Neural Architecture Search: The Next Frontier

Colin White, Abacus.ai

Neural Architecture Search: The Next Frontier

colin@abacus.ai

Slides (with hyperlinks): https://crwhite.ml/

Machine learning automation

Archai: Platform for Neural Architecture Search

Neural Network Intelligence

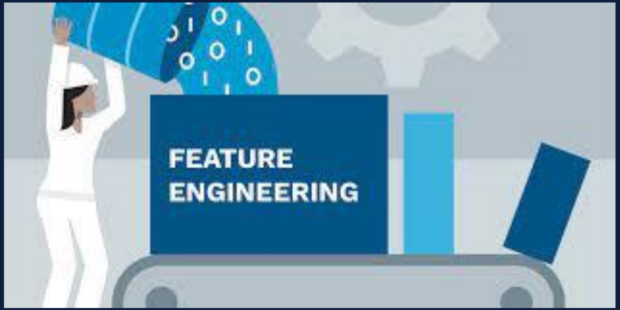
ABACUS.AI

1950s

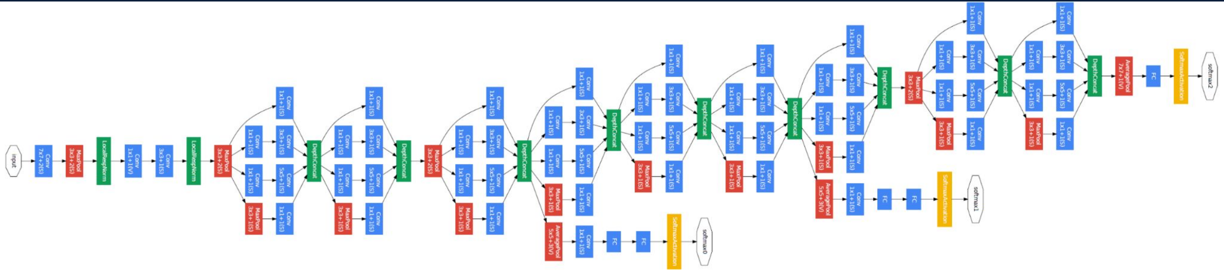
2013

2017

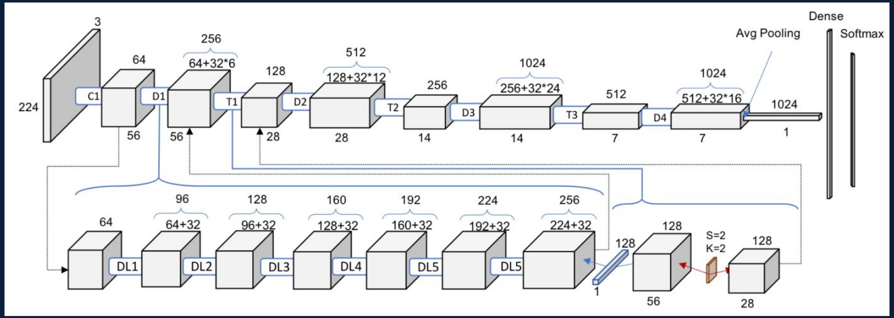
2022



Neural architecture search



GoogLeNet (2014)



DenseNet (2016)

Architectures are getting increasingly more specialized and complex

Machine learning automation

Archai

Archai: Platform for Neural Architecture Search

Neural Network Intelligence

ABACUS.AI

1950s

2013

2017

2022

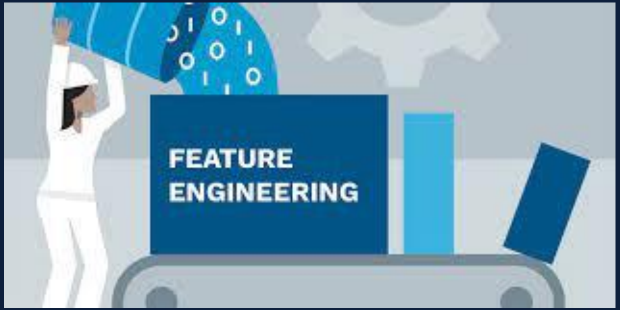
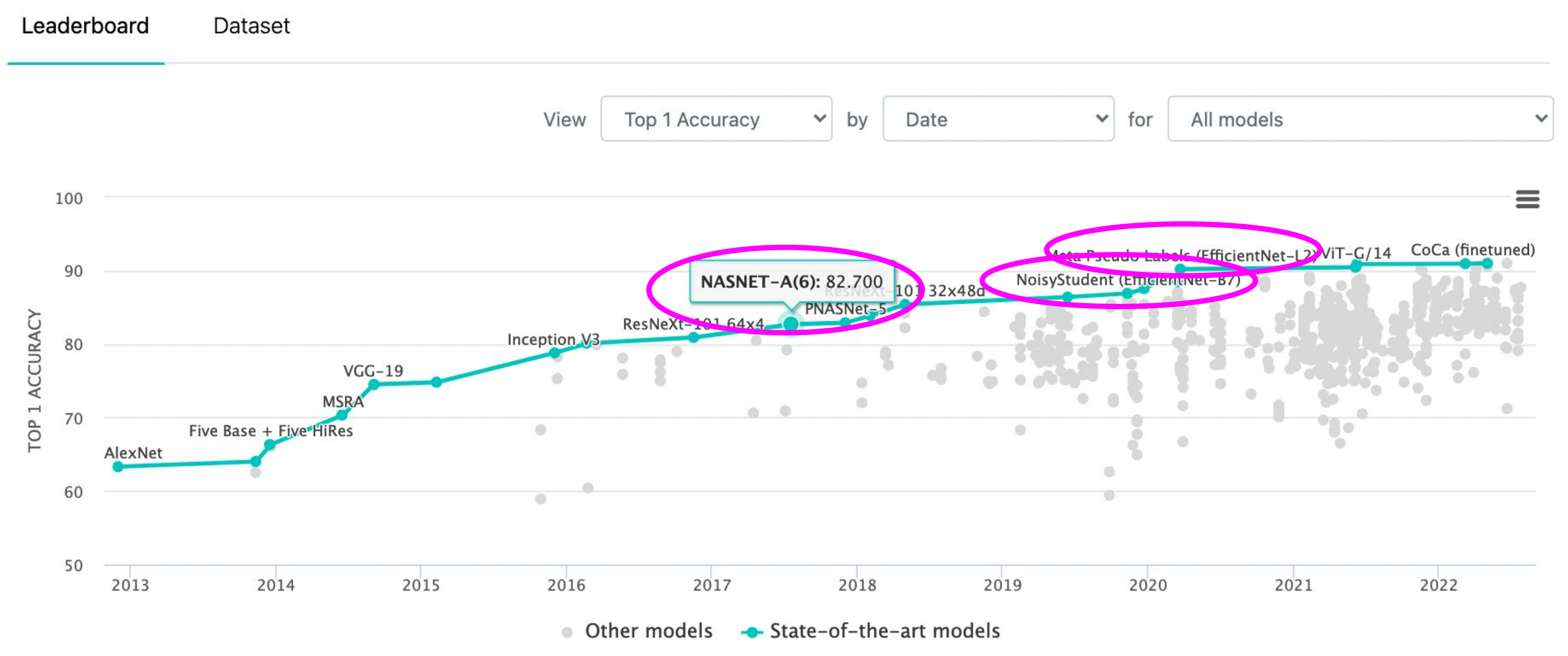


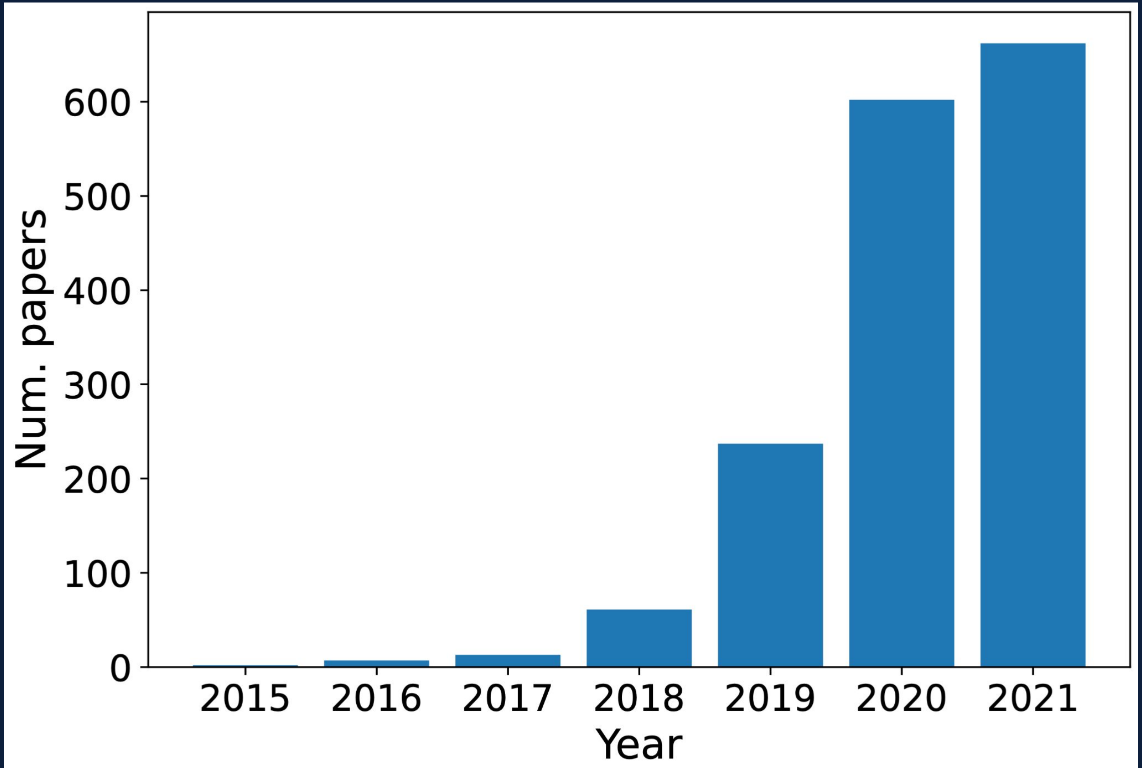
Image Classification on ImageNet



https://paperswithcode.com/sota/image-classification-on-imagenet

Neural architecture search

NAS: the process of **automating** the design of **neural architectures** for a given dataset.



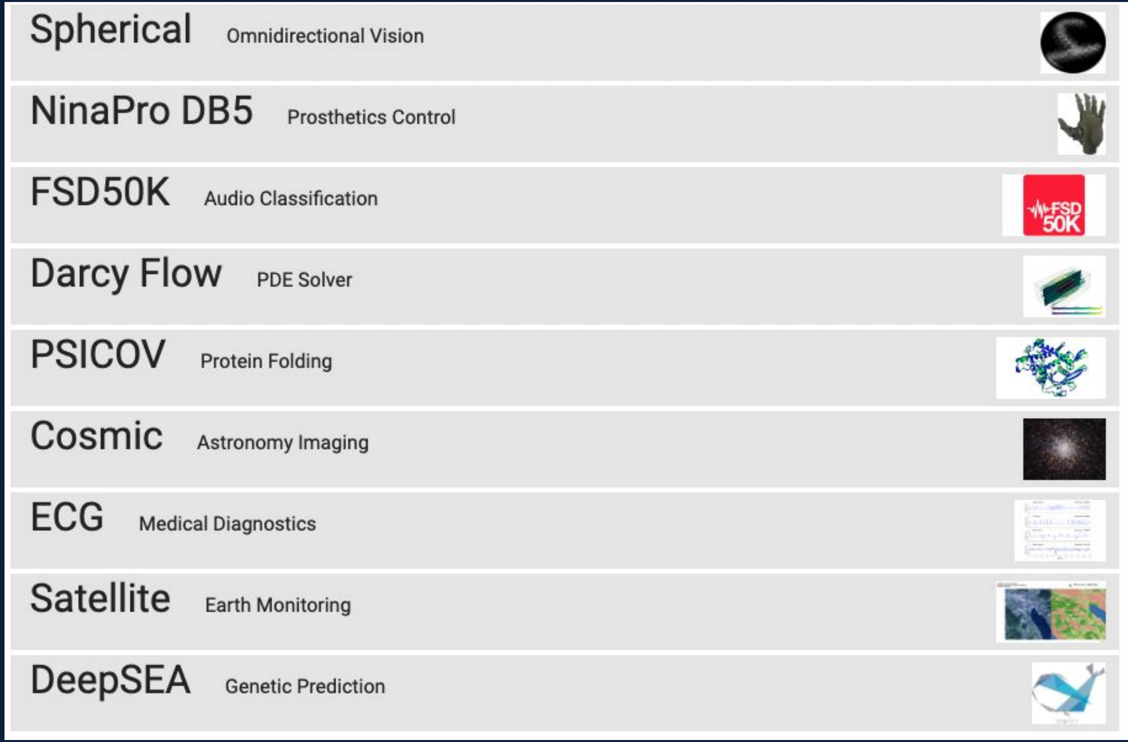
NAS: Basic Definition

- Define a search space ***A,***

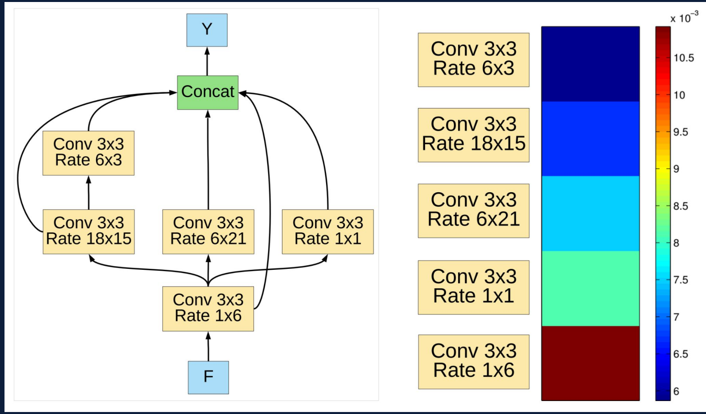
$$\min_{a \in \mathcal{A}} \quad \mathcal{L}_{\text{val}}\left(w^*(a), a\right)$$

$$\text{s.t.} \quad w^*(a) = \operatorname{argmin}_w \ \mathcal{L}_{\text{train}}\left(w, a\right)$$

NAS on new datasets / tasks



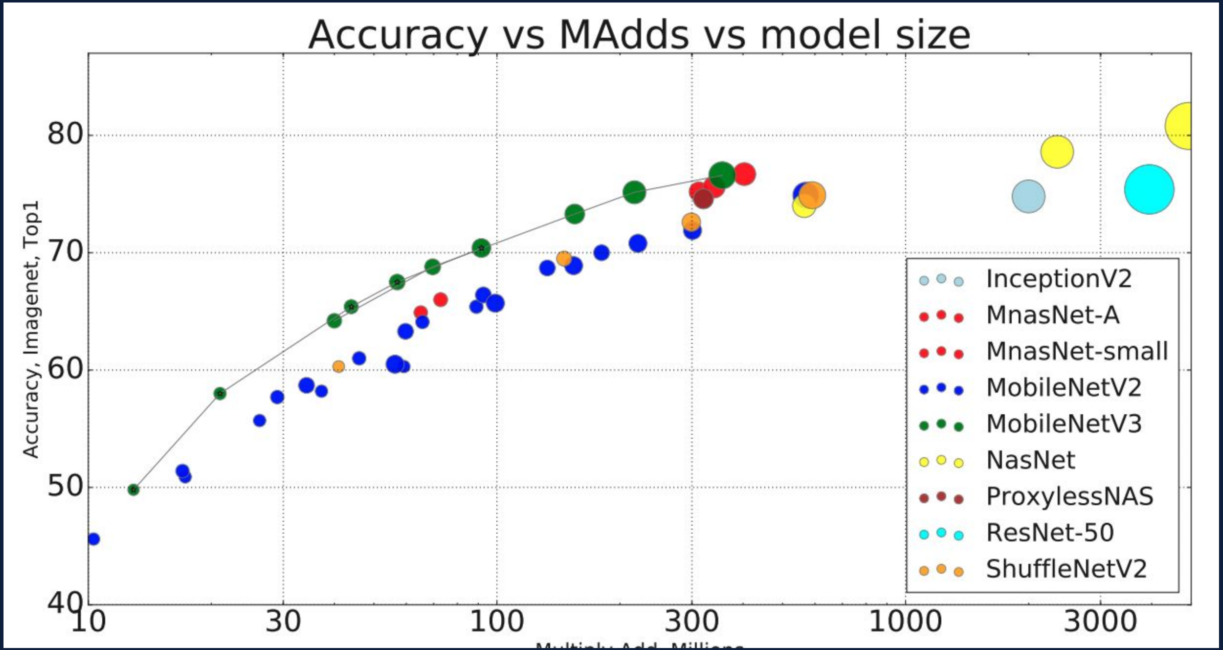
Graph neural networks . . . . .
Generative adversarial network
Dense prediction tasks . . . . .
Adversarial robustness . . . . .
Self-supervised learning for NAS



[NAS-Bench-360 (2021)]

[Chen et al. 2018]

Fitting Models on Edge Devices



Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

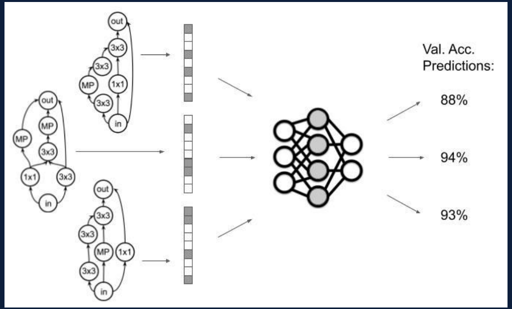
[MobileNetV3 (2019)]

Roadmap

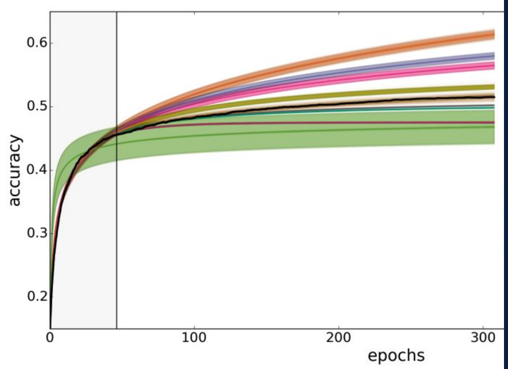
- Motivation and Introduction
- **Performance Prediction**
  - BANANAS
  - Learning curve extrapolation
  - Zero-cost proxies
- NAS Benchmarks
- Recommender Systems

Performance Predictors

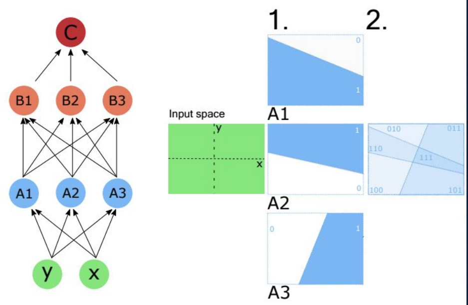
Predict the (relative) accuracy of an architecture, without fully training it.



Model-based



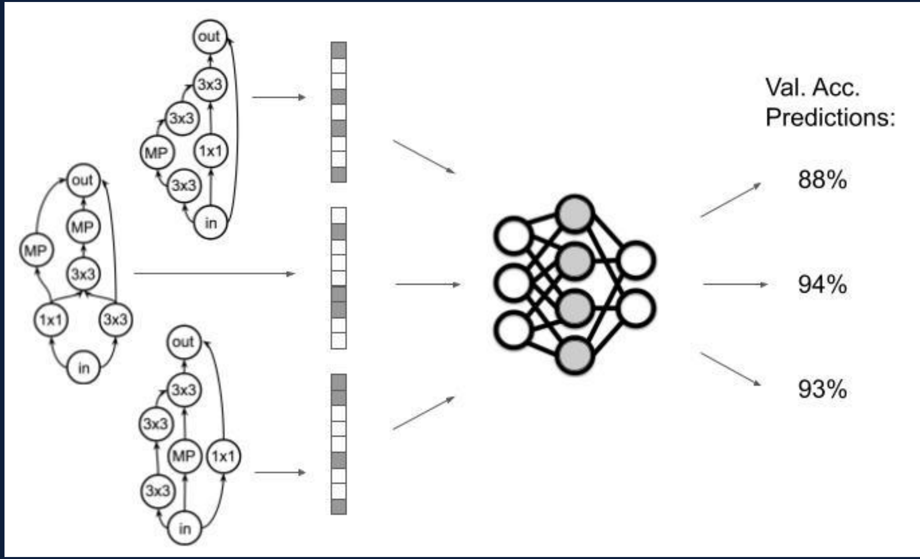
Learning curve extrapolation



Zero-cost proxies

Model-Based Predictors

Train a surrogate model



- Gaussian processes [Kandasamy et al. 2018], [Jin et al. 2018]
- Boosted trees [Luo et al. 2020], [Siems et al. 2020]
- GNNs [Shi et al. 2019], [Wen et al. 2019]
- Specialized encodings [White et al. 2019], [Ning et al. 2020]

High init time, **low query time**

“BO + Neural Predictor” Framework

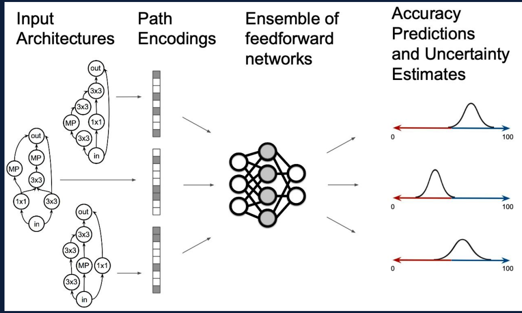
[NASGBO, 2019], [BONAS, 2019], [BANANAS, 2019]

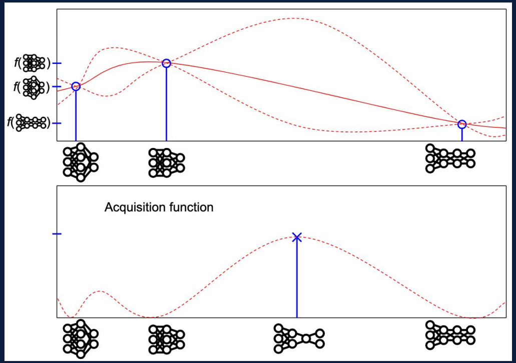
Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of meta neural networks on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.





Train 10 arch.’s each iteration

“BO + Neural Predictor” Components

Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of meta neural networks on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.

- Architecture encoding
- Uncertainty calibration
- Neural predictor architecture
- Acquisition optimization strategy
- Acquisition function

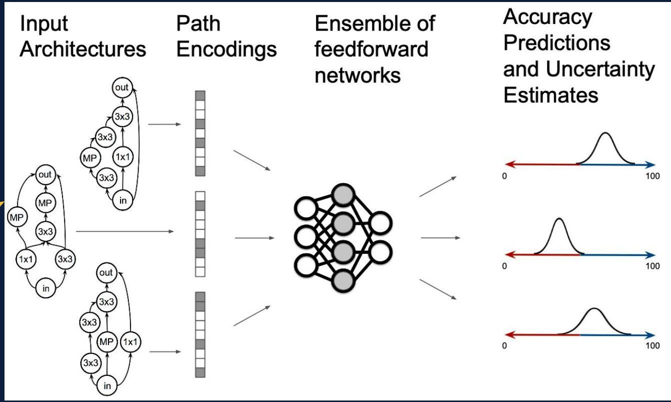
BANANAS

Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of meta neural networks on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.



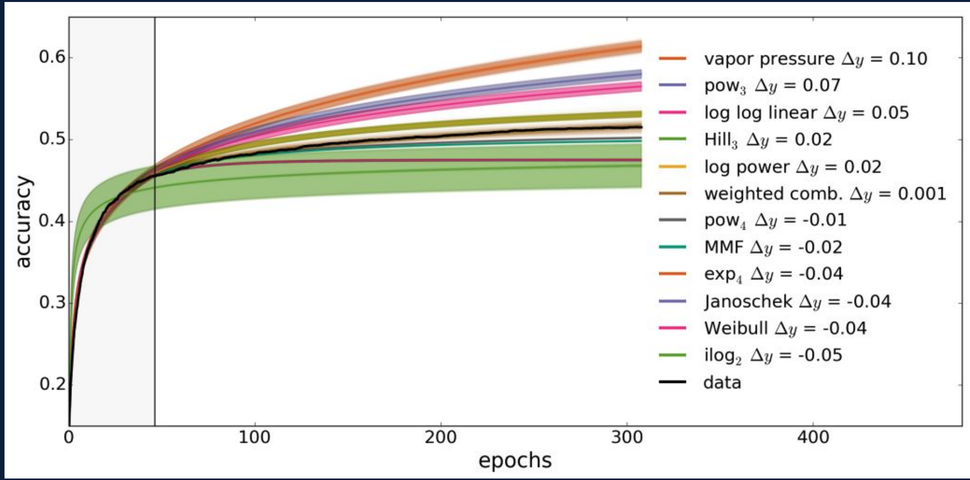
Path encoding, ensemble

Small mutations

Independent Thompson Sampling

[White, Neiswanger, Savani, AAAI 2021]

Learning curve based predictors



- Learning curve extrapolation
  - Fit partial learning curve to parametric model [Domhan et al. 2015]
  - Bayesian NN [Klein et al. 2017]
- LCE + Surrogate
  - SVR [Baker et al. 2017]
  - Full LC + Bayesian NN [Klein et al. 2017]

No init time, **high query time**

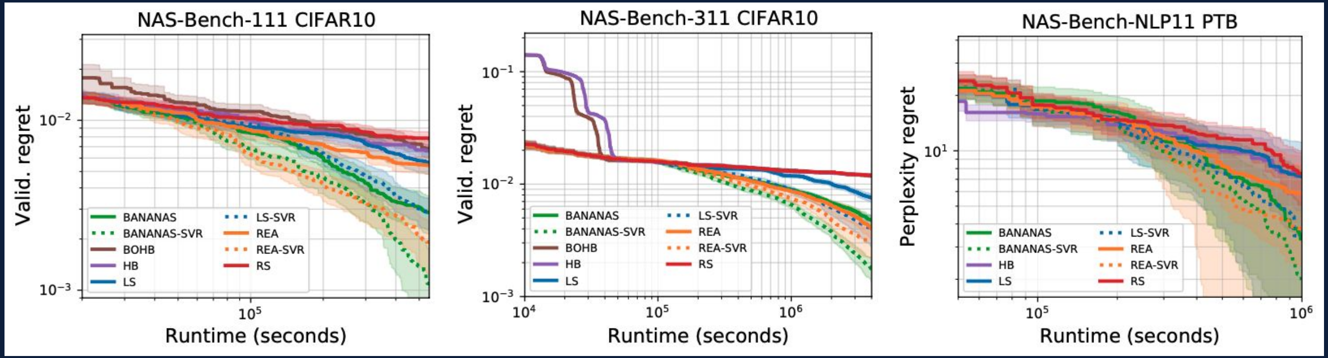
LCE Framework

Algorithm 1 Single-Fidelity Algorithm

initialize history
while $t < t_{\max}$:
  arches = gen_candidates(history)
  accs = train(arches, epoch=$E_{\max}$)
  history.update(arches, accs)
Return arch with the highest acc

Algorithm 2 LCE Framework

initialize history
while $t < t_{\max}$:
  arches = gen_candidates(history)
  accs = train(arches, epoch=E_{few})
  sorted_by_pred = LCE(arches, accs)
  arches = sorted_by_pred[:top_n]
  accs = train(arches, epoch=$E_{\max}$)
  history.update(arches, accs)
Return arch with the highest acc

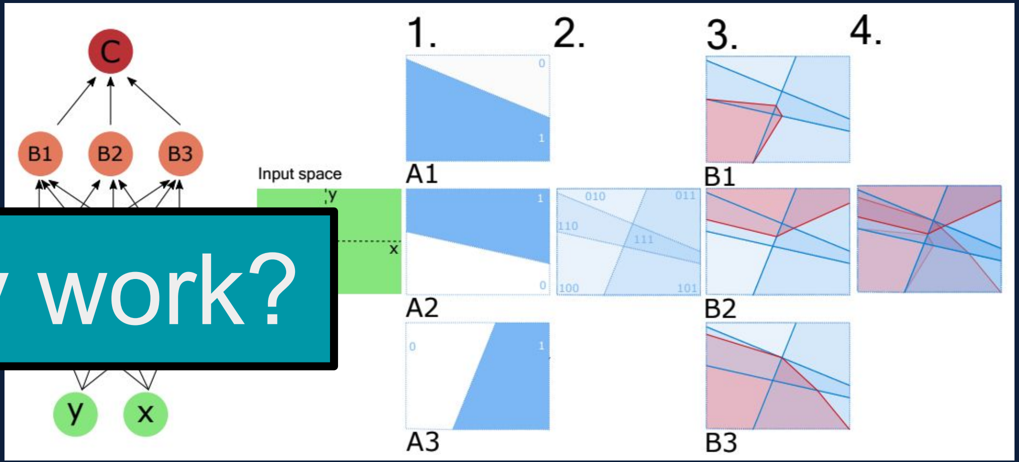


[Yan*, White*, Savani, Hutter, NeurIPS 2021]

Zero-cost proxies

epe-nas [21]	Jacobian
fisher [42]	Pruning-at-init
flops [25]	Baseline
grad-norm [1]	Pruning-at-init
grasp [43]	Pruning-at-init
l2-norm [1]	[illegible]
jacov [23]	[illegible]
nwot [23]	[illegible]
params [25]	Baseline
plain [1]	Baseline
snip [14]	Pruning-at-init
synflow [39]	Pruning-at-init
zen-score [16]	Piece. Lin.

Do they work?



[Mellor et al. 2020]

Compute an estimate in 5 seconds

Zero-cost proxies

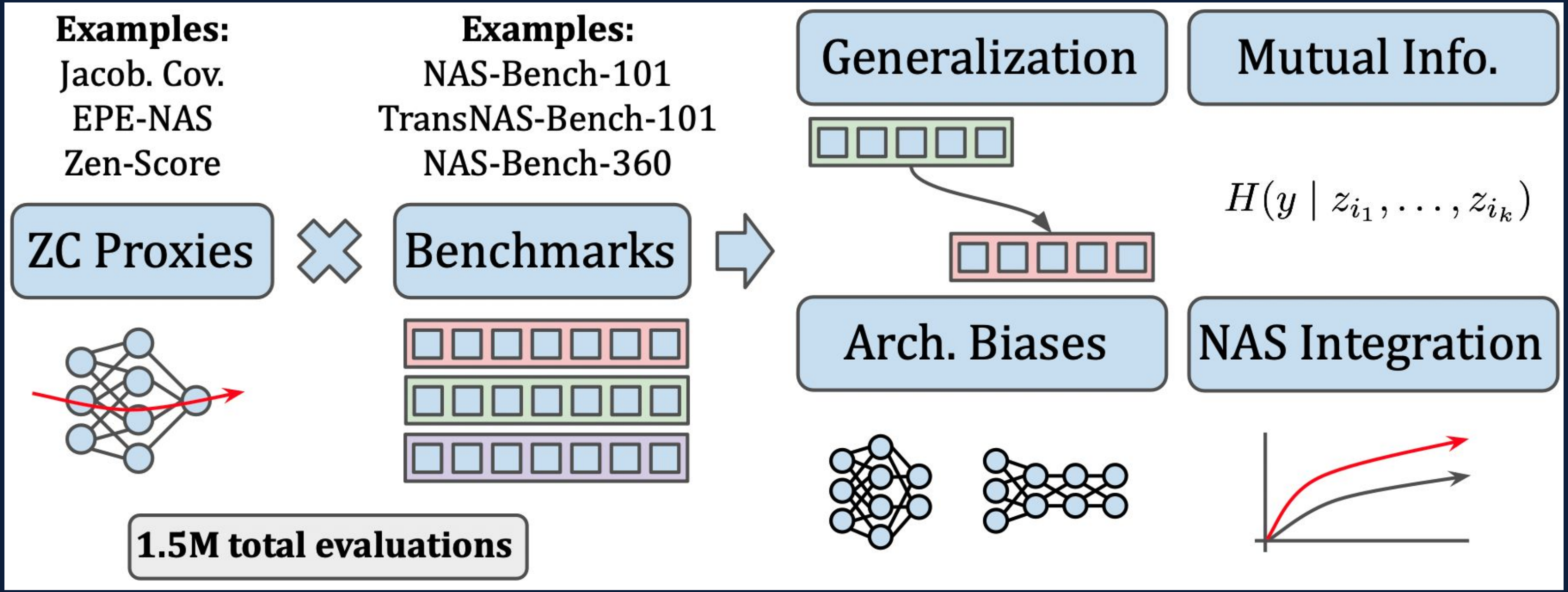
Table 4: Average ranking of each of the ZC proxies on each search space, and over all search spaces.

	`fisher`	`grad_norm`	`grasp`	`jacob_cov`	`snip`	`synflow`	`flops`	`params`
NATS-Bench TSS	6.0	6.0	5.0	4.0	5.67	**1.33**	4.0	4.0
DARTS	4.6	4.2	4.6	4.8	4.6	5.4	4.0	**3.8**
TransNAS-Bench-101	**2.75**	4.5	4.5	7.5	3.0	4.5	4.0	5.25
Overall	4.33	4.75	4.67	5.5	4.33	4.08	**4.0**	4.33

- Still do not consistently beat “flops”, “params”
- No single ZC proxy performs well consistently
- Promising when used in conjunction with other NAS techniques

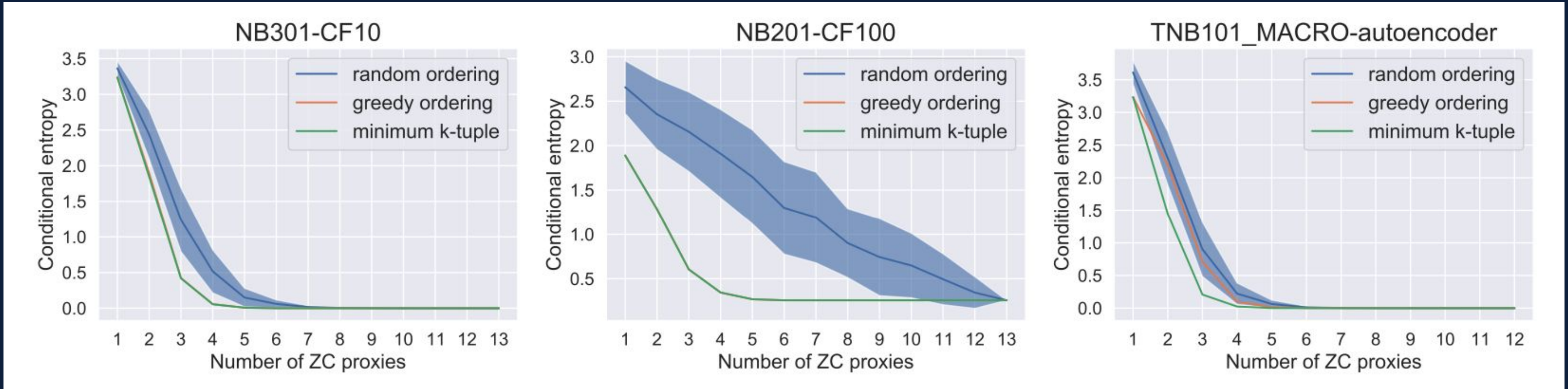
[White, Khodak, Tu, Shah, Bubeck, Dey, ICLR Blog Track, 2022]

NAS-Bench-Suite-Zero (28 tasks)



Krishnakumar*, White*, Zela*, Tu*, Safari, Hutter. Preprint 2022 https://github.com/automl/NASLib/tree/zerocost

Complementary info in ZC proxies

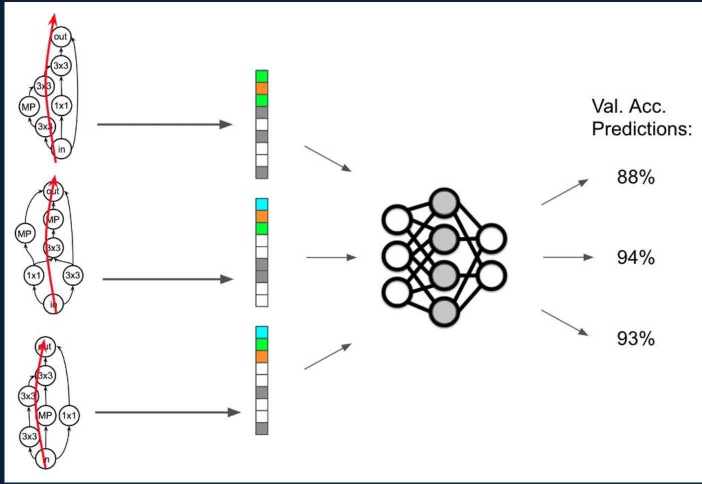


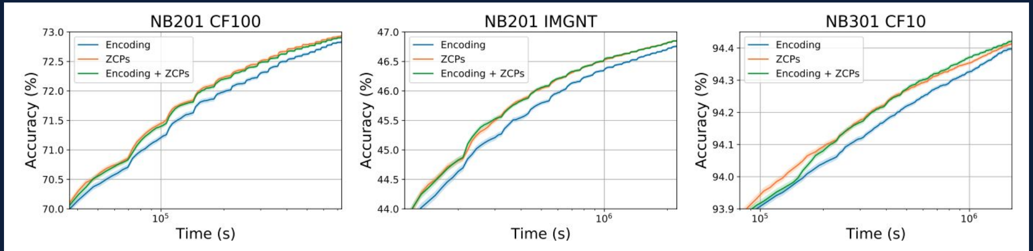
Conditional entropy $H(y \mid z_{i_1}, \ldots, z_{i_k})$ vs. k

Krishnakumar*, White*, Zela*, Tu*, Safari, Hutter. Preprint 2022

NAS integration

Features Benchmark	Encoding	ZC	Both	% Improvement (ZC)	% Improvement (Both)
NB101-CF10	0.546	0.708	0.718	29.67	31.50
NB201-CF10	0.622	0.905	0.906	45.50	45.66
NB201-CF100	0.640	0.907	0.908	41.71	41.87
NB201-IMGNT	0.683	0.879	0.883	28.70	29.28
NB301-CF10	0.314	0.405	0.465	28.98	48.09
TNB101_MACRO-AUTOENC	0.673	0.831	0.837	23.48	24.37
TNB101_MACRO-JIGSAW	0.809	0.706	0.809	-12.73	0.00
TNB101_MACRO-NORMAL	0.617	0.710	0.716	15.07	16.05
TNB101_MACRO-OBJECT	0.736	0.840	0.843	14.13	14.54
TNB101_MACRO-ROOM	0.683	0.589	0.707	-13.76	3.51
TNB101_MACRO-SCENE	0.832	0.891	0.899	7.09	8.05
TNB101_MACRO-SEGMENT	0.900	0.807	0.876	-10.33	-2.67
TNB101_MICRO-AUTOENC	0.714	0.754	0.803	5.60	12.46
TNB101_MICRO-JIGSAW	0.585	0.730	0.743	24.79	27.01
TNB101_MICRO-NORMAL	0.657	0.801	0.809	21.92	23.14
TNB101_MICRO-OBJECT	0.637	0.733	0.752	15.07	18.05
TNB101_MICRO-ROOM	0.582	0.843	0.844	44.85	45.02
TNB101_MICRO-SCENE	0.710	0.849	0.866	19.58	21.97
TNB101_MICRO-SEGMENT	0.767	0.886	0.897	15.51	16.95

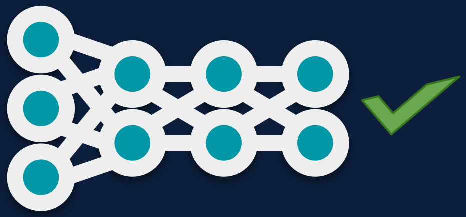


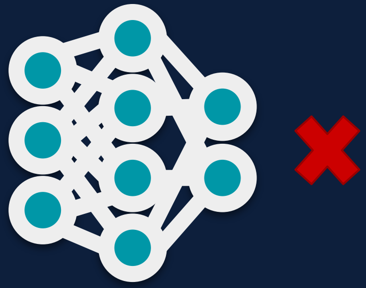


[Krishnakumar*, White*, Zela*, Tu*, Safari, Hutter. Preprint 2022]

Removing biases in ZC proxies

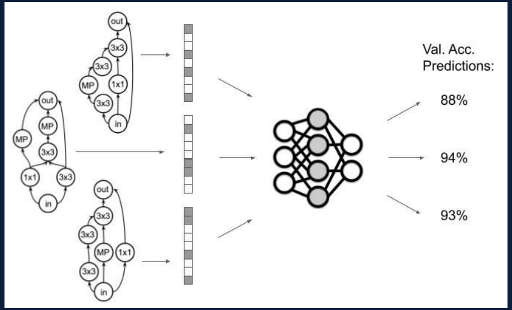
$$f'(a) = f(a) \cdot \frac{1}{b(a) + C}$$



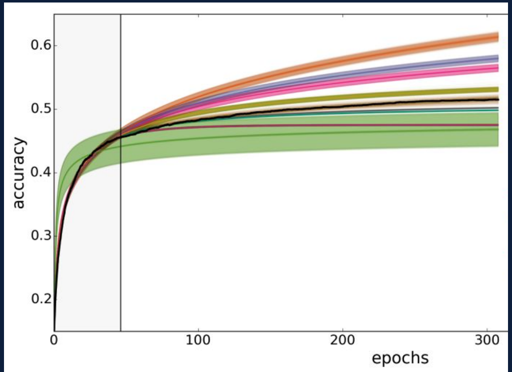


ZC proxy	dataset	bias metric	original bias	original perf.	new bias	new perf.	strategy
l2-norm	NB201-CF10	conv:pool	0.87	0.42	0.00	0.10	minimize
					0.37	0.11	equalize
					0.70	0.44	performance
nwot	NB301-CF10	conv:pool	0.78	0.49	0.00	0.03	minimize
					0.29	0.14	equalize
					0.78	0.49	performance
synflow	NB201-CF100	cell size	0.57	0.68	0.01	0.64	minimize
					0.35	0.71	equalize
					0.35	0.71	performance
synflow	NB201-IM	cell size	0.58	0.76	0.01	0.62	minimize
					0.43	0.76	equalize
					0.46	0.76	performance
flops	NB301-CF10	num. skip	-0.35	0.43	-0.01	0.06	minimize
					0.12	-0.05	equalize
					-0.35	0.43	performance

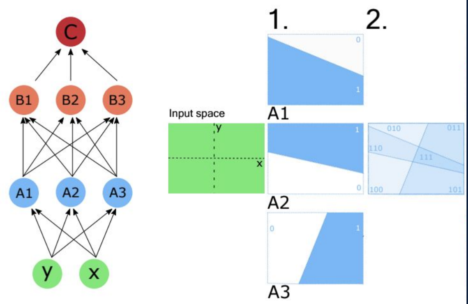
Performance predictor families



Model-based

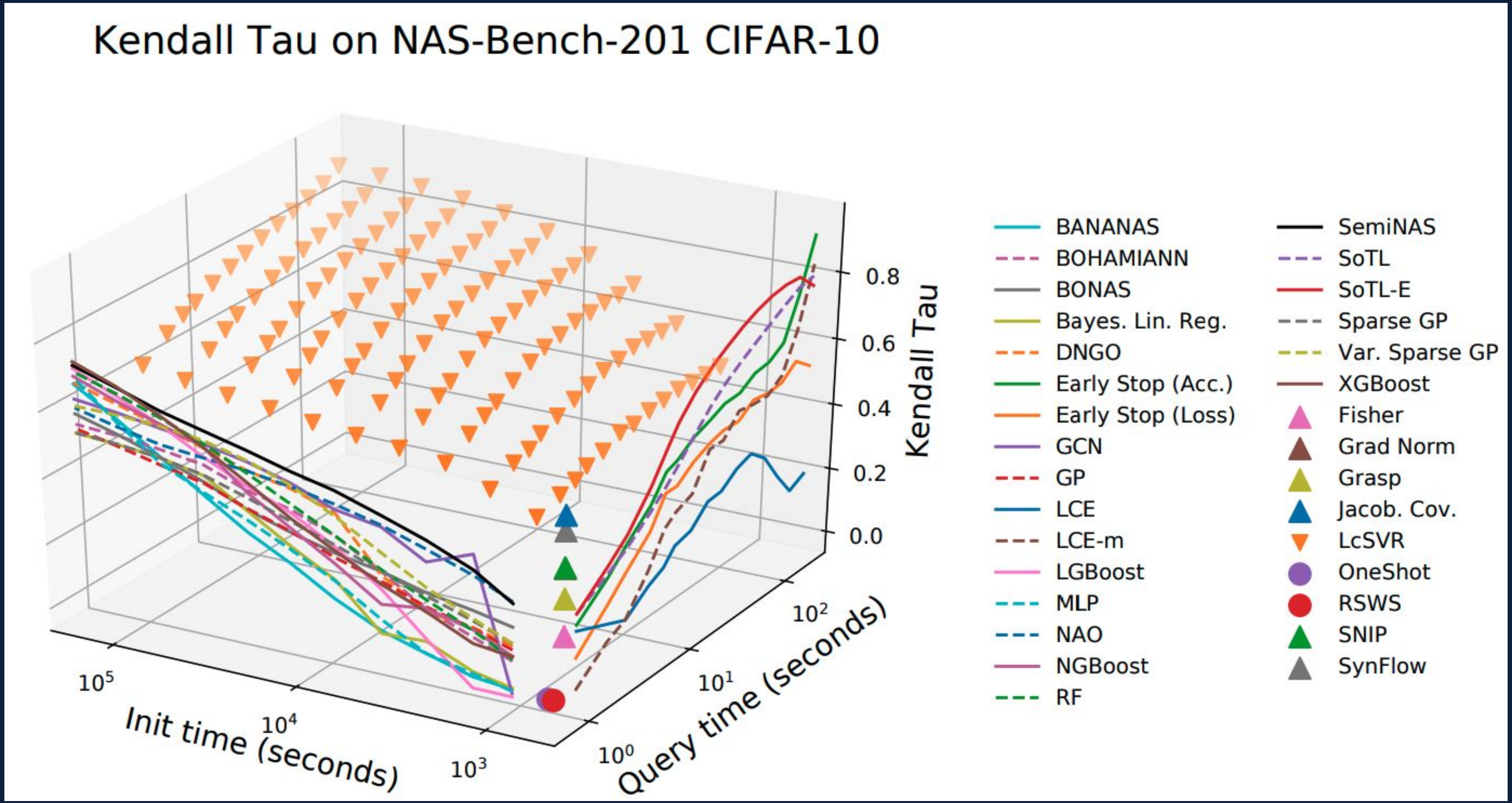


Learning curve extrapolation



Zero-cost proxies

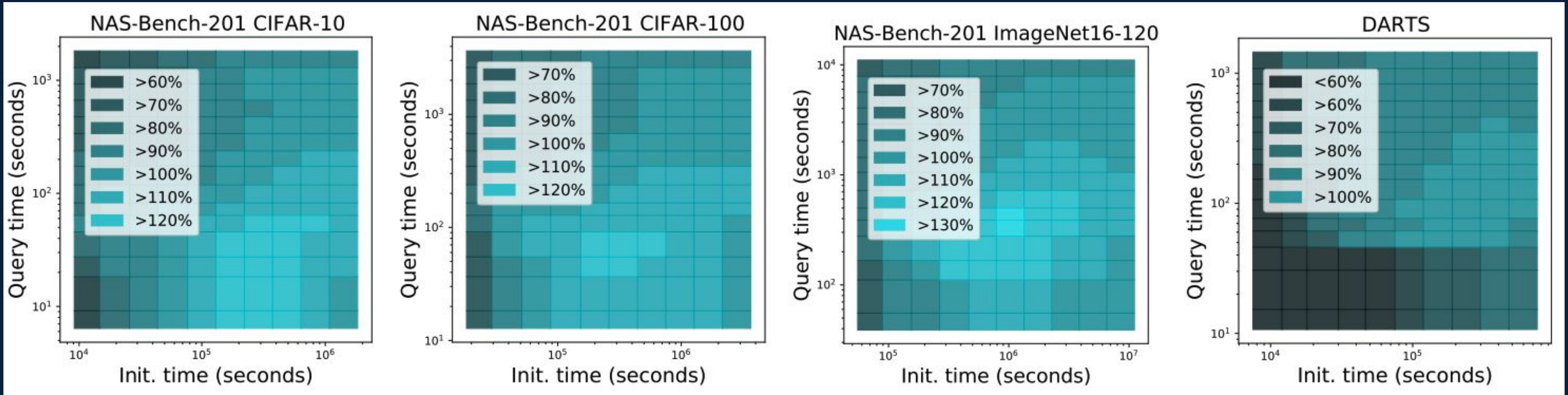
Performance predictors



White, Zela, Ru, Liu, Hutter, NeurIPS 2021

OMNI: The Omnipotent Predictor

Combine best predictors from each family



White, Zela, Ru, Liu, Hutter, NeurIPS 2021

Roadmap

- Motivation and Introduction
- Performance Prediction
  - BANANAS
  - Learning curve extrapolation
  - Zero-cost proxies
- NAS Benchmarks
- Recommender Systems

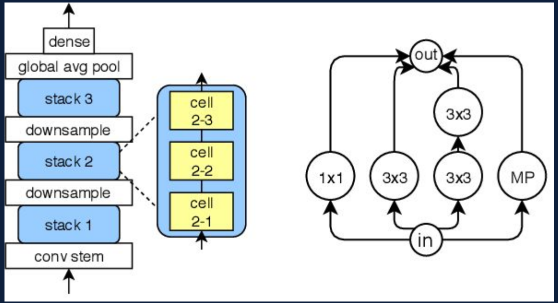
Tables of results

- Different epochs
- Different hardware
- Few trials

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	-	manual
NASNet-A (Zoph et al., 2018)	2.65	3.3	2000	RL
AmoebaNet-A (Real et [illegible]019)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-B (Real e[illegible]9)	2.55 ± 0.05	2.8	3150	evolution
PNAS (Liu et al., 2018)*	3.41 ± 0.09	[illegible]	225	SMBO
ENAS (Pham et al., 2018)	2.89	[illegible]	0.5	RL
NAONet (Luo et al., 2018)	3.53	[illegible]	0.4	NAO
SNAS (moderate) (Xie et al., 20[illegible]	[illegible]85 ± 0.0[illegible]	[illegible]8	1.5	gradient
GDAS (Dong & Yang, 2019)	[illegible].93	3.4	0.3	gradient
BayesNAS (Zhou et al., 2019)	[illegible]	3.4	0.2	gradient
ProxylessNAS (Cai et al., 2019)†	[illegible]	5.7	4.0	gradient
NASP (Yao et al., 2020)	2[illegible]	3.3	0.1	gradient
P-DARTS (Chen et al., 2019)	[illegible]	3.4	0.3	gradient
PC-DARTS (Xu et al., 2020)	[illegible] ± 0[illegible]	3.6	0.1	gradient
R-DARTS (L2) Zela et al. (2020)	[illegible]95 ± 0.2[illegible]	[illegible]	1.6	gradient
DARTS (Liu et al., 2019)	3.00 ± 0.14	[illegible]	0.4	gradient
SDARTS-RS (Chen & Hsie[illegible]	2.67 ± 0.03	[illegible]	0.4	gradient
SGAS (Cri 1. avg) (Li et[illegible])	2.66 ± 0.24	3[illegible]	0.25	gradient
DARTS+PT (avg)*	2.61 ± 0.08	3.0	0.8‡	gradient
DARTS+PT (best)	2.48	3.3	0.8‡	gradient
SDARTS-RS+PT (avg)*	2.54 ± 0.10	3.3	0.8‡	gradient
SDARTS-RS+PT (best)	2.44	3.2	0.8‡	gradient
SGAS+PT (Crit.1 avg)*	2.56 ± 0.10	3.9	0.29‡	gradient
SGAS+PT (Crit.1 best)	2.46	3.9	0.29‡	gradient

NAS-Bench-101

- Size 423k
- Used to **simulate** NAS experiments




```
# Load the data from file (this will take some time)
nasbench = api.NASBench('/path/to/nasbench.tfrecord')

# Create an Inception-like module (5x5 convolution replaced with two 3x3
# convolutions).
model_spec = api.ModelSpec(
    # Adjacency matrix of the module
    matrix=[[0, 1, 1, 1, 0, 1, 0],    # input layer
            [0, 0, 0, 0, 0, 0, 1],    # 1x1 conv
            [0, 0, 0, 0, 0, 0, 1],    # 3x3 conv
            [0, 0, 0, 0, 1, 0, 0],    # 5x5 conv (replaced by two 3x3's)
            [0, 0, 0, 0, 0, 0, 1],    # 5x5 conv (replaced by two 3x3's)
            [0, 0, 0, 0, 0, 0, 1],    # 3x3 max-pool
            [0, 0, 0, 0, 0, 0, 0]],   # output layer
    # Operations at the vertices of the module, matches order of matrix
    ops=[INPUT, CONV1X1, CONV3X3, CONV3X3, CONV3X3, MAXPOOL3X3, OUTPUT])

# Query this model from dataset, returns a dictionary containing the metrics
# associated with this model.
data = nasbench.query(model_spec)
```

[NAS-Bench-101 (2019)]

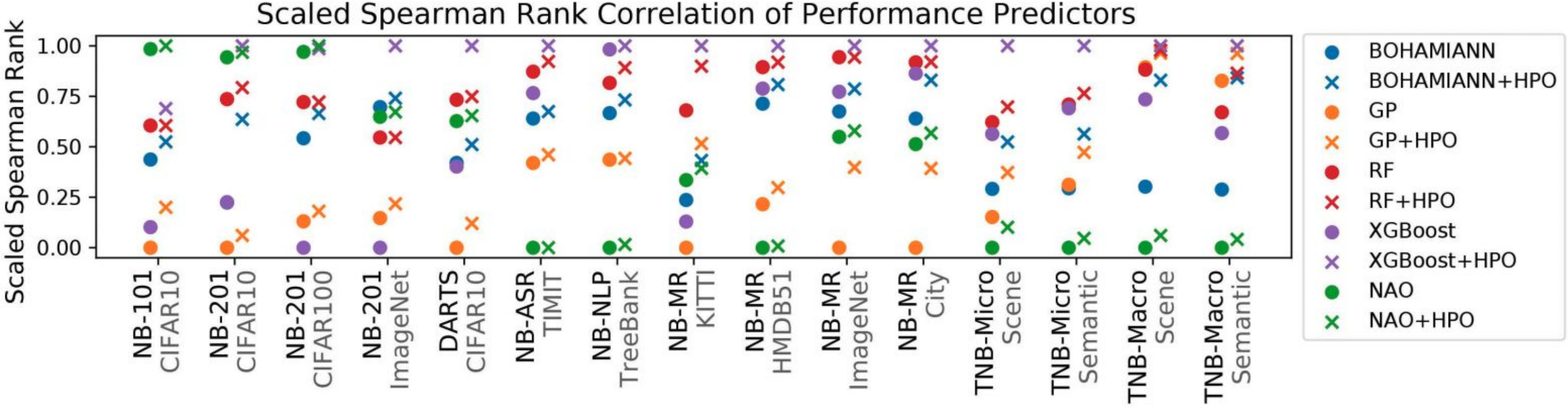
NAS Benchmarks

Benchmark	Size	Queryable		LCs	Macro	One-Shot	Task	#Tasks
		Tab.	Surr.					
NAS-Bench-101	423k	✓				✗	Image class.	1
NATS-Bench-TSS (NAS-Bench-201)	6k	✓		✓		✓	Image class.	3
NATS-Bench-SSS	32k	✓		✓	✓	✓	Image class.	3
NAS-Bench-NLP	$> 10^{53}$			✓		✗	NLP	1
NAS-Bench-1Shot1	364k	✓				✓	Image class.	1
Surr-NAS-Bench-DARTS (NAS-Bench-301)	10^{18}		✓			✓	Image class.	1
Surr-NAS-Bench-FBNet	10^{21}		✓			✗	Image class.	1
NAS-Bench-ASR	8k	✓			✓	✓	ASR	1
TransNAS-Bench-101-Micro	4k	✓		✓		✓	Var. CV	7
TransNAS-Bench-101-Macro	3k	✓		✓	✓	✗	Var. CV	7
NAS-Bench-111	423k		✓	✓		✗	Image class.	1
NAS-Bench-311	10^{18}		✓	✓		✓	Image class.	1
NAS-Bench-NLP11	$> 10^{53}$		✓	✓		✗	NLP	1
NAS-Bench-MR	10^{23}		✓		✓	✗	Var. CV	9
NAS-Bench-360	Var.				✓	✓	Var.	30
NAS-Bench-Macro	6k	✓			✓	✗	Image class.	1
HW-NAS-Bench-201	6k	✓		✓		✓	Image class.	3
HW-NAS-Bench-FBNet	10^{21}					✗	Image class.	1

NAS-Bench-Suite (25 tasks)

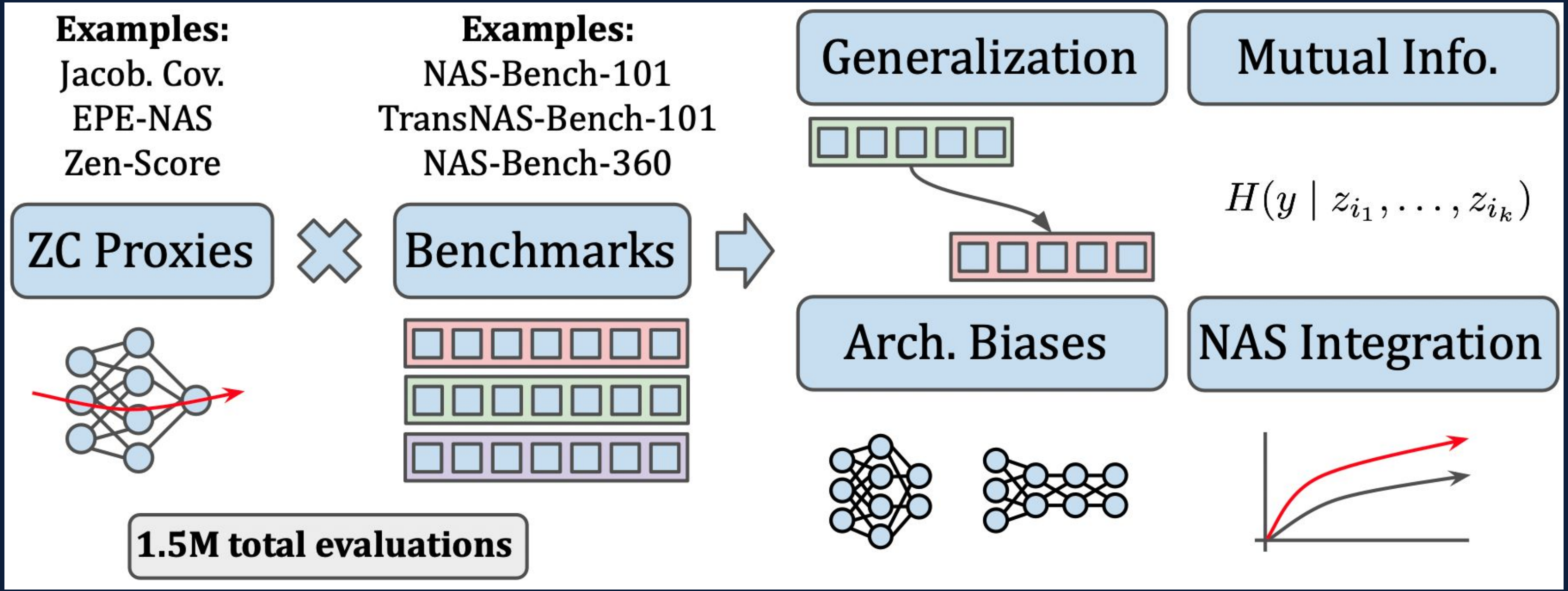
	NAS Algorithms					Performance Predictors				
	RS	RE	BANANAS	LS	NPENAS	GP	BOHAM.	RF	XGB	NAO
Avg.Rank, 101&201	4.50	3.00	3.50	**1.50**	2.50	4.67	2.83	2.17	4.17	**1.17**
Avg. Rank, non-101&201	3.06	**2.11**	2.83	3.13	3.87	4.08	3.06	**1.33**	2.46	4.08

❖ Conclusions drawn from just the popular NAS-Bench-101 and NAS-Bench-201 can be misleading!



Mehta*, White*, Zela, Krishnakumar, Zabergja, Moradian, Safari, Yu, Hutter, ICLR 2022 https://github.com/automl/NASLib

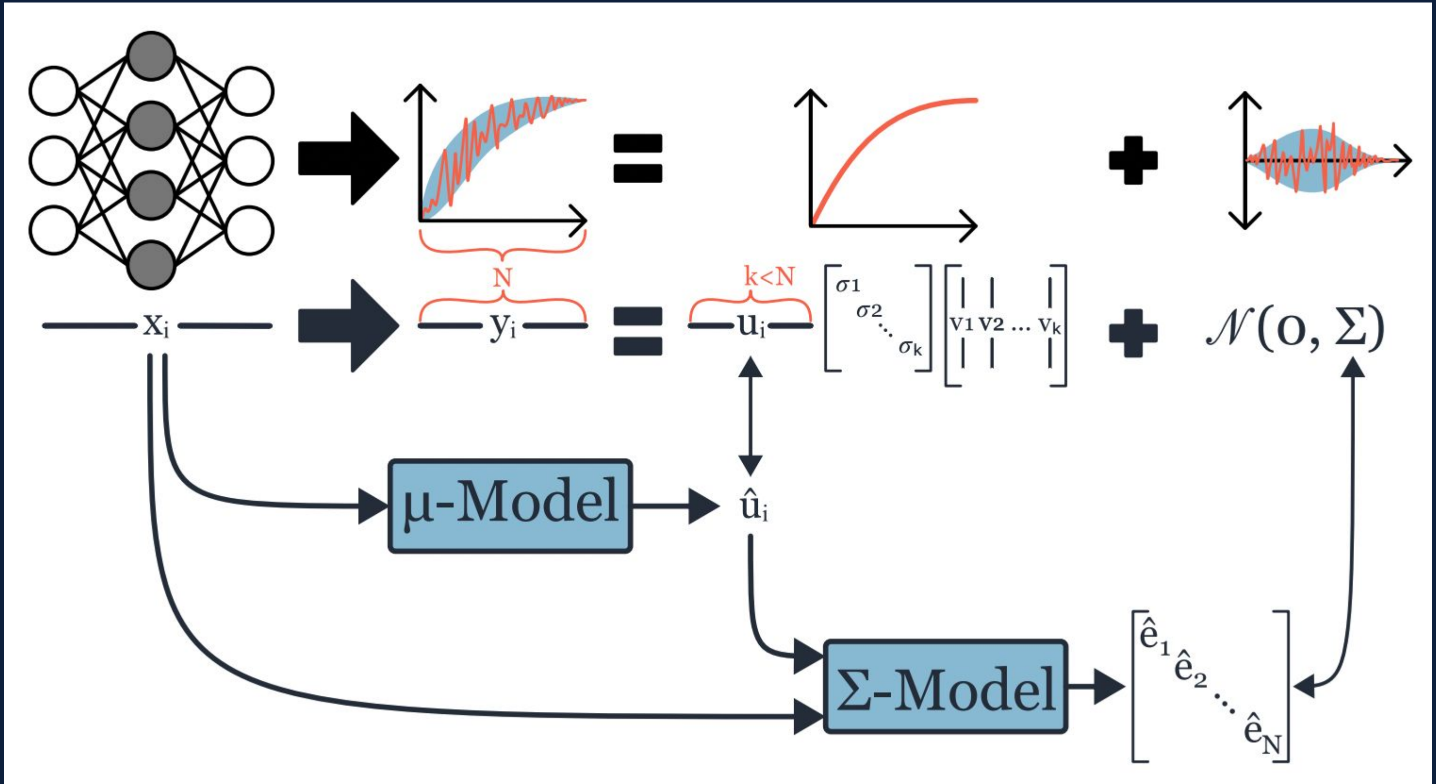
NAS-Bench-Suite-Zero (28 tasks)



Krishnakumar*, White*, Zela*, Tu*, Safari, Hutter. Preprint 2022

https://github.com/automl/NASLib/tree/zerocost

NAS-Bench-x11

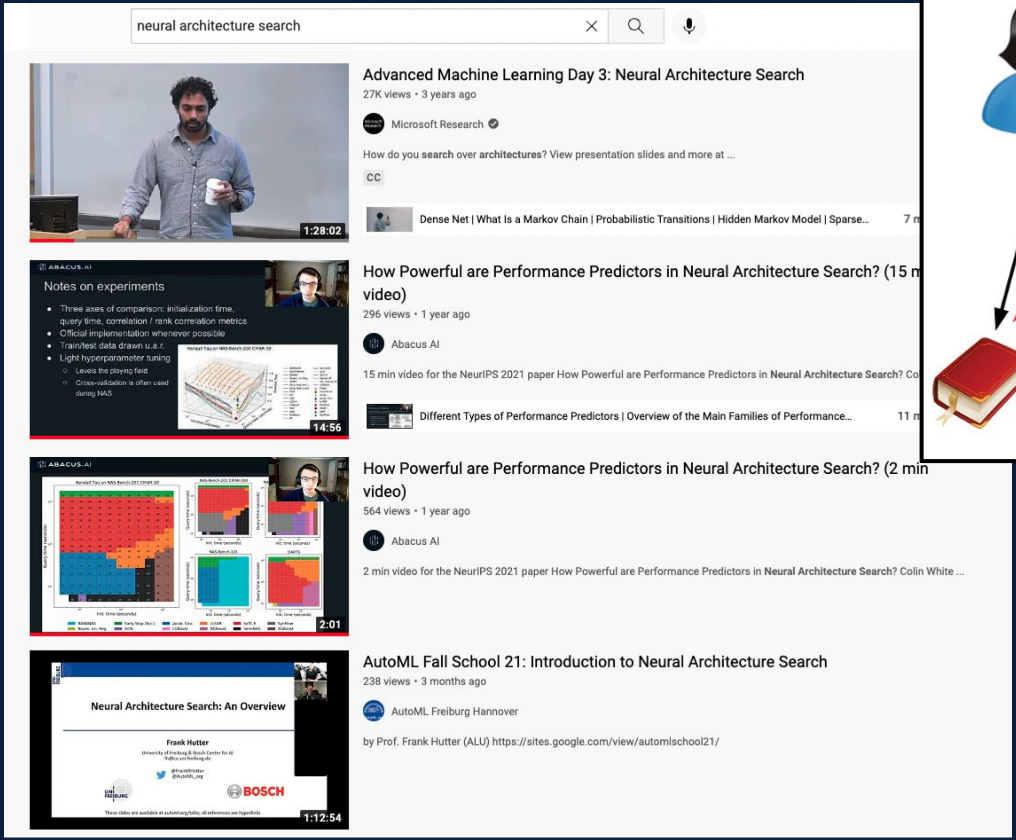


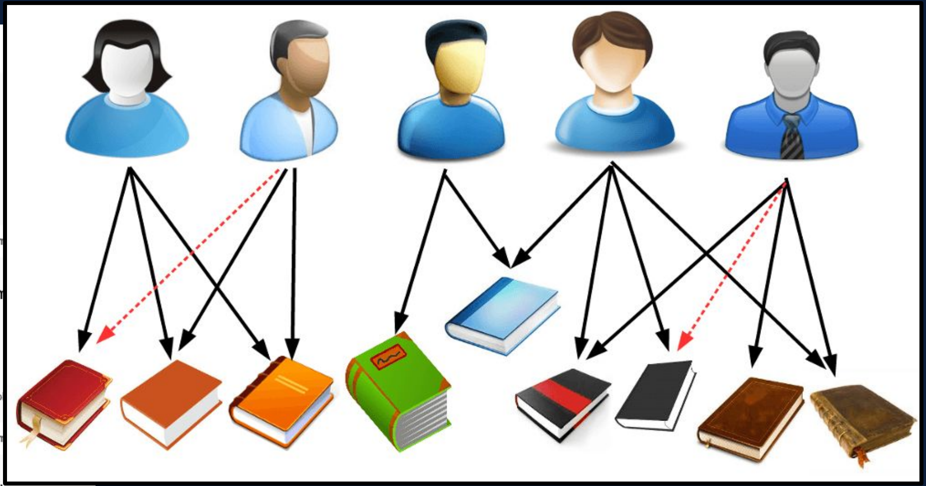
[Yan*, White*, Savani, Hutter, NeurIPS 2021]()

Roadmap

- Motivation and Introduction
- Performance Prediction
  - BANANAS
  - Learning curve extrapolation
  - Zero-cost proxies
- NAS Benchmarks
- Recommender Systems

Recommender Systems





A Worrying Analysis of Recommender Systems

Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches

Maurizio Ferrari Dacrema
Politecnico di Milano, Italy
maurizio.ferrari@polimi.it

Paolo Cremonesi
Politecnico di Milano, Italy
paolo.cremonesi@polimi.it

Dietmar Jannach
University of Klagenfurt, Austria
dietmar.jannach@aau.at

ABSTRACT

Deep learning techniques have become the method of choice for researchers working on algorithmic aspects of recommender systems. With the strongly increased interest in machine learning in general, it has, as a result, become difficult to keep track of what represents the state-of-the-art at the moment, e.g., for top-n recommendation tasks. At the same time, several recent publications point out problems in today's research practice in applied machine learning, e.g., in terms of the reproducibility of the results or the choice of the baselines when proposing new models.

In this work, we report the results of a systematic analysis of algorithmic proposals for top-n recommendation tasks. Specifically, we considered 18 algorithms that were presented at top-level research conferences in the last years. Only 7 of them could be reproduced with reasonable effort. For these methods, it however turned out that 6 of them can often be outperformed with comparably simple heuristic methods, e.g., based on nearest-neighbor or graph-based techniques. The remaining one clearly outperformed the baselines but did not consistently outperform a well-tuned non-

1 INTRODUCTION

Within only a few years, deep learning techniques have started to dominate the landscape of algorithmic research in recommender systems. Novel methods were proposed for a variety of settings and algorithmic tasks, including top-n recommendation based on long-term preference profiles or for session-based recommendation scenarios [36]. Given the increased interest in machine learning in general, the corresponding number of recent research publications, and the success of deep learning techniques in other fields like vision or language processing, one could expect that substantial progress resulted from these works also in the field of recommender systems. However, indications exist in other application areas of machine learning that the achieved progress—measured in terms of accuracy improvements over existing models—is not always as strong as expected.

Lin [25], for example, discusses two recent neural approaches in the field of information retrieval that were published at top-level conferences. His analysis reveals that the new methods do *not* significantly outperform existing baseline methods when these

Family	*Method*	*Description*
Non-personalized	TopPopular	Recommends the most popular items to everyone [18]
Nearest-Neighbor	UserKNN	User-based k-nearest neighbors [58]
	ItemKNN	Item-based k-nearest neighbors [61]
Graph-based	$P^3\alpha$	A graph-based method based on random walks [16]
	$RP^3\beta$	An extension of $P^3\alpha$ [54]
Content-Based and Hybrid	ItemKNN-CBF	ItemKNN with content-based similarity [43]
	ItemKNN-CFCBF	A simple item-based hybrid CBF/CF approach [50]
	UserKNN-CBF	UserKNN with content-based similarity
	UserKNN-CFCBF	A simple user-based hybrid CBF/CF approach
Non-Neural Machine Learning	iALS	Matrix factorization for implicit feedback data [33]
	pureSVD	A basic matrix factorization method [18]
	SLIM	A scalable linear model [36, 52]
	$EASE^R$	A recent linear model, similar to auto-encoders [63]

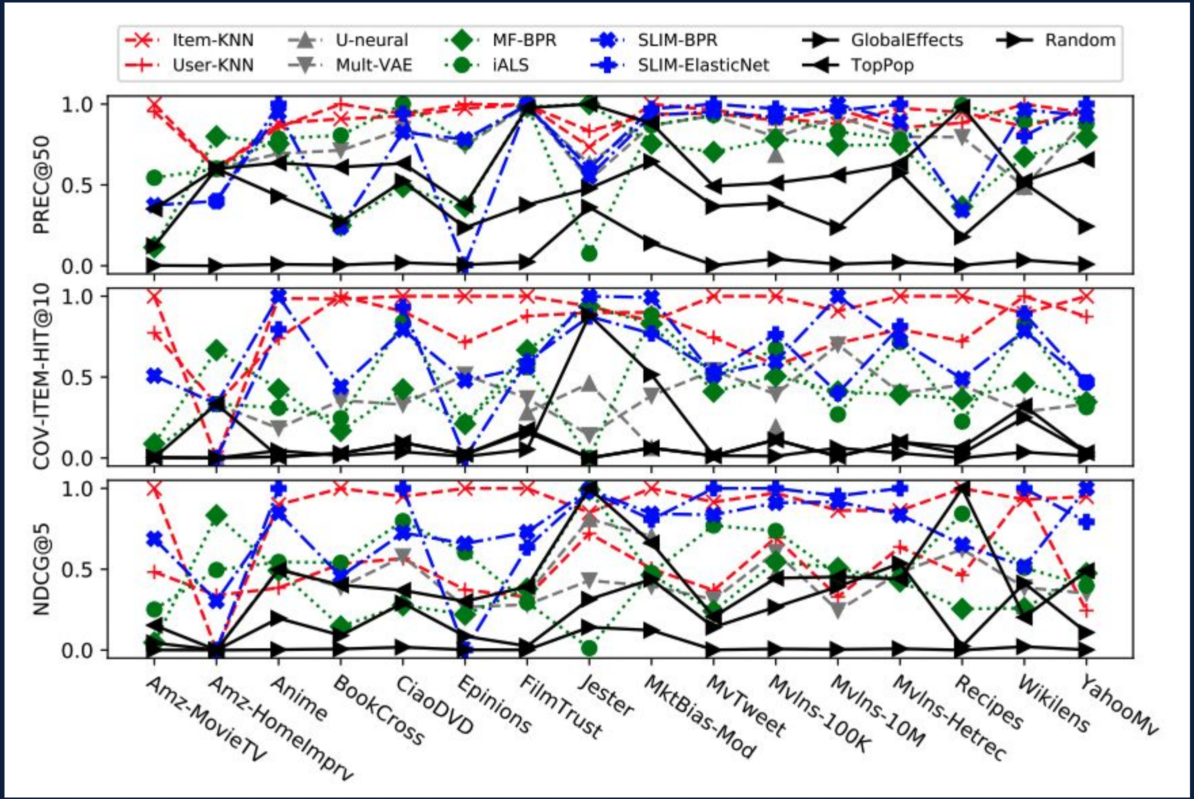
Meta-Learning for Recommender Systems

- 24 Algorithms, up to 100 hyperparameters, 85 datasets, 315 metrics

Rank	Item-KNN	P3alpha	SLIM-BPR	EASE-R	RP3beta	SVD	SLIM-ElasticNet	iALS	NMF	User-KNN	MF-Funk	TopPop	MF-Asy	MF-BPR	Mult-VAE	U-neural	GlobalEffects	CoClustering	Random	SlopeOne
Min.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	9	7
Max.	14	18	14	18	17	16	17	19	14	17	18	19	16	17	20	20	20	19	20	20
Mean	2.3	4.2	4.7	5.3	6	6	7	7	7.1	7.6	9.4	10.4	10.7	11.2	11.7	12.3	13.3	14.9	16.2	16.7

McElfresh, Khandagale, Valverde, Dickerson, White. Preprint 2022

Meta-Learning for Recommender Systems

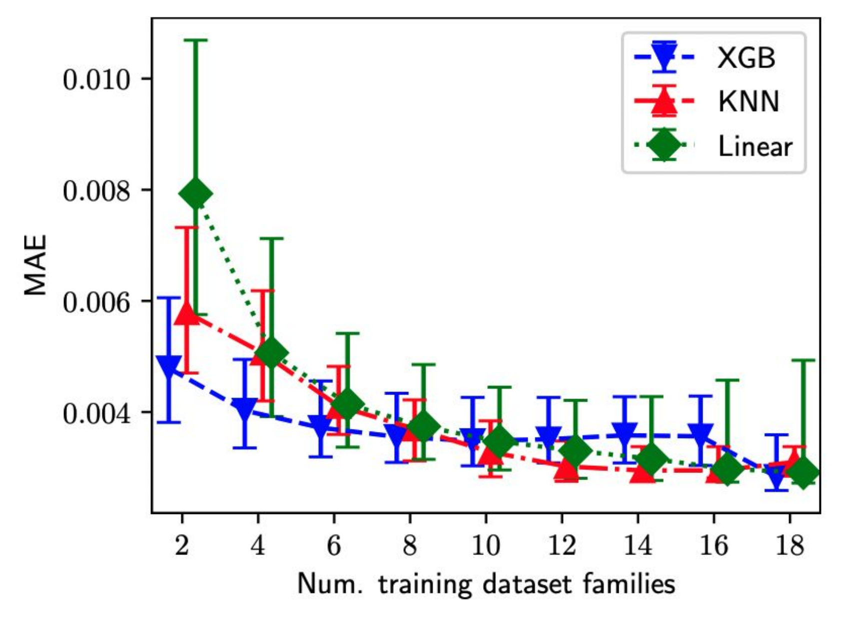


McElfresh, Khandagale, Valverde, Dickerson, White. Preprint 2022

RECZILLA

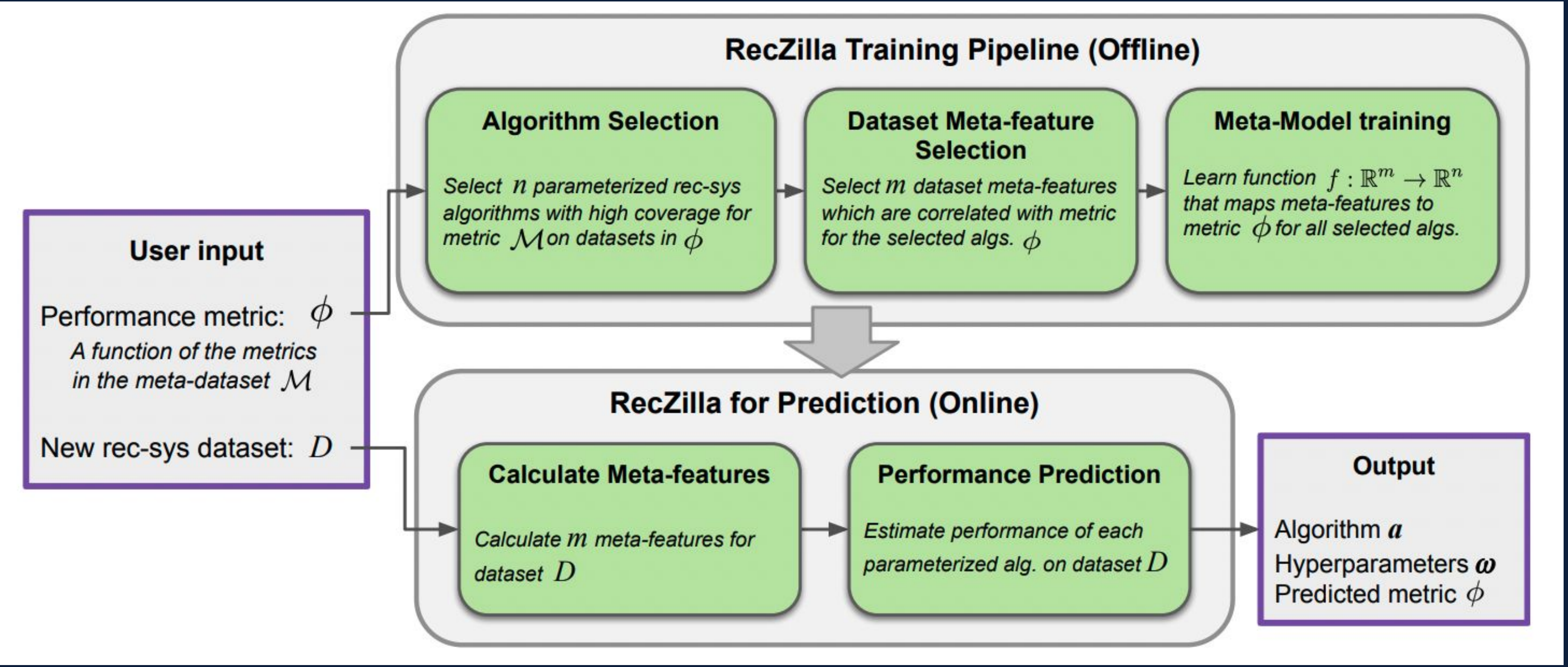
Dataset meta-features

- User distribution
- Item distribution
- Interaction distribution
- Landmarkers



McElfresh, Khandagale, Valverde, Dickerson, White. Preprint 2022

RECZILLA



McElfresh, Khandagale, Valverde, Dickerson, White. Preprint 2022

Thanks! Questions?

colin@abacus.ai   Slides (with hyperlinks): https://crwhite.ml/