Neural Architecture Search Benchmarks: Past, Present and Future

Frank Hutter

University of Freiburg & Bosch Center for AI
fh@cs.uni-freiburg.de

@FrankRHutter
@AutoML_org

Neural architecture search (NAS) is exploding!

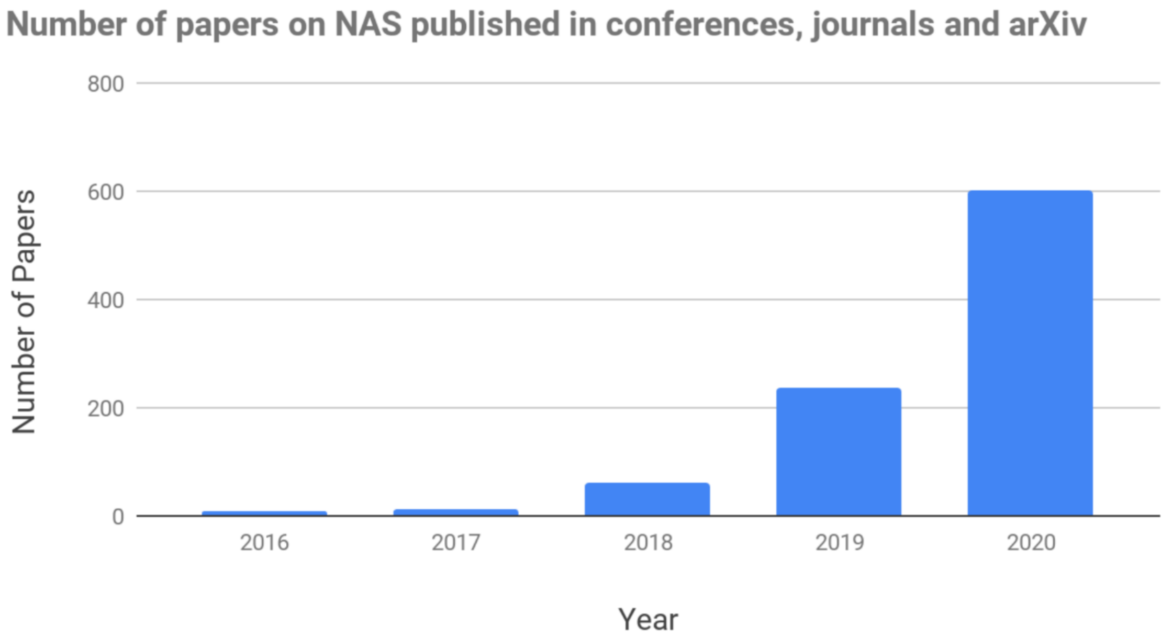
Journal of Machine Learning Research 20 (2019) 1-21 Submitted 9/18; Revised 3/19; Published 3/19

Neural Architecture Search: A Survey

Thomas Elsken THOMAS.ELSKEN@DE.BOSCH.COM
Bosch Center for Artificial Intelligence
71272 Renningen, Germany
and University of Freiburg

Jan Hendrik Metzen JANHENDRIK.METZEN@DE.BOSCH.COM
Bosch Center for Artificial Intelligence
71272 Renningen, Germany

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg
79110 Freiburg, Germany

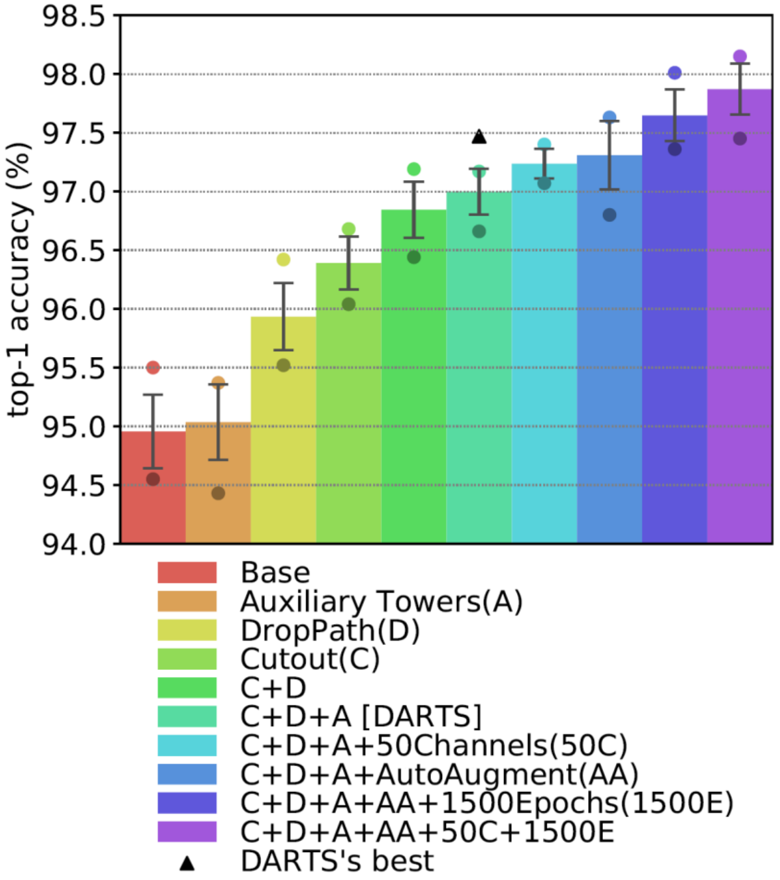


- The Past
  - Scientific best practices in NAS
  - First benchmarks
- The Present
- The Future

NAS Had To Overcome Some Childhood Problems 1/2

- **Poor performance compared to random search** [Li & Talwalkar, 2020, Yu et al, 2020]
- **Poor reproducibility** [Li & Talwalkar, 2020]
  - Even random seeds are very important
- **Training pipeline matters much more than architecture** [Yang et al, 2020]
- **Poor scientific practices** [Lindauer & Hutter, 2020]
  - Inavailability of code
  - Incomparable training code, search spaces, evaluation schemes, etc

Training pipeline matters much more than architecture



Model	Params	Test error (%)
DenseNet-BC (Huang [illegible]l., 2017)	25[illegible]	3.46
PyramidNet (Han et al., [illegible])	[illegible]M	3.31
Shake-Shake + c/o (DeVries [illegible]Taylor, 2017)	[illegible]6.2M	2.56
PyramidNet + SD (Yamada et a[illegible]018)	26.0M	2.31
ENAS + c/o (Pham et al., 2018)	4.6M	2.89
DARTS + c/o (Liu et al., 2018c)	3.4M	2.83
NASNet-A + c/o (Zoph et al., 201[illegible]	27.6M	2.40
PathLevel EAS + c/o (Cai et a[illegible]018b)	14.3M	2.30
AmoebaNet-B + c/o (Real [illegible]., 2018)	[illegible].9M	2.13
Proxyless-R + c/o (ours)	[illegible]	2.30
Proxyless-G + c/o (ou[illegible]	5.7[illegible]	**2.08**

Incomparable

- **Different training code** (often unavailable)
- **Different search spaces**
- **Different evaluation schemes**

[Yang et al, ICLR 2020: “NAS Evaluation is Frustratingly Hard“]

NAS Best Practices Checklist

[Lindauer and Hutter, JMLR 2020]

1. Releasing code

- Not just trained architectures

2. Properly comparing methods

- Proper scientific evaluations, powered by tabular/surrogate benchmarks for statistical significance

3. Reporting important details

- E.g., hyperparameter tuning

Suggestion to reviewers

– Deemphasize final results table on CIFAR-10 (or other datasets),
be aware of many confounding factors

The NAS Best Practices Checklist (version 1.0, September 6, 2019)
by Marius Lindauer and Frank Hutter

Best practices for releasing code

For all experiments you report, check if you released:

- ☐ Code for the training pipeline used to evaluate the final architectures
- ☐ Code for the search space
- ☐ The hyperparameters used for the final evaluation pipeline, as well as random seeds
- ☐ Code for your NAS method
- ☐ Hyperparameters for your NAS method, as well as random seeds

Note that the easiest way to satisfy the first three of these is to use *existing* NAS benchmarks, rather than changing them or introducing new ones.

Best practices for comparing NAS methods

- ☐ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same *dataset* (with the same training-test split), *search space* and *code* for training the architectures and *hyperparameters* for that code?
- ☐ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
- ☐ Did you run ablation studies?
- ☐ Did you use the same evaluation protocol for the methods being compared?
- ☐ Did you compare performance over time?
- ☐ Did you compare to random search?
- ☐ Did you perform multiple runs of your experiments and report seeds?
- ☐ Did you use tabular or surrogate benchmarks for in-depth evaluations?

Best practices for reporting important details

- ☐ Did you report how you tuned hyperparameters, and what time and resources this required?
- ☐ Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
- ☐ Did you report all the details of your experimental setup?

NAS-Bench-101: The First Tabular NAS Benchmark

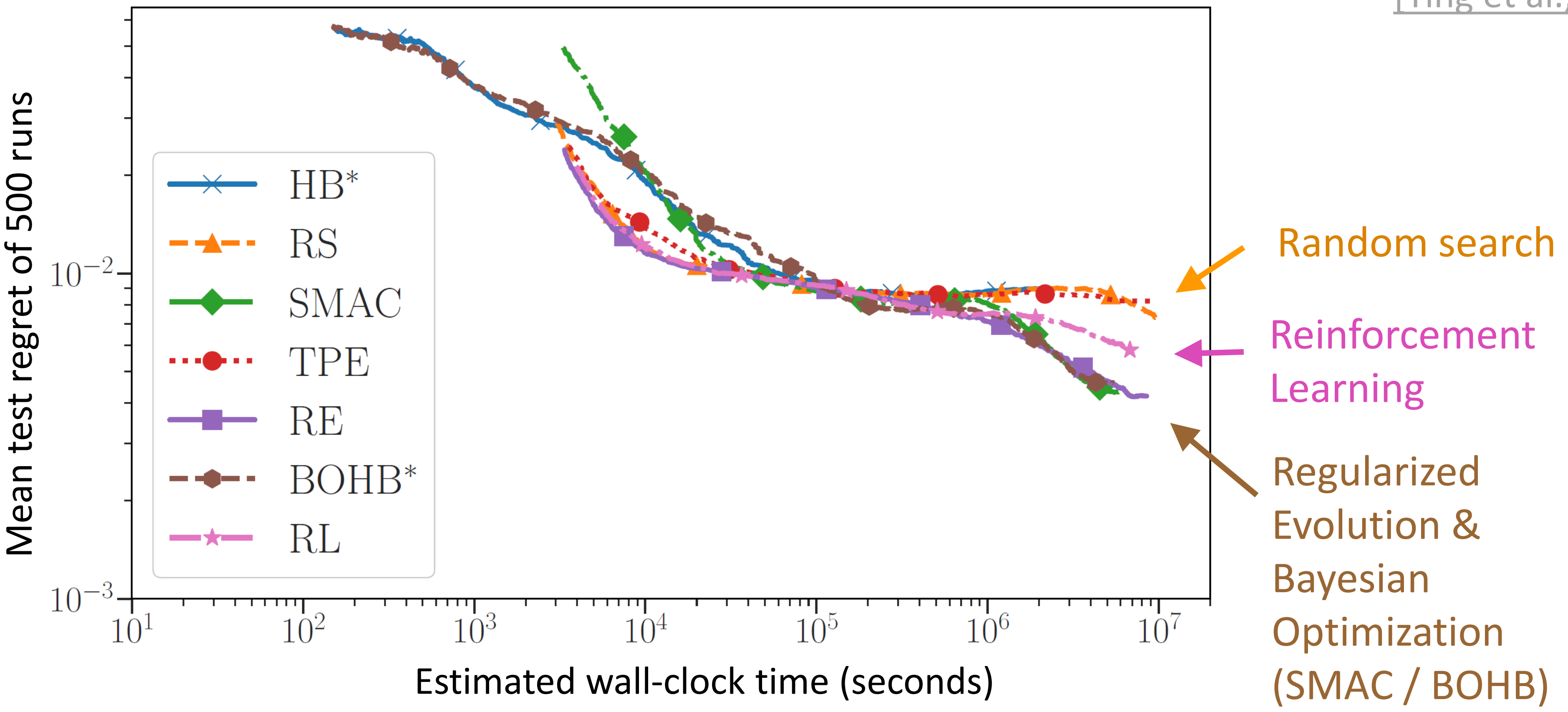
[Ying et al., ICML 2019]

- **Small cell search space that we exhaustively evaluated**
  - Enables evaluating a NAS method in minutes on a laptop
  - Enables proper scientific research:
    multiple runs, robustness studies, etc
  - Fair apples-to-apples evaluations by design
    (fixed final evaluation pipeline)
  - Of course, source code and scripts are available

- **423k architectures evaluated on CIFAR-10**
  - Nobody has to ever run this again
  - Only possible with Google resources (4.000 TPUs for months)
  - One-time cost already far more than amortized

NAS-Bench-101: Comparison of Optimizers

[Ying et al., ICML 2019]



- Note: **SMAC** (published 2011) **outperforms RL** (published 2016)
- Tabular NAS benchmarks finally allow us to do these analyses

- The Past
  - Scientific best practices in NAS
  - First benchmarks
- ➡ The Present
  - Surrogate Benchmarks
- The Future

Tabular NAS Benchmarks Really Caught on ☺

- **NAS-Bench-101** [Ying et al, ICML 2019] & **NAS-Bench-1Shot1** [Zela et al, ICLR 2020]
  - Up to 423k unique architectures
- **NAS-Bench-201** [Dong & Yang, ICLR 2020]
  - 6466 unique architectures
  - Extension: **NATS-Bench** with 32768 unique architectures
- **NAS-Bench-ASR**
  - 8242 unique architectures
- **NAS-Bench-NLP**
  - 14322 architectures evaluated

- But these NAS benchmarks are **too small to be realistic** ☹
  - E.g., local search is state of the art for such small space, but performs poorly on large ones, such as DARTS [White et al, AutoML 2020]
  - More realistically-sized search spaces
    - E.g., DARTS search space has $\approx 10^{18}$ architectures
    - E.g., FBNet search space is $\approx 10^{21}$ architectures

Surrogates: Going Beyond the Limits of Tabular NAS Benchmarks

- **Problem**
  - For any realistically-sized search space, there is no hope to evaluate it exhaustively to compute a table

- **NAS Surrogate Benchmark Methodology:**
  - Evaluate a subset of architectures
  - Fit a model to those; use model predictions in lieu of the real/tabular benchmark

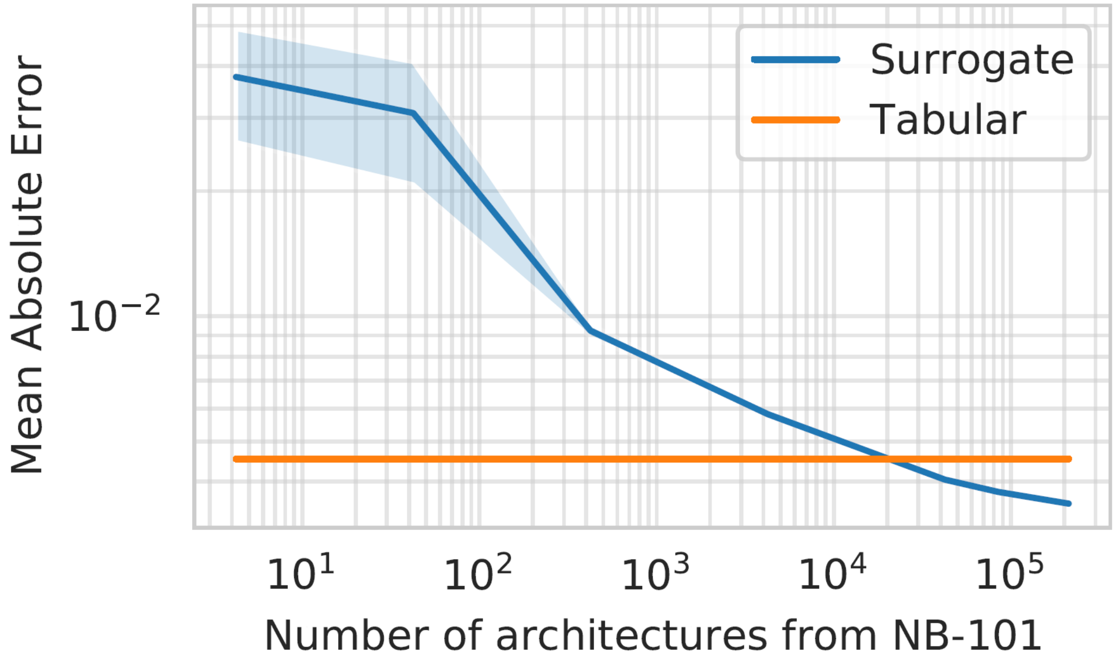
- **Many previous works** already used a surrogate model to predict the performance of untested architectures
  - All works on Bayesian optimization (SMAC, BOHB, NAS-BOWL, BANANAS, ...)
  - All works on „predictor-based NAS“ (NPE-NAS, BRP-NAS, etc)

- **The difference is in how we use the model:**
  not to speed up search, but **to define a benchmark**
  - Search algorithms only have a blackbox interface to the surrogate benchmark, just like for a tabular benchmark
  - Any improvements in surrogate modelling will improve surrogate NAS benchmarks

Surrogate Benchmarks Can Be More Accurate Than Tabular Ones

- The evaluations in a table come with a certain error due to the noise of SGD
  - Many NAS benchmarks quantify this error with 3-5 repetitions for (some subset of) the architectures

- From a machine learning perspective
  - A tabular NAS benchmark predicts a noisy function f(A) by evaluating at A a few times and returning the mean
    - This makes an independence assumption, not using data for similar architectures
  - A good model should do better than that ...
    - And indeed, it does ☺



Surr-NAS-Bench-DARTS (SNB-DARTS, aka NAS-Bench-301)

[Siems et al, arXiv 2021]

- Evaluated **50.000 architectures** in the DARTS search space using different optimizers

- Evaluated **broad range of regression** models to fit this data

- Best regression models
  - **Gradient boosting (XGB/LGB)**
  - **Graph convolutional networks**

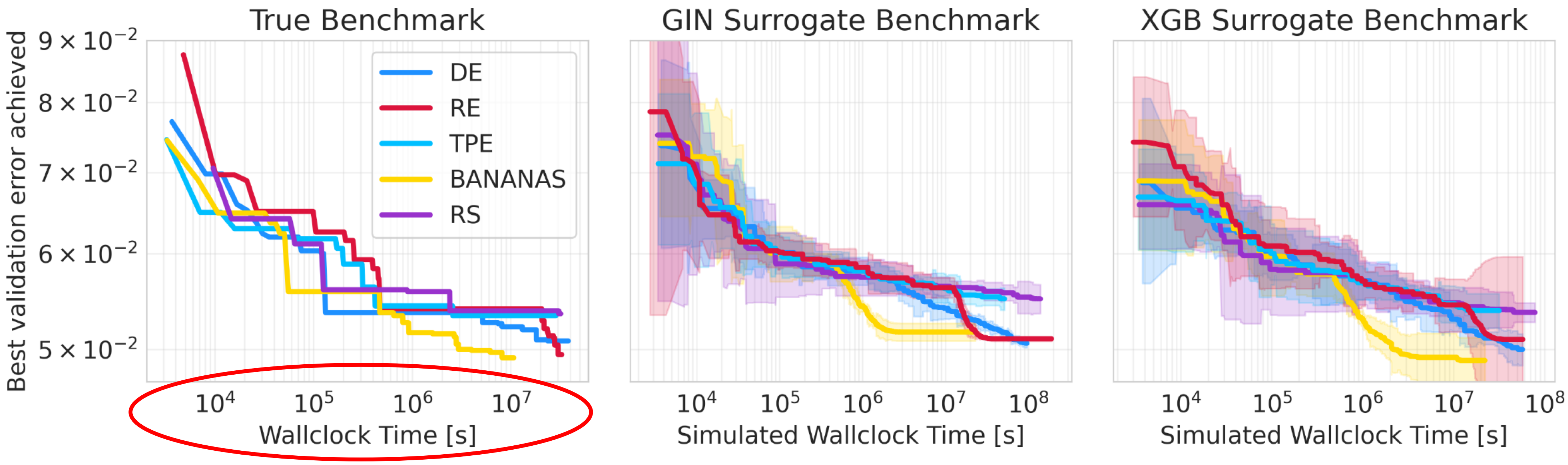
- Estimation errors lower than error due to noise in a single run of SGD

Optimizer			# Evaluations
Discrete		RS	24047
	Evolution	DE	7275
		RE	4639
	BO	TPE	6741
		BANANAS	2243
		COMBO	745
One-Shot		DARTS	2053
		GDAS	234
		RANDOM-WS	198
		PC-DARTS	149

Model	Test	
	R^2	sKT
LGBoost	**0.892**	0.816
XGBoost	0.832	**0.817**
GIN	0.832	0.778
NGBoost	0.810	0.759
μ-SVR	0.709	0.677
MLP (Path enc.)	0.704	0.697
RF	0.679	0.683
ϵ-SVR	0.675	0.660

Benchmarking NAS Methods on SNB-DARTS

[Siems et al, arXiv 2021]

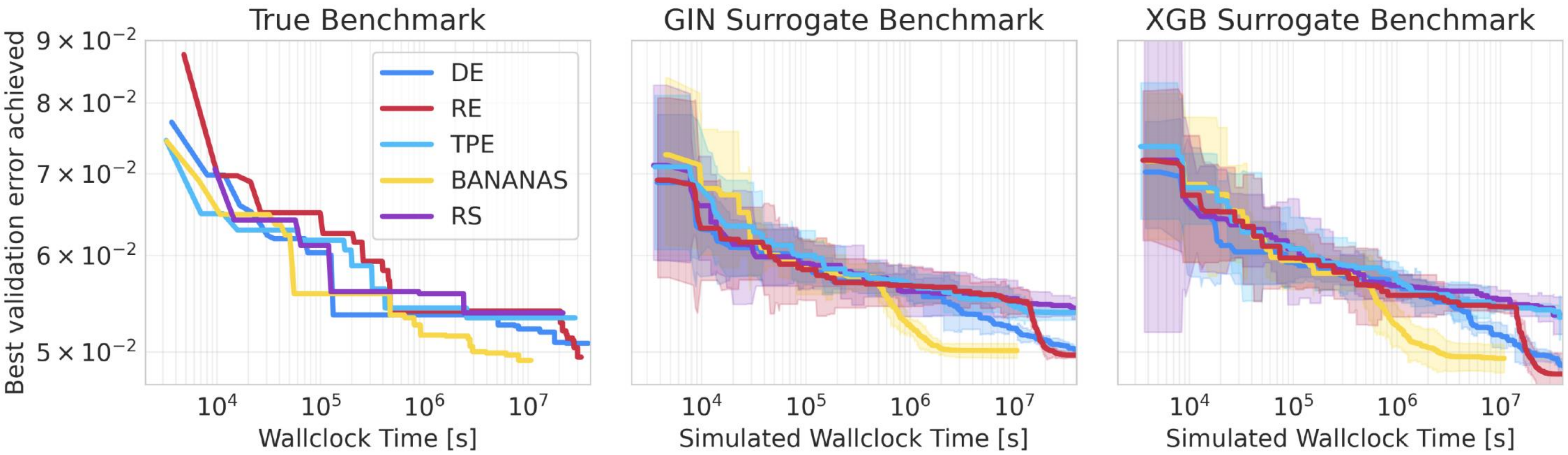


- Actual wallclock time required when run sequentially: **> 1 GPU year, per run**
- Surrogate benchmark: many orders of magnitude faster

- Note: **performance is smoother on the surrogates**,
since we could only afford 1 run on the true benchmark so far

Ablation: Only Using Random Samples to Generate SNB-DARTS

[Siems et al, arXiv 2021]



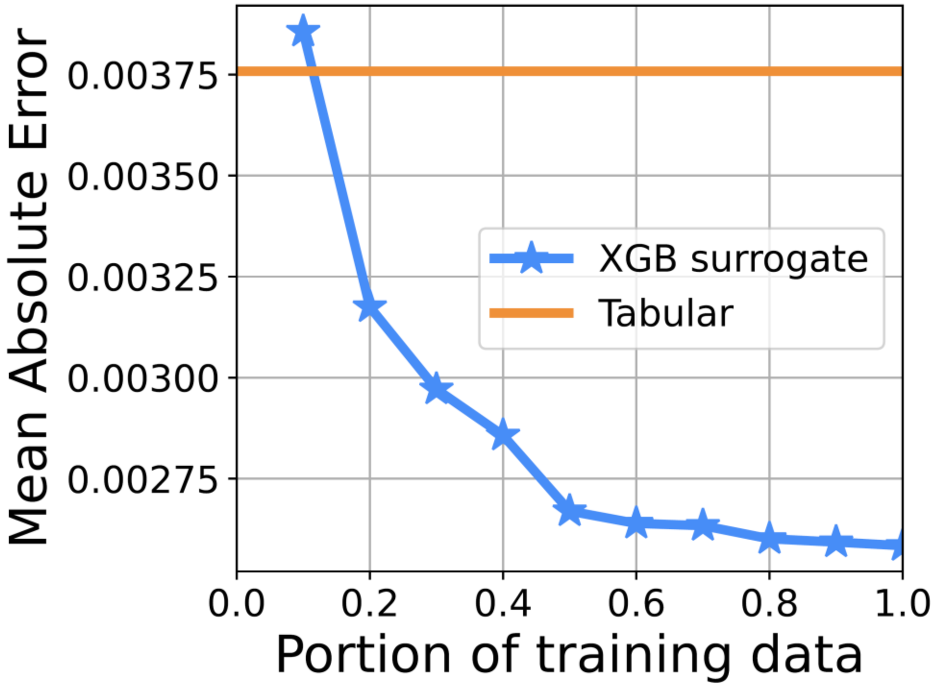
- **Randomly-gathered training data suffices**
  - At least to obtain truthful performance trajectories
  - Predictive performance for top-performing architectures a bit weaker

- Advantage:
  - No bias possible towards the optimizers used to generate the training data

Surr-NAS-Bench-FBNet (SNB-FBNet)

[Siems et al, arXiv 2021]

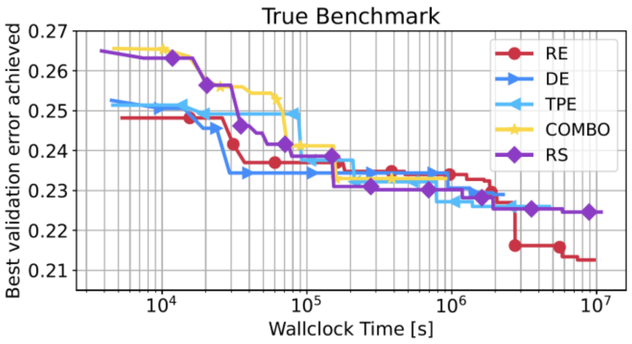
- Evaluated **25.000 random architectures** in the FBNet search space

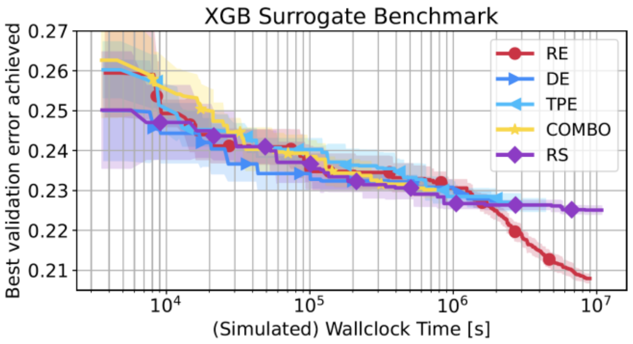


- Surrogate model: XGBoost

- Again, estimation errors lower than error due to noise in a single run of SGD

- Again, truthful trajectory plots





- The Past
  - Scientific best practices in NAS
  - First benchmarks
- The Present
  - Surrogate Benchmarks
  - Many new benchmarks
- The Future

NAS Benchmarks To Pave the Way to the Future of NAS

- **Goal: Discover Entirely New Architectures with NAS**
  - All the important architectures in deep learning were found manually
  - I hope that this will change over the next years

- **We need:**
  - Reliable & Efficient NAS Methods
    - Robust zero-cost proxies
    - Robust one-shot models
    - Efficient blackbox methods
  - Powerful search spaces
    - E.g., hierarchical spaces
  - NAS methods that are compatible with arbitrary search spaces

NAS-Bench-Suite: Many NAS Benchmarks with a Single Interface

- **Problem: Existing NAS Benchmarks are very different**
  - E.g., NAS-Bench-101 has the operations in the nodes while NAS-Bench-201 has them in the edges
  - As a result, NAS Algorithms often hardcoded the search space in their code

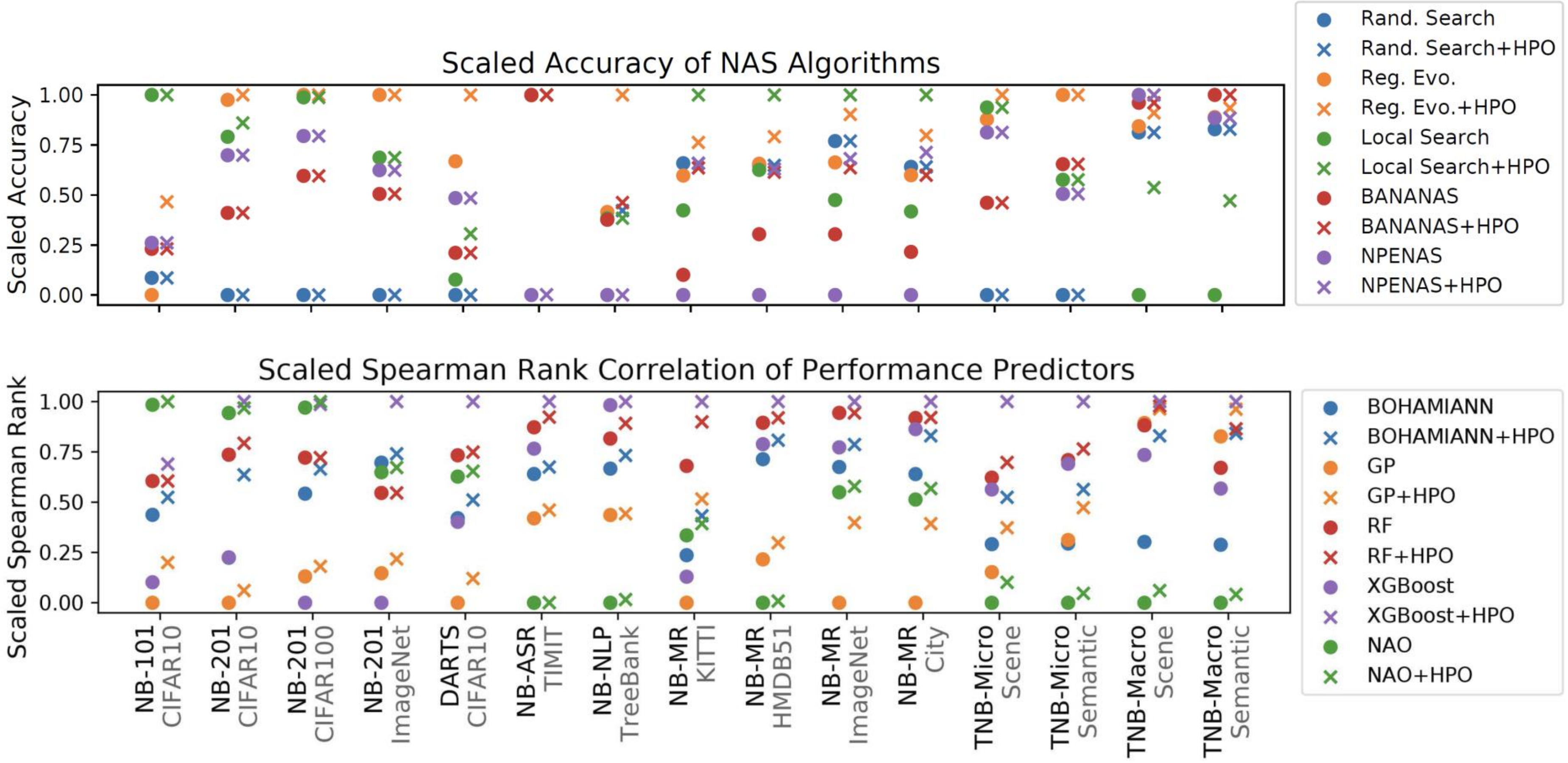
- **NASlib helps to unify the interface to different NAS benchmarks**

- This allows access to 25 (!) different NAS benchmarks
  - For the cost of a single implementation

		Queryable						
Benchmark	**Size**	**Tab.**	**Surr.**	**LCs**	**Macro**	**Type**	**#Tasks**	**NAS-Bench-Suite**
NAS-Bench-101	423k	✓				Image class.	1	✓
NAS-Bench-201	6k	✓		✓		Image class.	3	✓
NAS-Bench-NLP	10^{53}			✓		NLP	1	✓
NAS-Bench-1Shot1	364k	✓				Image class.	1	✓
NAS-Bench-301	10^{18}		✓			Image class.	1	✓
NAS-Bench-ASR	8k	✓			✓	ASR	1	✓
NAS-Bench-MR	10^{23}		✓		✓	Var. CV	4	✓
TransNAS-Bench	7k	✓		✓	✓	Var. CV	14	✓
NAS-Bench-111	423k		✓	✓		Image class.	1	✓
NAS-Bench-311	10^{18}		✓	✓		Image class.	1	✓
NAS-Bench-NLP11	10^{53}		✓	✓		NLP	1	✓

Do we really need so many different NAS benchmarks?

- Sadly, yes
  - Conclusions on „just“ NAS-Bench-101 & 3 NAS-Bench-201 datasets can be misleading

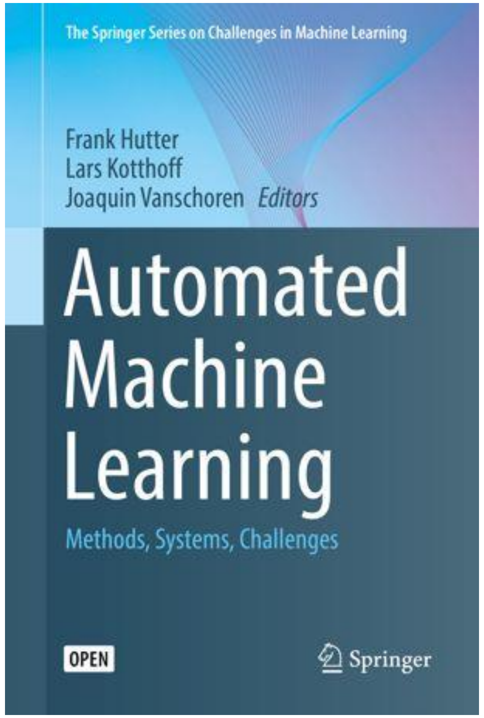


  - Tuning NAS algorithm hyperparameters on one benchmark can lead to poor performance on others

Take-Away: NAS Benchmarks are Coming Of Age

- **There are tons of tabular NAS benchmarks by now**
  - These enable scientific evaluations with minimal compute (i.e., carbon emissions)
- **Surrogates are the path to realistic search spaces**
  - They can even model performance more truthfully than tabular benchmarks
- **NAS-Bench-Suite has 25 queryable NAS benchmarks**
  - Available through a unified interface in NASLib (https://github.com/automl/NASLib)

- More information: http://automl.org
- Book on AutoML: http://automl.org/book



Thank you for your attention!

Funding sources

BOSCH

European Research Council

Emmy Noether-Programm
DFG Deutsche Forschungsgemeinschaft

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

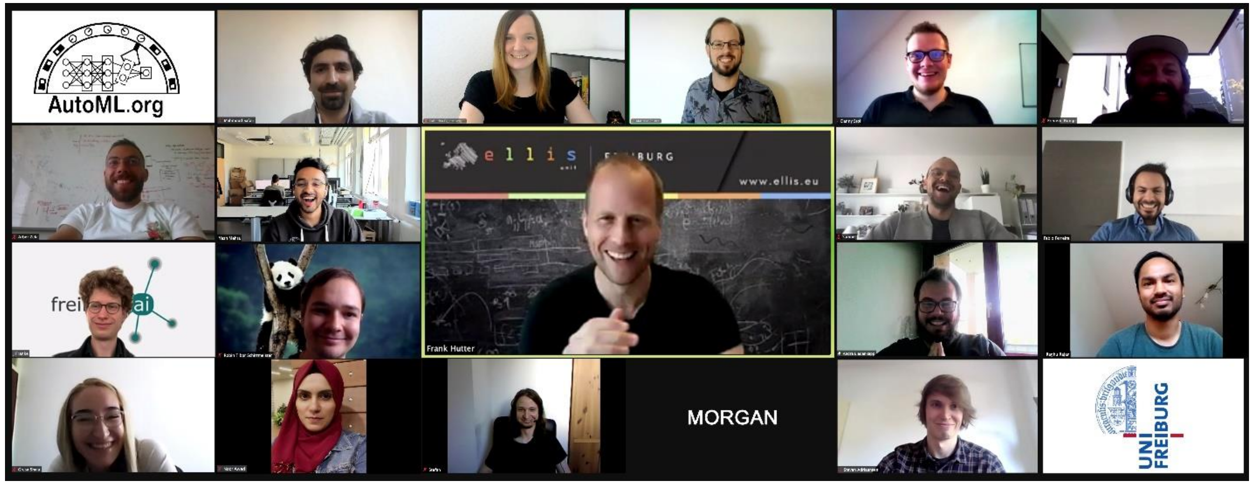
DFG
Priority Programme 1527

DFG Deutsche Forschungsgemeinschaft

BrainLinks BrainTools
acting.thoughts

European Commission

My fantastic team



I‘m looking for
additional great postdocs!

@FrankRHutter
@automl_org

AutoML.org