Accelerated NAS via pretrained ensembles and multi-fidelity Bayesian Optimization

Houssem Ouertatani, Cristian Maxim, Smail Niar, El-Ghazali Talbi

Accelerated NAS via pretrained ensembles and multi-fidelity Bayesian Optimization

Houssem Ouertatani[1,3], Cristian Maxim[1], Smail Niar[2], and El-Ghazali Talbi[3]

[1]IRT SystemX - France
[2]UPHF & LAMIH UMR CNRS, Valenciennes - France
[3]Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Bayesian optimization (BO) is a black-box search method particularly valued for its sample efficiency. It is especially effective when evaluations are very costly, such as in hyperparameter optimization or Neural Architecture Search (NAS). In this work, we design a fast NAS method based on BO. While Gaussian Processes underpin most BO approaches, we instead use deep ensembles. This allows us to construct a unified and improved representation, leveraging pretraining metrics and multiple evaluation fidelities, to accelerate the search. More specifically, we use a simultaneous pretraining scheme where multiple metrics are estimated concurrently. Consequently, a more general representation is obtained. A novel multi-fidelity approach is proposed, where the unified representation is improved both by high and low quality evaluations. These additions significantly accelerate the search time, finding the optimum on NAS-Bench-201 in an equivalent time and cost to performing as few as 50 to 80 evaluations. The accelerated search time translates to reduced costs, in terms of computing resources and energy consumption. As a result, applying this NAS method to real-world use cases becomes more practical and not prohibitively expensive. We demonstrate the effectiveness and generality of our approach on a custom search space. Based on the MOAT architecture, we design a search space of CNN-ViT hybrid networks. The search method yields a better-performing architecture than the baseline in only 70 evaluations.

1 Introduction

The design of high-quality neural network architectures has been a major driver of the impressive advances seen in deep learning and its applications. Many of the introduced improvements in widely-used architectures come from years of iterations and trial-and-error, guided by a set of general best practices and intuitions.

Neural architecture search (NAS) [1] seeks to automate this design process. The goal is to find the best performing architecture(s), according to one or many criteria, in a large search space. NAS can be useful in adapting a widely used architecture to a specific use case, or in building new architectures.

NAS consists of 3 main components: The search space, the search method, and the architecture performance evaluation method. Many search methods have been successfully used to perform NAS, including reinforcement learning [30, 9], evolutionary algorithms [4, 13, 14], local search [26], and Bayesian optimization [24]. The evaluation of potential architectures is usually expensive, as it

consists of a training phase and a test phase. With considerations of cost, energy consumption, and practicality, there is a special focus to explore these search spaces efficiently. This can be achieved by speeding up each evaluation, by making as few evaluations as possible, or a combination of the two.

1.1 Bayesian optimization

Surrogate model-based optimization methods offer a viable solution for searching in large search spaces when evaluations are expensive. Instead of performing many costly evaluations, a model approximates the expensive objective function. In particular, Bayesian optimization (BO) relies on a probabilistic model to guide the search by suggesting which points to evaluate next at each iteration. This makes BO an effective blackbox search approach, particularly noted for its good sample efficiency. As a result, it is a useful method in Neural Architecture Search.

In Bayesian optimization literature, the model is most often a Gaussian Process (GP) [7]. However, GPs have certain shortcomings. A commonly mentioned limitation is scalability: as they scale in $O(n^3)$ w.r.t the number of datapoints, they can become computationally expensive. In addition, GP-based Bayesian optimization can be difficult to apply for use cases like Neural Architecture Search. More specifically, the required kernel function and distance function can be non-trivial for complicated spaces like neural network architecture spaces. A number of kernels and distance measures have been suggested, including optimal transport metrics for architectures of neural networks (OTMANN) [11], Weisfeiler-Lehman kernels [17], edit-distance neural network kernels [10], and phenotypic distance [8].

However, it is possible to use alternative models for Bayesian Optimization, such as random forests and Bayesian neural networks [7]. For NAS in particular, models like graph convolutional networks or Bayesian graph NNs have yielded good results [16, 21].

1.2 Deep ensembles and NAS

Ensembling methods [6] involve aggregating the predictions of a set of diverse models. Using an aggregate prediction is more robust and reliable than using an individual prediction model. The models can compensate mutual weaknesses and offer complementary strengths. A deep ensemble (DE) is the ensemble of a number of independently-trained neural networks.

DEs can reliably quantify uncertainty in many scenarios [12], and are effective in out-of-distribution cases. With a large number of parameters in neural networks, there are many low-loss regions which can approximate the training data. Deep ensembles can produce a set of diverse networks representing different low-loss regions. In practice, compared to other models such as Bayesian NNs, DEs are also easier to implement and to parallelize.

The Trieste BO-based search framework [20] presents an example of using a deep ensemble as a probabilistic model for Bayesian optimization. In the NAS context, the most prominent example is BANANAS [24], where the authors thoroughly analyzed many components of BO applied to architecture search: predictor choice, acquisition functions, etc.. This led to the design of a successful NAS method based on an ensemble of networks, coupled with a novel architecture encoding scheme: path encoding. This work clearly demonstrated the potential of deep ensemble-based NAS.

Compared to GPs, the previously mentioned methods eliminate the need for kernel and distance functions. Flexibility in the choice of architecture for the ensemble networks is another advantage.

For example, a GNN-based predictor can be used for graph-based search spaces. The ensembling allows an ordinarily non-probabilistic model to be an effective predictor for BO.

1.3 Contributions

Our NAS method is built on the idea of deep-ensembles as a predictor for BO-based NAS, focusing specifically on the capacity and advantages of **weights reuse**. In fact, DEs inherently enable the use of their weights to approximate many prediction targets. As a result, instead of only relying on the expensive evaluations of the objective function, we can leverage less expensive data to improve the internal representations of the ensemble. The idea is to incorporate useful information from cheaper-to-evaluate sources in the weights of the ensemble, with the aim of reducing search time and cost. We illustrate this with two complementary ideas: **simultaneous pretraining** on zero-cost metrics, and **multi-fidelity search**.

While pretraining is widely-used in machine learning, our main insight in the **simultaneous pretraining** section is how to efficiently combine multiple pretraining targets. Instead of pretraining the ensemble on one metric at a time, we use a shared embedding to predict all the metrics at once. This ensures the internal representation is more generic, and results in a significant speedup of the BO search using the pretrained ensemble.

The notion of shared embeddings also allows us to share information between different fidelity levels, allowing us to trade a few high quality evaluations for many lower quality evaluations. This **multi-fidelity BO** approach also considerably accelerates the search.

Our experiments on a NAS benchmark have shown that these two additions to deep ensemble-based NAS significantly reduce the search cost and boost the performance. In order to test our method's generality and applicability outside of NAS benchmarks, we construct a custom search space of ViT-CNN networks. The search space is based on an existing family of architectures, the MOAT family [29]. Our search method quickly yields an improved architecture.

We summarize our contributions as follows:

- Simultaneous pretraining on multiple zero-cost metrics, which outperforms classic pretraining

- Multi-fidelity BO using shared weights

- An application example on a custom search space

2 NAS method description

2.1 Bayesian optimization with DEs

At a glance, the Bayesian optimization (BO) procedure with deep ensembles is similar to BO with Gaussian processes (Figure 1). At each iteration, the probabilistic model is used to select the best points to be evaluated, and the evaluations are in turn used to update the model and improve it in preparation for the next iterations.

The acquisition function is the criteria used to select the most interesting points to evaluate. It relies on the predictions and uncertainty measures provided by the probabilistic model. Different acquisition functions strike various balances between exploration and exploitation.

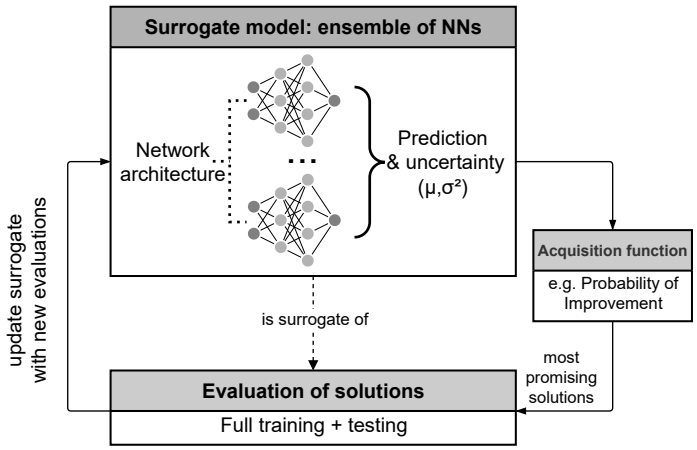


Figure 1: Overview of Bayesian optimization with deep ensembles

The surrogate model we use is a deep ensemble, made up of a number of neural networks which take an architecture (described by an encoding), and predict its performance. Owing to the different initializations, the networks in the ensemble occupy different low-loss regions in the loss landscape. As a result, their predictions for a particular candidate architecture are different, and combining them yields a more robust prediction.

We used the probability of improvement (PI) as acquisition function. It selects the points with the highest probability of improving over the current best observed value. Although in GP-based BO other acquisition functions (e.g. Expected Improvement) usually outperform PI, our tests showed PI to result in slightly faster search times. We use the predictions computed by the ensemble networks as MC samples to approximate the acquisition function PI.

At each iteration, the newly acquired observations are added to the training set of the deep ensemble, and the networks of the DE are trained to incorporate the new information.

2.2 Simultaneous pretraining

With the performance gains it unlocks, pretraining is a widely-used technique in machine learning. Pretraining gives the model a head-start, by allowing it to learn general patterns in the data beforehand, which boosts its training and performance later.

In Bayesian Optimization, there are methods to pretrain GPs, such as HyperBO [23]. For Neural Architecture Search, FBNetV3 [2] is a notable example of successfully using a pretrained predictor in the search process. In a first step, architecture statistics are used to pretrain a predictor, which is then used as a proxy for the objective function in a predictor-based evolutionary search. The pretraining procedure led to an important boost in the sample efficiency of the predictor.

In the context of Bayesian optimization, the use of deep ensembles significantly simplifies this pretraining step. As opposed to GPs and other probabilistic models, pretraining a deep ensemble is simple: it trivially consists of pretraining its component networks. The impact of pretraining in general, and the way to pretrain the ensemble, is well established. Instead, our main insight is related to the pretraining data we use, and especially to the manner in which the pretraining on many metrics is performed.

$$\mathbf{y} = L_n(L_{n-1}(....(L_1(\mathbf{x}))))$$

$$\begin{aligned} \text{metric}_1 &= f_1(\mathbf{y}) \\ \text{metric}_2 &= f_2(\mathbf{y}) \\ &... \\ \text{metric}_m &= f_m(\mathbf{y}) \end{aligned}$$

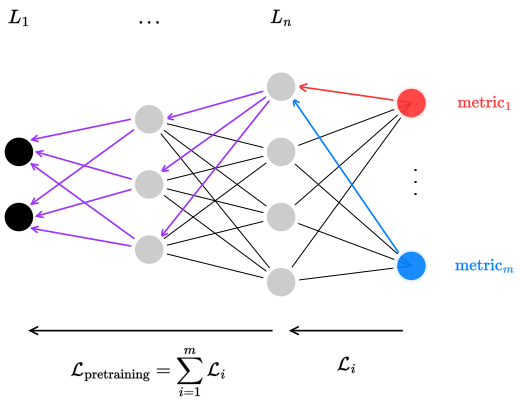


Figure 2: Backpropagation step of the simultaneous pretraining scheme, with a separate prediction head for each zero-cost metric

For neural networks in the search space, we can quickly calculate a number of metrics to use as pretraining data. We focus on zero-cost metrics, ie. metrics whose computation or estimation doesn't require prior training of the neural network in question. These metrics depend only on the architecture, or hardware, but not on whether it is trained or not. Among such zero-cost metrics, we use the number of trainable parameters, the average inference latency, the number of floating point operations (FLOPs) and memory footprint.

With access to multiple pretraining metrics, the main question we focused on is **which pretraining method is the best way to fully leverage these multiple metrics**.

Our idea is to force the common representation generated by the networks to remain as generic as possible, while retaining relevant information. Since these metrics are not our real prediction target, simultaneous pretraining is used to keep the embedding from specializing on any one metric to the detriment of our real prediction target. Practically, this is achieved by using a common embedding to predict all the metrics simultaneously.

In practice, each ensemble network is composed of a section of n layers $(L_1, .., L_n)$, followed by prediction head(s). This first section is mutualized to all the pretraining metrics. In the forward step, this shared section produces a common embedding $\mathbf{y}$ of the input data $\mathbf{x}$. The embedding $\mathbf{y}$ is subsequently used to predict each of the m pretraining metrics, using a separate prediction head f_i for each metric.

In other words, all the metrics are predicted using a common vector $\mathbf{y}$ representing the input architecture. The loss is the sum of losses of each pretraining metric, as illustrated in Figure 2. As a result of this setup, the backpropagation step updates the weights of the shared section using combined loss information from all the zero-cost metrics.

In our experiments, we measure the impact of correlations between the metrics, and the impact of the number of metrics used. We also compare this pretraining scheme to performing no pretraining and to sequential pretraining.

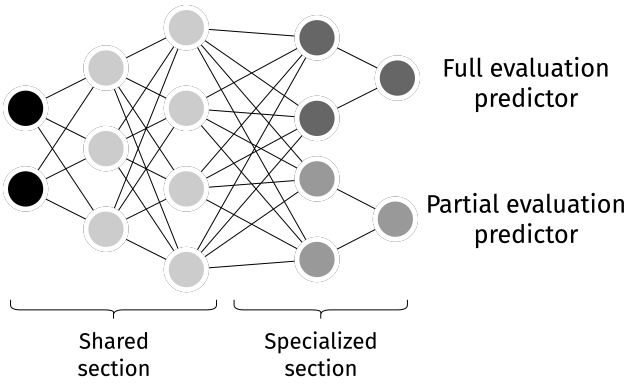


Figure 3: Ensemble network architecture for multi-fidelity search

2.3 Multi-fidelity search

The motivation for multi-fidelity search is to replace a low number of high quality evaluations, with a high number of lower quality evaluations. For example, we can use the validation score after fully training the model (N epochs) as the high-fidelity evaluation. The low-fidelity evaluation is then the validation score after a smaller number n of training epochs ($n < N$).

The practical implementation is similar to simultaneous pretraining: a common embedding is used to estimate both fidelity levels of the evaluation. As a result, in a similar way to the multi-head pretraining method, our ensemble networks have a shared section, followed by a specialized section (a dense layer and prediction head) for each level of fidelity. Figure 3 illustrates this architecture.

The training procedure is as follows:

- At each iteration, perform p low-fidelity evaluations.

- Every f iterations, perform q high-fidelity evaluations.

Both of these steps update the shared sections of the ensemble networks, which results in this section being updated more frequently than in the mono-fidelity case. The high-fidelity evaluations are given more importance than the low-fidelity evaluations, by training the ensemble on their data for more epochs.

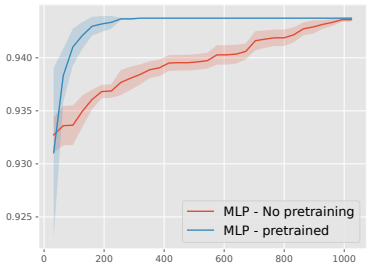
In subsequent experiments, we compare this training procedure to the mono-fidelity approach, and illustrate whether its impact is additive or contradictory with the impact of simultaneous pretraining.

3 Experiments and results

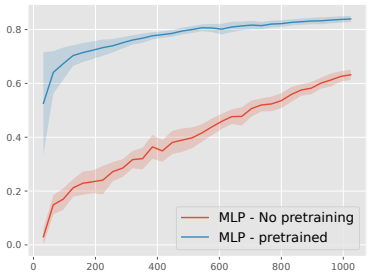
We use an MLP-based architecture for the deep ensemble networks, both in the benchmark and the custom search space (section 4).

3.1 Simultaneous pretraining experiments

In figure 4, we compare the evolution of the best value and the Spearman rank correlation with and without the simultaneous pretraining, as we advance along the search procedure. The Spearman rank

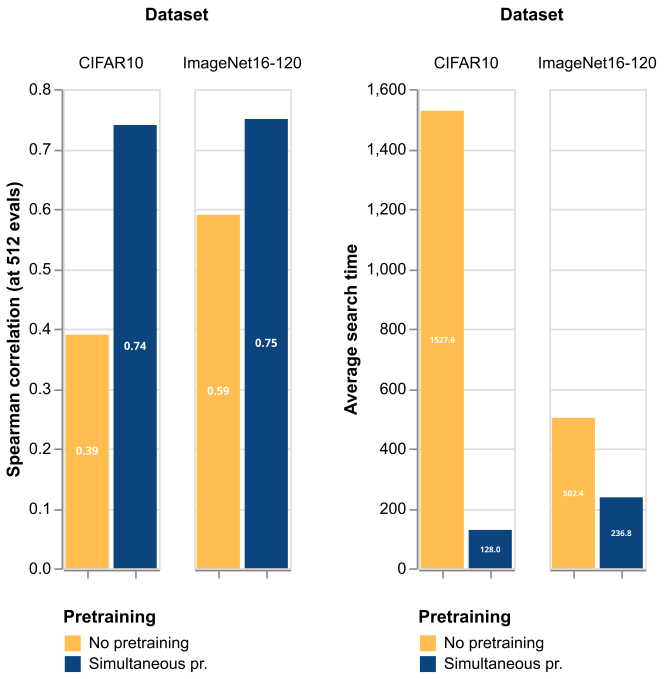


(a) Best value



(b) Spearman ρ

Figure 4: Best value and correlation plots (CIFAR10 - 20 run averages)



(a) Spearman ρ (b) Avg. search time

Figure 5: Impact of pretraining on correlation and search time (C10 and ImageNet16-120)

correlation designates how well the model ranks the architectures in the benchmark: the correlation between the true ranking, and the ranking based on the model's prediction. The x-axis tracks the number of evaluations performed.

In figure 5, we report the average search time, i.e. the number of evaluations needed to reach the optimum, for CIFAR10 and ImageNet16-120. We also report the Spearman rank correlation values ρ after 512 evaluations.

The results show that pretraining has a significant impact on the predictive performance of the ensemble: the increase in rank correlations means the model suggests better points to be evaluated, and as a result the best value plot shows a significant advantage for the pretrained DE.

3.1.1 Impact of the pretraining method

To specifically evaluate the impact of **simultaneous pretraining**, we compare it to **sequential pretraining**, where pretraining is performed on metric_1, then metric_2, and so on.

We varied the number of pretraining epochs to account for the different hyperparameters each pretraining mode might require. We reported the average search time in figure 6. The results show a clear trend where the search is significantly accelerated by simultaneous pretraining.

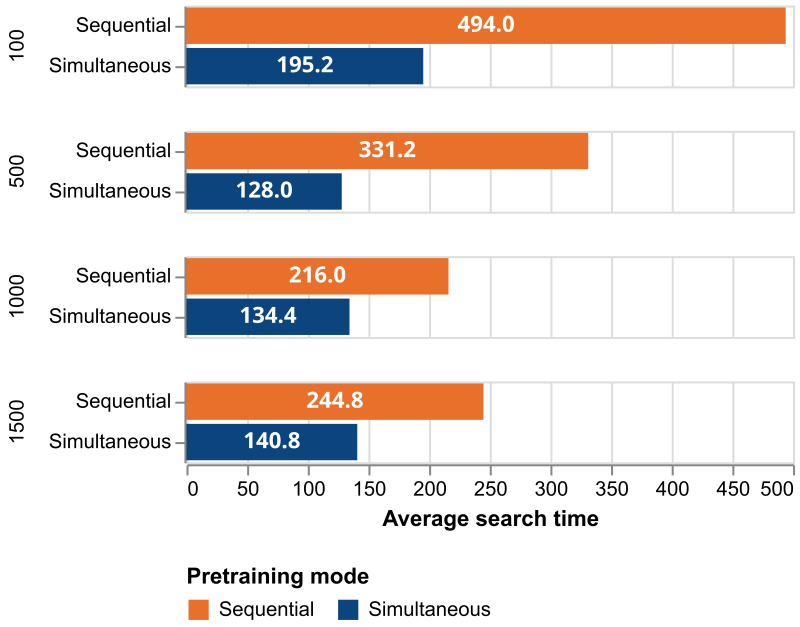


Figure 6: Average search times using sequential and simultaneous pretraining. The rows indicate the number of pretraining epochs.

Table 1: Pairwise correlations of pretraining metrics

	N.params	Latency	Hi-Fi acc.	Lo-Fi acc.
N.params	1	0.68	0.32	0.38
Latency	0.68	1	0.38	0.48

Table 2: Effect of the number of metrics on search time

N.metrics	1	2	3
Search time (in evaluations)	113.4	96	78.9

3.1.2 Pairwise correlations and number of pretraining metrics

We used the following zero-cost metrics: number of trainable parameters, FLOPs, average inference time. The NAS-Bench-201 [3] benchmark allows us to calculate some pairwise correlations between the different metrics and the evaluations at high and low fidelities (Table 1). We also measured the impact of the number of used metrics on the search time (Table 2). The search time corresponds here to the average number of evaluations needed to find the optimal architecture in the benchmark.

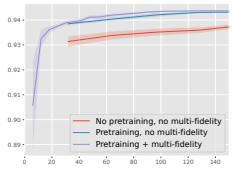
While testing with 2 metrics, we found that the differences in pairwise correlations have a small impact on the search time. The number of metrics has a much more significant impact. Therefore, using more metrics, even if that includes highly correlated metrics (e.g. number of parameters and

FLOPs), yields the fastest search times.

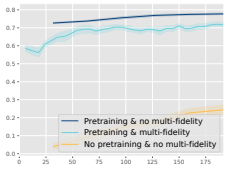
3.2 Multi-fidelity search experiments

NAS-Bench-201 includes validation scores after 12 epochs and after 200 epochs of training, which we use respectively as the low-fidelity and the high-fidelity evaluations. We use values of $p = 5$, $q = 3$, $f = 10$, ie. 3 full evaluations are performed for every 50 partial evaluations.

In figures 7 and 8, we plot the average best values found and Spearman rank correlation with and without multi-fidelity training. For CIFAR10, we also include the values without any pretraining or multi-fidelity training, to illustrate the combined impact of these two procedures. On the x-axis, we track the current cost in terms of "equivalent full evaluations", to be able to compare results with the mono-fidelity approach. We calculate this as the total number of epochs at that point, divided by the number of epochs for a full evaluation.

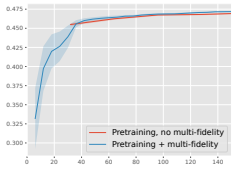


(a) Best value

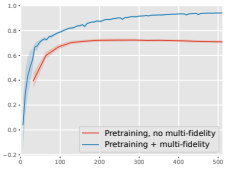


(b) Spearman ρ

Figure 7: CIFAR10



(a) Best value



(b) Spearman ρ

Figure 8: ImageNet120-16

Our experiments show that multi-fidelity search with shared weights improves the search speed with or without pretraining, and that combining both produces much faster search speeds. The next section provides an overview of the method, and its main results on NAS-Bench-201.

3.3 NAS Method overview and results

Figure 9 sums up the NAS approach in its two steps: the simultaneous pretraining, and the multi-fidelity BO loop. Figure 10 provides an overview of the impact simultaneous pretraining and multi-fidelity search have on search time.

In table 3, we report statistics about the number of evaluations needed to reach the optimum on NAS-Bench-201 datasets. We compare to other NAS methods in the literature where it was possible to access this data, most notably using the NASZilla library [27], which includes BANANAS [24] and the local search approach [25].

In table 4, we report the best accuracy value found by the search method, after respectively 100 and 200 evaluations (or queries) on NAS-Bench-201. We compare this with other NAS methods in the literature, either using our own tests or reported results from their respective papers.

On the 100 evaluation budget, our method finds architectures with the top value in all 3 benchmark datasets, finding the optimum for C100. By the 200 evaluations mark, our method find all 3 optima for the NAS-Bench-201 dataset.

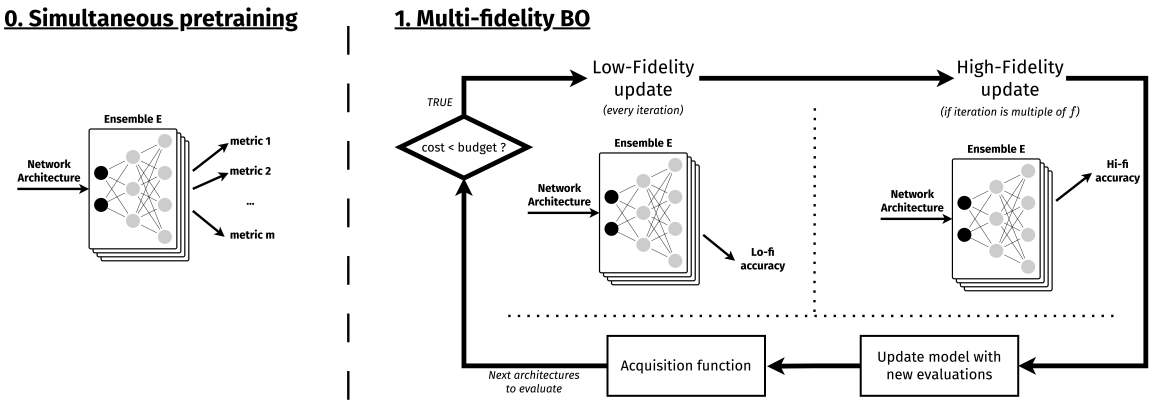


Figure 9: General overview of the NAS method. Ensemble E's weights are pretrained in **step 0**, then updated both by high fidelity and low fidelity evaluation data. It is used at each iteration to suggest the next points to evaluate.

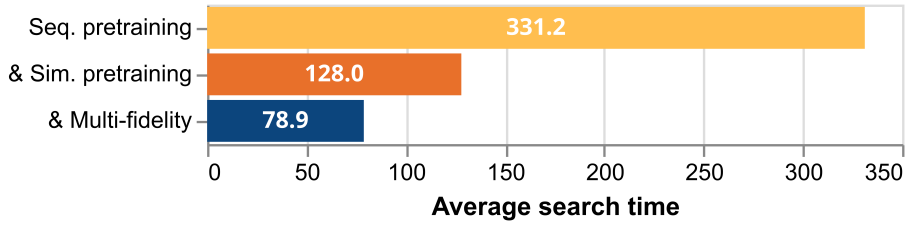


Figure 10: Combined effect of simultaneous pretraining and multi-fidelity search on the search time (CIFAR10, 20 run average)

Table 3: Statistics about the number of evaluations to reach optimum (20-run averages)

		Min	Max	Mean	Std
CIFAR10	Random search	280	13690	7717.0	4498.8
	Evolution	20	1000	250.0	247.41
	BANANAS	20	470	128.5	118.59
	Local search	30	190	99.0	56.82
	Ours	36	132	**78.9**	25.45
CIFAR100	Random search	520	14690	7141.5	4137.08
	Evolution	40	650	247.5	154.2
	BANANAS	40	310	101.0	54.49
	Local search	20	490	118.5	96.4
	Ours	18	96	**53.7**	24.03

Table 4: Best value after 100 and 200 epochs (20-run averages)

NAS Method	CIFAR10	CIFAR100	ImageNet16-120	Queries
Optimum*	**94.37**	**73.51**	**47.31**	
Arch2vec & BO [28]	94.18	73.37	46.27	100
AG-Net [15]	94.24	73.12	46.20	100
Random search	93.70	70.94	45.44	100
Local search	94.26	73.15	46.09	100
Evolution	94.20	72.70	46.03	100
BANANAS	94.15	73.28	46.46	100
Ours	**94.36**	**73.51***	**47.19**	100
Random search	93.86	71.78	45.70	200
Local search	**94.37***	73.48	46.21	200
Evolution	94.24	73.07	46.30	200
BANANAS	94.20	73.43	46.51	200
AG-Net	**94.37***	**73.51***	46.43	192
Ours	**94.37***	**73.51***	**47.31***	200

NB: There are of course better-performing networks on these datasets in the literature in general, but here we are interested in how fast the different methods perform the search, and are restricted to the architectures contained in the benchmark. As a result, the accuracies have an upper bound (first line).

4 Application to a custom search space

The architecture of the ensemble networks we used are generic, not tailored to the graph-based search space of the benchmarks. To further test the generality of this method, we decided to apply it to a very different search space, and kept the same ensemble architecture and hyperparameters.

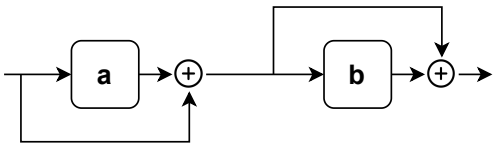
4.1 Search space design

MOAT [29] is a family of hybrid CNN-ViT networks for vision tasks. They are built on a hybrid block which effectively merges mobile convolution with transformer blocks: the MOAT block. The design of the MOAT block involves replacing the MLP in the transformer block with a mobile convolution, and rearranging the components.

Inspired by the architecture of the MOAT block, we built a search space of hybrid CNN-ViT networks. It contains purely mobile convolution-based networks including MobileNetV2 [19], purely transformer-based networks including ViT [22], and various hybrid configurations including MOAT.

It is a cell-based design, made up of 4 stages, where the image resolution is divided by 2 at each new stage. An architecture is described by which cell design is selected for each section of the network. Our optimization objective is to find the optimal sequence of cell types associated with the stages. This way, we allow sufficient flexibility, but avoid having an unreasonably large space potentially full of ineffective or unfeasible architectures, keeping it at a similar size to benchmark search spaces like NAS-Bench-201.

The possible cell designs were constructed by leveraging an architectural pattern common to the MOAT block, the ViT block and the MBConv block. They can all be described using a succession of two mini-blocks, each with a skip connection. Figure 11 illustrates this pattern.



Block	(a) miniblock	(b) miniblock
MBConv [19]	MBConv & SE	Zero
ViT [22]	Self-attention	MLP
MOAT [29]	MBConv, no SE	Self-attention

Figure 11: Meta-architecture of a cell in the search space

Table 5: Performance comparison of the searched architecture and the baseline MOAT architecture

Experiment	Baseline	Ours
Imagewoof - 200 epochs (Objective function)	81.13	83.76
Imagewoof - 300 epochs	84.34	86.54
Imagewoof - 300 epochs (MOAT-0 channels)	89.82	90.86
ImageNet-1k - 90 epochs	61.84	67.74

The cell types are described by specifying the miniblocks **(a)** and **(b)** from a list of possible miniblocks: {*MBConv with Squeeze-Excitation (SE), MBConv with no SE, Self-attention, MLP, Zero*}. We eliminate unfeasible designs, resulting in a total of 7 possible cell designs. We add two experimental cell types, with miniblocks **(a)** and **(b)** in parallel.

4.2 Experiment and results

The objective function is the validation score on Imagewoof [5], a challenging 10-class subset of ImageNet [18]. The high-fidelity evaluation is computed after 200 epochs of training epochs, and the low-fidelity evaluation is after 15 training epochs. We compute 4 pretraining metrics using PyTorch Profiler: FLOPs, inference time, GPU memory footprint and number of trainable parameters.

We use the `tiny-moat-0` configuration and number of channels as a baseline. It is a small network of around 2.7M parameters, and the ImageNet results should be interpreted in comparison to this baseline network and not performances of much bigger networks.

The baseline architecture **tiny-moat-0** is as follows:

```
(MBConv) | (MBConv) | (MOAT block)| (MOAT block)
```

The search procedure yielded the following architecture after approximately 70 evaluations:

```
(MBConv no SE) | (MOAT block) | (MBConv & SE + MLP) | (MOAT block & SE)
```

We report the validation results on Imagewoof and ImageNet-1k (with no pretraining) in table 5. We also test a bigger version of the baseline and the searched architecture, using the `MOAT-0` channel sizes. The search procedure has produced a better-performing architecture than the baseline we started with.

5 Conclusion

Deep ensembles are an effective probabilistic model for Bayesian optimization applied to NAS. They present a number of upsides, especially the flexibility to leverage additional sources of information to improve the predictive performance. We showcase this in practice with simultaneous pretraining on zero-cost metrics and multi-fidelity search. These additions lead to significant speedups of the search procedure, respectively accelerating the search 2.5 and 1.6 times on NAS-Bench-201 (CIFAR10). We also highlight how our method can be applied more generally outside the scope of usual NAS benchmarks, by successfully applying it on a fully custom and complex hybrid CNN-ViT search space, outperforming the baseline on ImageNet-1k. As a conclusion, our NAS method is a practical and effective way to search for architectures in customized and novel search spaces.

References

[1] Hadjer Benmeziane et al. "Hardware-Aware Neural Architecture Search: Survey and Taxonomy". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Aug. 2021, pp. 4322–4329.

[2] Xiaoliang Dai et al. "Fbnetv3: Joint architecture-recipe search using predictor pretraining". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 16276–16285.

[3] 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.

[4] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Efficient multi-objective neural architecture search via lamarckian evolution". In: *arXiv preprint arXiv:1804.09081* (2018).

[5] Fast.ai. *Imagewoof*. https://github.com/fastai/imagenette. [Online; 2023-07-20]. 2019.

[6] M.A. Ganaie et al. "Ensemble Deep Learning: A Review". In: *Eng. Appl. Artif. Intell.* 115.C (Oct. 2022). ISSN: 0952-1976. DOI: 10.1016/j.engappai.2022.105151. URL: https://doi.org/10.1016/j.engappai.2022.105151.

[7] Roman Garnett. *Bayesian Optimization*. to appear. Cambridge University Press, 2023.

[8] Alexander Hagg et al. "Prediction of Neural Network Performance by Phenotypic Modeling". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1576–1582. ISBN: 9781450367486. DOI: 10.1145/3319619.3326815. URL: https://doi.org/10.1145/3319619.3326815.

[9] Chi-Hung Hsu et al. "Monas: Multi-objective neural architecture search using reinforcement learning". In: *arXiv preprint arXiv:1806.10332* (2018).

[10] Haifeng Jin, Qingquan Song, and Xia Hu. "Auto-keras: An efficient neural architecture search system". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 1946–1956.

[11] Kirthevasan Kandasamy et al. "Neural architecture search with bayesian optimisation and optimal transport". In: *Advances in neural information processing systems* 31 (2018).

[12] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6405–6416. ISBN: 9781510860964.

[13] Zhichao Lu et al. "Nsga-net: neural architecture search using multi-objective genetic algorithm". In: *Proceedings of the genetic and evolutionary computation conference*. 2019, pp. 419–427.

[14] Zhichao Lu et al. "Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer. 2020, pp. 35–51.

[15] Jovita Lukasik, Steffen Jung, and Margret Keuper. "Learning Where To Look–Generative NAS is Surprisingly Efficient". In: *ECCV 2022*. 2022.

[16] Lizheng Ma, Jiaxu Cui, and Bo Yang. "Deep neural architecture search with deep graph bayesian optimization". In: *IEEE/WIC/ACM International Conference on Web Intelligence*. 2019, pp. 500–507.

[17] Binxin Ru et al. "Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=j9Rv7qdXjd.

[18] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[19] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

[20] Secondmind-Labs. *Bayesian Optimization with Deep Ensembles*. Accessed: August 2023. 2020. URL: %5Curl%7Bhttps://secondmind-labs.github.io/trieste/1.0.0/notebooks/deep_ensembles.html%7D.

[21] Han Shi et al. "Bridging the gap between sample-based and one-shot neural architecture search with bonas". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1808–1819.

[22] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[23] Zi Wang et al. "Pre-trained Gaussian processes for Bayesian optimization". In: *arXiv preprint arXiv:2109.08215* (2021).

[24] Colin White, Willie Neiswanger, and Yash Savani. "Bananas: Bayesian optimization with neural architectures for neural architecture search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10293–10301.

[25] Colin White, Sam Nolen, and Yash Savani. "Exploring the Loss Landscape in Neural Architecture Search". In: *Uncertainty in Artificial Intelligence*. PMLR. 2021.

[26] Colin White, Sam Nolen, and Yash Savani. "Exploring the loss landscape in neural architecture search". In: *Uncertainty in Artificial Intelligence*. PMLR. 2021, pp. 654–664.

[27] Colin White et al. "A Study on Encodings for Neural Architecture Search". In: *Advances in Neural Information Processing Systems*. 2020.

[28] Shen Yan et al. "Does unsupervised architecture representation learning help neural architecture search?" In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12486–12498.

[29] Chenglin Yang et al. “Moat: Alternating mobile convolution and attention brings strong vision models”. In: *arXiv preprint arXiv:2210.01820* (2022).

[30] Barret Zoph and Quoc Le. “Neural Architecture Search with Reinforcement Learning”. In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=r1Ue8Hcxg.