesh Ramakrishnan, Rishabh Iyer, et al. Automata: Gradient based data subset selection for compute-efficient hyper-parameter tuning. *arXiv preprint arXiv:2203.08212*, 2022.

[11] Krishnateja Killamsetty, S Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pages 5464–5474. PMLR, 2021.

[12] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. Glister: Generalization based data subset selection for efficient and robust learning. *arXiv preprint arXiv:2012.10630*, 2020.

[13] Kevin Alexander Laube and Andreas Zell. Prune and replace nas. *arXiv preprint arXiv:1906.07528*, 2019.

[14] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *MLSys Conference*, 2020.

[15] Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[16] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *JMLR*, 2018.

[17] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019.

[19] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[20] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020.

[21] Byunggook Na, Jisoo Mok, Hyeokjun Choe, and Sungroh Yoon. Accelerating neural architecture search via proxy data. *arXiv preprint arXiv:2106.04784*, 2021.

[22] Aditya Rawal, Joel Lehman, Felipe Petroski Such, Jeff Clune, and Kenneth O. Stanley. Synthetic petri dish: A novel surrogate model for rapid architecture search, 2020.

[23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[24] Jae-hun Shim, Kyeongbo Kong, and Suk-Ju Kang. Core-set sampling for efficient neural architecture search. *arXiv preprint arXiv:2107.06869*, 2021.

[25] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data, 2019.

[26] Manoel Tenorio and Wei-Tsih Lee. Self organizing neural networks for the identification problem. *Advances in Neural Information Processing Systems*, 1, 1988.

[27] Tanja Tornede, Alexander Tornede, Jonas Hanselle, Marcel Wever, Felix Mohr, and Eyke Hüllermeier. Towards green automated machine learning: Status quo and future directions. *arXiv preprint arXiv:2111.05850*, 2021.

[28] Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*, 2021.

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

[30] Colin White, Arber Zela, Binxin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *arXiv preprint arXiv:2104.01177*, 2021.

[31] 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 *ICLR*, 2019.

[32] Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. In *NeurIPS*, 2021.

[33] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

[34] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11396–11404, 2020.

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