

# Neural Architecture Search: Foundations and Trends

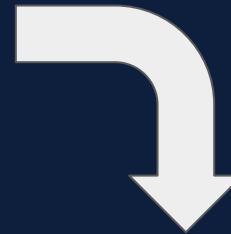
Colin White  
Abacus.AI  
[colin@abacus.ai](mailto:colin@abacus.ai)



Debadeepta Dey  
Microsoft Research  
[dedey@microsoft.com](mailto:dedey@microsoft.com)

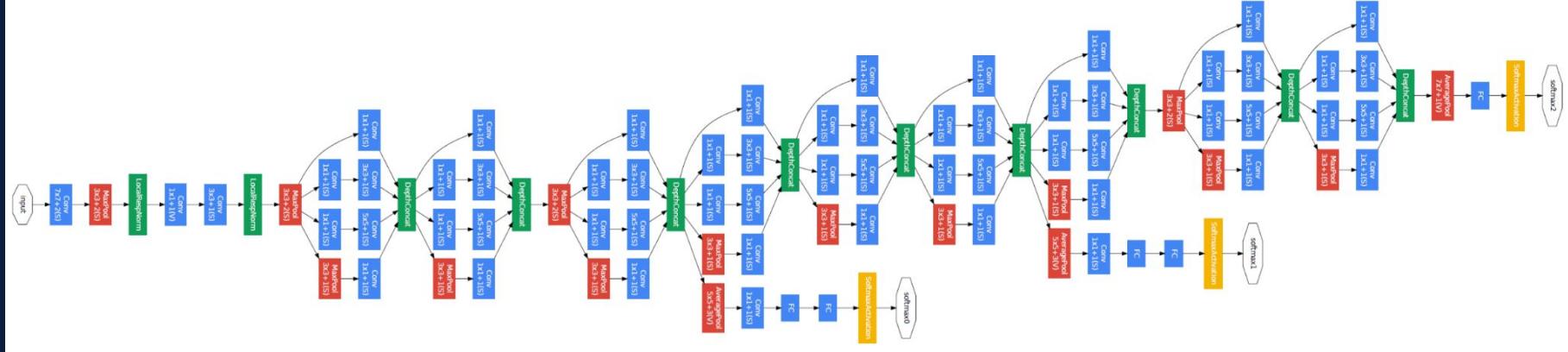


# Machine learning: a story of automation



```
from torchvision.models import resnet50, ResNet50_Weights  
  
# Best available weights (currently alias for IMAGENET1K_V2)  
# Note that these weights may change across versions  
resnet50(weights=ResNet50_Weights.DEFAULT)
```

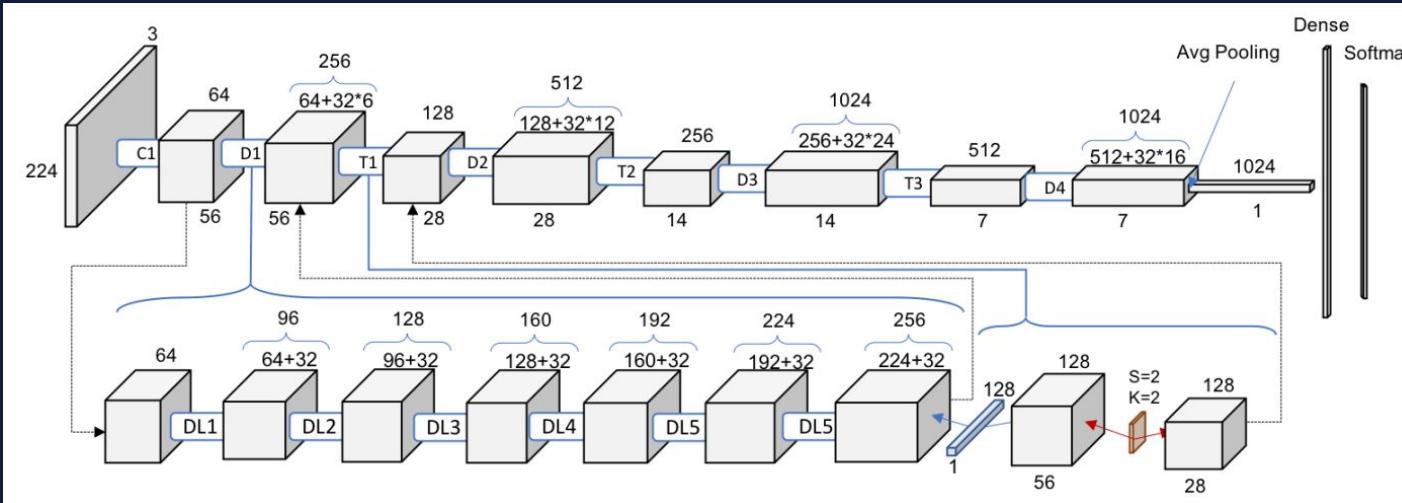
# Neural architecture search



GoogLeNet (2014)

Architectures are getting increasingly more specialized and complex

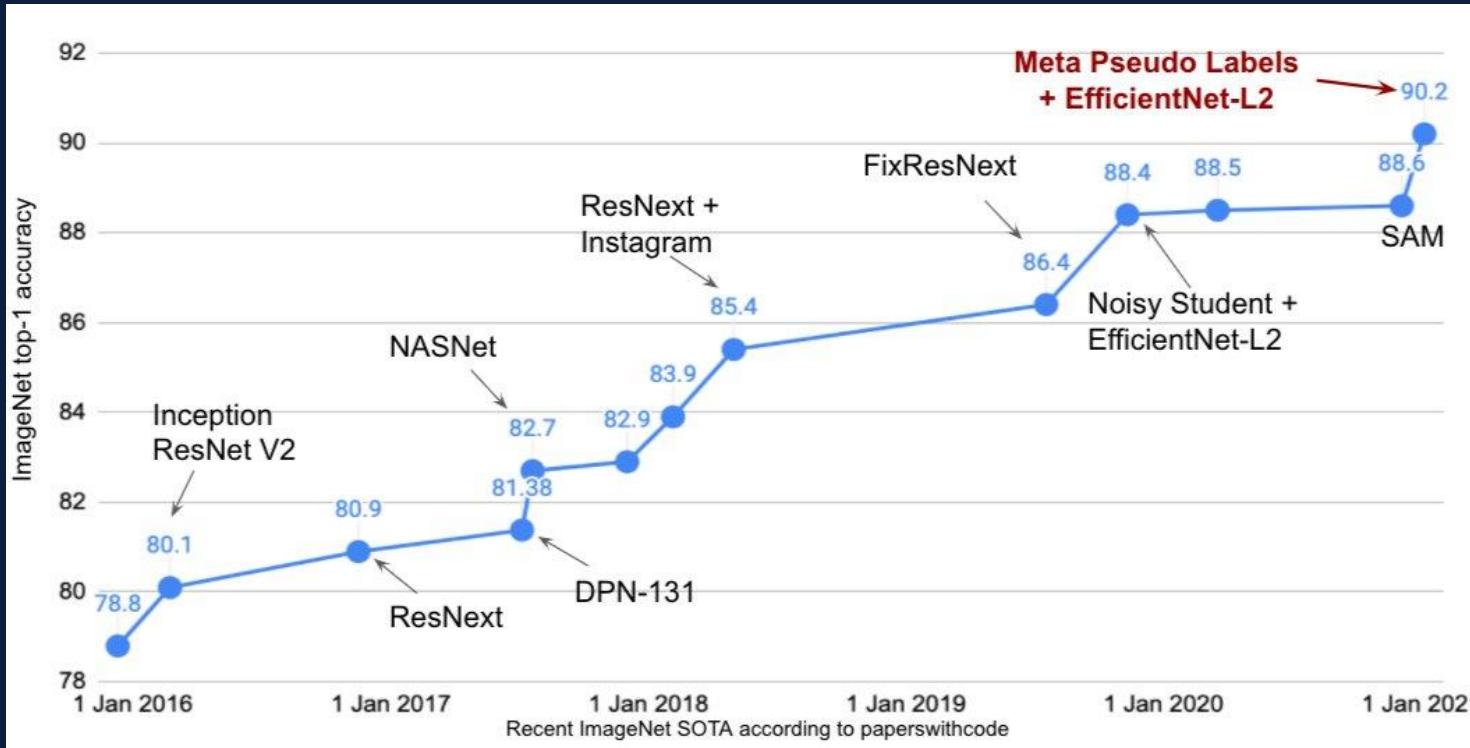
# Neural architecture search



DenseNet (2016)

Architectures are getting increasingly more specialized and complex

# Neural architecture search

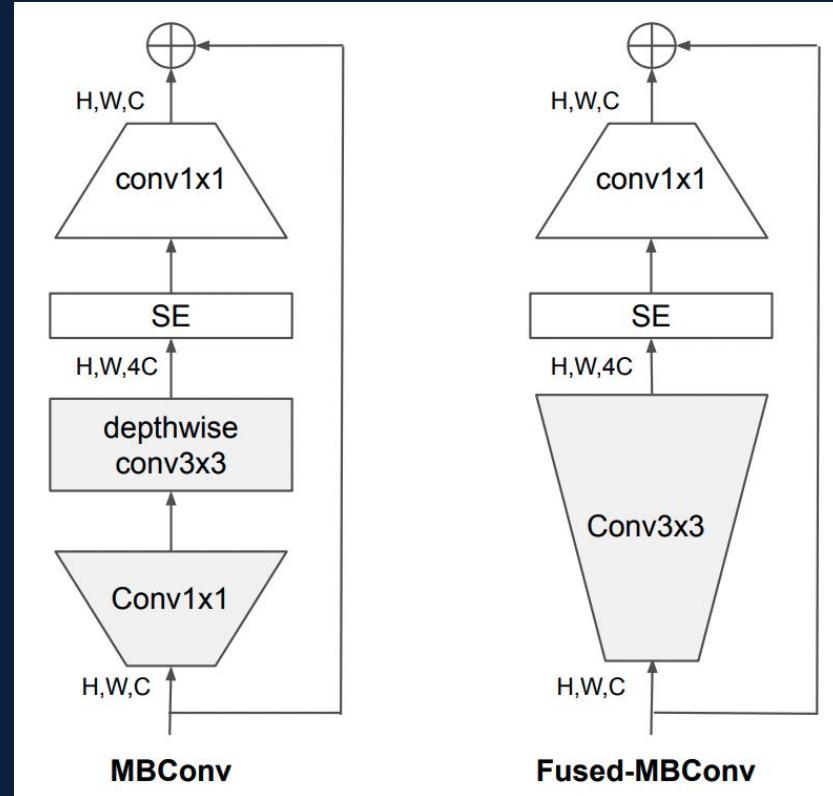


**Searched** models are replacing **human-designed** models

# Human + Neural architecture search

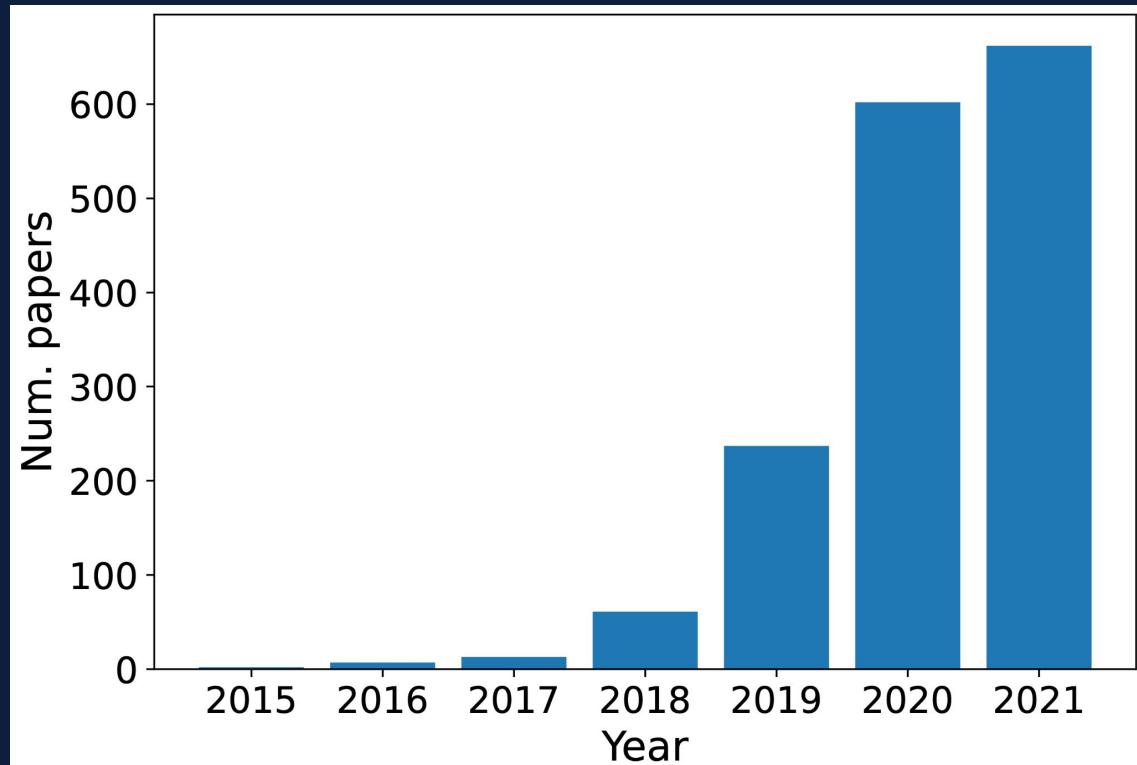
Stage $i$	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	$224 \times 224$	32	1
2	MBConv1, k3x3	$112 \times 112$	16	1
3	MBConv6, k3x3	$112 \times 112$	24	2
4	MBConv6, k5x5	$56 \times 56$	40	2
5	MBConv6, k3x3	$28 \times 28$	80	3
6	MBConv6, k5x5	$14 \times 14$	112	3
7	MBConv6, k5x5	$14 \times 14$	192	4
8	MBConv6, k3x3	$7 \times 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \times 7$	1280	1

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1



# Neural architecture search

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



# New Datasets

Spherical Omnidirectional Vision



NinaPro DB5 Prosthetics Control



FSD50K Audio Classification



Darcy Flow PDE Solver



PSICOV Protein Folding



Cosmic Astronomy Imaging



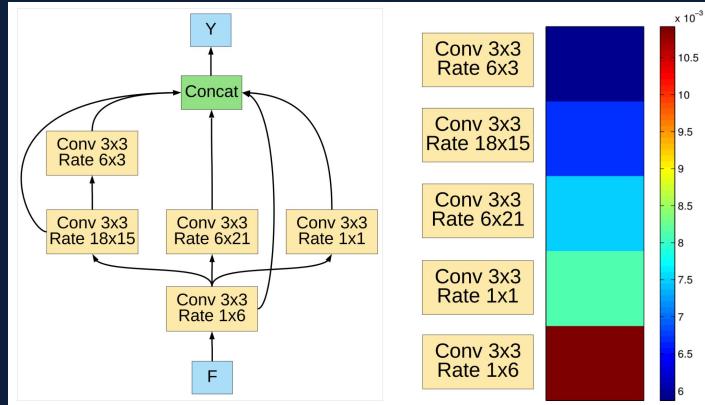
ECG Medical Diagnostics



Satellite Earth Monitoring

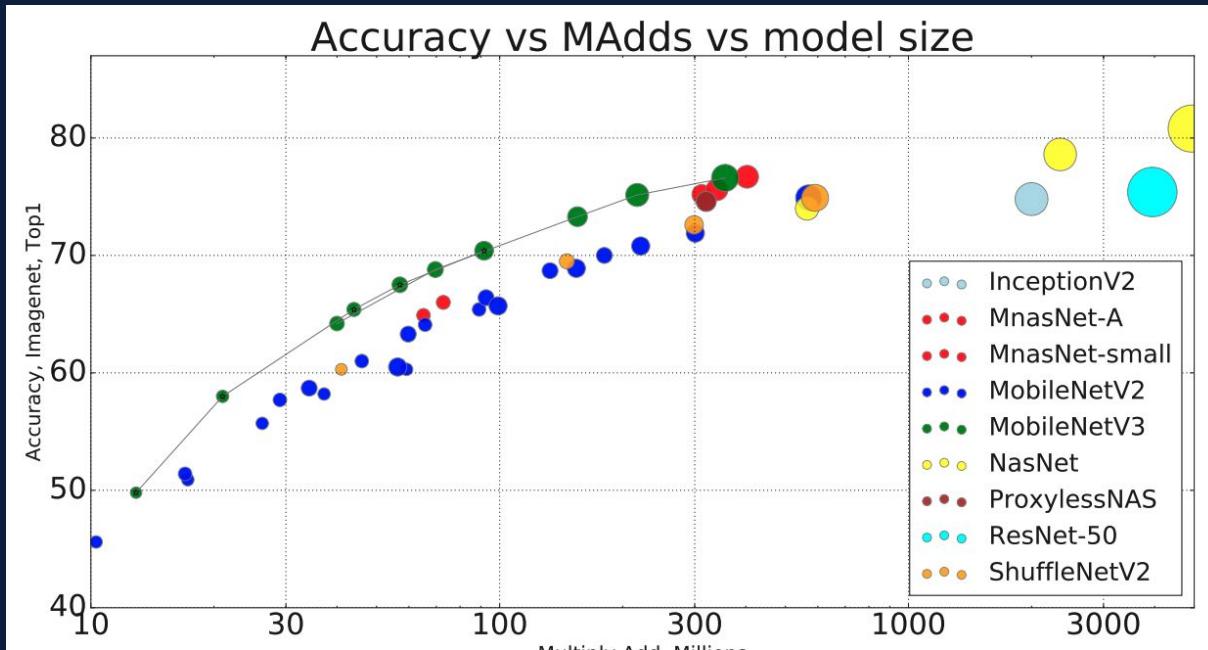


DeepSEA Genetic Prediction



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

# Fitting Models on Edge Devices



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

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

The best and most efficient architectures today are found automatically

Source: [MobileNetV3 \(2019\)](#)

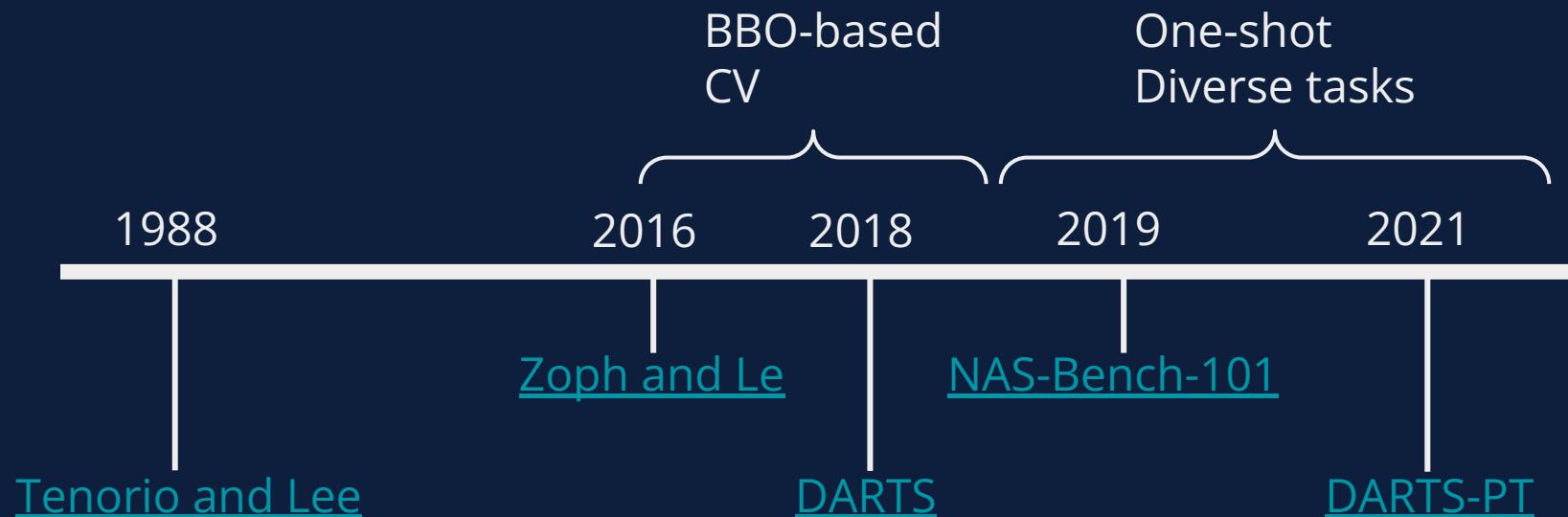
# Motivation - Summary

- Widely-used benchmarks
  - New datasets
  - Constrained / multi-objective problems
- 
- Democratizing deep learning
    - Latest techniques are just a few GPU-hours

# NAS: A History

Studied since at least the late 1980s

Resurgence in late 2016



# NAS: Basic Definition

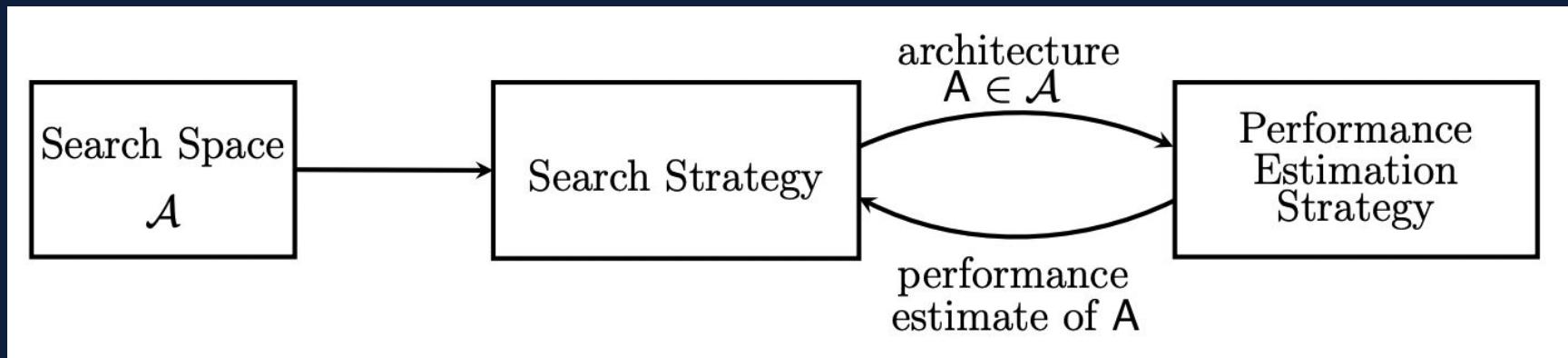
- Define a search space  $\mathcal{A}$ ,

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

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

# Three Pillars of NAS

- Search space
  - Search strategy
  - Performance estimation strategy
- } Coupled, for one-shot methods

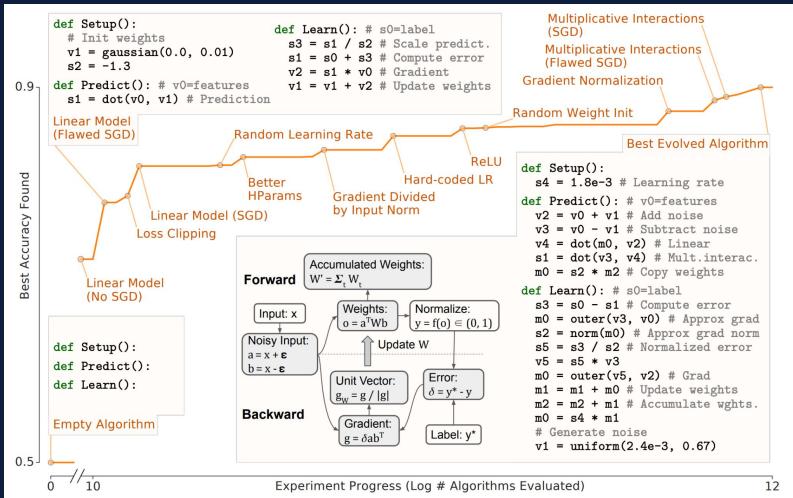


[Elsken et al. \(2018\)](#)

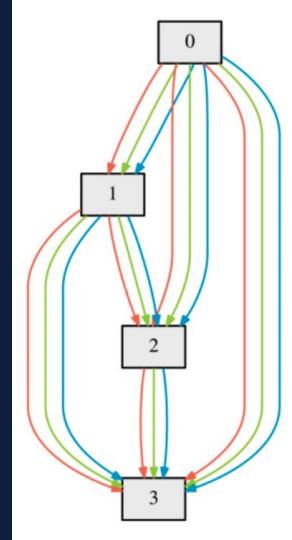
# Roadmap - Part 1

- Motivation and introduction
- Search spaces
  - Macro
  - Cell-based
  - Hierarchical
  - Encodings
- Black-box optimization methods
  - Baselines
  - Bayesian optimization
  - Evolution
- Performance prediction

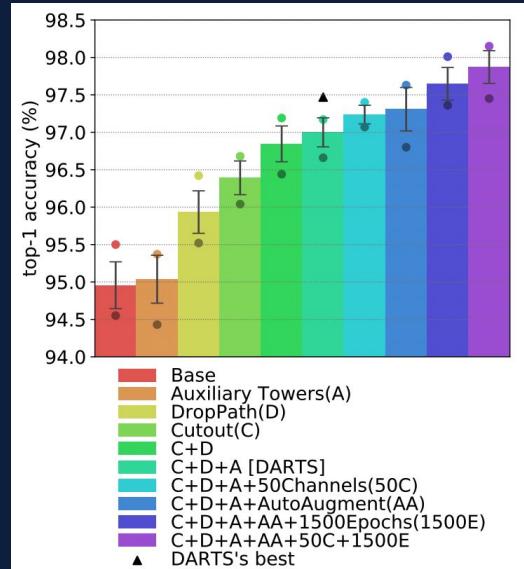
# Search Spaces: Exploration vs. Exploitation



AutoML-Zero (2020)



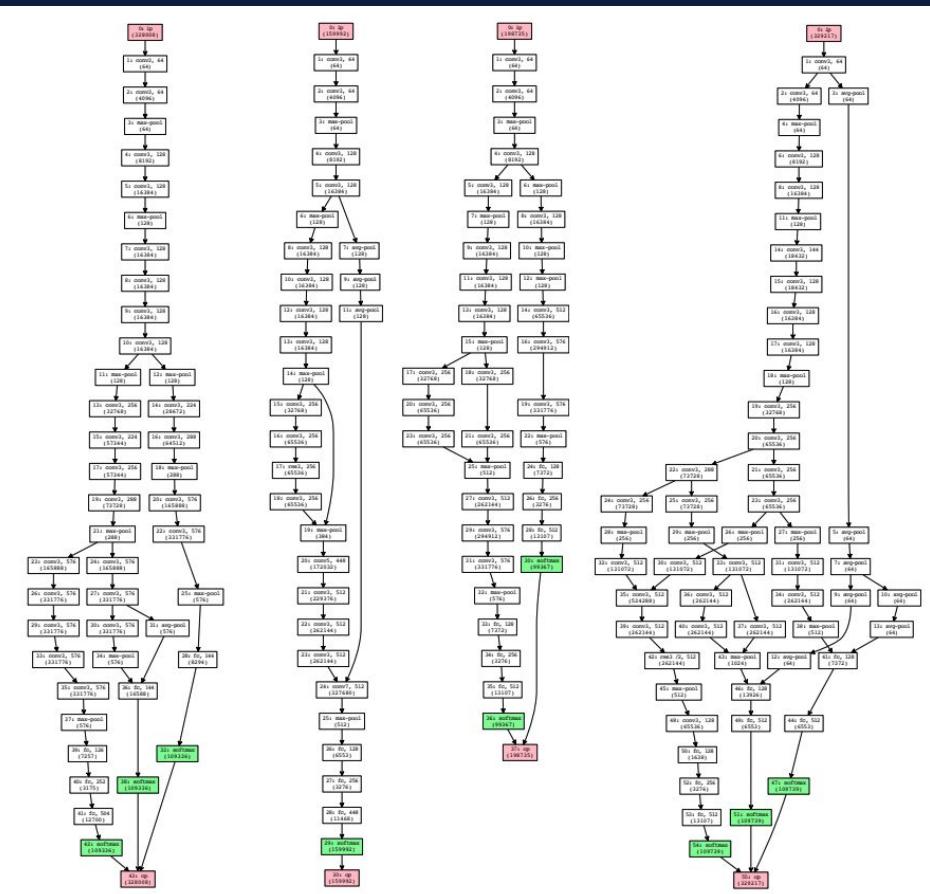
DARTS (2018)



Yang et al. (2019)

# Macro Search Spaces

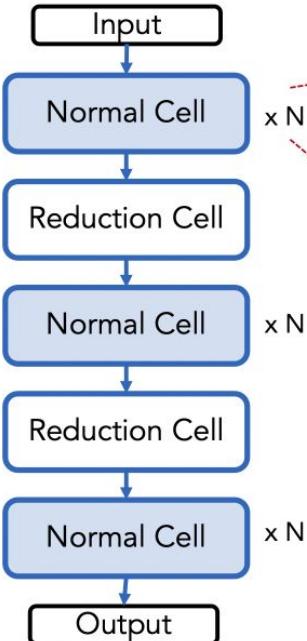
- Define a set of operations
  - Iteratively add more nodes



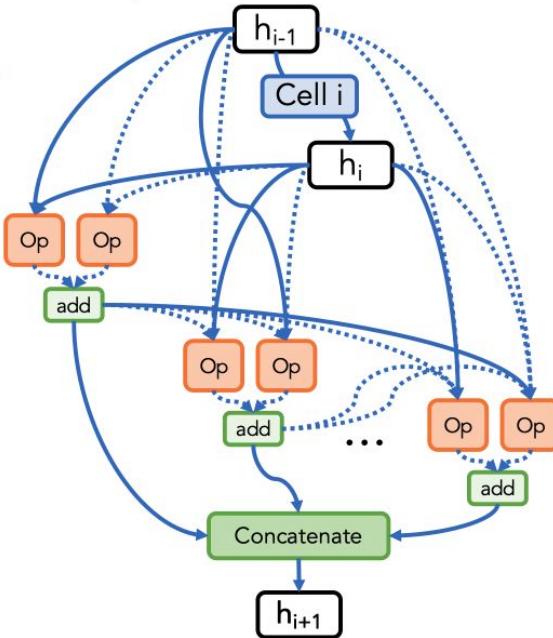
(NASBOT 2018)

# Cell-based search spaces

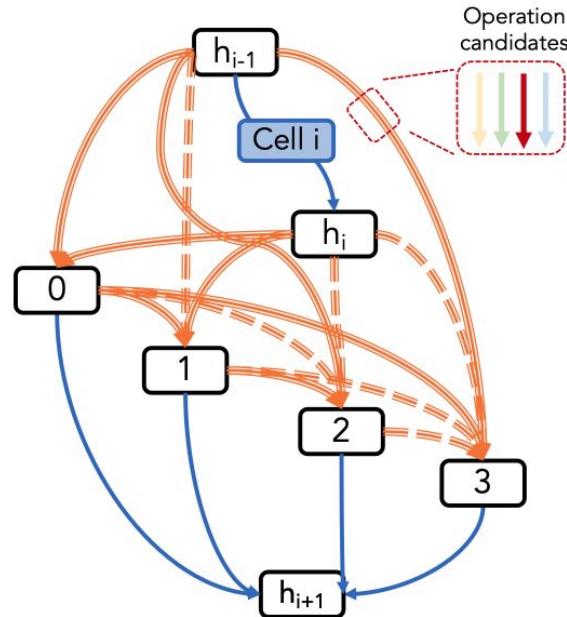
Architecture



NASNet Cell  
(Operations on Nodes)



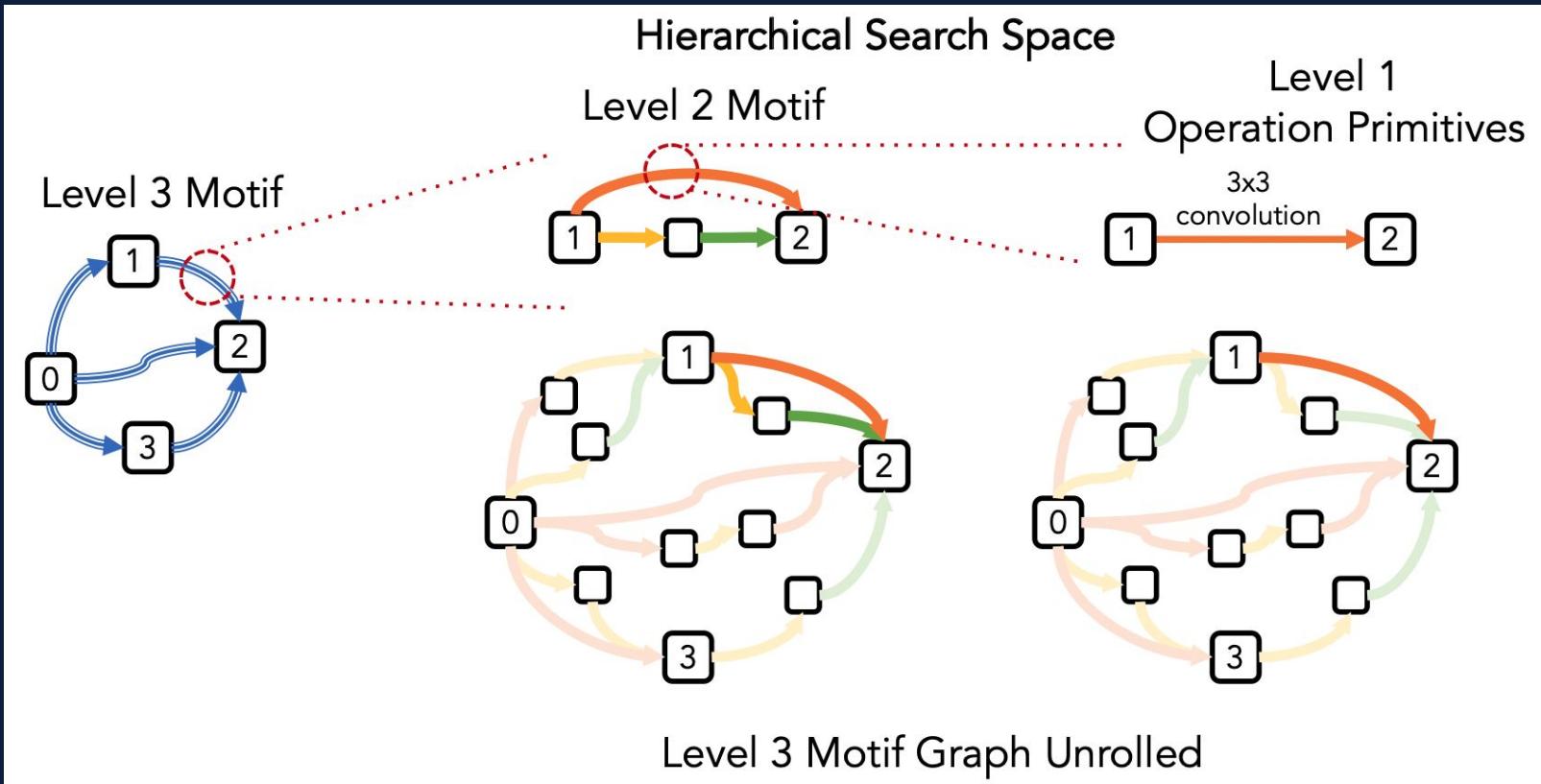
DARTS Cell  
(Operations on Edges)



[NASNet \(2017\)](#)

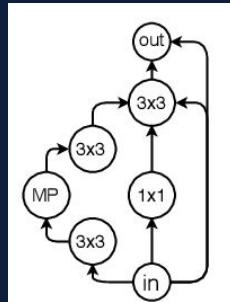
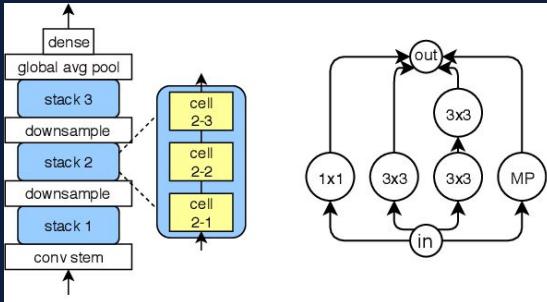
[DARTS \(2018\)](#)

# Hierarchical Search Spaces



# NAS-Bench-101

- Size 423k
- Used to **simulate** NAS experiments
- Allows for more principled research
  - Fixed training pipeline
  - Can run many trials



```
# 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, 1],      # 1x1 conv
            [0, 0, 0, 0, 0, 1],      # 3x3 conv
            [0, 0, 0, 1, 0, 0],      # 5x5 conv (replaced by two 3x3's)
            [0, 0, 0, 0, 0, 1],      # 5x5 conv (replaced by two 3x3's)
            [0, 0, 0, 0, 0, 1],      # 3x3 max-pool
            [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])

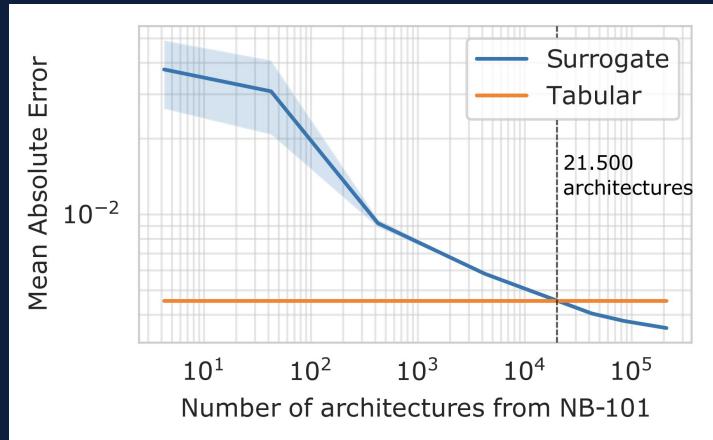
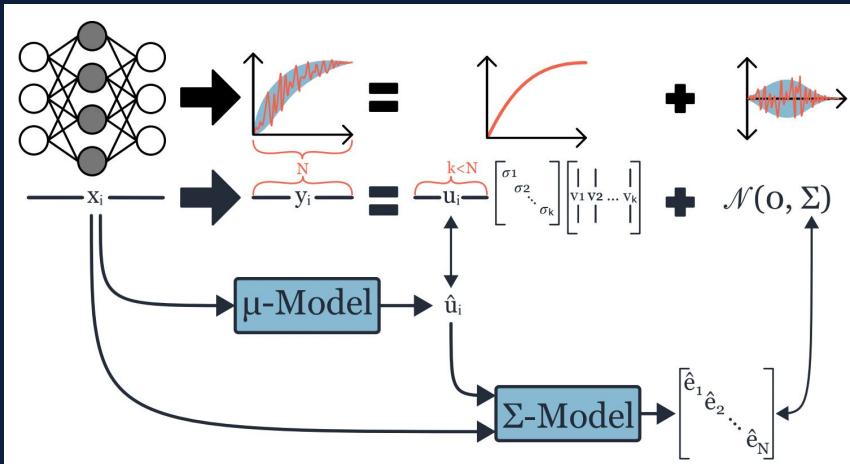
# Query this model from dataset, returns a dictionary containing the metrics
# associated with this model.
data = nasbench.query(model_spec)
```

# NAS Benchmarks

Benchmark	Size	Queryable						#Tasks
		Tab.	Surr.	LCs	Macro	One-Shot	Task	
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

# Surrogate NAS Benchmarks

- Surr-NAS-Bench-DARTS (NAS-Bench-301)
- Surr-NAS-Bench-FBNet
- NAS-Bench-111
- NAS-Bench-311
- NAS-Bench-NLP11



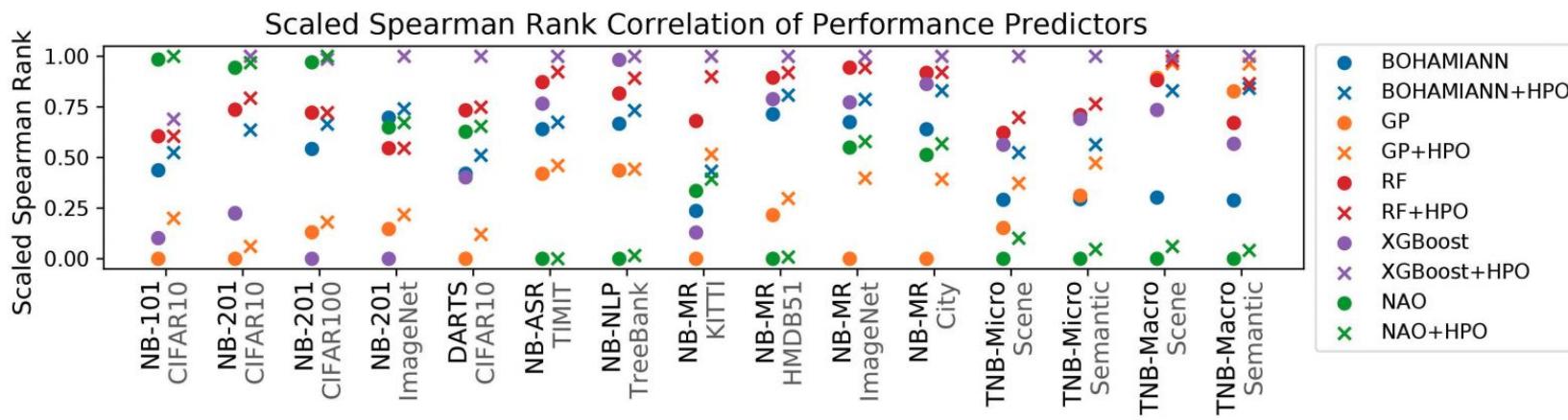
Surr-NAS-Bench (2020)

NAS-Bench-x11 (2021)

# NAS-Bench-Suite (25 tasks)

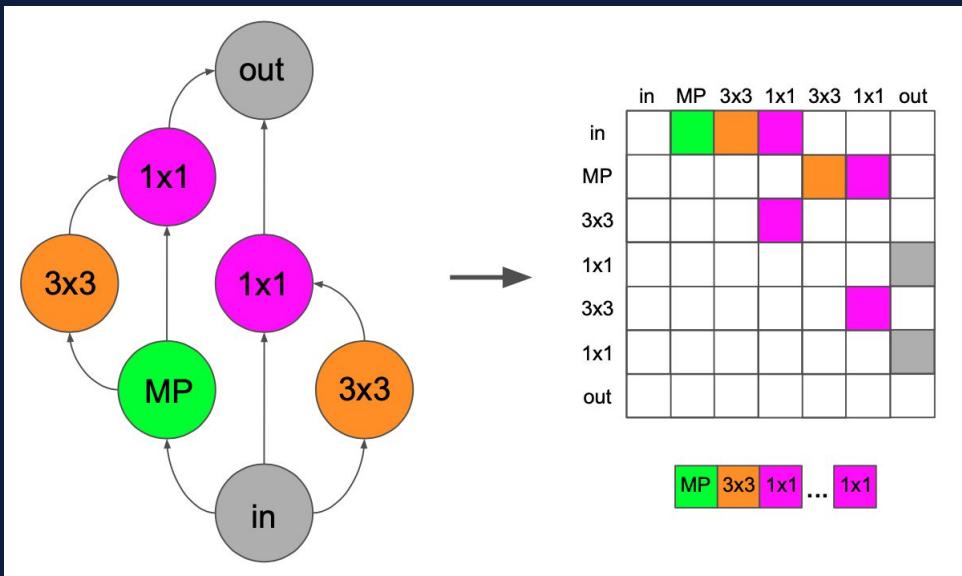
	NAS Algorithms						Performance Predictors			
	RS	RE	BANANAS	LS	NPENAS	GP	BOHAM.	RF	XGB	NAO
Avg.Rank, 101&201	4.50	3.00	3.50	<b>1.50</b>	2.50	4.67	2.83	2.17	4.17	<b>1.17</b>
Avg. Rank, non-101&201	3.06	<b>2.11</b>	2.83	3.13	3.87	4.08	3.06	<b>1.33</b>	2.46	4.08

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



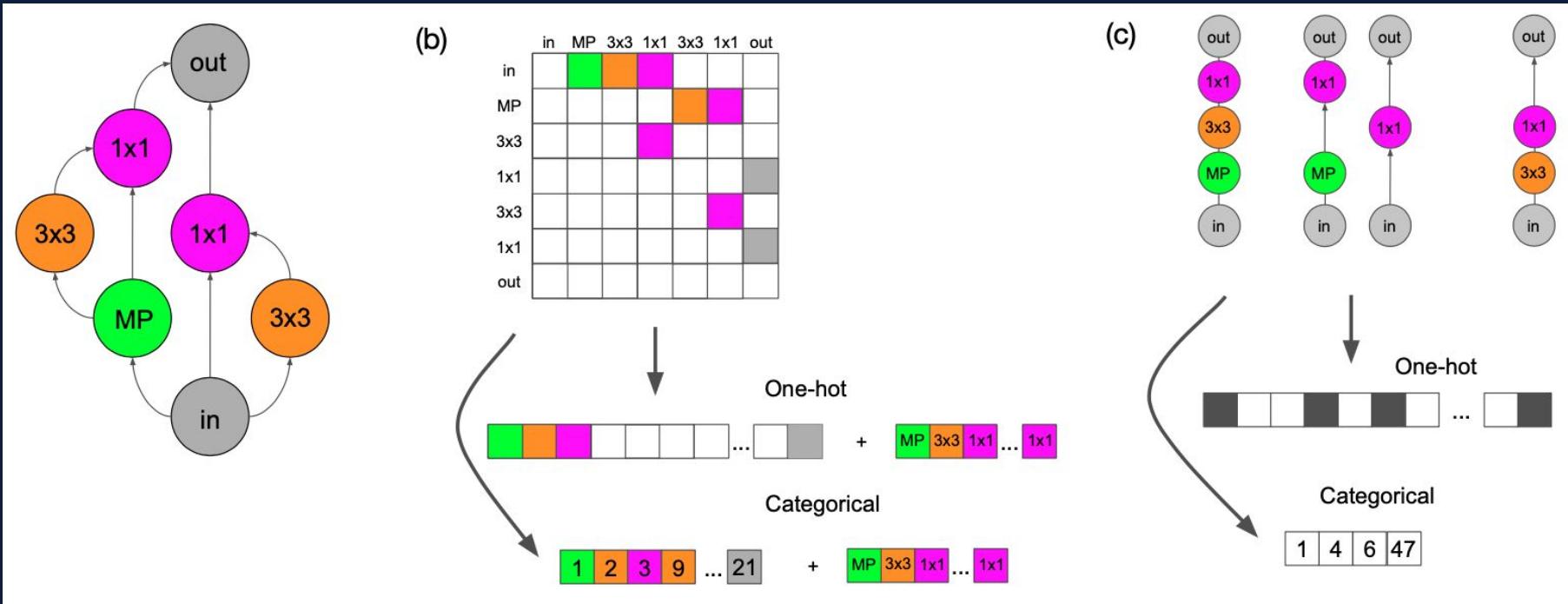
# Encodings for NAS

Most NAS algorithms search over DAG-based architectures, which must be encoded into a tensor



[White et al. \(2020\)](#)

# Encodings for NAS



# NAS encoding-dependent subroutines

Many NAS algos can be composed from three subroutines

- Sample random architecture
- Perturb architecture
- Train predictor model

## Algorithm 1 BANANAS

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

1. Draw  $t_0$  architectures  $a_0, \dots, a_{t_0}$  uniformly at random from  $A$  and train them on  $D$ .
2. For  $t$  from  $t_0$  to  $T$ ,
  - i. Train an ensemble of neural predictors on  $\{(a_0, f(a_0)), \dots, (a_t, f(a_t))\}$  using the path encoding to represent each architecture.
  - ii. Generate a set of  $c$  candidate architectures from  $A$  by randomly mutating the  $x$  architectures  $a$  from  $\{a_0, \dots, a_t\}$  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:**  $a^* = \operatorname{argmin}_{t=0, \dots, T} f(a_t)$ .

## Algorithm 1 Aging Evolution

```
population ← empty queue                                ▷ The population.  
history ← ∅                                              ▷ Will contain all models.  
while |population| < P do                                ▷ Initialize population.  
    model.arch ← RANDOMARCHITECTURE()  
    model.accuracy ← TRAINANDEVAL(model.arch)  
    add model to right of population  
    add model to history  
end while  
while |history| < C do                                    ▷ Evolve for  $C$  cycles.  
    sample ← ∅                                              ▷ Parent candidates.  
    while |sample| < S do  
        candidate ← random element from population  
        ▷ The element stays in the population.  
        add candidate to sample  
    end while  
    parent ← highest-accuracy model in sample  
    child.arch ← MUTATE(parent.arch)  
    child.accuracy ← TRAINANDEVAL(child.arch)  
    add child to right of population  
    add child to history  
    remove dead from left of population                      ▷ Oldest.  
    discard dead  
end while  
return highest-accuracy model in history
```

**Algorithm 1** Bayesian Optimized Neural Architecture Search (BONAS).  $\mathcal{A}$  is the given search space,  $N$  is the number of initial architectures,  $k$  is the ratio of GCN / BLR update times.

```
1: initialize random  $N$  fully-trained architectures:  $\mathcal{D} = \{(A_i, X_i, t_i)\}_{i=1}^N$  from search space  $\mathcal{A}$ ;  
2: initial training of GCN using  $\mathcal{D}$  with proposed loss;  
3: replace the final layer of GCN with BLR;  
4: initialize Sampler;  
5: repeat  
6:   for iteration = 1, 2, ...,  $k$  do  
7:     sample candidate pool  $\mathcal{C}$  from  $\mathcal{A}$ ;  
8:     for each candidate  $m$  in  $\mathcal{C}$  do  
9:       embed  $m$  using GCN;  
10:      compute  $\mu$  and  $\sigma^2$  in (4) and (5) using BLR;  
11:      compute expected improvement (EI) in (2);  
12:    end for  
13:     $M \leftarrow$  candidate with the highest EI score;  
14:    fully train  $M$  to obtain its actual performance;  
15:    add  $M$  and its actual performance to  $\mathcal{D}$ ;  
16:    update BLR using the enlarged  $\mathcal{D}$ ;  
17:    update Sampler;  
18:  end for  
19:  retrain GCN using the enlarged  $\mathcal{D}$  with proposed loss;  
20: until stop criterion satisfy.
```

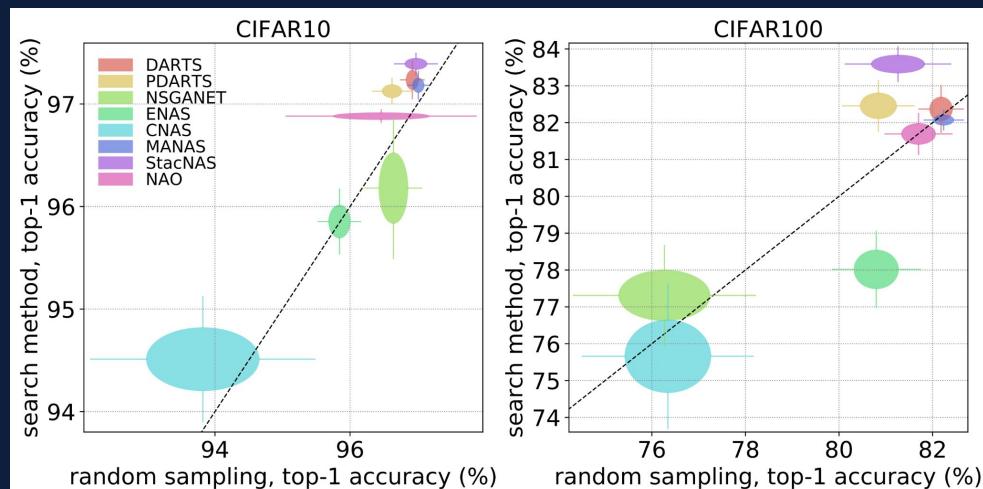
# Roadmap - Part 1

- Motivation and introduction
- Search spaces
  - Macro
  - Cell-based
  - Hierarchical
  - Encodings
- Black-box optimization methods
  - Baselines
  - Bayesian optimization
  - Evolution
- Performance prediction

# Random Search & Random Sampling

Random search is surprisingly competitive [\[Li and Talwalkar, 2019\]](#), [\[Yang et al., 2019\]](#), [\[Sciuto et al., 2020\]](#)

Random sampling:  
performance of a  
randomly drawn  
architecture.



[Yang et al. \(2019\)](#)

# Local Search

- Five lines of code
- Performs surprisingly well on popular benchmarks

---

**Algorithm 1** Local search

---

**Input:** Search space  $A$ , objective function  $\ell$ , neighborhood function  $N$

1. Pick an architecture  $v_1 \in A$  uniformly at random
2. Evaluate  $\ell(v_1)$ ; denote a dummy variable  $\ell(v_0) = \infty$ ; set  $i = 1$
3. While  $\ell(v_i) < \ell(v_{i-1})$  :
  - i. Evaluate  $\ell(u)$  for all  $u \in N(v_i)$
  - ii. Set  $v_{i+1} = \operatorname{argmin}_{u \in N(v_i)} \ell(u)$ ; set  $i = i + 1$

**Output:** Architecture  $v_i$

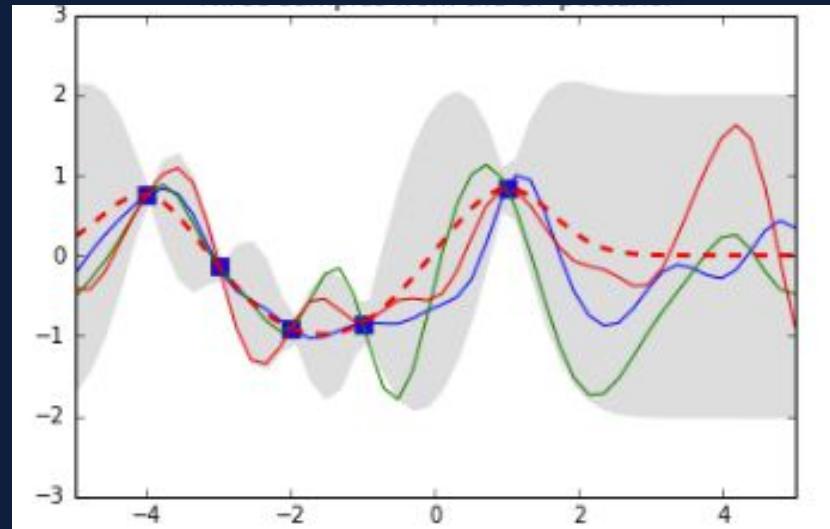
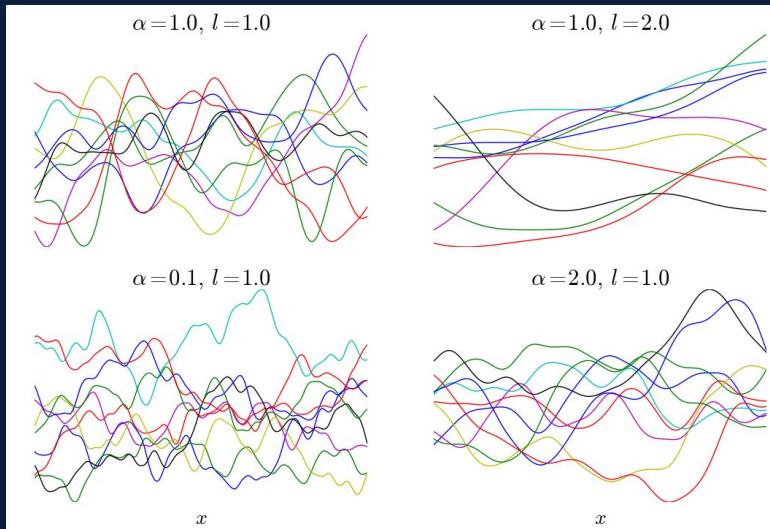
---

[Ottelander et al. 2020]

[White et al. 2020]

# Bayesian optimization

- [NASBOT, 2018], [Auto-Keras, 2018], [NASBOWL, 2020]



Source: <http://keyonvafa.com/gp-tutorial/>,  
<https://katbailey.github.io/post/gaussian-processes-for-dummies/>

# “BO + Neural Predictor” Framework

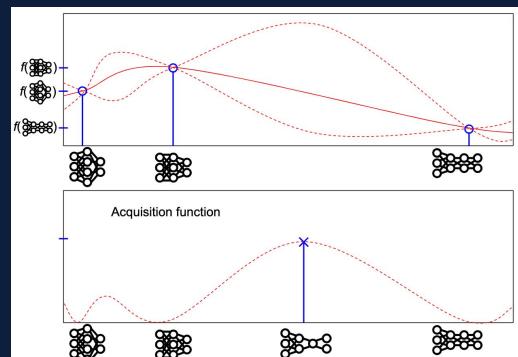
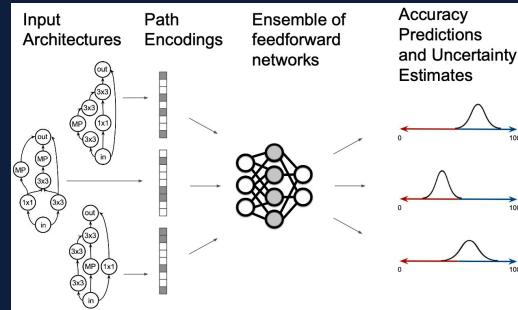
[\[NASGBO, 2019\]](#), [\[BONAS, 2019\]](#), [\[BANANAS, 2021\]](#)

## Algorithm 1 BANANAS

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

1. Draw  $t_0$  architectures  $a_0, \dots, a_{t_0}$  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  $\{(a_0, f(a_0)), \dots, (a_t, f(a_t))\}$ .
  - ii. Generate a set of  $c$  candidate architectures from  $A$  by randomly mutating the  $x$  architectures  $a$  from  $\{a_0, \dots, a_t\}$  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:**  $a^* = \operatorname{argmin}_{t=0, \dots, T} f(a_t)$ .



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  $\phi$ , function  $f(a)$  returning validation error of  $a$  after training.

1. Draw  $t_0$  architectures  $a_0, \dots, a_{t_0}$  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  $\{(a_0, f(a_0)), \dots, (a_t, f(a_t))\}$ .
  - ii. Generate a set of  $c$  candidate architectures from  $A$  by randomly mutating the  $x$  architectures  $a$  from  $\{a_0, \dots, a_t\}$  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:**  $a^* = \operatorname{argmin}_{t=0, \dots, T} f(a_t)$ .

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

# BANANAS

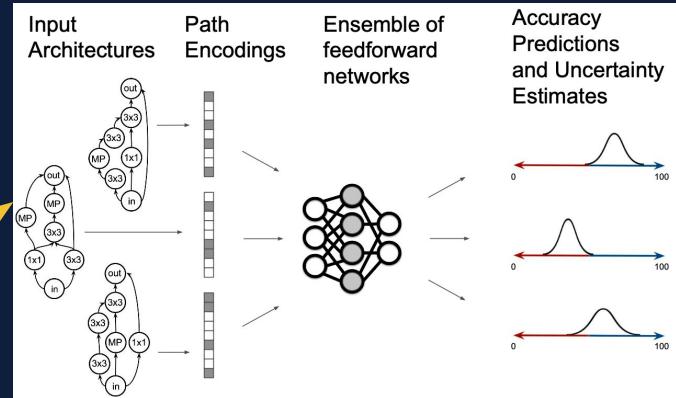
[BANANAS, 2021]

## Algorithm 1 BANANAS

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

1. Draw  $t_0$  architectures  $a_0, \dots, a_{t_0}$  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  $\{(a_0, f(a_0)), \dots, (a_t, f(a_t))\}$ .
  - ii. Generate a set of  $c$  candidate architectures from  $A$  by randomly mutating the  $x$  architectures  $a$  from  $\{a_0, \dots, a_t\}$  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:**  $a^* = \operatorname{argmin}_{t=0, \dots, T} f(a_t)$ .



Path encoding, ensemble

Small mutations

Independent Thompson Sampling

# Evolution

---

**Algorithm 1** General Evolutionary Algorithm

---

**Input:** Initial population of architecture data  $\mathcal{D}_0 = (a_i, y_i)_{i=1}^{N_0}$ , objective function  $f$ , total number of evolution steps  $T$

**Output:** The optimal architecture  $a_T^*$

**for**  $t = 1, \dots, T$  **do**

    Sample a set of parent architectures  $\mathcal{S}_{parents} = \{(a_j, y_j)\}_{j=1}^{N_p}$  from the population  $\mathcal{D}_{t-1}$

    Generate children architectures by mutating parent architectures and evaluate their performance  
    to obtain  $\mathcal{S}_{children} = \{(a_k, y_k)\}_{k=1}^{N_c}$

    Update the population with  $\mathcal{S}_{children}$  to obtain  $\mathcal{D}_t$

**end for**

---

Decisions: sampling the initial population, selecting the parents,  
generating the offspring

[Real et al. \(2018\)](#)

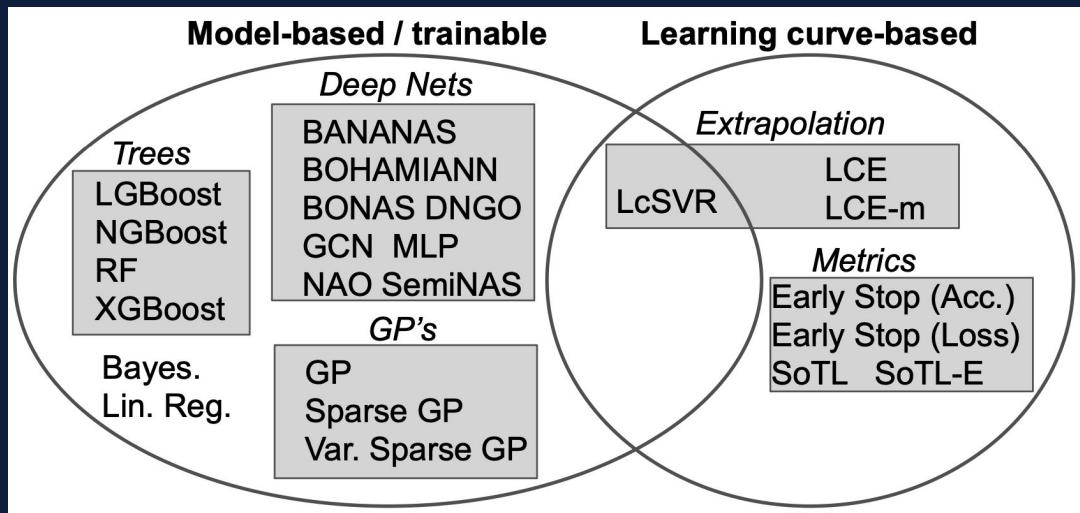
# Roadmap - Part 1

- Motivation and introduction
- Search spaces
  - Macro
  - Cell-based
  - Hierarchical
  - Encodings
- Black-box optimization methods
  - Baselines
  - Bayesian optimization
  - Evolution
- Performance prediction

# Performance Predictors

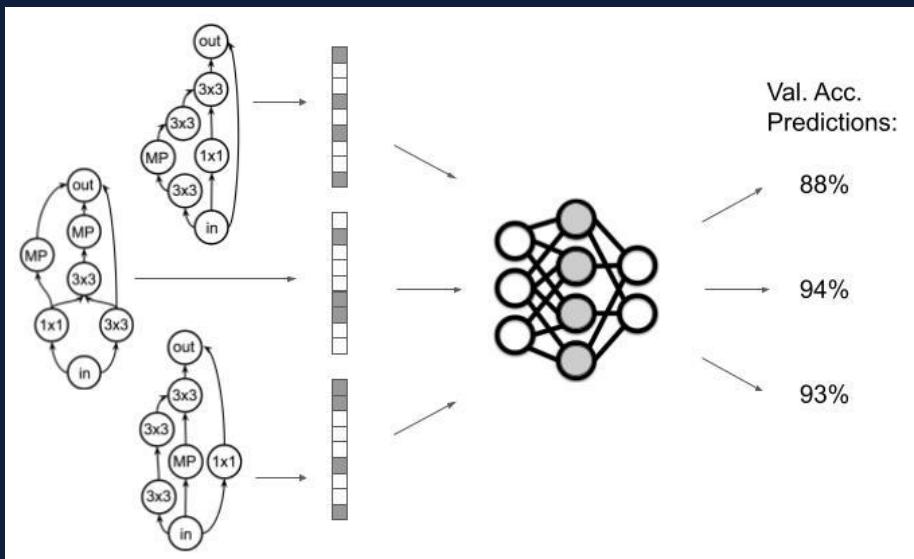
Any technique which predicts the (relative) accuracy of an architecture, without fully training it.

- **Initialization:** performs any necessary pre-computation
- **Query:** take any architecture as input, and output predicted accuracy



# Model-Based Predictors

- Supervised learning - regression
  - X - the architecture encoding (e.g. one-hot adjacency matrix)
  - Y - validation accuracy of trained architecture



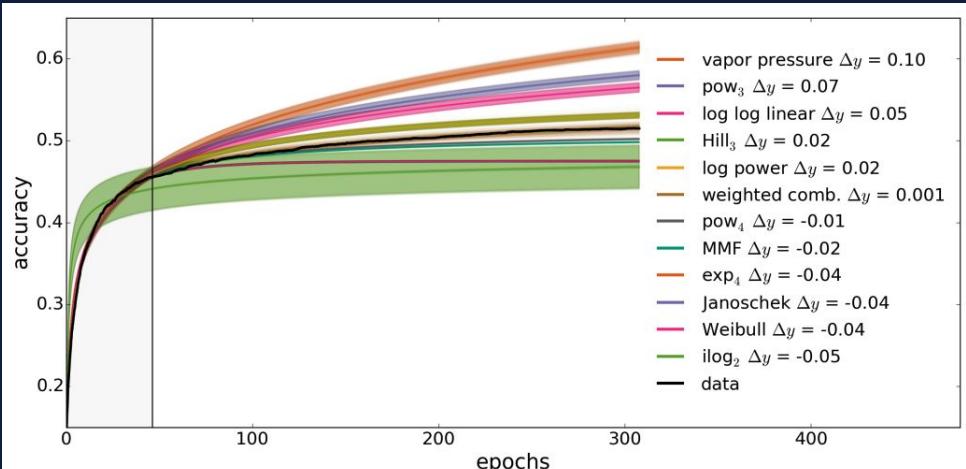
[\[White et al. 2019\]](#)

- 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**

# Learning curve based predictors

- Learning curve extrapolation
  - Fit partial learning curve to parametric model [\[Domhan et al. 2015\]](#)
  - Bayesian NN [\[Klein et al. 2017\]](#)
- Training statistics
  - Early stopping (val acc) [\[Elsken et al. 2018\]](#)
  - Sum of training losses [\[Ru et al. 2020\]](#)



[\[Elsken et al. 2018\]](#)

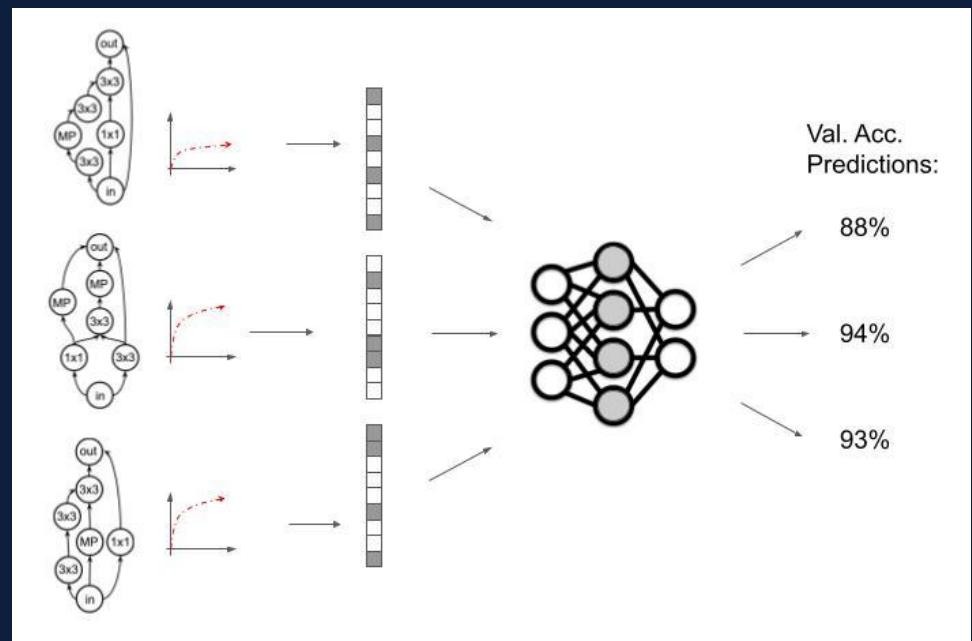
No init time, high query time

# Hybrid model-based + LC predictors

Train a model, using partial learning curve + hyperparams, to predict final accuracy

- First and second derivatives as features, SVR [\[Baker et al. 2017\]](#)
- Full LC as features, Bayesian NN [\[Klein et al. 2017\]](#)

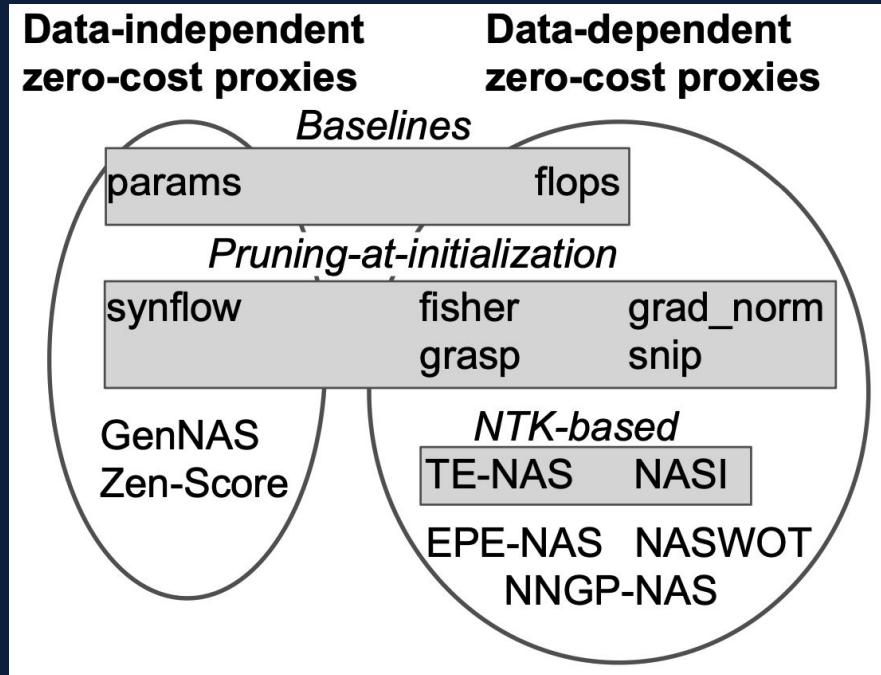
**High init time, high query time**



# “Zero-cost” proxies

Compute a statistic of an architecture in 3-5 seconds

- Jacobian covariance [\[Mellor et al. 2020\]](#)
- Synaptic Flow [\[Abdelfattah et al. 2021\]](#)
  - SNIP [\[Lee et al. 2018\]](#)



**Low init time, low query time**

[\[Abdelfattah et al. 2021\]](#)

$$\text{snip} : \mathcal{S}_p(\theta) = \left| \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta \right|, \quad \text{grasp} : \mathcal{S}_p(\theta) = -\left( H \frac{\partial \mathcal{L}}{\partial \theta} \right) \odot \theta, \quad \text{synflow} : \mathcal{S}_p(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta$$

# OMNI: The Omnipotent Predictor

---

**Algorithm 1** OMNI predictor

---

**Input:** Search space  $A$ , dataset  $D$ , initialization time budget  $B_{\text{init}}$ , query time budget  $B_{\text{query}}$ .

**Initialization():**

- $\mathcal{D}_{\text{train}} \leftarrow \emptyset$
- While  $t < B_{\text{init}}$ 
  - Draw an architecture  $a$  randomly from  $A$
  - Train  $a$  to completion to compute val. accuracy  $y_a$
  - $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(a, y_a)\}$
- Train an NGBoost model  $m$  to predict the final val. accuracy of architectures from  $\mathcal{D}_{\text{train}}$ , using the architecture encoding, SoTL-E, and Jacob. cov. as input features.

**Query(architecture  $a_{\text{test}}$ ):**

- While  $t < B_{\text{query}}$ , train  $a_{\text{test}}$
  - Compute SoTL-E using the partial learning curve, and compute Jacob. cov., and the arch. encoding of  $a_{\text{test}}$
  - Predict val. acc. of  $a_{\text{test}}$  using  $m$  and the above features.
-

Thanks! Questions?