Automated Machine Learning (AutoML): A Tutorial

Matthias Feurer
University of Freiburg
feurerm@cs.uni-freiburg.de

Thomas Elsken
Bosch Center for Artificial Intelligence & University of Freiburg
Thomas.Elsken@de.bosch.com
Outline

1. General AutoML
2. Neural Architecture Search
3. Meta Learning & Learning to Learn

For more details, see: automl.org/book

AutoML: true end-to-end learning

Feurer and Elsken: AutoML
Neural Architecture Search - Motivation

Bigger, more complex architectures...

Slide courtesy Nikhil Naik

Inception-V4 modules

Feurer and Elsken: AutoML

[Canziani et al., preprint 2017]

[Szegedy et al., AAAI 2017]
Can we automatically design neural network architectures?
Outline

Neural Architecture Search

- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

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

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

More complex space with multiple branches and skip connections

Elsken et al., JMLR 2019

Feurer and Elsken: AutoML
Cell Search Spaces

Introduced by Zoph et al. [CVPR 2018]

Architecture composed of stacking together individual cells

Two possible cells

**normal cell:** preservess spatial resolution

**reduction cell:** reduce spatial resolution

Feurer and Elsken: AutoML
Outline

Neural Architecture Search

- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

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

- 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

[Feurer and Elsken: AutoML]

[Zoph & Le, ICLR 2017]
**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

[Real et al., AAAI 2019]

final evaluation

[Real et al., AAAI 2019]
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]
## Some numbers (Cifar-10)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Error (%)</th>
<th>Params (Millions)</th>
<th>GPU Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zoph and Le (2017)</td>
<td>3.65</td>
<td>37.4</td>
<td>22,400</td>
</tr>
<tr>
<td>Zoph et al. (2018)</td>
<td>3.41</td>
<td>3.3</td>
<td>2,000</td>
</tr>
<tr>
<td>EA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real et al. (2017)</td>
<td>5.40</td>
<td>5.4</td>
<td>2,600</td>
</tr>
<tr>
<td>Real et al. (2019)</td>
<td>3.34</td>
<td>3.2</td>
<td>3,150</td>
</tr>
</tbody>
</table>

[Wistuba et al., preprint 2019]

Feurer and Elsken: AutoML

Going to cell search space
Blackbox optimization is expensive!
Can we do better?
Outline

Neural Architecture Search

- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

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

- Weight inheritance & network morphisms

- Weight sharing & one-shot models

- Multi-fidelity optimization
  - do search on smaller models, less training epochs, fewer data,...

- Meta-learning
  - Avoid running NAS from scratch for every task
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 network

\[ N_1(x) = \text{Softmax}_{w_{1,1}} \circ \text{ReLU} \circ \text{Conv}_{w_{1,2}}(x) \]

want to add another Relu-Conv block

\[ N_2(x) = \text{Softmax}_{w_{2,1}} \circ \text{ReLU} \circ \text{Conv}_{w_{2,2}} \circ \text{ReLU} \circ \text{Conv}_{w_{2,3}}(x) \]

copy

\[ w_{2,1} = w_{1,1}, \quad w_{2,3} = w_{1,2} \]

and set \( w_{2,2} \) so that \( \text{Conv}_{w_{2,2}}(x) = x \)

Then:
Weight inheritance & network morphisms

[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]

→ enables efficient architecture search

Feurer and Elsken: AutoML
Multi-objective NAS: LEMONADE

- 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

More work on designing efficient architectures, e.g.:
- NAS for compression [Cao et al., ICLR 2019; He et al., ECCV 2018], NAS with hardware-constrained objectives [Tan et al., CVPR 2019, Tan et al., ICML 2019, Cai et al., ICLR 2019]
Some numbers (Cifar-10)

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

[Wistuba et al., preprint 2019]

Feurer and Elsken: AutoML
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
- solely 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 hyernetwork that generates weights of models
DARTS: Differentiable Architecture Search

- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight $\alpha$ for each operator
  - Mixed operator:
    $$\tilde{o}^{(i,j)}(x) = \sum_{o \in O} \sum_{o' \in O} \exp(\alpha_{o}^{(i,j)}) \cdot o(x)$$

- Solve a bi-level optimization problem (c)
  $$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
  $$\text{s.t. } w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha)$$

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

Feurer and Elsken: AutoML
Very fast:
  - By alternating SGD steps for $\alpha$ and $w$ runtime only a bit higher than SGD for $w$ alone

Very brittle optimization:
  - Requires hyperparameter tuning for new problems

One-shot models needs to be kept in GPU memory

Discretization at end of search degenerates performance; re-training necessary

Already lots of follow-up work trying to solve these problems

[Xie et al., ICLR 2019, Cai et al., ICLR 2019, Dong et al., CVPR 2019]
## Some numbers (Cifar-10)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Error (%)</th>
<th>Params (Millions)</th>
<th>GPU Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoph and Le (2017)</td>
<td>3.65</td>
<td>37.4</td>
<td>22,400</td>
</tr>
<tr>
<td>Cai et al. (2018a)</td>
<td>4.23</td>
<td>23.4</td>
<td>10</td>
</tr>
<tr>
<td>Zoph et al. (2018)</td>
<td>3.41</td>
<td>3.3</td>
<td>2,000</td>
</tr>
<tr>
<td>Zoph et al. (2018) + Cutout</td>
<td>2.65</td>
<td>3.3</td>
<td>2,000</td>
</tr>
<tr>
<td>Cai et al. (2018b)</td>
<td>2.99</td>
<td>5.7</td>
<td>200</td>
</tr>
<tr>
<td>Cai et al. (2018b) + Cutout</td>
<td>2.49</td>
<td>5.7</td>
<td>200</td>
</tr>
<tr>
<td>Real et al. (2017)</td>
<td>5.40</td>
<td>5.4</td>
<td>2,600</td>
</tr>
<tr>
<td>Liu et al. (2018b)</td>
<td>3.75</td>
<td>15.7</td>
<td>300</td>
</tr>
<tr>
<td>Real et al. (2019)</td>
<td>3.34</td>
<td>3.2</td>
<td>3,150</td>
</tr>
<tr>
<td>Elsken et al. (2018)</td>
<td>5.2</td>
<td>19.7</td>
<td>1</td>
</tr>
<tr>
<td>Wistuba (2018a) + Cutout</td>
<td>3.57</td>
<td>5.8</td>
<td>0.5</td>
</tr>
<tr>
<td>Pham et al. (2018)</td>
<td>3.54</td>
<td>4.6</td>
<td>0.5</td>
</tr>
<tr>
<td>Pham et al. (2018) + Cutout</td>
<td>2.89</td>
<td>4.6</td>
<td>0.5</td>
</tr>
<tr>
<td>Bender et al. (2018)</td>
<td>4.00</td>
<td>5.0</td>
<td>N/A</td>
</tr>
<tr>
<td>Casale et al. (2019) + Cutout</td>
<td>2.81</td>
<td>3.7</td>
<td>1</td>
</tr>
<tr>
<td>Liu et al. (2019b) + Cutout</td>
<td>2.76</td>
<td>3.3</td>
<td>4</td>
</tr>
<tr>
<td>Xie et al. (2019b) + Cutout</td>
<td>2.85</td>
<td>2.8</td>
<td>1.5</td>
</tr>
<tr>
<td>Cai et al. (2019) + Cutout</td>
<td>2.08</td>
<td>5.7</td>
<td>8.33</td>
</tr>
<tr>
<td>Brock et al. (2018)</td>
<td>4.03</td>
<td>16.0</td>
<td>3</td>
</tr>
<tr>
<td>Zhang et al. (2019)</td>
<td>2.84</td>
<td>5.7</td>
<td>0.84</td>
</tr>
</tbody>
</table>

| 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 |

[Preprint 2019]
NAS for dense prediction tasks

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

Optimized:
(3 GPU days search on Cityscapes)
NAS for dense prediction tasks

- **AutoDispNet** [Saikia et al., ICCV 2019]
  - Introduce upsampling cells in addition to normal and reduction cells to allow for encoder-decoder architectures
- DARTS [Liu et al., ICLR 2019] for architecture optimization
- BOHB [Falkner et al., ICML 2018] for hyperparameters
Remarks on Experimentation in NAS

- Final results are often *incomparable* due to
  - Different training pipelines without available source code
    - Releasing the final architecture does not help for comparisons
  - Different hyperparameter choices
    - Very different hyperparameters for training and final evaluation
  - Different search spaces / initial models
    - Starting from random or from state-of-the-art?

→ Need for looking beyond the error numbers on CIFAR
→ Need for benchmarks including training pipeline & hyperparams

- Experiments are often very expensive

→ Need for cheap benchmarks that allow for many runs,
  e.g., Ying et al., ICML 2019
Outline

1. General AutoML
2. Neural Architecture Search
3. Meta Learning & Learning to Learn

For more details, see: automl.org/book

AutoML: true end-to-end learning

Feurer and Elsken: AutoML
“Meta-learning, or learning to learn, is the science of systematically observing how different machine learning approaches perform on a wide range of learning tasks, and then learning from this experience, or meta-data, to learn new tasks much faster than otherwise possible.”

[Vanschoren, Chapter 2 of the AutoML book]
Meta Learning & Learning to Learn

- Meta Learning for Few Shot Learning
- Learning to Optimize
Meta Learning & Learning to Learn

- Meta Learning for Few Shot Learning
- Learning to Optimize
Few-Shot Learning

- Classic deep learning setting: large, diverse data
- What if we don’t have a large dataset? (medical data sets, personalized education, speech for rare languages, …)
- We can still learn from related tasks and experience.

Task 1

Task 2

Task 3

Task 4

Quick, Draw! Dataset

- Few-Shot Learning setting: many small but related tasks

Slide inspired by Chelsea Finn
Feurer and Elsken: AutoML
Model-Agnostic Meta-Learning (MAML) [Finn et al., ICML 2017]

- Learn initialization for weights $\theta$ of neural network that quickly adapts to weights $\theta_i'$ for new task $T_i$
while not done:

1. sample tasks $T_i$

2. update task weights

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{T_i}(f_{\theta})$$

3. update meta weights $\theta$ by solving

$$\min_{\theta} \sum_{T_i} \mathcal{L}_{T_i}(f_{\theta'}) = \sum_{T_i} \mathcal{L}_{T_i}(f_{\theta - \alpha \nabla_{\theta} \mathcal{L}_{T_i}(f_{\theta})})$$

See also

REPTILE [Nichol et al., arXiv preprint 2018], PLATIPUS [Finn et al., NeurIPS 2018]

Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples [Triantafillou et al., NeurIPS MetaLearn 2018]
Learning Initializations for Few-Shot Learning

- Transferring Knowledge Across Learning (LEAP)
  [Flennerhag et al., ICLR 2019]

- Look at training process rather than final weights only
- Iteratively learn initialization with
  a) shorter expected gradient path that
  b) also improves performance

**Gradient path length**

**Meta objective:**

\[
\min_{\theta^0} F(\theta^0) = \mathbb{E}_{\tau \sim p(\tau)} [d(\theta^0; M_\tau)]
\]

s.t. \( \theta^{i+1}_\tau = u_\tau(\theta^i_\tau), \quad \theta^0_\tau = \theta^0, \quad \theta^0 \in \Theta = \cap_\tau \{ \theta^0 \mid f_\tau(\theta^{K_\tau}_\tau) \leq f_\tau(\psi^{K_\tau}_\tau) \} \)

**Improved performance**
Meta Learning & Learning to Learn

- Meta Learning for Few Shot Learning
- Learning to Optimize
Many machine learning problems solved by applying some form of gradient descent

\[ \theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \]

Much modern work in optimization based on designing algorithms for specific class of problems

e.g., for deep learning: AdaGrad [Duchi et al., JMLR 2011], Adam [Kingma et al., ICLR 2015], RMSProp, ...

Can we learn better optimizers / update rules?
Learning to Learn by Gradient Descent by Gradient Descent
[Andrychowicz et al., NeurIPS 2016]

- Replace classic gradient-based updates such as
  \[ \theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \]

- by learning model \( g_t \) providing updates
  \[ \theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) \]

- \( g_t \) trained by optimizing meta objective
  \[ \mathcal{L}(\phi) = \mathbb{E}_f \left[ \sum_{t=1}^{T} w_t f(\theta_t) \right] \]
Learning to Optimize (with RNNs)

[Andrychowicz et al., NeurIPS 2016]

Training MLP

- Quadratics
  - ADAM
  - RMSprop
  - SGD
  - NAG
  - LSTM

Training ConvNet

- CIFAR-10
- CIFAR-5

Feurer and Elsken: AutoML
Does the learned optimizer generalize to optimizing other neural networks architectures?

Learned optimizer trained on optimizing MLP with 20 hidden units, 1 layer, sigmoid activation function.

- More hidden units
- More layers
- Different activation function
Learning to Learn **without** Gradient Descent by Gradient Descent [Chen et al., ICML 2017]

- Directly propose next query point $x_t$ rather than update rule

$$h_t, x_t = \text{RNN}_\theta(h_{t-1}, x_{t-1}, y_{t-1})$$

- No need for gradients during meta-test time!
- Meta objective: e.g., maximize Expected Improvement

$$L_{\text{EI}}(\theta) = -\mathbb{E}_{f, y_{1:T-1}} \left[ \sum_{t=1}^{T} \text{EI}(x_t \mid y_{1:t-1}) \right]$$

---

Feurer and Elsken: AutoML
Neural Optimizer Search with RL [Bello et al., ICML 2017]

- Use RL to train RNN controller to sample update rules for training neural networks

- Allow updates of the form

\[ \Delta w = b(u_1(op_1), u_2(op_2)) \]

- Binary functions which map \((x, y)\) to \(x + y\) (addition), \(x - y\) (subtraction), \(x \cdot y\) (multiplication), \(\frac{x}{y+\epsilon}\) (division) or \(x\) (keep left).

- Unary functions which map input \(x\) to: \(x, -x, e^x, \log |x|, \text{clip}(x, 10^{-5}), \text{clip}(x, 10^{-4}), \text{clip}(x, 10^{-3}), \text{drop}(x, 0.1), \text{drop}(x, 0.3), \text{drop}(x, 0.5)\) and \(\text{sign}(x)\).

- Operands: \(g, g^2, g^3, \hat{m}, \hat{v}, \hat{\gamma}, \text{sign}(g), \text{sign}(\hat{m})\), 1, small constant noise, \(10^{-4}w, 10^{-3}w, 10^{-2}w, 10^{-1}w\), ADAM and RMSProp.

+ update rules of above form
**Learning to Optimize (with RL)**

**Top 5 update rules**

\[
[e^{\text{sign}(g)\times\text{sign}(m)} + \text{clip}(g, 10^{-4})] \times g \\
\text{clip}(\hat{m}, 10^{-4}) \times e^{v} \\
\hat{m} \times e^{v} \\
g \times e^{\text{sign}(g)\times\text{sign}(m)} \\
\text{drop}(g, 0.3) \times e^{\text{sign}(g)\times\text{sign}(m)}
\]

- Method can also be used to search for activation functions [Ramachandran et al., ICLR WS 2018]
- See also [Li and Malik, ICLR 2017]
Learning to Optimize for Unsupervised Learning

- Meta-Learning Update Rules for Unsupervised Representation Learning [Metz et al., ICLR 2019]
- Unsupervised Learning: discover data representations without access to supervised labels
Generalization of learned unsupervised learning rules ... to different data sets ...

... to different architectures

Learning to Optimize for Unsupervised Learning

([Metz et al., ICLR 2019])
Learn the parameters $\psi$ of a simulator via RL by solving

$$\psi^* = \arg\min_{\psi} \sum_{(x, y) \in D_{\text{val}}} \mathcal{L}(y, h_\theta(x; \theta^*(\psi)))$$

subject to

$$\theta^*(\psi) = \arg\min_{\theta} \sum_{(x, y) \in D_q(x, y|\psi)} \mathcal{L}(y, h_\theta(x, \theta))$$

tune $\psi$ on validation data given a model $h_\theta$ trained on simulated data

train model $h_\theta$ on data generated by simulator with parameter $\psi$
NAS & Meta Learning Wrap-up

- NAS is not insanely expensive anymore; several ways to speed up blackbox NAS
  - Weight inheritance
  - Weight sharing & one-shot models
  - Meta Learning (so far mostly unexplored)
  → by now: resources required for running NAS methods often in same order of magnitude as simply training a network

- Exciting research fields, lots of progress but also lots of open problems:
  - meaningful benchmarks missing
  - NAS beyond image classification
  - NAS for multi-task, multi-objective problems
  - Understanding why architectures found by NAS work well

- Meta Learning and Learning to Learn very natural concepts to look at
  - Humans almost never learn from scratch
  - Has the potential to significantly speed up learning of new tasks and improve performance when limited data is available

Feurer and Elsken: AutoML
Thank you for your attention!


Contact:
thomas.elsken@de.bosch.com
New NAS papers over time

Number of papers on architecture search

#Papers

Year

2015  2016  2017  2018  2019 (first half)

Feurer and Elsken: AutoML
## New NAS papers over time

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

Feurer and Elsken: AutoML
NAS and compression:

– Efficient Neural Architecture Compression (ESNAC) [Cao et al., ICLR 2019]
  • learn embedding space over architectures via bi-directional LSTM
  • Use Bayesian Optimization in embedded space to compress architectures

– AutoML for Model Compression (AMC) [He et al., ECCV 2018]
  • Optimize per-layer compression rate via RL
Open-source Automated Deep Learning Systems

- **Auto-Net** [Mendoza et al, AutoML 2016]
  - First system AutoML 2016
  - Based on SMAC and Lasagne (→ deprecated)

- **Auto-Keras** [Jin, Song & Hu, AutoML 2018]
  - Based on network morphisms and Keras; [code](#)

- **Auto-PyTorch** [unpublished]
  - Based on PyTorch and BOHB; [code (pre-alpha)](#)

- Neural Network Intelligence by Microsoft
  - Based on various techniques; [code](#)