# Automated Machine Learning (AutoML): A Tutorial


**Matthias Feurer**

University of Freiburg
feurerm@cs.uni-freiburg.de
@__mfeurer__

**Frank Hutter**

University of Freiburg & Bosch
fh@cs.uni-freiburg.de
@FrankRHutter

@automlfreiburg


Tutorial based on Chapters 1-3 of the book *Automated Machine Learning*
Slides and video available at automl.org/events/tutorials
(all references are clickable links)

# Outline

## Part 1: General AutoML (Matthias)

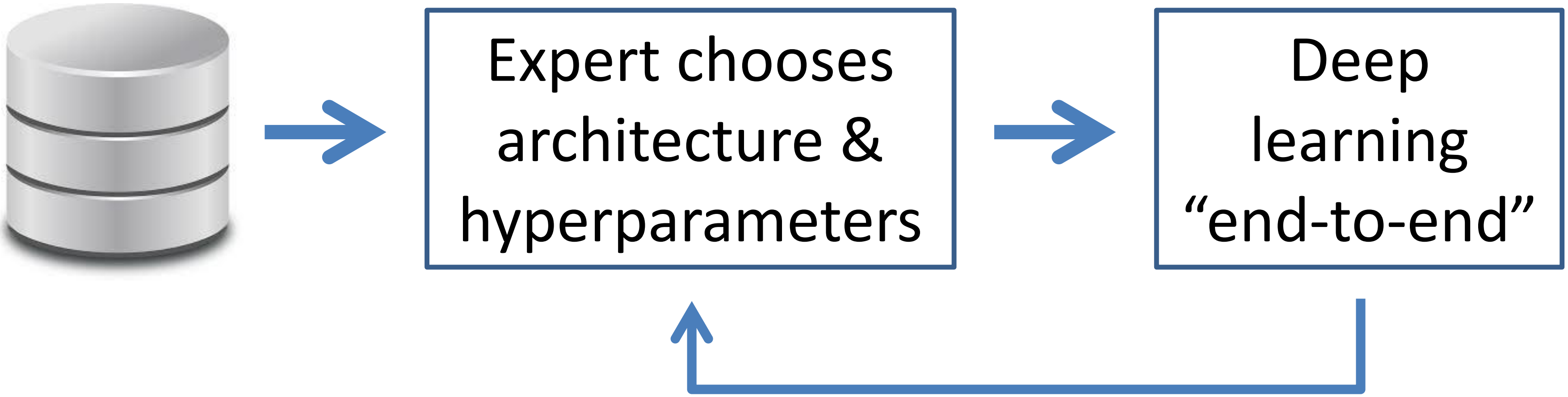
1. AutoML by Hyperparameter Optimization
2. Black-box Hyperparameter Optimization
3. Beyond black-box optimization
4. Examples of AutoML
5. Wrap-up & Conclusion

## Part 2: Neural Architecture Search (Frank)

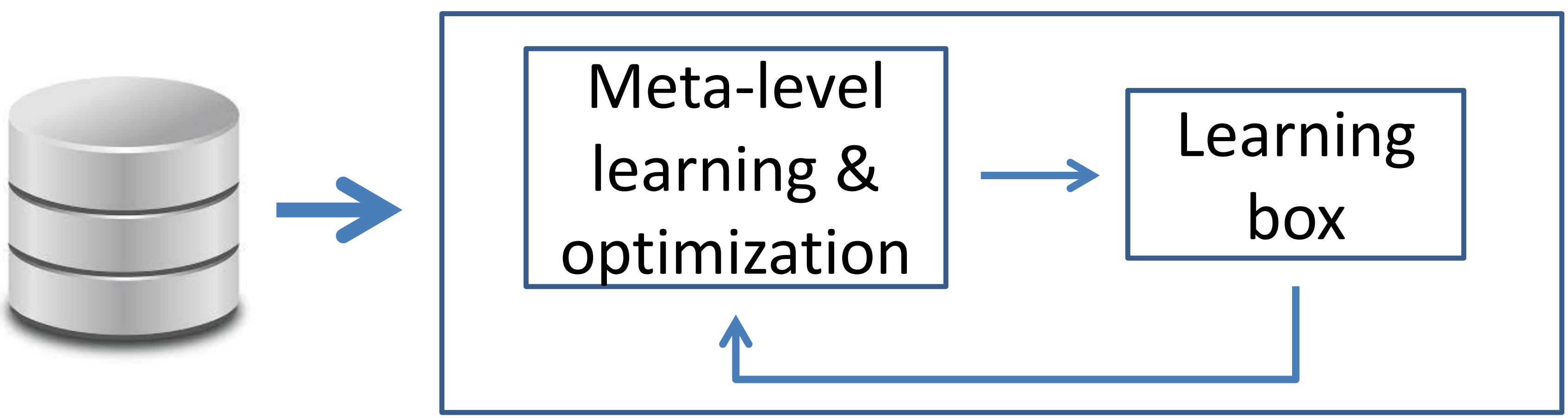
1. Search Spaces
2. Black-box Optimization
3. Beyond Black-box Optimization
4. Best Practices

# Deep Learning and AutoML

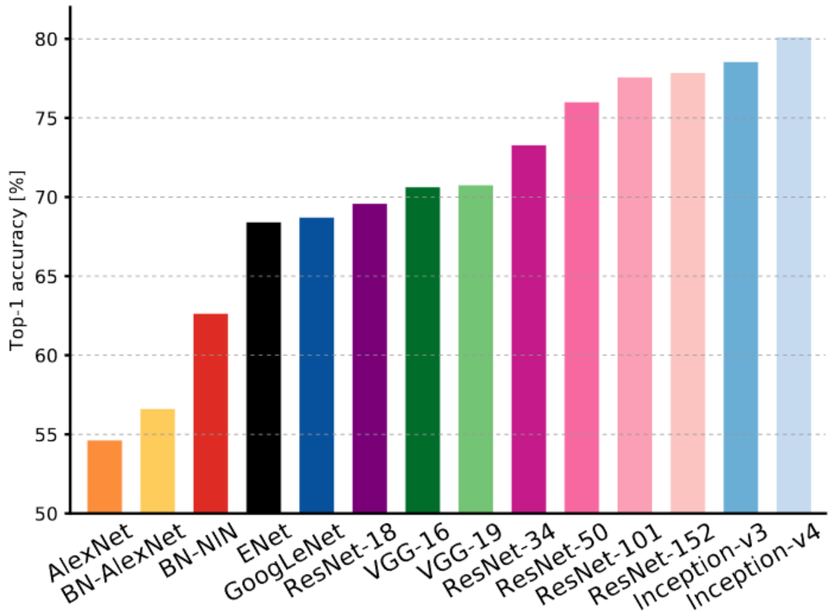
## Current deep learning practice

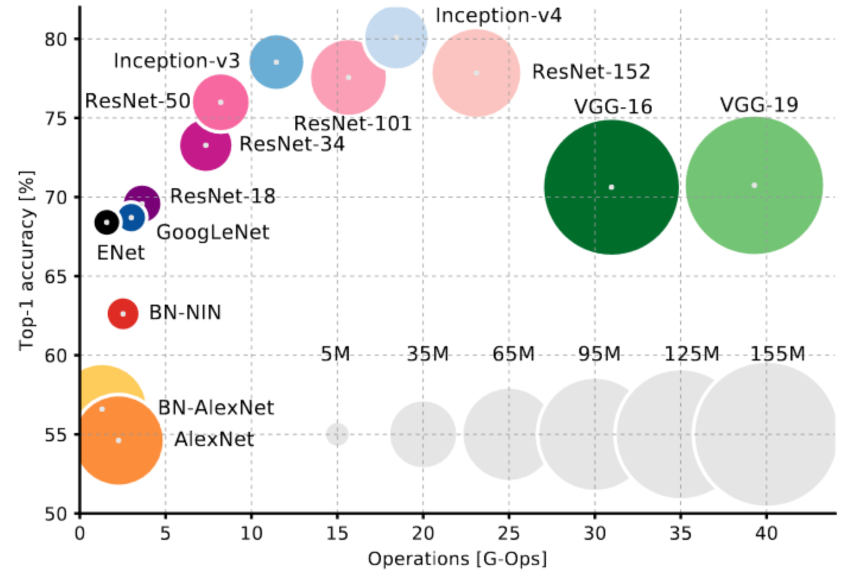



## AutoML: true end-to-end learning



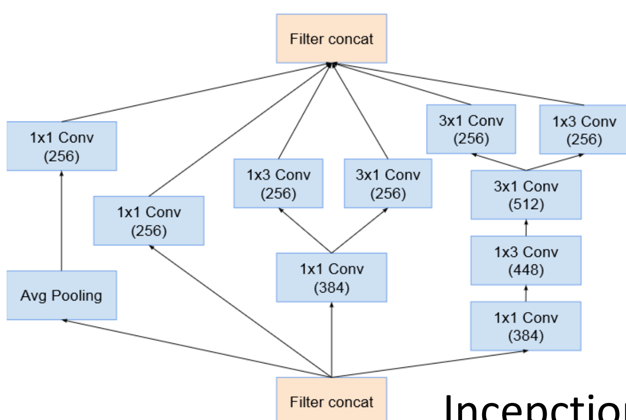
# Neural Architecture Search - Motivation

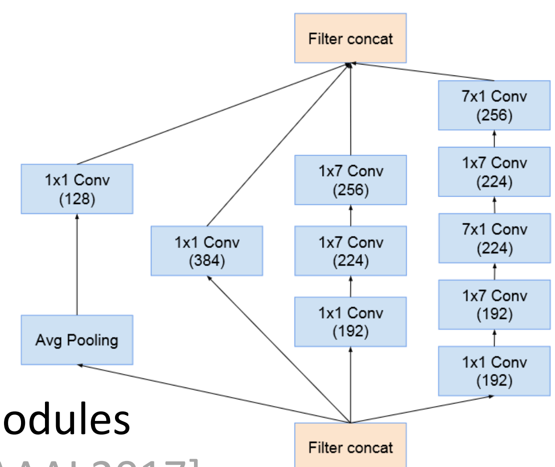



[Canziani et al., preprint 2017]

Bigger,
more complex
architectures...



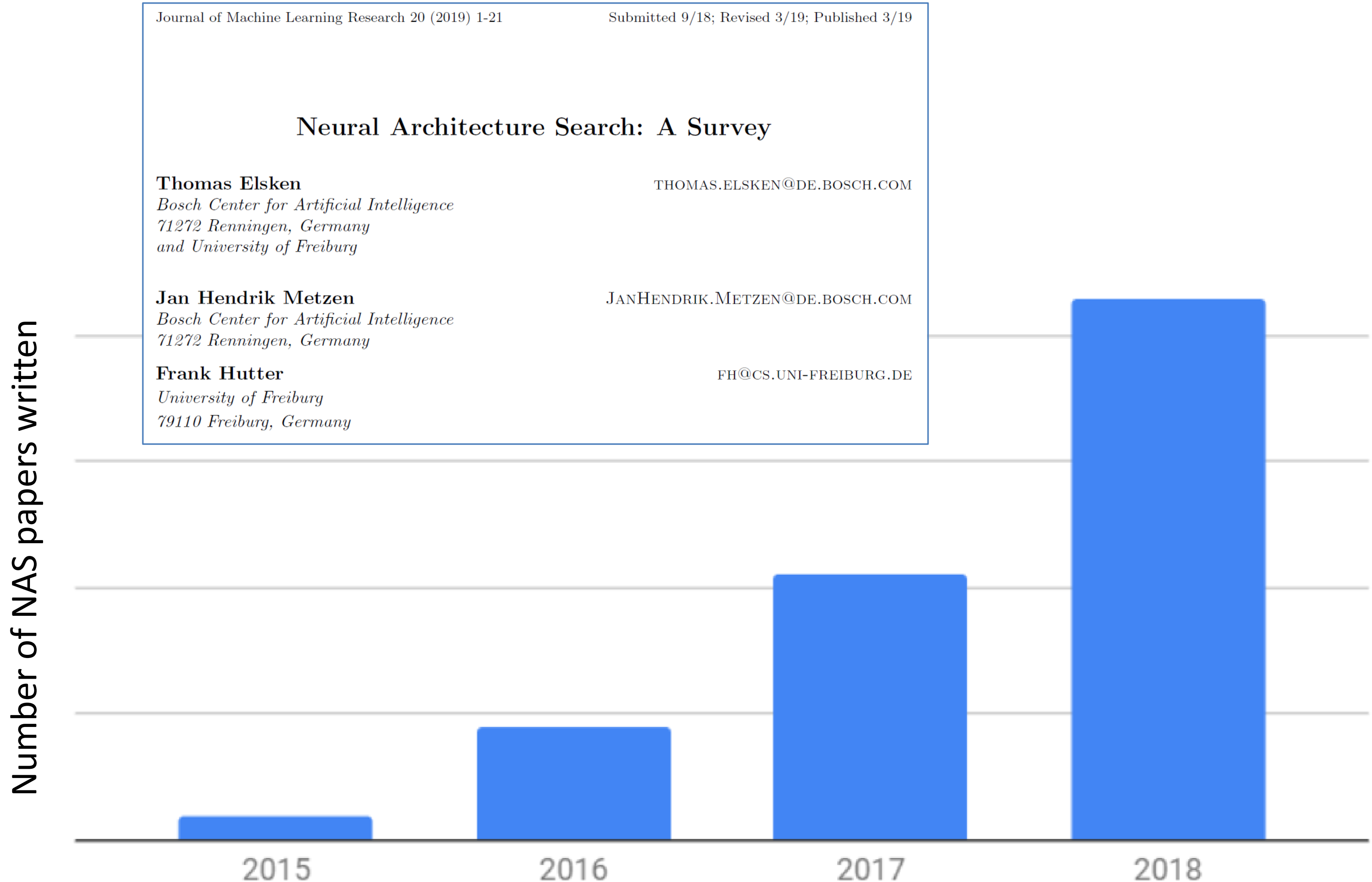

Incepction-V4 modules

[Szegedy et al., AAAI 2017]

Slide courtesy of Nikhil Naik

# Can we automatically design neural network architectures?

# Yes, we can – and NAS has become a hot topic!



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*

# Outline

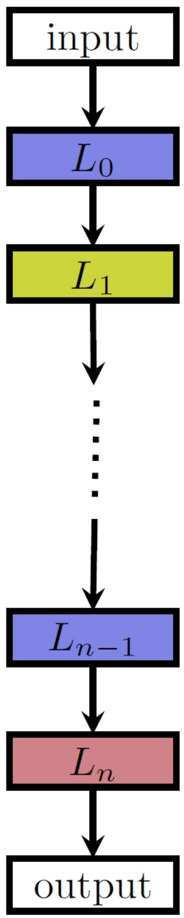
## Part 2: Neural Architecture Search

1. Search Spaces
2. Black-box Optimization
3. Beyond Black-box Optimization
4. Best Practices

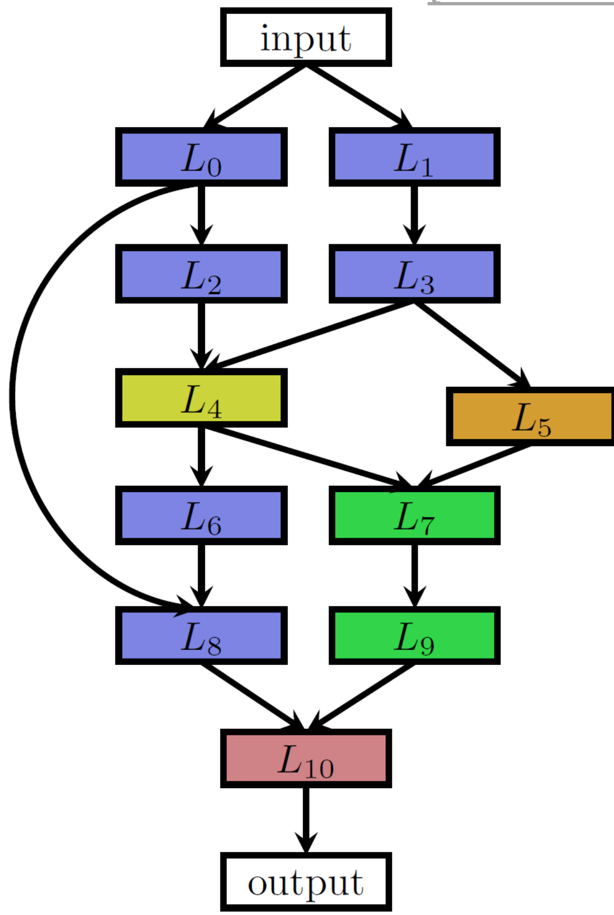
Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, JMLR 2019; also Chapter 3 of the AutoML book]

# Basic Neural Architecture Search Spaces

[Elsken et al., JMLR 2019]




Chain-structured space
(different colours:
different layer types)




More complex space
with multiple branches
and skip connections

# Cell Search Spaces

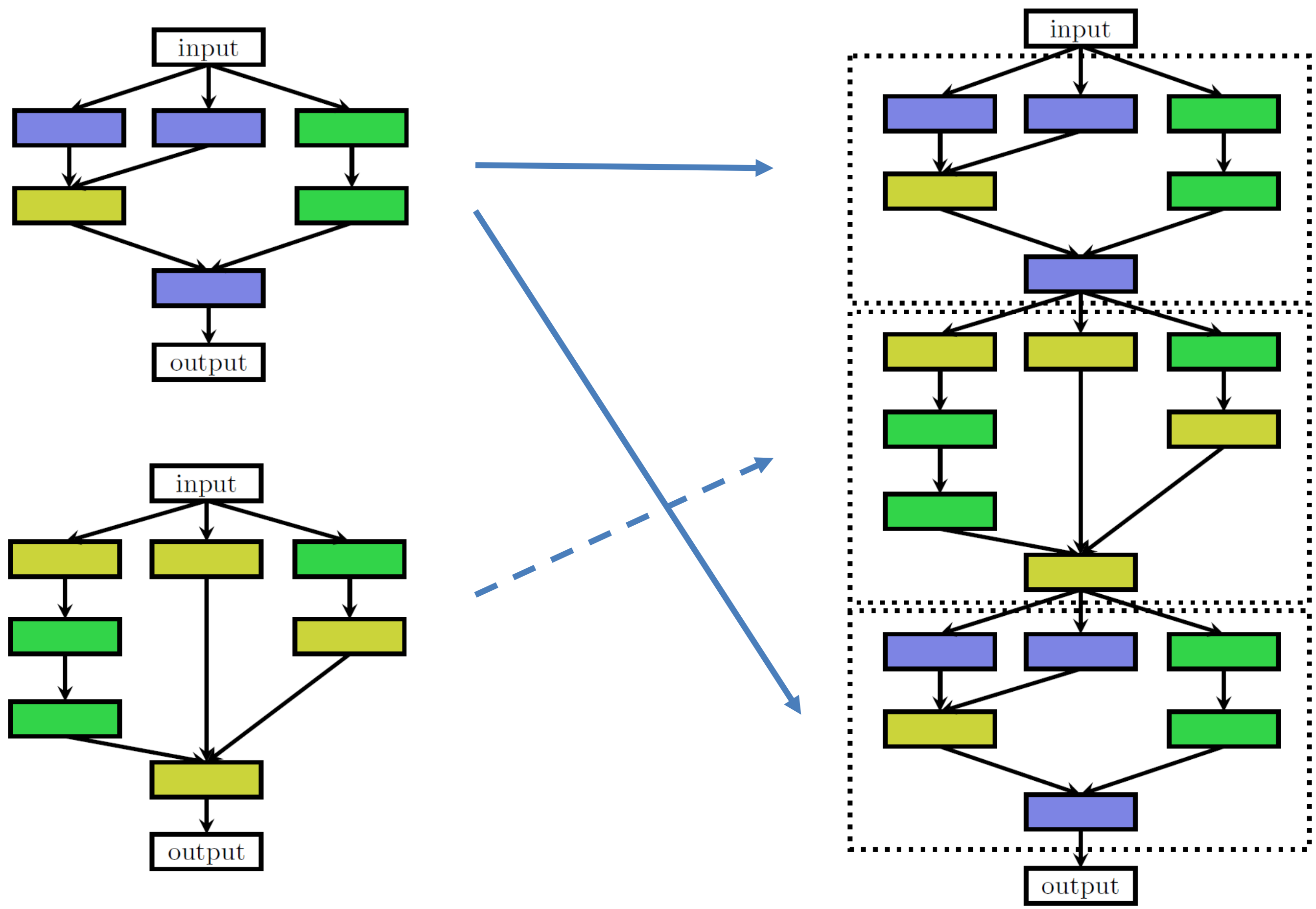
Introduced by Zoph et al. [CVPR 2018]

Two possible cells

Architecture composed of stacking together individual cells

normal cell: preserves spatial resolution

reduction cell: reduces spatial resolution



# Outline

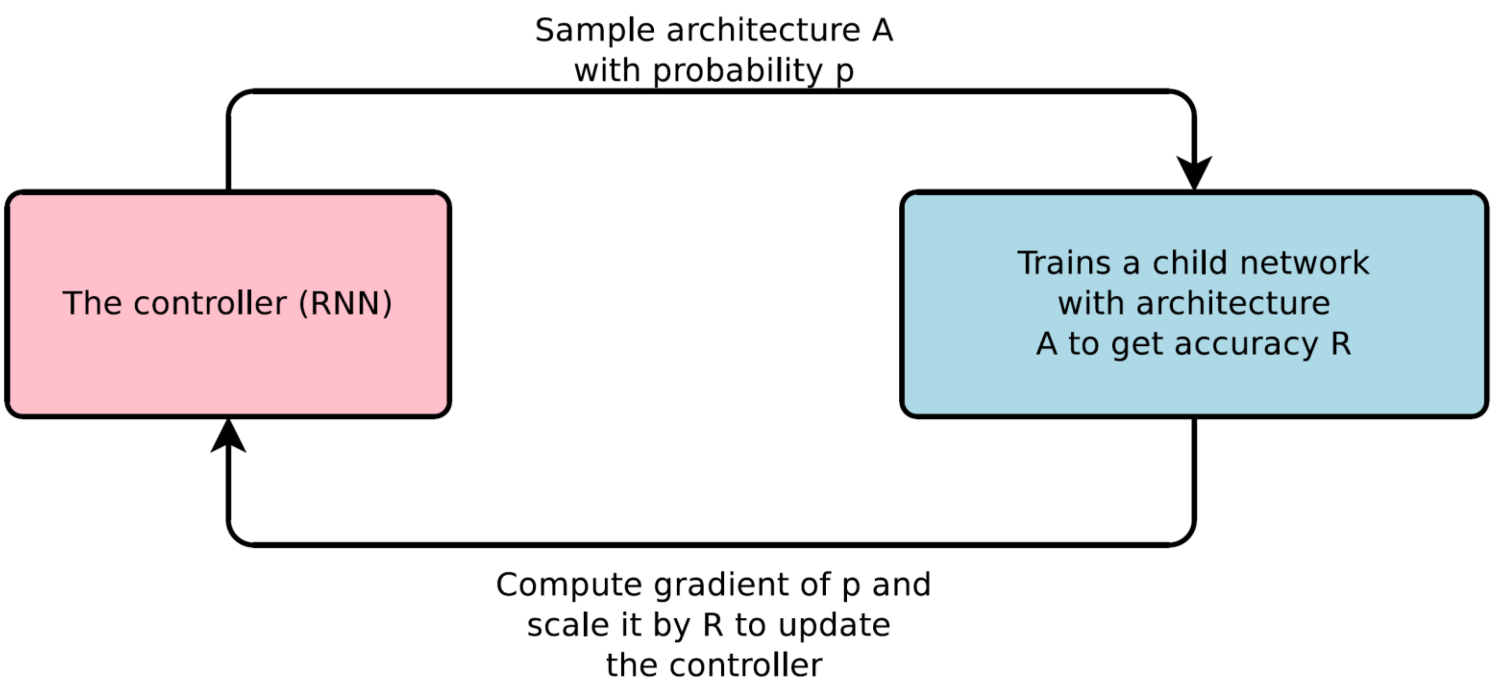
## Part 2: Neural Architecture Search

1. Search Spaces
2. Black-box Optimization
3. Beyond Black-box Optimization
4. Best Practices

Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, JMLR 2019; also Chapter 3 of the AutoML book]

# NAS with Reinforcement Learning

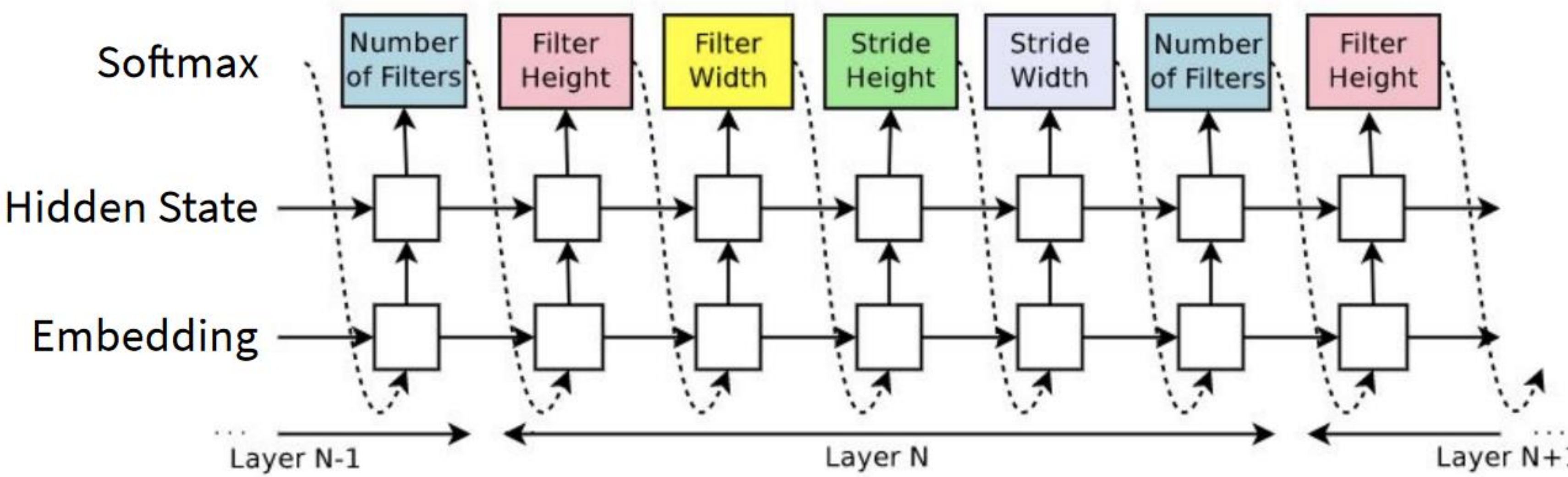
- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands:
    **800 GPUs for 3-4 weeks, 12.800 architectures trained**



# NAS with Reinforcement Learning

[Zoph & Le, ICLR 2017]

- Architecture of neural network represented as string
  e.g., [“filter height: 5”, “filter width: 3”, “# of filters: 24”]
- Controller (RNN) generates string that represents architecture



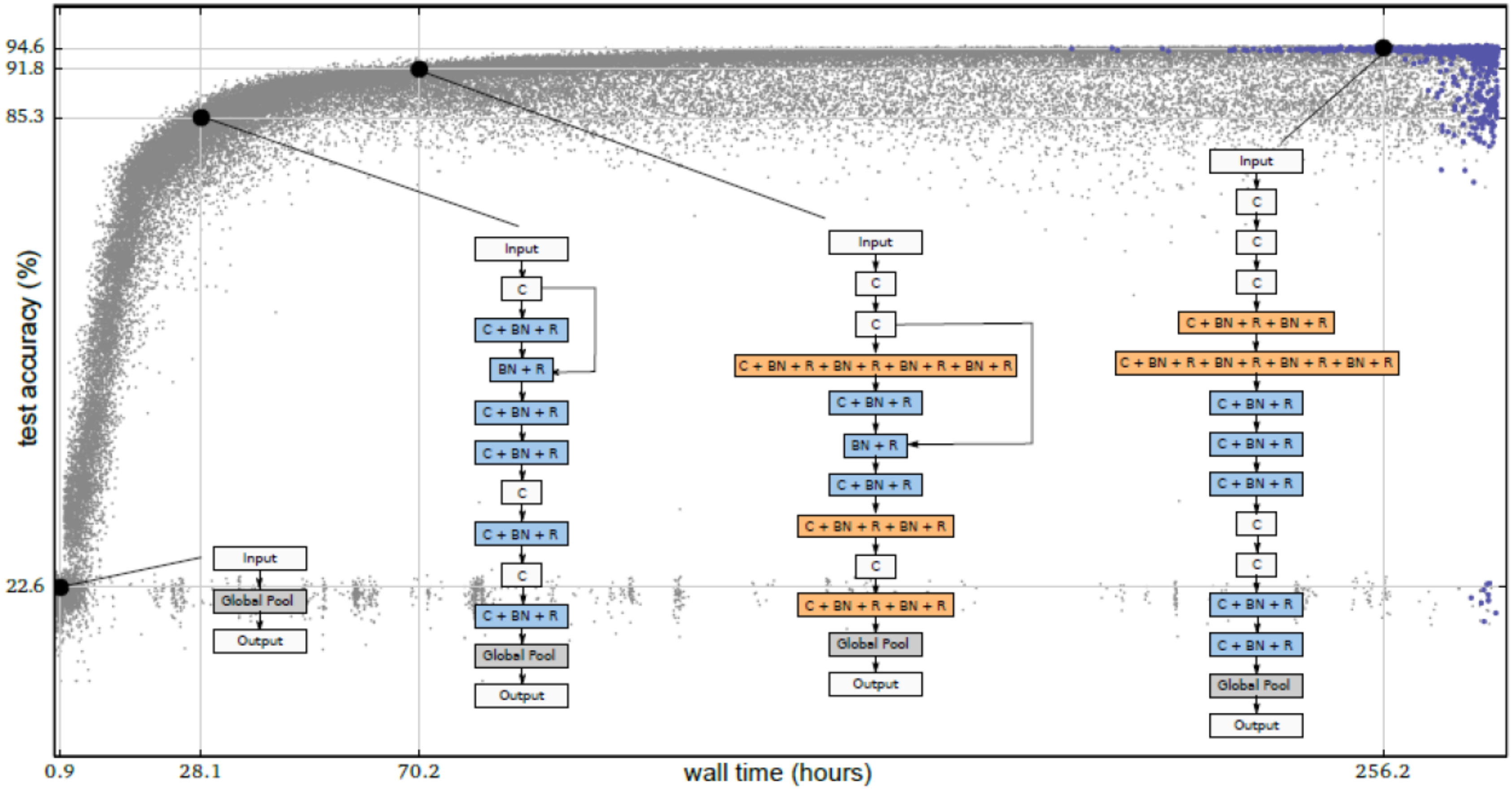
# NAS as Hyperparameter Optimization

[Zoph & Le, ICLR 2017]

- Architecture of neural network represented as string
  e.g., [“filter height: 5”, “filter width: 3”, “# of filters: 24”]
- We can simply treat these as categorical parameters
  - E.g., 25 cat. parameters for each of the 2 cells in Zoph et al [CVPR 2018]

# NAS with Evolution

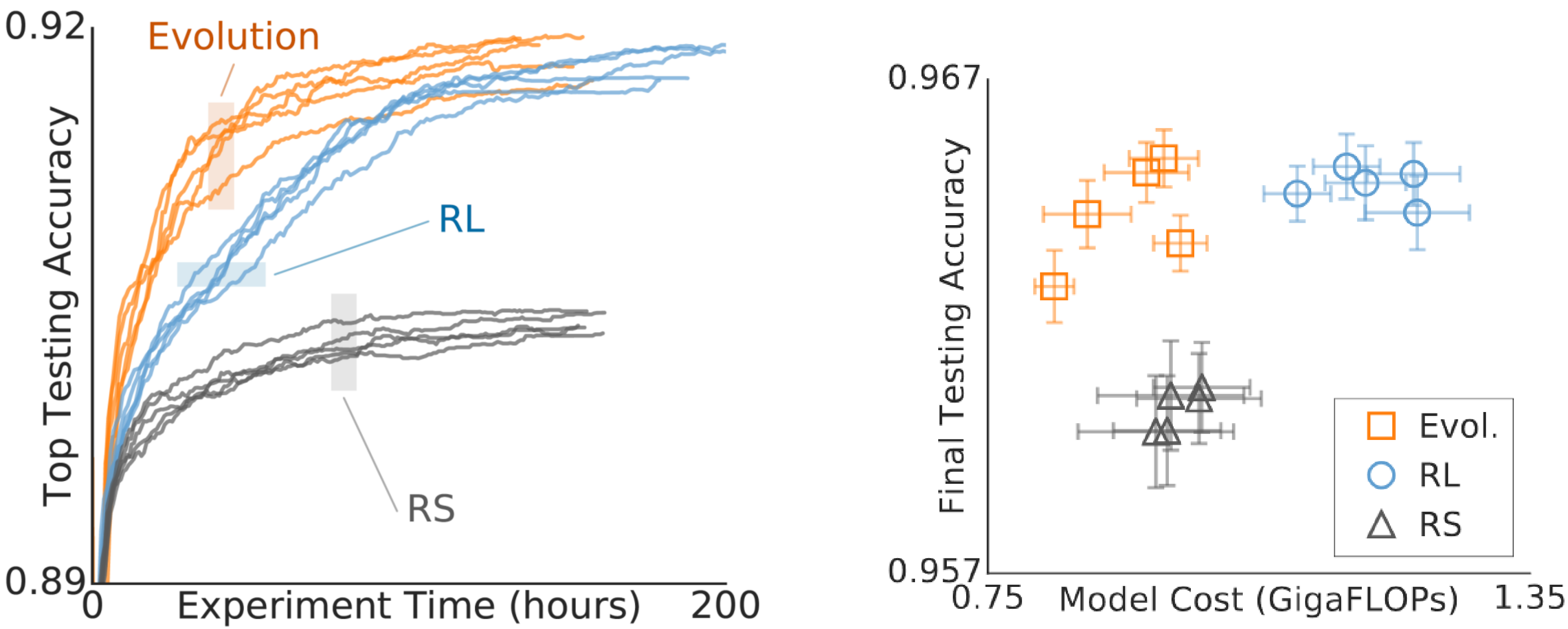
- Neuroevolution
  (already since the 1990s [Angeline et al., 1994; Stanley and Miikkulainen, 2002])
  - Mutation steps, such as adding, changing or removing a layer
  [Real et al., ICML 2017; Miikkulainen et al., arXiv 2017]



# RL vs. Evolution vs. Random Search

## during architecture search

## final evaluation




[Real et al., AAAI 2019]

# Bayesian Optimization

- Joint optimization of a vision architecture with 238 hyperparameters with TPE [Bergstra et al, ICML 2013]
- Auto-Net
  - Joint architecture and hyperparameter search with SMAC
  - First Auto-DL system to win a competition dataset against human experts [Mendoza et al, AutoML 2016]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al, BayesOpt 2013]
  - NASBOT [Kandasamy et al, NIPS 2018]
- Sequential model-based optimization
  - PNAS [Liu et al, ECCV 2018]

# Blackbox methods require a lot of compute (CIFAR-10)

| | Reference | Error (%) | Params (Millions) | GPU Days |
|---|---|---|---|---|
| RL | Zoph and Le (2017) | 3.65 | 37.4 | 22,400 |
| RL | Zoph et al. (2018) | 3.41 | 3.3 | 2,000 |
| EA | Real et al. (2017) | 5.40 | 5.4 | 2,600 |
| EA | Real et al. (2019) | 3.34 | 3.2 | 3,150 |

Going to cell search space

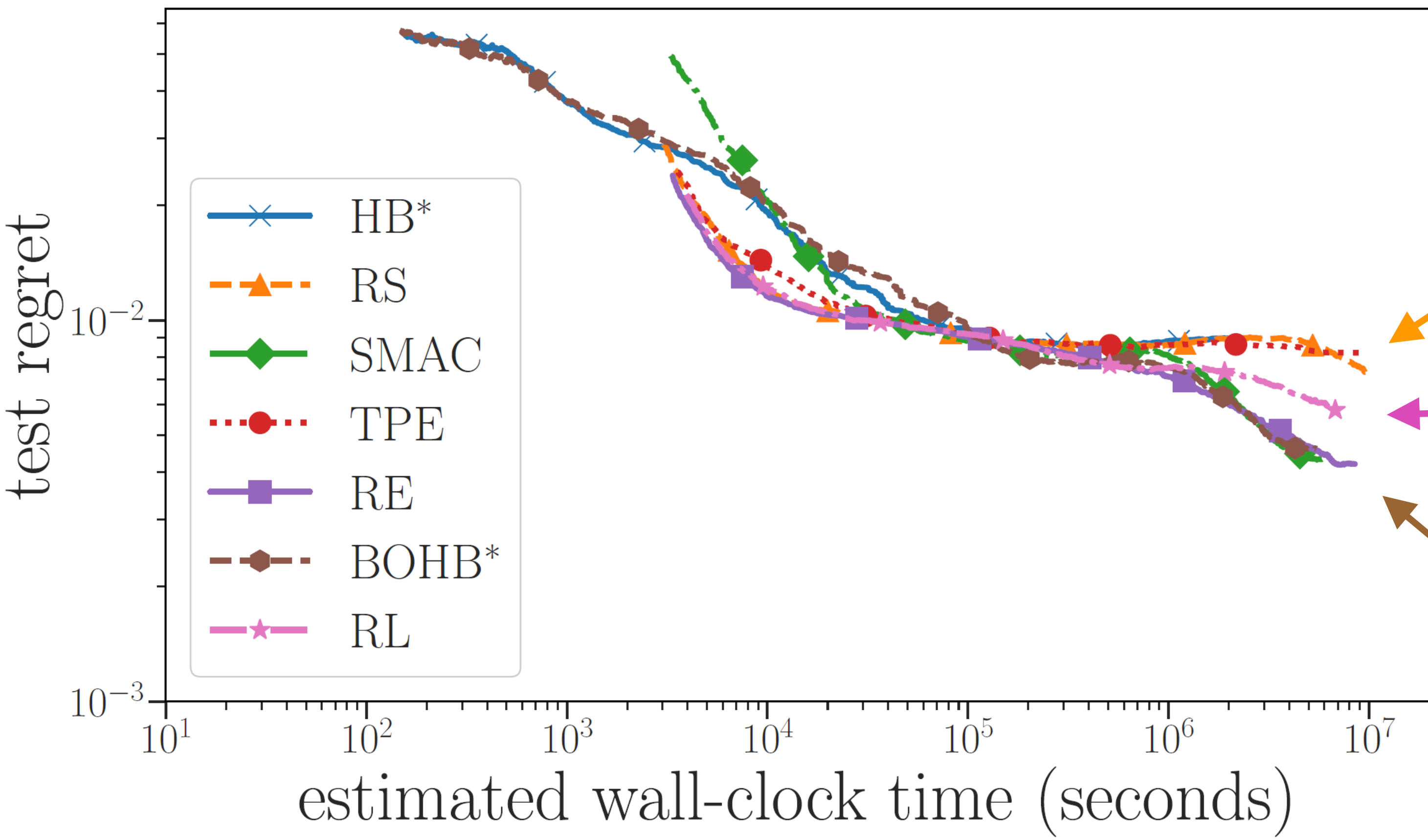
[Wistuba et al., preprint 2019]

# Democratizing NAS Research: NAS-Bench-101

[Ying et al., ICML 2019]

- Table with exhaustive evaluations of a small search space
  - Enables evaluating a NAS method in minutes on a laptop
  - Enables proper scientific research:
    multiple runs, robustness studies, etc
  - Fair head-to-head evaluations by design
    (fixed final evaluation pipeline & hyperparameters)
  - Of course, source code and scripts are available
- 423k cell architectures evaluated on CIFAR-10
  - Only possible with Google resources (4.000 TPUs for months)
  - One-time cost already far more than amortized

# NAS-Bench-101: Comparison of Optimizers




Random search

Reinforcement Learning

Regularized Evolution & Bayesian Optimization (SMAC / BOHB)

Still, blackbox optimization is expensive! Can we do better?

# Outline

## Part 2: Neural Architecture Search

1. Search Spaces
2. Black-box Optimization
3. Beyond Black-box Optimization
4. Best Practices

Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, JMLR 2019; also Chapter 3 of the AutoML book]

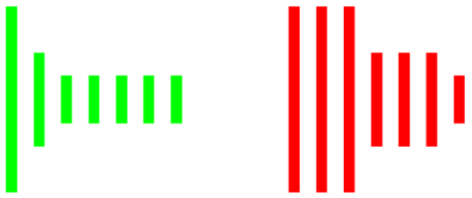
# Main approaches for making NAS efficient

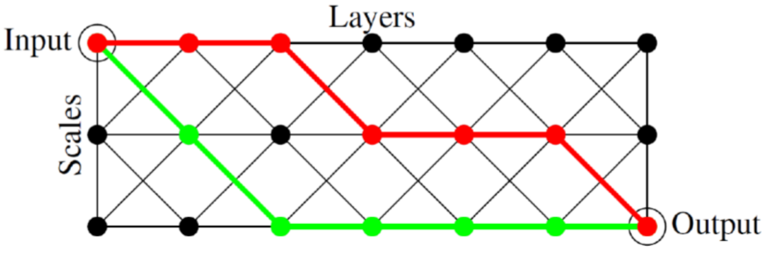
- Multi-fidelity optimization
  [Zela et al, AutoML 2018, Runge et al, MetaLearn 2018]

- Meta-learning [Wong et al, NeurIPS 2018]

- Weight inheritance & network morphisms

- Weight sharing & one-shot models





# Weight inheritance & network morphisms

- Network morphisms [Chen et al., 2016; Wei et al., 2016]
  - Change the network structure, but not the modelled function (i.e., for every input, the network yields the same output as before applying the network morphism)

  - Can use this in NAS algorithms as operations to generate new networks
  - Avoids costly training from scratch

# Network morphism example

We have trained a network

$$N_1(x) = Softmax_{w_{1,1}} \circ ReLU \circ Conv_{w_{1,2}}(x)$$

More details in our blog post.

… and want to add another Relu-Conv block

$$N_2(x) = Softmax_{w_{2,1}} \boxed{\circ ReLU \circ Conv_{w_{2,2}}} \circ ReLU \circ Conv_{w_{2,3}}(x)$$

copy

$$w_{2,1} = w_{1,1}, \quad w_{2,3} = w_{1,2}$$

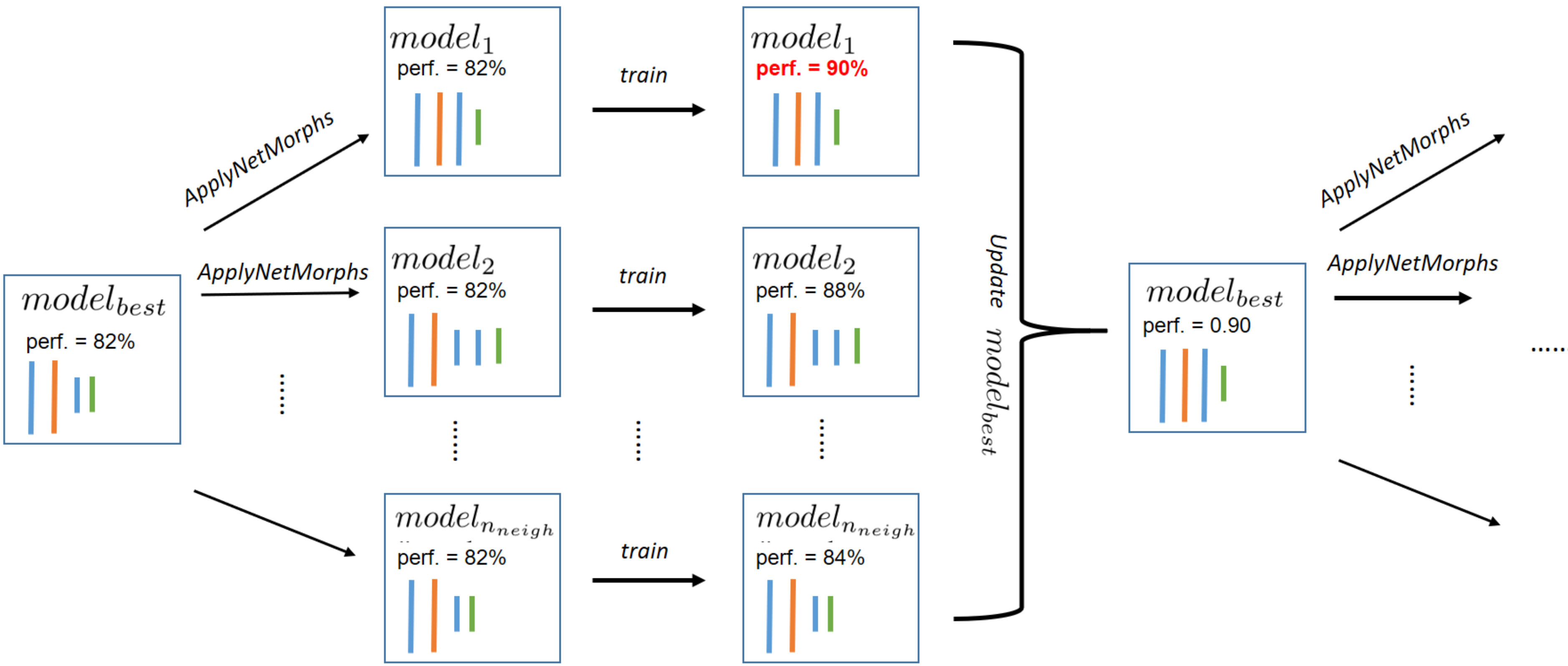
and set $w_{2,2}$ so that $Conv_{w_{2,2}}(x) = x$

Then:

$$\begin{aligned}
N_2(x) &= Softmax_{w_{2,1}} \circ ReLU \circ Conv_{w_{2,2}} \circ ReLU \circ Conv_{w_{2,3}}(x) \\
&= Softmax_{w_{1,1}} \circ ReLU \circ Conv_{w_{2,2}} \circ ReLU \circ Conv_{w_{1,2}}(x) \quad \text{// copy weights} \\
&= Softmax_{w_{1,1}} \circ ReLU \circ Id \circ ReLU \circ Conv_{w_{1,2}}(x) \quad \text{// chose } w_{2,2} \text{ to be Id} \\
&= Softmax_{w_{1,1}} \circ ReLU \circ Conv_{w_{1,2}}(x) \quad \text{// ReLU is idempotent} \\
&= N_1(x)
\end{aligned}$$

# Weight inheritance & network morphisms in NAS

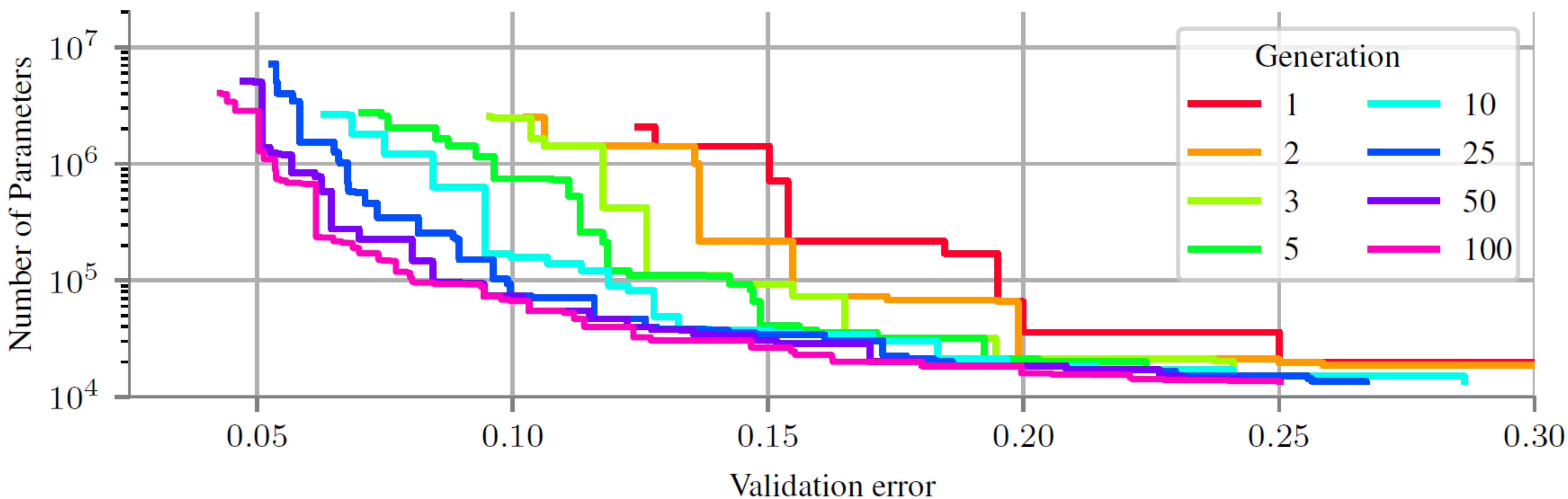
[Cai et al, AAAI 2018; Elsken et al, NeurIPS MetaLearn 2017; Cortes et al, ICML 2017; Cai et al, ICML 2018; Elsken et al, ICLR 2019]




$\rightarrow$ enables efficient architecture search

# Designing Efficient Architectures

- Multi-objective NAS: LEMONADE [Elsken et al., ICLR 2019, blog post]
  - Multi-objective evolutionary method
  - Objectives such as accuracy, # parameters, # flops, latency
  - Outputs Pareto-front wrt. multiple objectives
  - No need to specify tradeoff between objectives a-priori



# Some numbers (CIFAR-10)

| | Reference | Error (%) | Params (Millions) | GPU Days |
|---|---|---|---|---|
| RL | Baker et al. (2017) | 6.92 | 11.18 | 100 |
| | Zoph and Le (2017) | 3.65 | 37.4 | 22,400 |
| | Cai et al. (2018a) | 4.23 | 23.4 | 10 |
| | Zoph et al. (2018) | 3.41 | 3.3 | 2,000 |
| | Zoph et al. (2018) + Cutout | 2.65 | 3.3 | 2,000 |
| | Zhong et al. (2018) | 3.54 | 39.8 | 96 |
| | Cai et al. (2018b) | 2.99 | 5.7 | 200 |
| | Cai et al. (2018b) + Cutout | 2.49 | 5.7 | 200 |
| EA | Real et al. (2017) | 5.40 | 5.4 | 2,600 |
| | Xie and Yuille (2017) | 5.39 | N/A | 17 |
| | Suganuma et al. (2017) | 5.98 | 1.7 | 14.9 |
| | Liu et al. (2018b) | 3.75 | 15.7 | 300 |
| | Real et al. (2019) | 3.34 | 3.2 | 3,150 |
| | Elsken et al. (2018) | 5.2 | 19.7 | 1 |
| | Wistuba (2018a) + Cutout | 3.57 | 5.8 | 0.5 |

NAS with weight inher. / network morphisms

[Wistuba et al., preprint 2019]

# Weight Sharing & One-shot Models

- Embed architectures from search space into single network, the *„one-shot model“*
- Each path through the one-shot model is an architecture
- Only need a single training of the one-shot model
- Weights are shared across architectures embedded in one-shot model

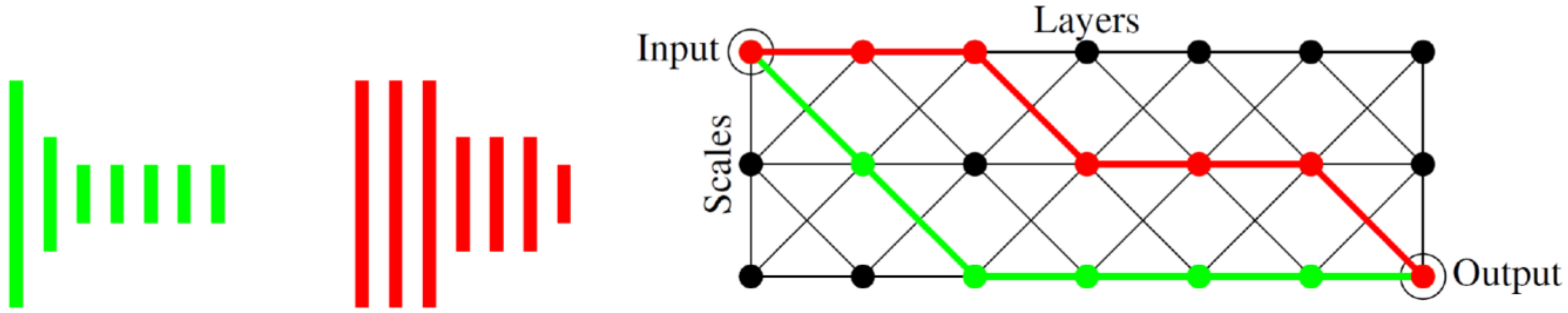
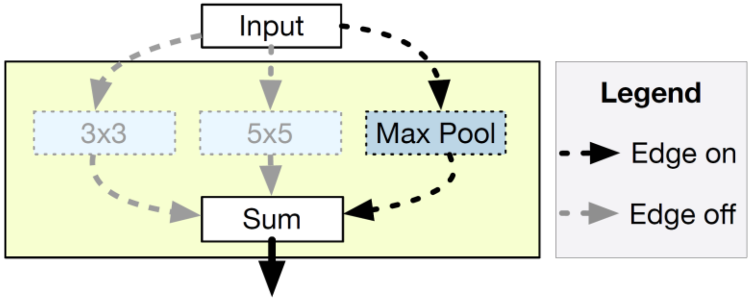



Figure: embeddings of two 7-layer CNNs (red, green) [Saxena & Verbeek, NeurIPS 2016]

- Problems/ limitations:
  - Search space restricted to one-shot model
  - One-shot model needs to be kept in GPU-memory
  - Search bias?

# Weight Sharing & One-shot Models

- Simplifying One-Shot Architecture Search
  [Bender et al., ICML 2018]
  – Use path dropout to make sure the individual models perform well by themselves

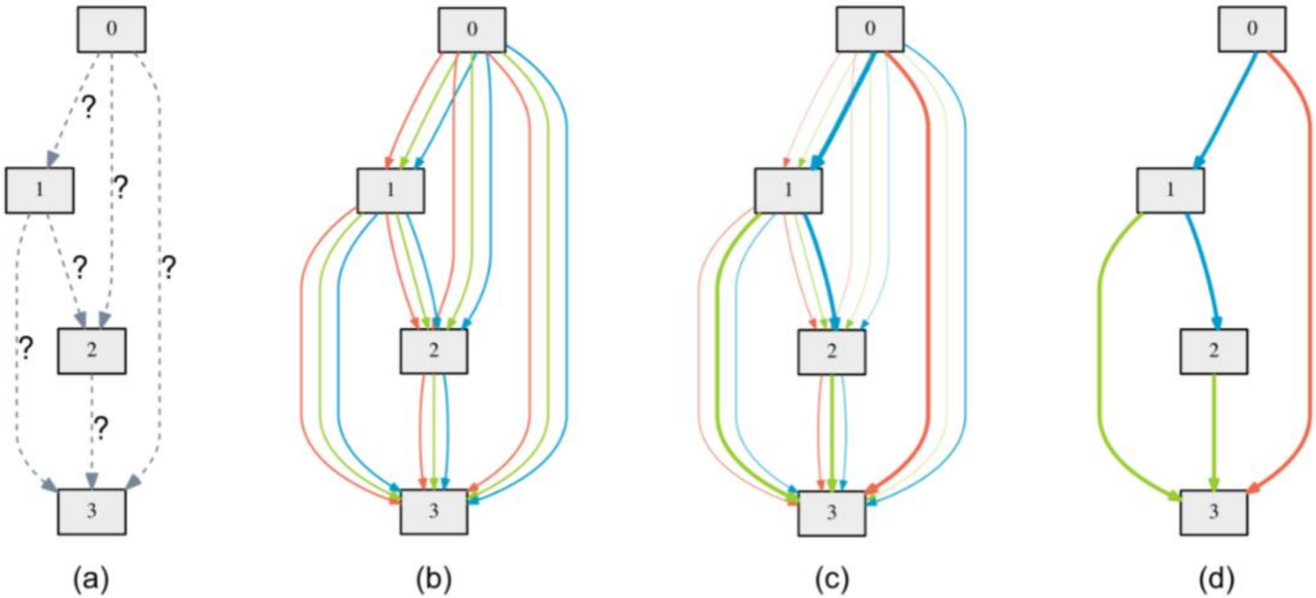



- ENAS [Pham et al., ICML 2018]
  – Use RL to sample paths (=architectures) from one-shot model

- SMASH [Brock et al., MetaLearn 2017]
  – Train hypernetwork that generates weights of models

# DARTS: Differentiable Architecture Search

[Liu et al., ICLR 2019]

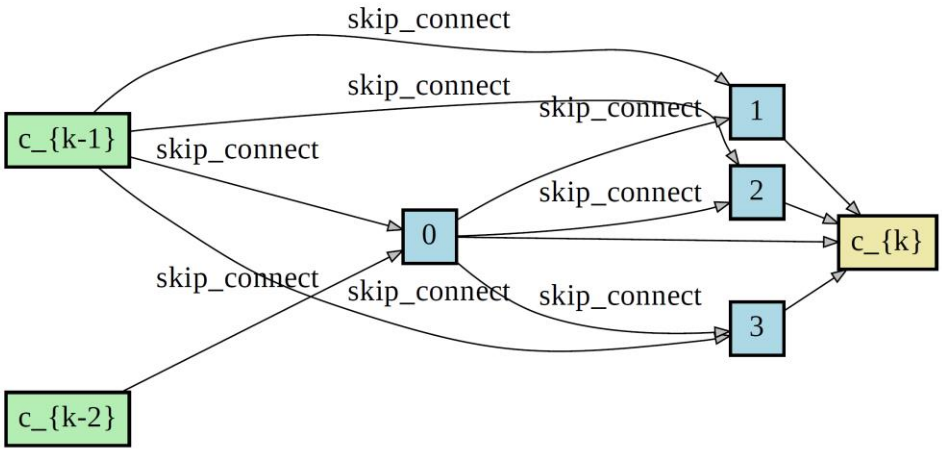



- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight α for each operator
  - Mixed operator: $$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$
- Solve a bi-level optimization problem (c)

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

- In the end, discretize to obtain a single architecture (d)

- Very fast:
  - By alternating SGD steps for α and w runtime only a bit higher than SGD for w alone
- Very brittle optimization:
  - Requires hyperparameter tuning for new problems
  - Discretization in step (d) can deteriorate performance a lot




- One-shot model needs to fit into GPU memory
- Already lots of follow-up work to solve these problems

[Xie et al., ICLR 2019, Cai et al., ICLR 2019, Dong et al., CVPR 2019, Zela et al, arXiv 2019]

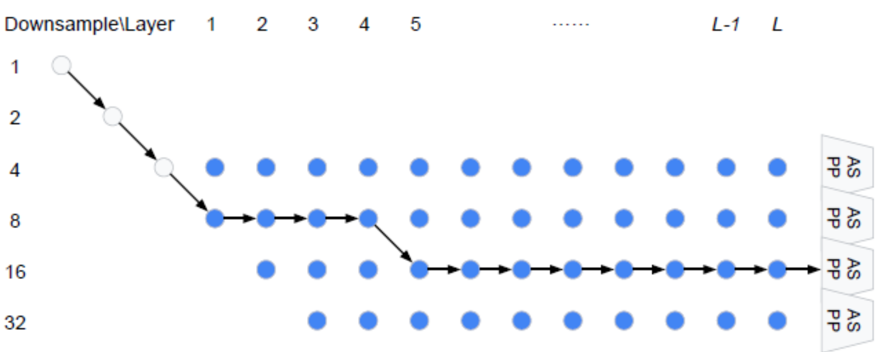
# Some numbers (CIFAR-10)

| | Reference | Error (%) | Params (Millions) | GPU Days |
|---|---|---|---|---|
| RL | Zoph and Le (2017) | 3.65 | 37.4 | 22,400 |
| | Cai et al. (2018a) | 4.23 | 23.4 | 10 |
| | Zoph et al. (2018) | 3.41 | 3.3 | 2,000 |
| | Zoph et al. (2018) + Cutout | 2.65 | 3.3 | 2,000 |
| | Cai et al. (2018b) | 2.99 | 5.7 | 200 |
| | Cai et al. (2018b) + Cutout | 2.49 | 5.7 | 200 |
| EA | Real et al. (2017) | 5.40 | 5.4 | 2,600 |
| | Liu et al. (2018b) | 3.75 | 15.7 | 300 |
| | Real et al. (2019) | 3.34 | 3.2 | 3,150 |
| | Elsken et al. (2018) | 5.2 | 19.7 | 1 |
| | Wistuba (2018a) + Cutout | 3.57 | 5.8 | 0.5 |
| One-Shot | Pham et al. (2018) | 3.54 | 4.6 | 0.5 |
| | Pham et al. (2018) + Cutout | 2.89 | 4.6 | 0.5 |
| | Bender et al. (2018) | 4.00 | 5.0 | N/A |
| | Casale et al. (2019) + Cutout | 2.81 | 3.7 | 1 |
| | Liu et al. (2019b) + Cutout | 2.76 | 3.3 | 4 |
| | Xie et al. (2019b) + Cutout | 2.85 | 2.8 | 1.5 |
| | Cai et al. (2019) + Cutout | 2.08 | 5.7 | 8.33 |
| | Brock et al. (2018) | 4.03 | 16.0 | 3 |
| | Zhang et al. (2019) | 2.84 | 5.7 | 0.84 |
| Random | Liu et al. (2018b) | 3.91 | N/A | 300 |
| | Luo et al. (2018) | 3.92 | 3.9 | 0.3 |
| | Liu et al. (2019b) + Cutout | 3.29 | 3.2 | 4 |
| | Li and Talwalkar (2019) + Cutout | 2.85 | 4.3 | 2.7 |

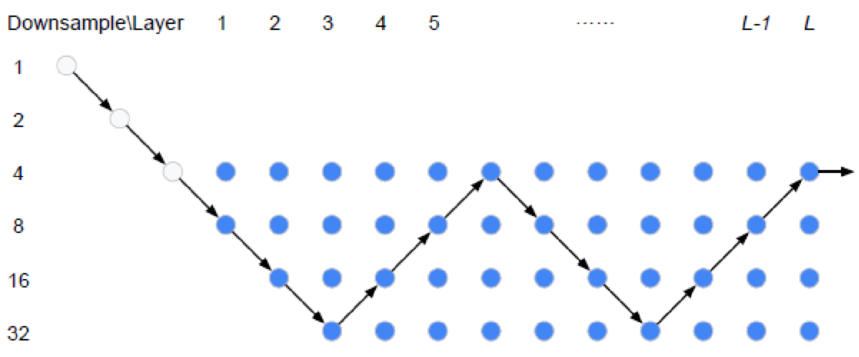
# Application of DARTS: Semantic Segmentation

# Application of DARTS: Semantic Segmentation

- Auto-DeepLab [Liu et al., CVPR 2019]
  - Also optimize downsampling factor for each layer
  - 3 GPU days search on Cityscapes
  - Based on DARTS

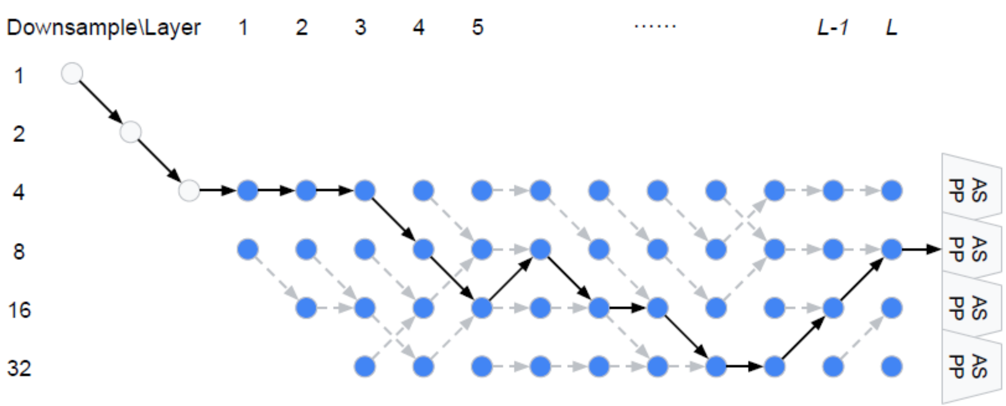



(a) Network level architecture used in DeepLabv3 [9].




(c) Network level architecture used in Stacked Hourglass [55].

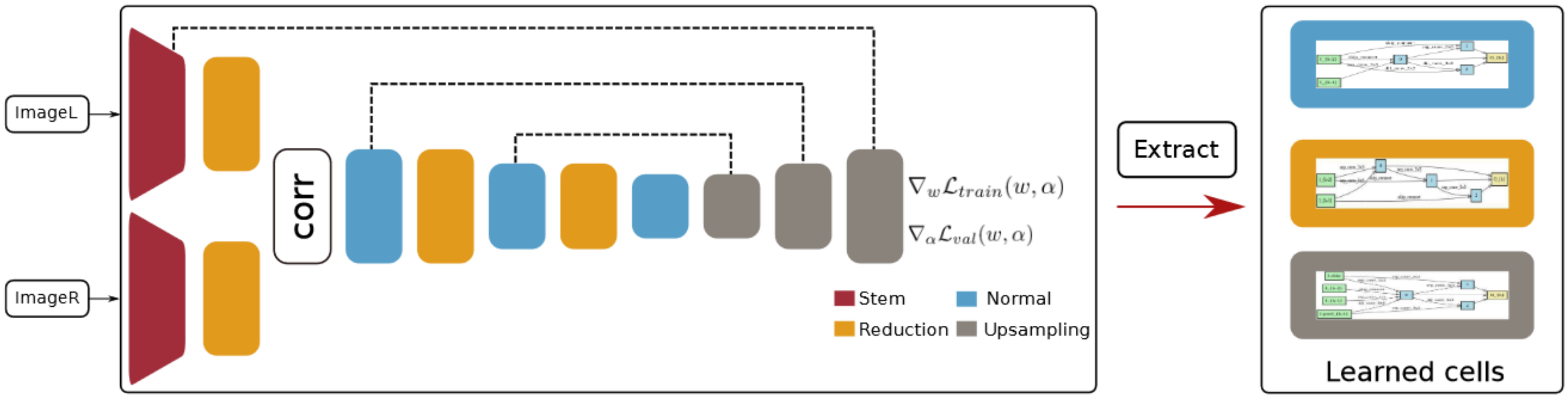
**Optimized**:
(3 GPU days search on Cityscapes)



# Application of DARTS: Disparity Estimation

# Application of DARTS: Disparity Estimation

- AutoDispNet [Saikia et al., ICCV 2019]
  - Introduce upsampling cells in addition to normal and reduction cells to allow for encoder-decoder architectures
  - Supports U-Net like encoder-decoder architectures




  - NAS with DARTS, then HPO with BOHB
  - Better performance than human domain experts
    - E.g., EPE on Sintel: 2.36 -> 2.14 (NAS) -> 1.94 (NAS+HPO)

# Outline

## Part 2: Neural Architecture Search

1. Search Spaces
2. Black-box Optimization
3. Beyond Black-box Optimization
4. Best Practices

# The Need for Proper Scientific Practice in NAS

- Reproducibility
  - Many ML methods are not very reproducible
  - NAS research is particularly hard to reproduce
- Fair comparisons of NAS methods
  - Using the same NAS Benchmarks
  - Definition: NAS Benchmark
    - Dataset (with predefined train/test split)
    - Search space
    - Available code with fixed hyperparameters for training final architecture
- Reporting important details
  - Hyperparameter optimization can drastically improve results

# A False Friend: The Final Results Table for CIFAR-10

| | Reference | Error (%) | Params (Millions) | GPU Days |
|---|---|---|---|---|
| RL | Baker et al. (2017) | 6.92 | 11.18 | 100 |
| | Zoph and Le (2017) | 3.65 | 37.4 | 22,400 |
| | Cai et al. (2018a) | 4.23 | 23.4 | 10 |
| | Zoph et al. (2018) | 3.41 | 3.3 | 2,000 |
| | Zoph et al. (2018) + Cutout | 2.65 | 3.3 | 2,000 |
| | Zhong et al. (2018) | 3.54 | 39.8 | 96 |
| | Cai et al. (2018b) | 2.99 | 5.7 | 200 |
| | Cai et al. (2018b) + Cutout | 2.49 | 5.7 | 200 |
| EA | Real et al. (2017) | 5.40 | 5.4 | 2,600 |
| | Xie and Yuille (2017) | 5.39 | N/A | 17 |
| | Suganuma et al. (2017) | 5.98 | 1.7 | 14.9 |
| | Liu et al. (2018b) | 3.75 | 15.7 | 300 |
| | Real et al. (2019) | 3.34 | 3.2 | 3,150 |
| | Elsken et al. (2018) | 5.2 | 19.7 | 1 |
| | Wistuba (2018a) + Cutout | 3.57 | 5.8 | 0.5 |
| SMBO | Kandasamy et al. (2018) | 8.69 | N/A | 1.7 |
| | Liu et al. (2018a) | 3.41 | 3.2 | 225 |
| | Luo et al. (2018) | 3.18 | 10.6 | 200 |
| One-Shot | Pham et al. (2018) | 3.54 | 4.6 | 0.5 |
| | Pham et al. (2018) + Cutout | 2.89 | 4.6 | 0.5 |
| | Bender et al. (2018) | 4.00 | 5.0 | N/A |
| | Casale et al. (2019) + Cutout | 2.81 | 3.7 | 1 |
| | Liu et al. (2019b) + Cutout | 2.76 | 3.3 | 4 |
| | Xie et al. (2019b) + Cutout | 2.85 | 2.8 | 1.5 |
| | Cai et al. (2019) + Cutout | 2.08 | 5.7 | 8.33 |
| | Brock et al. (2018) | 4.03 | 16.0 | 3 |
| | Zhang et al. (2019) | 2.84 | 5.7 | 0.84 |
| Random | Liu et al. (2018b) | 3.91 | N/A | 300 |
| | Luo et al. (2018) | 3.92 | 3.9 | 0.3 |
| | Liu et al. (2019b) + Cutout | 3.29 | 3.2 | 4 |
| | Li and Talwalkar (2019) + Cutout | 2.85 | 4.3 | 2.7 |
| Human | Zagoruyko and Komodakis (2016) | 3.87 | 36.2 | - |
| | Gastaldi (2017) (26 2x32d) | 3.55 | 2.9 | - |
| | Gastaldi (2017) (26 2x96d) | 2.86 | 26.2 | - |
| | Gastaldi (2017) (26 2x112d) | 2.82 | 35.6 | - |
| | Yamada et al. (2016) + ShakeDrop | 2.67 | 26.2 | - |

- Different code for training networks (often unavailable)
  - Performance hugely affected by cutout, Auto-Augment, mixup, scheduled drop-path, cosine annealing, etc
- Different search spaces
- Different evaluation schemes
- No repeated runs

# **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.

# NAS Best Practices Checklist 2/3

**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?

# NAS Best Practices Checklist 3/3

**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?

# Further Ways Forward for the NAS Community

- Need for more good NAS Benchmarks
  - Only experiments on the same benchmarks are comparable
  - Have we overfit on CIFAR-10 and PTB?
  - By now, lots of non-standard applications of NAS, e.g., semantic segmentation, disparity estimation, reinforcement learning, machine translation, GANs, image restoration, etc.

- Need for an open-source library of NAS methods
  - To avoid confounding factors & understand the true causes of good and bad performance on a NAS benchmark
  - To be able to mix & match components of different algorithms
  - We‘re developing NASlib and Auto-PyTorch

# Wrapup

## Part 2: Neural Architecture Search

1. Search Spaces
2. Black-box Optimization
3. Beyond Black-box Optimization
4. Best Practices

- NAS is key for new application areas of deep learning
- By now, NAS is a topic all of academia can partake in
- NAS & HPO are both important for strong performance

# 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
@automlfreiburg

AutoML.org