Neural Architecture Tuning: A BO-Powered NAS Tool

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

Neural Architecture Tuning: A BO-Powered NAS Tool

H. Ouertatani[1], C. Maxim[2], S. Niar[3], and E-G. Talbi[4]

[1] IRT SystemX & INRIA Lille - France
houssem.ouertatani@irt-systemx.fr
[2] IRT SystemX - France
cristian.maxim@irt-systemx.fr
[3] UPHF & LAMIH UMR CNRS, Valenciennes - France
smail.niar@uphf.fr
[4] University of Lille & INRIA - France
el-ghazali.talbi@univ-lille.fr

Abstract. Neural Architecture Search (NAS) consists of applying an optimization technique to find the best performing architecture(s) in a defined search space, with regard to an objective function. The practical implementation of NAS currently carries certain limitations, including prohibitive costs with the need for a large number of evaluations, an inflexibility in defining the search space by often having to select from a limited set of possible design components, and a difficulty of integrating existing architecture code by requiring a specialized design language for search space specification. We propose a simplified search tool, with efficiency in the number of evaluations needed to achieve good results, and flexibility by design, allowing for an easy and open definition of the search space and objective function. Interoperability with existing code or newly released architectures from the literature allows the user to quickly and easily tune architectures to produce well-performing solutions tailor-made for particular use cases. We practically apply this tool to certain vision search spaces, and showcase its effectiveness.

Keywords: Neural Architecture Search · Bayesian Optimization · Custom Search Space · Multi-fidelity search

1 Introduction

1.1 Neural Architecture Search (NAS)

Neural networks have in recent years represented a tidal wave of impressive advances in a large spectrum of applications, including but not limited to images, video, audio, 3D data, language, graphs, time series, etc.. A long list of innovations in architectures have enabled this evolution, with a succession of incremental improvements as well as breakthroughs resulting in widely-used reference architectures trained and tested on large benchmarks. These architectures are then applied to use cases to solve real-world problems.

Neural Architecture Search (NAS) aims to automate the design of neural architectures. It requires the definition of a search space which delimits the considered architectures, and the specification of an objective function by which to measure the quality of the architectures. A search strategy is then applied to find the best-performing architectures in this search space. Reinforcement learning was one of the earliest search strategies used in this context [26, 10], and powers certain leading cloud NAS platforms [1]. Local search [25], evolutionary algorithms [6, 14, 13] and bayesian optimization [24] have been proposed as suitable black-box search algorithms. Differentiable NAS [12] is also a family of approaches to efficiently perform the search.

1.2 Architecture adaptation and customisation

Neural Architecture Search (NAS) has been a driver for many of the aforementioned architectural innovations, helping to find many state-of-the-art models, especially in computer vision, e.g. NASNet [27] and EfficientNet [22]. This demonstrates that NAS has been a useful approach to the discovery and design of new reference architectures. However, the potential usefulness of NAS goes beyond this aspect. In fact, efficient and easy NAS can pave the way to its usage by a more general audience than exclusively researchers devising newer architectures.

New architectural innovations are published very frequently in the literature, often tested on large reference benchmarks. If search space definition is too constrained, search spaces can be left behind quickly and superseded by newer, better-performing components.

In addition, directly applying reference architectures from the literature is often suboptimal for particular use cases. An example of this is the usual practice of providing a set number of variants of various sizes (number of parameters). These variants often differ not only in depth (number of blocks or layers), or width (number of channels), but also certain architectural choices at times. For a particular use case, we might be seeking different compromises than the ones struck by the discrete variants, for instance accepting a slightly bigger model than one particular variant for more performance, or vice versa. Use case circumstances sometimes impose hard limits on memory consumption or latency.

Automatically tuning architectures is a useful idea in this context. Using architectures in the literature as reference points, general deep learning practitioners could automatically find the best models to suit their specific application and constraints. As a result, specificities about data, available resources, and specific quality metrics can be addressed in a more principled way by tuning available architectures to these characteristics.

1.3 NAS frameworks

General-purpose black-box optimization tools, such as packages for evolutionary algorithms (EAs) or Bayesian Optimization (BO), often require specialized knowledge. For instance, the user would need to define mutation and crossover

operators suitable for how their search space is structured for EAs, or to select a kernel and a distance function for Gaussian Process-based BO.

Many NAS frameworks have been proposed in the literature, with significant differences in their objectives as well as the methods used to perform the search. Some are designed to advance NAS research by streamlining the comparison of different approaches and facilitating innovations in NAS methods, such as [19, 17]. Certain frameworks focus on specific search spaces, e.g. Convolutional Capsule Networks [15], focus on certain optimization objectives [16]. Many of the frameworks in the literature describe specific search spaces and are difficult to re-apply to new search spaces, complicating the incorporation of the latest architecture innovations. More general-purpose NAS frameworks, such as Vertex AI [1] or NNI [18], use a specialized design language or API for search space specification, and AutoKeras [11] uses similar syntax to the Keras functional API. It is not a straightforward process to build search spaces from publicly available architecture code, often written in one particular deep learning framework, without having to convert it to the specialized design language, the API, or the different deep learning framework.

1.4 Proposed contribution

We instead propose a NAS framework built with the following design objectives in mind:

- Fast search: the BO-based method, especially when coupled with certain prebuilt improvements like pretraining and multi-fidelity search, is very fast on NAS benchmarks and custom-built search spaces and finds better architectures in the search space in fewer than 100 total evaluations.
- Few assumptions: the search space and objective function can be defined in a non-constrained way. The user can define any search space and any objective function.
- Simple incorporation of existing code: it is easy to include newer architectural advancements in the literature, custom-made architectural elements, specific and custom training and testing routines, etc...
- Sharing search spaces which can be directly applied to new objective functions (e.g. on a different dataset, using a different training recipe, or measuring a different quality metric).

This is achieved by only requiring the user to provide the list of encodings their search space is made of, and a function to measure the quality of a solution represented by an encoding. Two optional additions can be made to significantly speed up the search:

- Providing an additional, lower-fidelity but faster objective function. For instance, if the full objective function is the test score after the model was trained for 300 epochs, the low-fidelity objective function could be the test score after only 15 epochs of training.

– Pretraining data: zero-cost metrics like number of parameters, FLOPs, latency or memory footprint, for a subset of the search space networks. If the user is optimising the architecture of PyTorch models, they can instead provide a function which takes an encoding and returns a PyTorch `nn.Module` object, and a sample input for it (e.g. a random image-shaped tensor for a vision model). The framework automatically calculates the pretraining data.

This flexibility is intended to facilitate usage when an architecture for a problem has already been implemented and some architectural parameters (e.g. width, depth, type of block or layer, or other more specific aspects) are to be tuned. As a result, this tool can be useful to further improve an existing architecture, or to adapt a reference architecture from the literature and tailor it to a custom use case (e.g. adjusting width and depth or replacing certain blocks). This also doesn't limit the search space definition to a set of existing components.

2 Search method

A detailed description of the search strategy is outside the scope of the paper, but we provide a general overview of its most important aspects in the following as well as its performance on a NAS benchmark.

In Neural Architecture Search, evaluating solutions can be expensive. In order to reduce the total number of evaluations performed, we opted for a Bayesian Optimization (BO)-based search method, as it offers good sample efficiency.

Bayesian optimization (BO) uses a probabilistic model to approximate the expensive evaluation function. More specifically, an acquisition function uses this model to decide which points are to be evaluated next. In turn, the new evaluations are used to improve the model in preparation for the next iterations.

Gaussian Processes (GPs) are usually the model of choice for BO [8], offering mathematical convenience and good predictive performance, but limited scalability because of its cubic complexity, as well as requiring certain design decisions to be specified (e.g. kernel, distance function), complicating its applicability on complex structures such as neural network architecture search spaces.

Instead, we use a deep ensemble: a set of neural networks with different random initialisations, which end up occupying different low-loss regions and providing a robust combined prediction. This allows a greater flexibility and ease-of-use, and produces a generic NAS method which has been tested successfully on very different search spaces, like cell-based vision search spaces, and graph-based search spaces such as NAS benchmarks.

One main motivation for this choice come from the desire to exploit different data sources to accelerate the search. Using shared weights for multiple prediction targets, we can improve the internal representation used for the predictions. More precisely, we use a number of zero-cost metrics, as well as combining low and high quality evaluations of the objective function. This additional data improves the shared internal representation, improving the model's predictive performance in a cost-effective manner and by extension accelerate the search.

Simultaneous pretraining

We use a set of zero-cost pretraining metrics (e.g. number of trainable parameters, FLOPs, average latency, memory footprint) to jump-start the model with good internal representations of the search space.

To preserve a generic representation, we pretrain on all metrics simultaneously. This is achieved by using a single representation for all the metrics, which is used by different prediction heads for each metric. As a result, this internal representation is affected by all the predictions at once at the backpropagation step.

We performed tests on the NAS-Bench-201 [5] benchmark, with the following observations:

- The pretraining significantly accelerates the search time.
- Simultaneous pretraining outperforms a simpler pretraining scheme where the ensemble is pretrained sequentially, ie. one metric at a time.

Multi-fidelity training

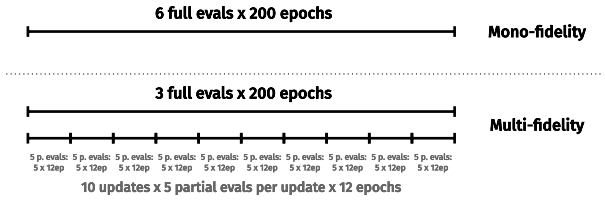

Fig. 1: Mono-fidelity and multi-fidelity training at equivalent costs

The idea for this multi-fidelity search implementation is to trade a few full-quality evaluations for many more low-quality evaluations (Fig 1). Compared to what could be possible with a GP, we can use one shared deep ensemble with only a few specialized weights for each specific fidelity level. As a result, the shared representation is affected both by the low-quality and the high-quality evaluations, the latter being given more importance (i.e. the ensemble is trained for more epochs on the high-quality data).

Search method overview

The combination of these elements in the search method contributes to an accelerated search, which reaches the optimum on NAS-Bench-201 in fewer evaluations than competing methods: under 80 evaluations for CIFAR10, and under 60 evaluations for CIFAR100.

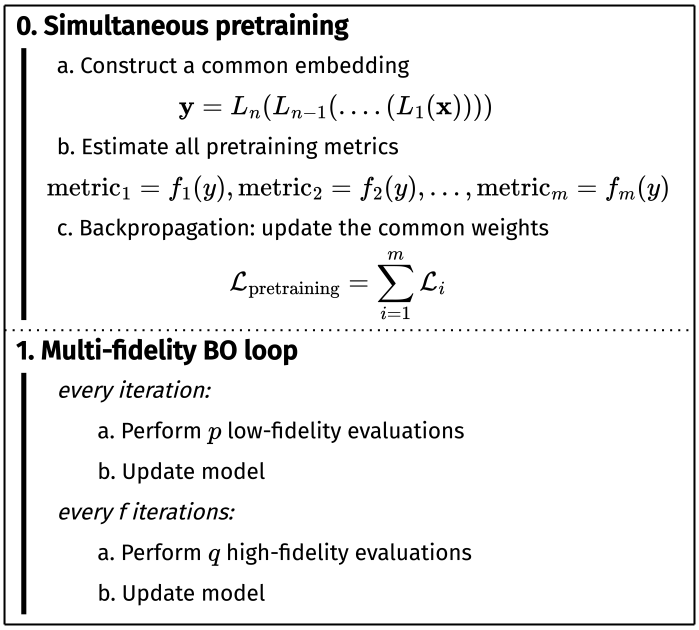

Fig. 2: Overview of the search procedure

3 NAS framework usage

Search space definition

A `SearchSpace` object is created for a new search space. To define it, the user only needs to provide the encodings, and optionally a function to convert from an encoding to a PyTorch `nn.Module`.

```
ss = create_search_space(name='search-space',
                         save_filename='ss.dill',
                         encodings=encodings,
                         encoding_to_net=encoding_to_net)
```

The `preprocess` function automatically generates the pretraining data, using the `PyTorch Profiler` tools, then creates and pretrains the deep ensemble associated with this search space. It requires a sample input suitable for the networks in the search space, e.g. here we generate a random image-shaped tensor for a vision search space, for the profiling step. The user can instead provide custom pretraining data if for instance a different deep learning framework is in use.

```
ss.preprocess(sample_input=torch.rand(16, 3, 224, 224),
              threads=16)
```

Reusability The `SearchSpace` object is saved to a file which can be shared, providing a pretrained deep ensemble ready to be used for the search step imme-

diately. It enables the launch of a new search on a user-defined objective function, e.g. on a new dataset, or using new data augmentation or training techniques.

Launching the search To initiate a new search, the user defines the objective function (the high-fidelity evaluation), as well as an optional but recommended low-fidelity function. This is defined in a `SearchInstance` object, which encapsulates the current search progress, logs, and data. It is saved in a file, which can be loaded using the `dill` package to resume a previous search.

```
s = SearchInstance(name='search-inst',
                   save_filename='search.dill',
                   search_space_filename='ss.dill',
                   hi_fi_eval = hi_fi_eval,
                   hi_fi_cost = 240,
                   lo_fi_eval= lo_fi_eval,
                   lo_fi_cost = 12)
```

To run the search for an evaluation budget n:

```
s.run_search(eval_budget=n)
```

For image classification, a helper function provides code to train in distributed mode (using `torchrun`) for a user-specified number of epochs and then test the networks. Any dataset from the `datasets` package can be specified.

```
evaluator = create_img_class_evaluator(dataset,
                                       n_classes=num_classes,
                                       n_gpus=n_gpus,
                                       config_to_model_file=filename,
                                       dataset_config=dataset_config,
                                       eval_split=eval_split,
                                       reparam=True)
```

4 Practical application examples on image classification

4.1 MobileOne-based CNN search space

MobileOne architecture

MobileOne [23] is a family of efficient vision backbones targeted towards mobile devices. They are purely CNN-based, with numerous improvements to traditional efficient CNN designs aimed at minimizing the latency on a mobile device.

One key aspect of the MobileOne architecture is re-parameterization, building on advances in [2–4]. Different architectures are used during training and inference. At train time, parallel branches incorporate convolution and batch normalization operations with diverse kernel sizes. At inference time, these branches are fused into an equivalent block with a much simpler architecture. This design leads to performance improvements across many vision tasks (image classification, object detection, semantic segmentation) without sacrificing low latency.

Table 1: MobileOne-based CNN search space

	Stage 1		Stage 2		Stage 3		Stage 4		Use SE	N. conv. branches
Base width	**64**		**128**		**256**		**512**			
	Depth	**Width multiplier**	**Depth**	**Width multiplier**	**Depth**	**Width multiplier**	**Depth**	**Width multiplier**		
MobileOne-s0	2	0.75	8	1.0	10	1.0	1	2.0	No	4
Value ranges	{0,1,2, 5,8,10}	[0.25..4] in steps of 0.25	{0,1,2, 5,8,10}	[0.25..4] in steps of 0.25 ($\geq$ Stage 1)	{0,1,2, 5,8,10}	[0.25..4] in steps of 0.25 ($\geq$ Stage 2)	{0,1,2, 5,8,10}	[0.25..4] in steps of 0.25 ($\geq$ Stage 3)	{Yes, No}	{1, 4}

Search space description

We use the MobileOne architecture as a base to construct a simple search space around it. We specifically start with the `MobileOne-s0` variant, and vary the depth and width in each of its 4 stages (Table 1)

We aim to search for a better configuration of the `MobileOne-s0` architecture on a different dataset, with a similar number of total parameters. `M1-s0` has 21 as the total depth of the 4 stages, and 1.06M parameters at inference time.

Therefore, we filter the resulting search space as follows:

$$19 \leq \text{Total depth} \leq 22$$
$$0.86\text{M} \leq \text{n. params} \leq 1.26\text{M}$$

This yields a total of 55870 architectures. We perform the search using as objective function the validation accuracy on the Imagenette [9] image classification dataset, a 10-class subset of ImageNet [20]. The high quality evaluation has 240 epochs during training, while the low quality evaluation has 12 epochs.

Search results

We run the search for the equivalent of 60 evaluations. We compare the resulting architecture and the baseline M1-s0 architecture's results in table 2, where we report the value of the high quality evaluation (equivalent to the objective function during the search) i.e. 240 epochs, as well as the evaluation at 300 epochs.

Table 2: Search results for the M1-based search space on Imagenette

	N. params (train-time)	**N.params (test-time)**	**240 epochs**	**300 epochs**
MobileOne-s0	4.28M	1.06M	89.34	90.09
Search result	0.95M	0.93M	**90.37**	**90.77**

The search process successfully found a smaller and better-performing depth and width configuration for the M1-s0 model, as it applies to this dataset.

4.2 Hybrid CNN-ViT search space

SwiftFormer architecture

SwiftFormer [21] introduces a novel attention mechanism designed with low latency in mind: efficient additive attention. Instead of an expensive operation with quadratic complexity w.r.t the input resolution, self-attention is implemented as a fast linear operation.

In the SwiftFormer family of models, the architecture is a succession of stages separated by downsampling steps, each stage being a succession of convolution-based blocks followed by one attention-based block. This is an original setup only possible because of the efficiency of the self-attention mechanism: where efficient ViT-CNN models generally reserve attention modules to the latter stages where the resolution is lowest, SwiftFormer can apply self-attention anywhere in the network, with minimal impact to efficiency and latency.

Search space description

Starting from the SwiftFormer-XS variant, we built a search space incorporating and generalising both MobileOne-s0 and SwiftFormer-XS architectures. For each stage, 2 variables have to be decided: the type of convolution block to use, and the number of attention blocks at the end of the stage. In the attention block(s), the attention mechanism is preceded by a mini-convolution block. This is of the same type as the other convolution blocks in the stage.

Table 3: MobileOne-SwiftFormer hybrid search space

	Stage 1		**Stage 2**		**Stage 3**		**Stage 4**	
	Conv. type	**Attn. blocks**	**Conv. type**	**Attn. blocks**	**Conv. type**	**Attn. blocks**	**Conv. type**	**Attn. blocks**
SwiftFormer	`conv-enc`	`last-1`	`conv-enc`	`last-1`	`conv-enc`	`last-1`	`conv-enc`	`last-1`
MobileOne (s0-s3)	`mo`	`none`	`mo`	`none`	`mo`	`none`	`mo`	`none`
Value ranges	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}	{`conv-enc`, `mo`, `mo-se`}	{`none`, `last-1`, `last-2`, `all`}

- `conv-enc`: SwiftFormer's convolution-based block design
- `mo`: MobileOne block
- `mo-se`: MobileOne block with SE (Squeeze-and-Excitation)

For each stage, 12 possible combinations exist, and with 4 stages the search space spans $12^4 = 20736$ architectures, including SwiftFormer-XS and MobileOne-s0.

Search results

We perform the search using two different datasets. Along with Imagenette, we also use the Alzheimer MRI disease classification dataset [7], a 4-class dataset which seems to be slightly more challenging for the tested networks. For Imagenette, two search instances were tested, one with the number of branches in the MobileOne blocks set to 4, and a second one with this value set to 1. Tables 4 and 5 contain the search results, and figures 3 and 4 illustrate the evolution of the best found scores as the search progressed.

Table 4: Search results on the Imagenette dataset

Architecture	Accuracy (240 epochs)
MobileOne-s0	89.94
SwiftFormer-XS	91.08
Search result (4 branches)	91.18
Search result (1 branch)	92.05

Table 5: Search results on the Alzheimer-MRI dataset

Architecture	Accuracy (200 epochs)
MobileOne-s0	74.53
SwiftFormer-XS	64.60
Search result	76.95

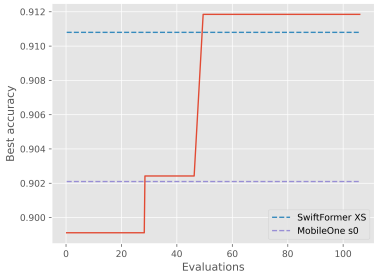

Fig. 3: Best value evolution during search - Imagenette (4 branches)

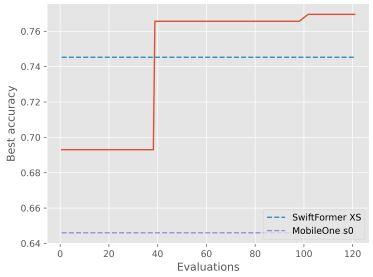

Fig. 4: Best value evolution during search - Alzheimer-MRI dataset

5 Conclusion

In this work, we described a framework to perform NAS quickly and with maximum flexibility for incorporating custom and new architectural components. It mainly relies on Bayesian Optimization with deep ensembles, a pretraining scheme and multiple fidelities to accelerate the search. We argue that this method can be used effectively to improve existing architectures or find new ones, with the simple tuning and adaptation of reference architectures from the literature a target use case. Starting from baseline architectures from the literature, we construct search spaces, and launch search instances to find the top-performing architectures on different datasets.

6 Acknowledgements

This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute within the Confiance.ai project

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

1. Cloud, V.A.G.: Vertex ai neural architecture search, https://cloud.google.com/vertex-ai/docs/training/neural-architecture-search/overview, accessed: February 2024
2. Ding, X., Guo, Y., Ding, G., Han, J.: Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1911–1920 (2019)
3. Ding, X., Zhang, X., Han, J., Ding, G.: Diverse branch block: Building a convolution as an inception-like unit. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10886–10895 (2021)
4. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: Making vgg-style convnets great again. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 13733–13742 (2021)
5. Dong, X., Yang, Y.: Nas-bench-201: Extending the scope of reproducible neural architecture search. In: International Conference on Learning Representations (ICLR) (2020), https://openreview.net/forum?id=HJxyZkBKDr
6. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution. arXiv preprint arXiv:1804.09081 (2018)
7. Falah.G.Salieh: Alzheimer mri dataset (2023), https://huggingface.co/datasets/Falah/Alzheimer_MRI
8. Garnett, R.: Bayesian Optimization. Cambridge University Press (2023), to appear
9. Howard, J.: Imagenette: A smaller subset of 10 easily classified classes from imagenet (March 2019), https://github.com/fastai/imagenette
10. Hsu, C.H., Chang, S.H., Liang, J.H., Chou, H.P., Liu, C.H., Chang, S.C., Pan, J.Y., Chen, Y.T., Wei, W., Juan, D.C.: Monas: Multi-objective neural architecture search using reinforcement learning. arXiv preprint arXiv:1806.10332 (2018)
11. Jin, H., Song, Q., Hu, X.: Auto-keras: An efficient neural architecture search system. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 1946–1956 (2019)
12. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
13. Lu, Z., Deb, K., Goodman, E., Banzhaf, W., Boddeti, V.N.: 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. pp. 35–51. Springer (2020)
14. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the genetic and evolutionary computation conference. pp. 419–427 (2019)

15. Marchisio, A., Massa, A., Mrazek, V., Bussolino, B., Martina, M., Shafique, M.: Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20). p. 9. Institute of Electrical and Electronics Engineers (2020). https://doi.org/10.1145/3400302.3415731, https://arxiv.org/abs/2008.08476
16. Marchisio, A., Mrazek, V., Massa, A., Bussolino, B., Martina, M., Shafique, M.: Rohnas: A neural architecture search framework with conjoint optimization for adversarial robustness and hardware efficiency of convolutional and capsule networks. IEEE Access **10**, 109043–109055 (2022)
17. Mehta, Y., White, C., Zela, A., Krishnakumar, A., Zabergja, G., Moradian, S., Safari, M., Yu, K., Hutter, F.: Nas-bench-suite: Nas evaluation is (now) surprisingly easy. In: International Conference on Learning Representations (2022)
18. Microsoft: Neural Network Intelligence (1 2021), https://github.com/microsoft/nni
19. Ruchte, M., Zela, A., Siems, J., Grabocka, J., Hutter, F.: Naslib: A modular and flexible neural architecture search library. https://github.com/automl/NASLib (2020)
20. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
21. Shaker, A., Maaz, M., Rasheed, H., Khan, S., Yang, M.H., Khan, F.S.: Swiftformer: Efficient additive attention for transformer-based real-time mobile vision applications (2023)
22. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning. pp. 6105–6114. PMLR (2019)
23. Vasu, P.K.A., Gabriel, J., Zhu, J., Tuzel, O., Ranjan, A.: Mobileone: An improved one millisecond mobile backbone. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7907–7917 (2023)
24. White, C., Neiswanger, W., Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 10293–10301 (2021)
25. White, C., Nolen, S., Savani, Y.: Exploring the loss landscape in neural architecture search. In: Uncertainty in Artificial Intelligence. pp. 654–664. PMLR (2021)
26. Zoph, B., Le, Q.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=r1Ue8Hcxg
27. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)