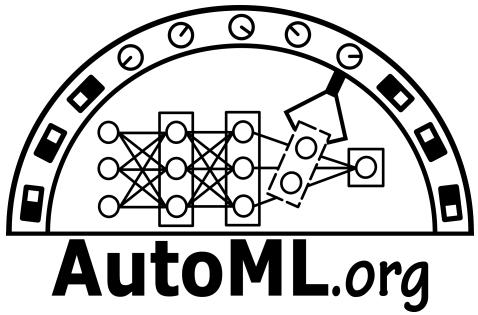
Automated Machine Learning
(AutoML)

M. Lindauer F. Hutter

University of Freiburg



Lecture 7: Neural Architecture Search (Part 1)

Speaker: Arber Zela

Where are we? The big picture

- Introduction
- Background
  - Design spaces in ML
  - Evaluation and visualization
- Hyperparameter optimization (HPO)
  - Bayesian optimization
  - Other black-box techniques
  - More details on Gaussian processes
- Pentecost (Holiday) – no lecture

→ Architecture search I + II

- Meta-Learning
- Learning to learn & optimize
- Beyond AutoML: algorithm configuration and control
- Project announcement and closing

Learning Goals

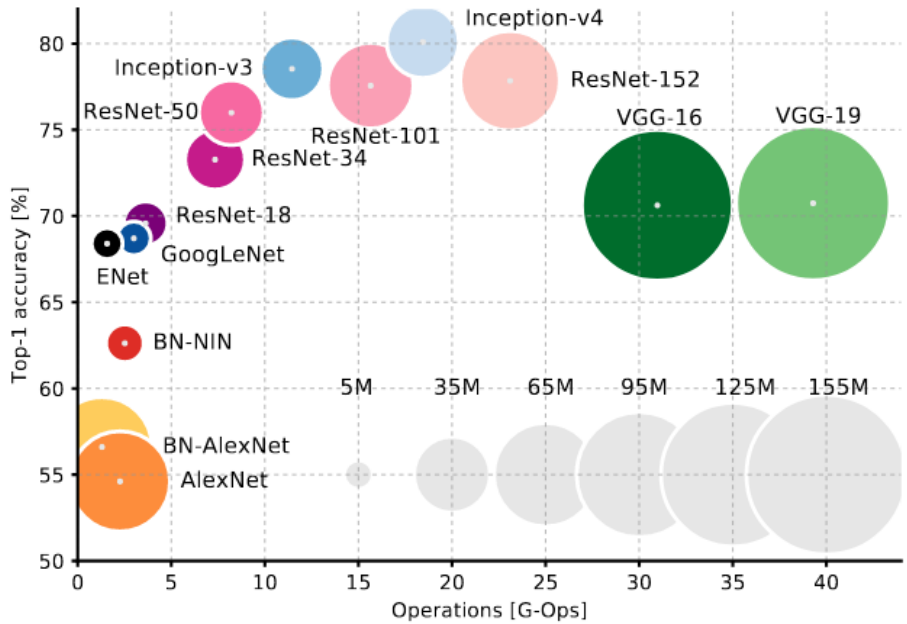
After this lecture, you will be able to . . .

- describe different search spaces for neural architecture search (NAS)
- describe various optimization methods for NAS
- explain various speedup techniques for NAS

Lecture Overview

Motivation for Neural Architecture Search

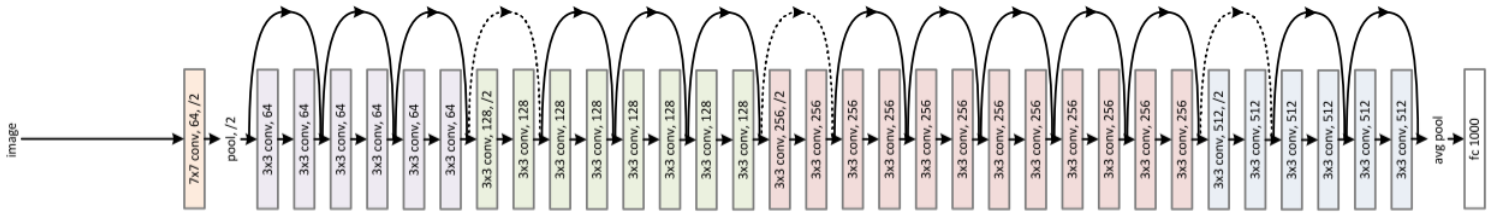
- Performance improvements on various tasks mostly due to novel architectural design choices
- Designing neural network architectures is hard, requiring lots of human efforts
- Can we automate this design process, potentially discovering new components/topologies?



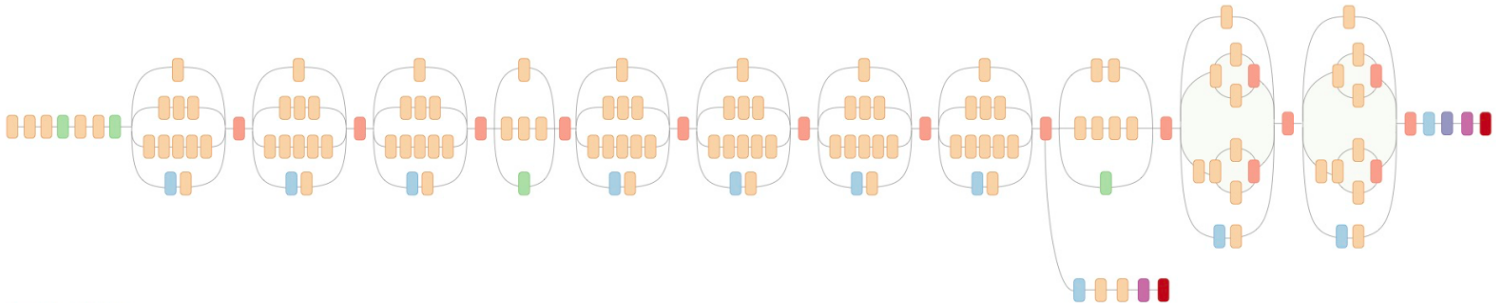
Larger circles, more network parameters [Canziani et al. 2017]

Successful Human-designed Architectures

34-layer ResNet [He et al. 2015]

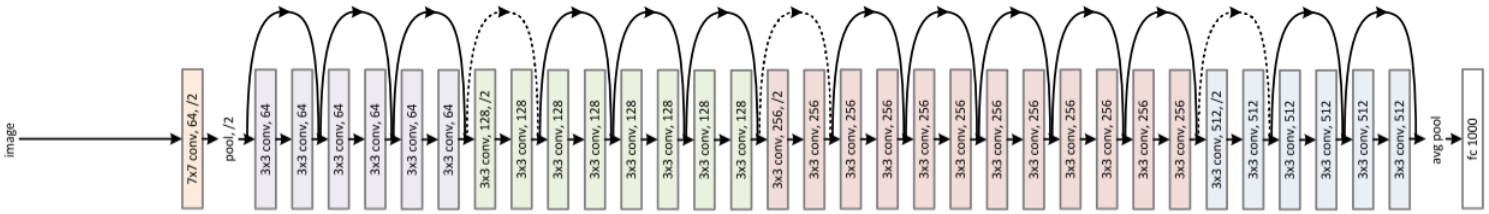


Inception-v3 [Szegedy et al. 2016]

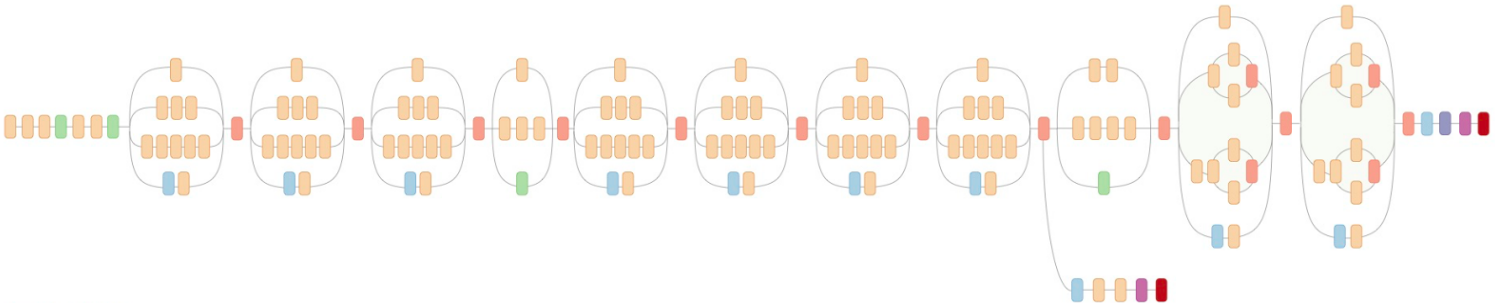


Successful Human-designed Architectures

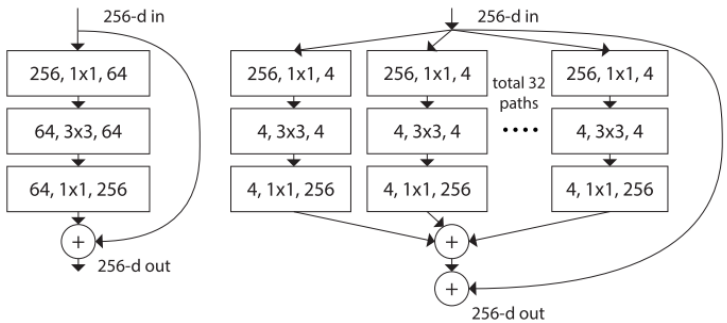
34-layer ResNet [He et al. 2015]



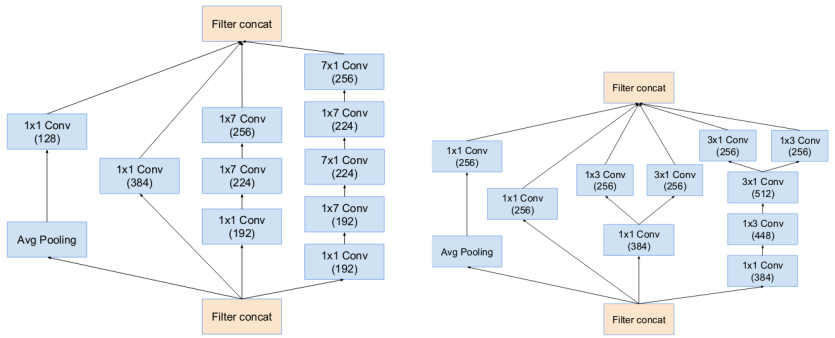
Inception-v3 [Szegedy et al. 2016]



ResNet/ResNeXt blocks [He et al. 2015, 2016; Xie et al. 2016]

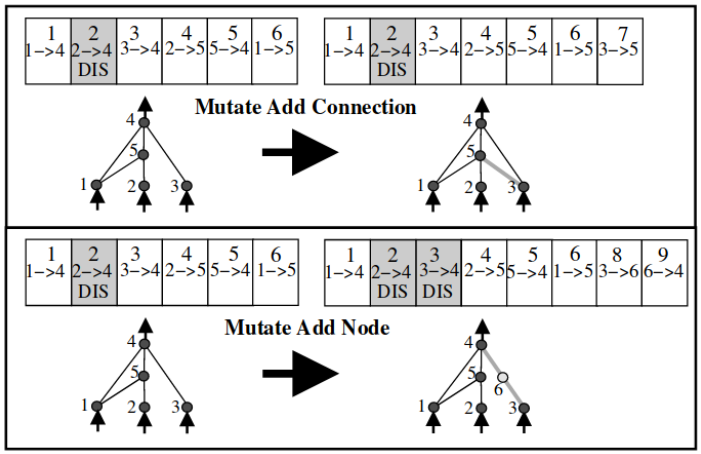


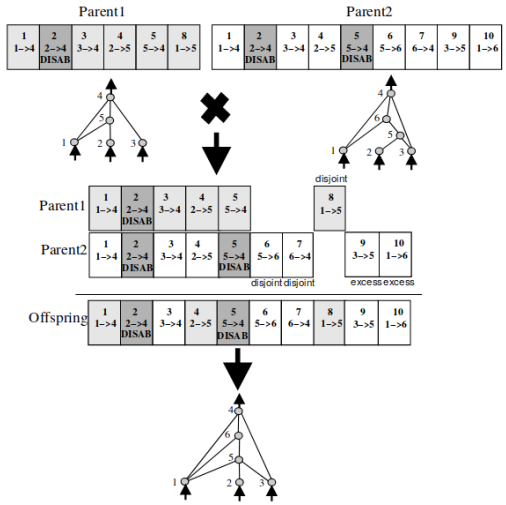
Inception-v4 blocks [Szegedy et al. 2017]



Brief history and early approaches

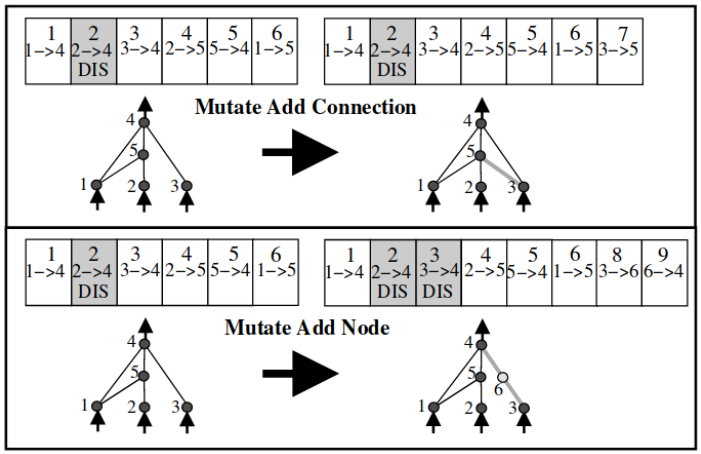
- Neuroevolution (already since the 1990s)
  - Some authors used evolutionary algorithms for the architecture, optimizing the weights with SGD [Stanley and Miikkulainen, 2002]
  - Others optimized both architecture and weights with evolutionary methods

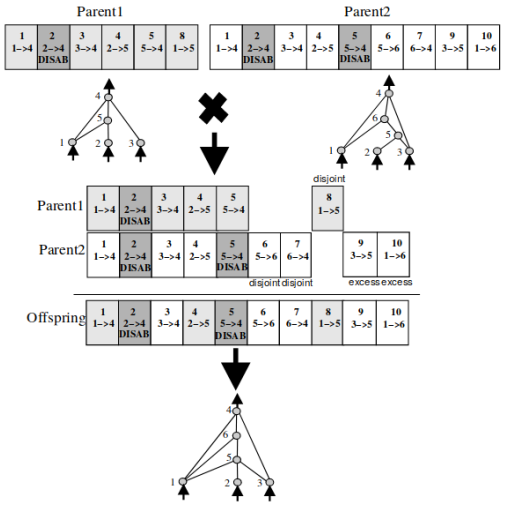




Brief history and early approaches

- Neuroevolution (already since the 1990s)
  - Some authors used evolutionary algorithms for the architecture, optimizing the weights with SGD [Stanley and Miikkulainen, 2002]
  - Others optimized both architecture and weights with evolutionary methods





- Bayesian optimization (BO)
  - Encode the architecture as a vector space
  - i.e. optimize number of layers, number of filters, kernel width, etc.
  - Use BO to search in this vectorized space [Bergstra et al. 2013, Domhan et al. 2015, Mendoza et al. 2015]

Since Then: Many Works on Architecture Search

- RL & Evolution for NAS by Google Brain [Quoc Le's group, '16-'18]
  - New state-of-the-art results for CIFAR-10, ImageNet, Penn Treebank
  - Large computational demands
    - 800 GPUs for 2 weeks; 12800 architectures evaluated
  - Hyperparameter optimization only as postprocessing

Since Then: Many Works on Architecture Search

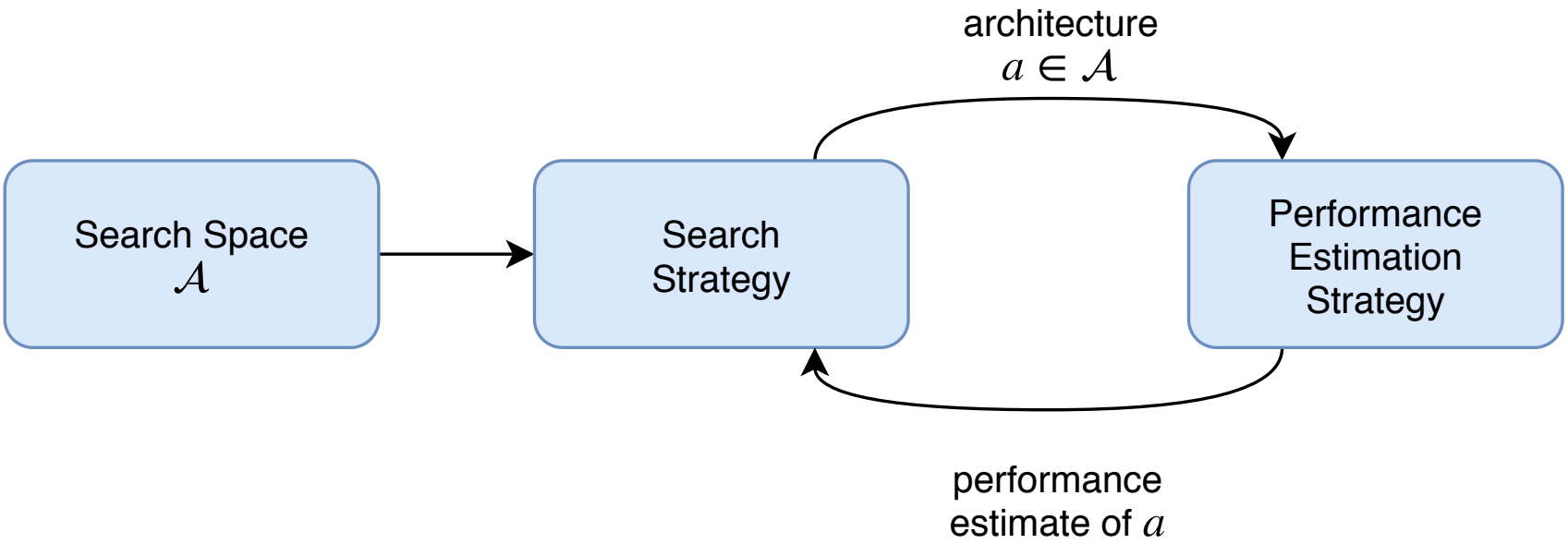
- RL & Evolution for NAS by Google Brain [Quoc Le's group, '16-'18]
  - New state-of-the-art results for CIFAR-10, ImageNet, Penn Treebank
  - Large computational demands
    - 800 GPUs for 2 weeks; 12800 architectures evaluated
  - Hyperparameter optimization only as postprocessing
- NAS for dense prediction, video classification and machine translation
  - AutoDeepLab for semantic segmentation and AutoDispNet for disparity estimation tasks [Liu et al.'18; Saikia et al.'19]
  - Improved accuracy for action classification on Charades and Moments in Time datasets with AssembleNet [Ryoo et al.'19]
  - SOTA results in translation tasks with Evolved Transformer [So et al.'19]

Since Then: Many Works on Architecture Search

- RL & Evolution for NAS by Google Brain [Quoc Le's group, '16-'18]
  - New state-of-the-art results for CIFAR-10, ImageNet, Penn Treebank
  - Large computational demands
    - 800 GPUs for 2 weeks; 12800 architectures evaluated
  - Hyperparameter optimization only as postprocessing
- NAS for dense prediction, video classification and machine translation
  - AutoDeepLab for semantic segmentation and AutoDispNet for disparity estimation tasks [Liu et al.'18; Saikia et al.'19]
  - Improved accuracy for action classification on Charades and Moments in Time datasets with AssembleNet [Ryoo et al.'19]
  - SOTA results in translation tasks with Evolved Transformer [So et al.'19]
- Recent work aims for efficiency
  - Weight sharing [Pham et al,'18; Bender et al, '18; Liu et al, '19; Xie et al. '19; Cai et al. '19, Zhang et al. '19]
  - Network morphisms [Chen et al, '16; Cai et al, '17 & '18; Elsken et al, '17 & 18]
  - Multi-fidelity optimization [Klein et al, '16; Li et al, '18; Falkner et al, '18]

Lecture Overview

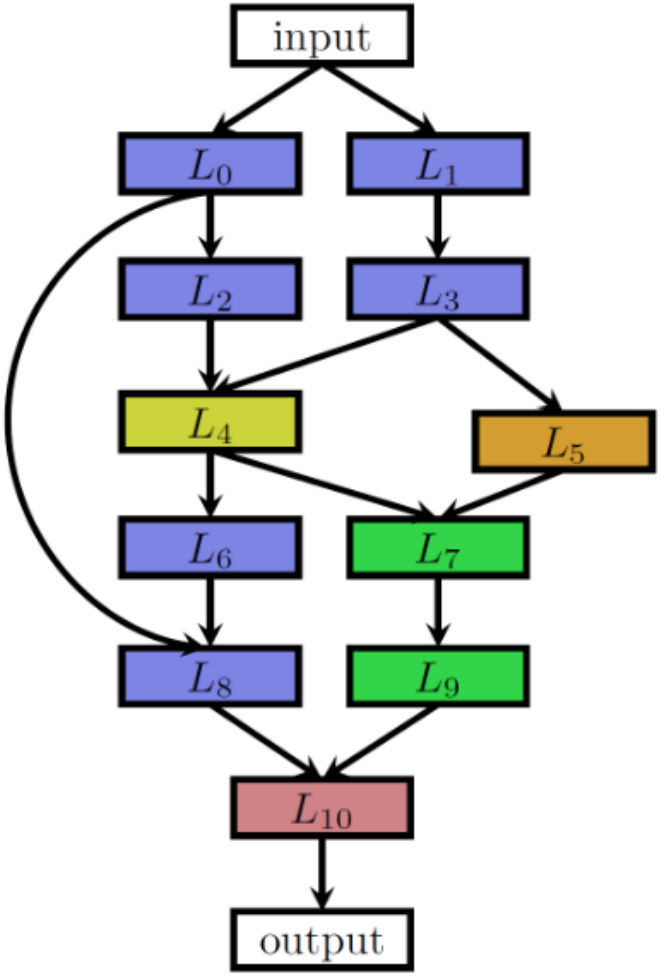
NAS components



- Search Space: *macro* or *micro* search space
- Search Strategy: blackbox (RL, ES, BO, etc.) or gradiend-based methods
- Performance Estimation Strategy: Lower fidelity estimates, curve extrapolation, etc.

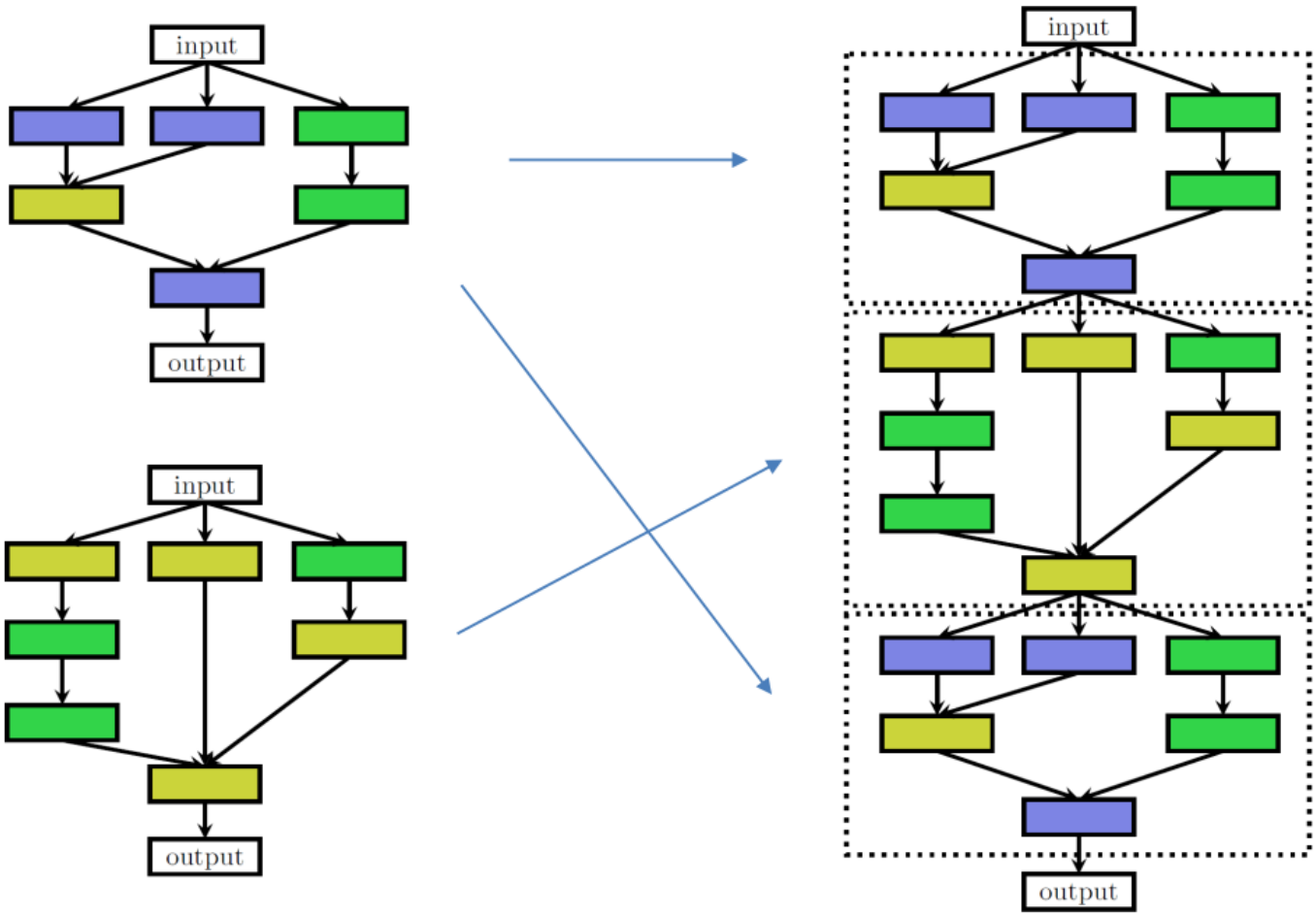
Basic Neural Architecture Search Spaces

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



More complex space
with multiple branches
and skip connections

Cell Search Spaces



Two possible cells

Architecture composed of stacking together individual cells

Introduced by Zoph et al.'18

Pros and Cons of Cell Search Space

What are some pros and cons of the cell search space compared to the chain-structured one? [2min]

Pros and Cons of Cell Search Space

Pros:

- Reduced search space size; speed-ups in term of search time.
- Transferrability to other datasets (e.g. cells found on CIFAR-10 transferred to ImageNet)
- Stacking repeating patterns proven to be useful design principle (ResNet, Inception, etc.)

Pros and Cons of Cell Search Space

Pros:

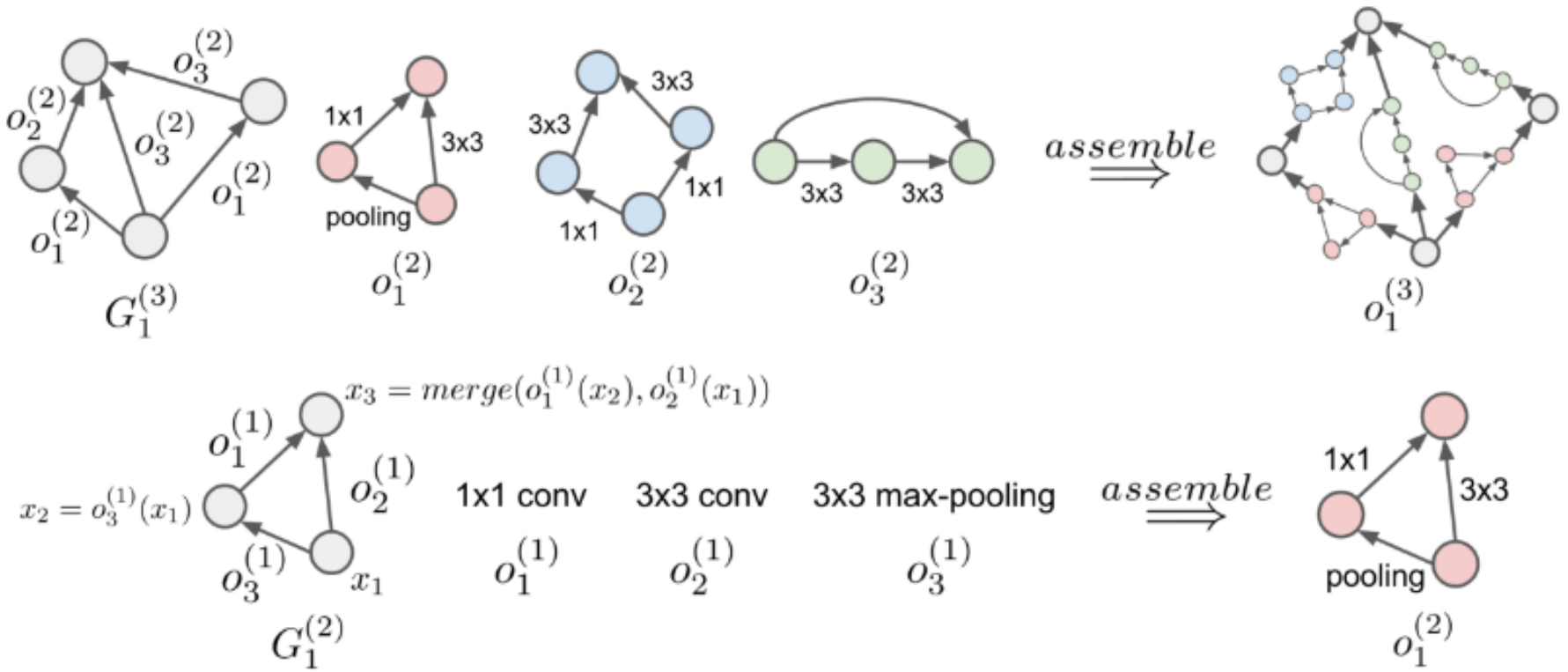
- Reduced search space size; speed-ups in term of search time.
- Transferrability to other datasets (e.g. cells found on CIFAR-10 transferred to ImageNet)
- Stacking repeating patterns proven to be useful design principle (ResNet, Inception, etc.)

Cons:

- Manually determining the *macro* architecture, i.e. the way cells are connected.

Hierarchical representation of search space [Liu et al.'17]

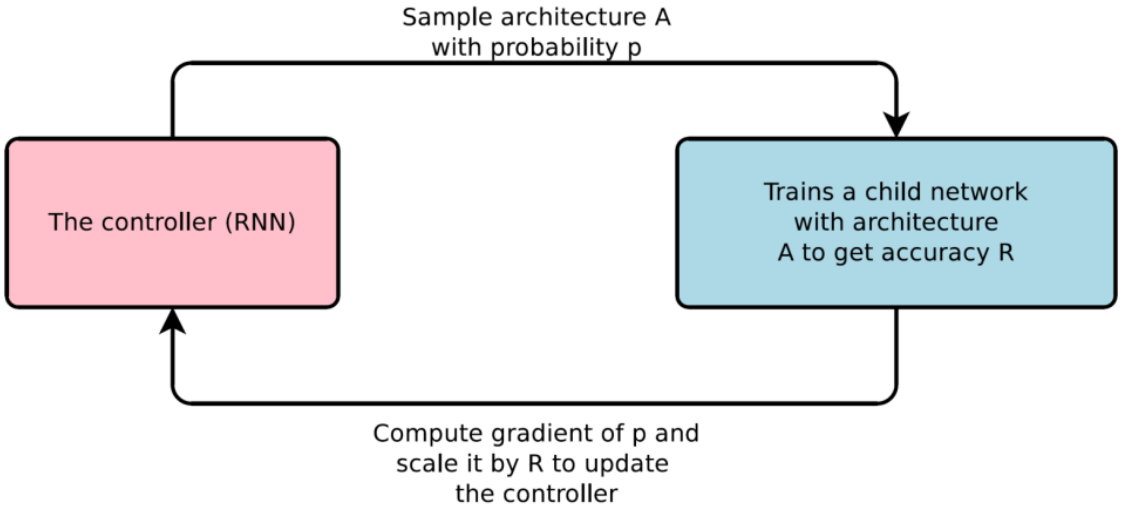
- Directed Acyclic Graph (DAG) representation of architectures
- Each node a latent representation; each edge an operation/motif
- Multiple levels of hierarchy for motifs; e.g. $o_1^{(2)}$ is a level-2 motif, $o_1^{(3)}$ is a level-3 motif



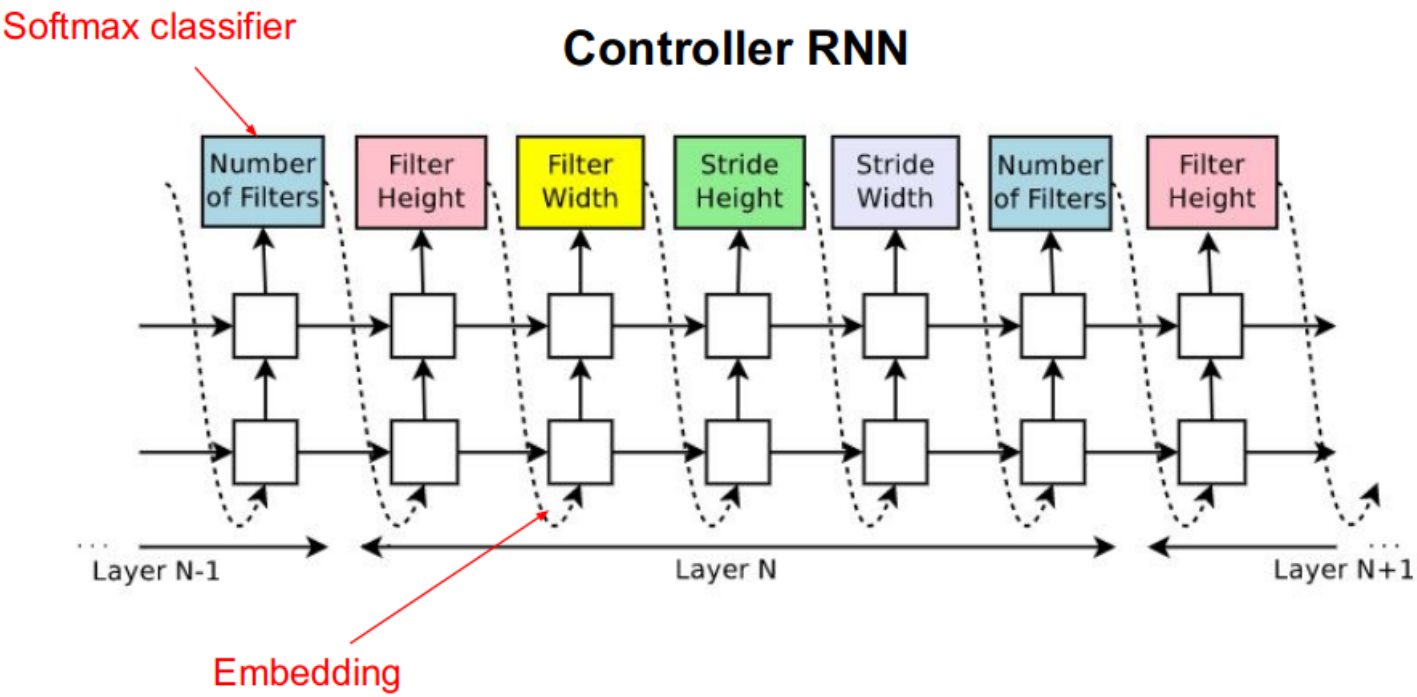
Lecture Overview

Reinforcement Learning [Zoph & Le, ICLR 2017]

- Key idea: Use a configuration string to represent the structure and connections in a NN
  - E.g.: ["Input to layer 1: 0", "Input to layer 2: 0", "Layer 2: conv"]
- Use RNN ("Controller") to generate this string that specifies the NN
- Train this NN ("Child Network") and see how it performs on validation set
- Use Reinforcement Learning (RL) to update the parameters of the Controller RNN based on the performance of child models

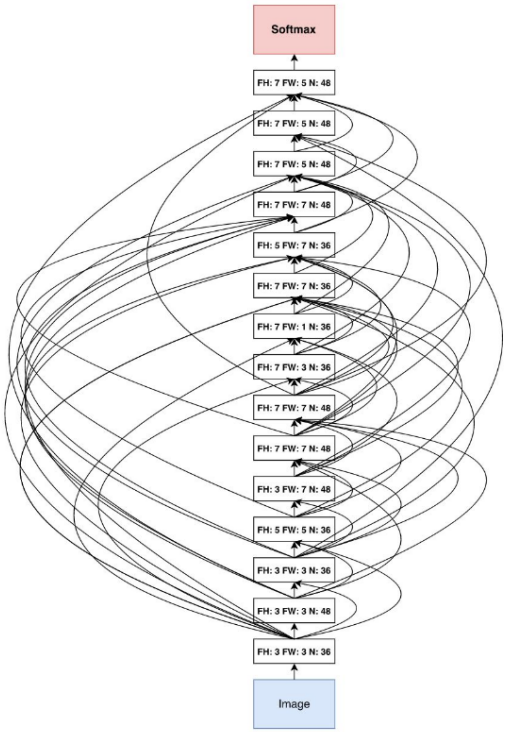


Learning CNNs with RL [Zoph & Le, '17]



- Maximize expected reward (validation accuracy): $J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$
- For a fixed number of layers predict:
  - Filter width/height, stride width/height, number of filters
- Each child model was trained for 50 epochs
- The final chosen architecture is trained for longer (600 epochs)

Learning CNNs with RL [Zoph & Le, '17]



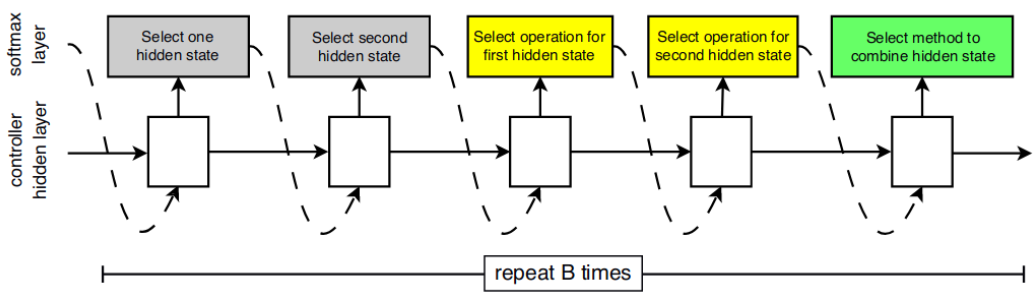
Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

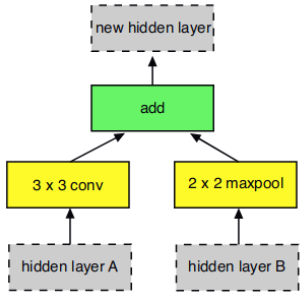
5% faster

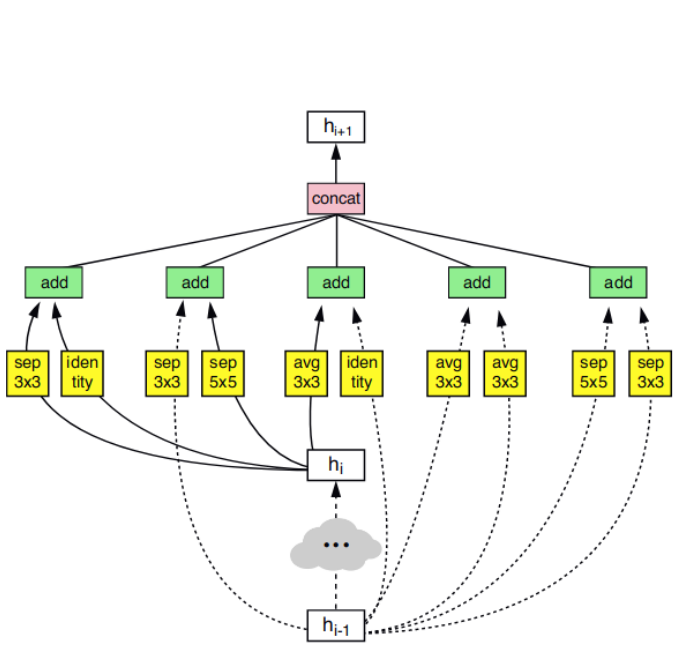
- State-of-the-art results for CIFAR-10, Penn Treebank architecture, optimizing the weights with SGD
- Large computational demands 800 GPUs for 2 weeks, 12.800 architectures evaluated
- Hyperparameter optimization only as postprocessing, which may potentially play an important role during search

Learning CNN cells with RL [Zoph et al. '18]

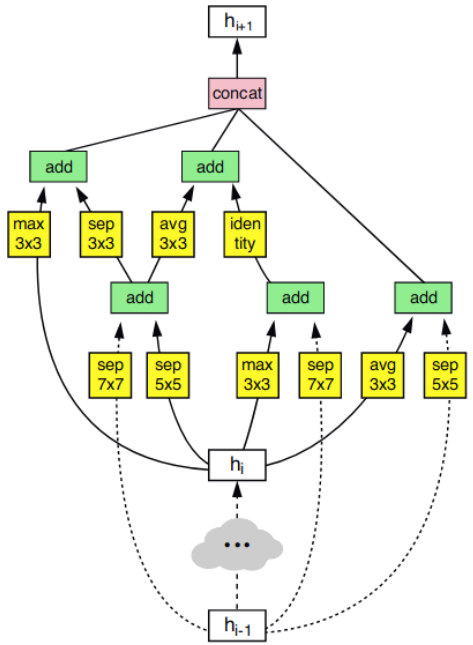
- RNN controller makes $2 \times 5B$ softmax predictions in total (5B for *normal* and 5B for *reduction* cells)
- Determines both topology and operations in each node of the graph







Normal Cell



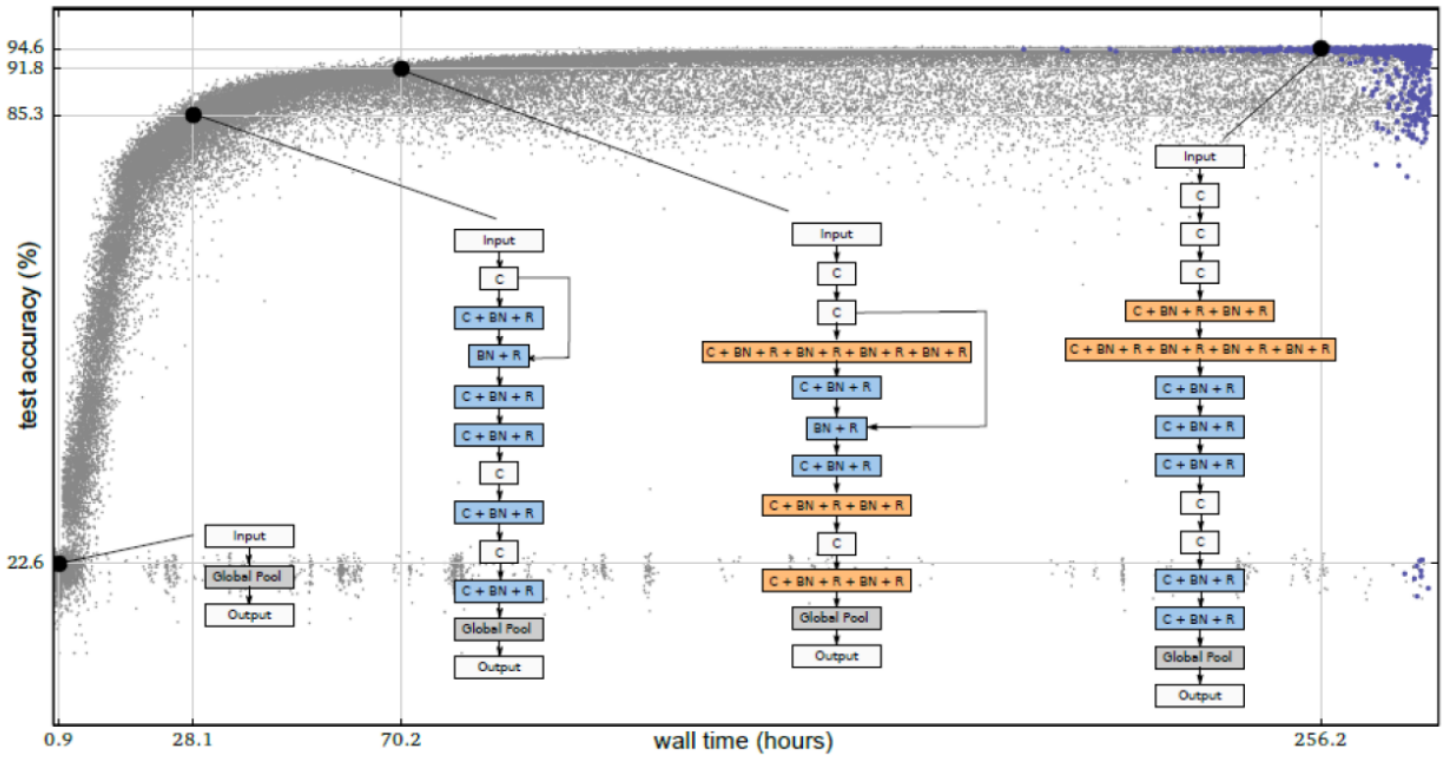
Reduction Cell

NAS as Hyperparameter Optimization

- Chain-structured space:
  - Hyperparameters of layer k conditional on depth $\geq k$
- Cell search space
  - Fixed (or maximum) number of layers per cell (e.g., 5)
  - For each: categorical choice between various operations, e.g., between {conv3x3, conv5x5, max pool, separable conv3x3, ...}

Evolution

- Neuroevolution (already since the 1990s)
  - Typically optimized both architecture and weights with evolutionary methods [Angeline et al. 1994; Stanley and Miikkulainen, 2002]
  - Mutation steps, such as adding, changing or removing a layer [Real et al. '17; Liu et al. '17; Miikkulainen et al. '17]

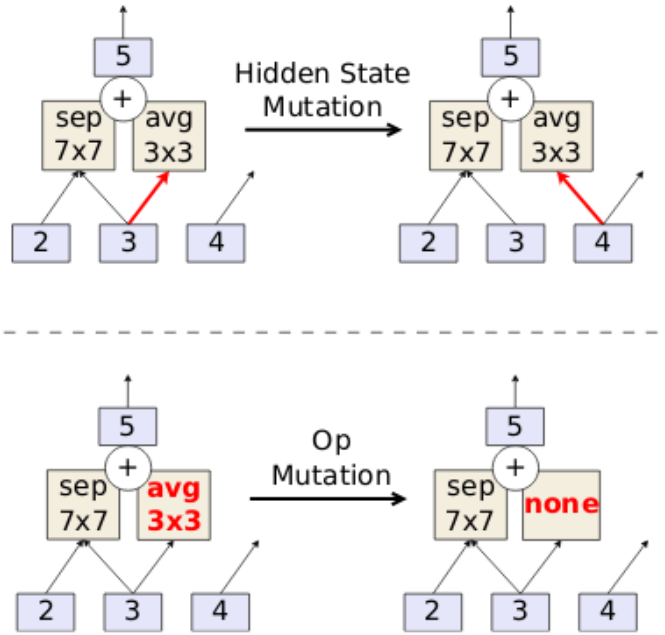


Regularized/Aging Evolution [Real et al. '19]

- Standard evolutionary algorithm, but oldest solutions are dropped from population P, instead of the worst.
- Same cell search space as in Zoph et al '18

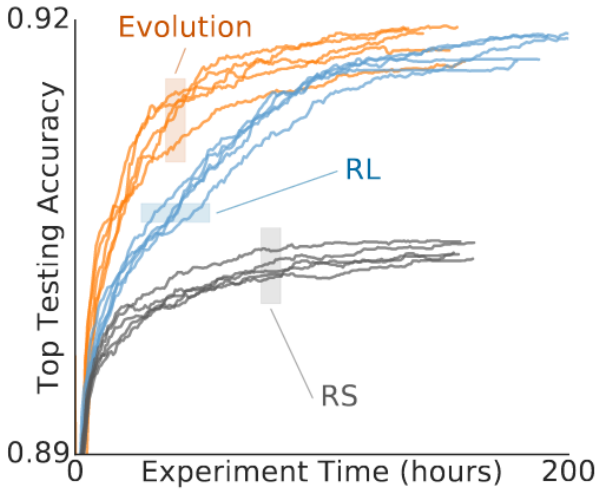
Algorithm 1 Aging Evolution

```
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
```



Regularized/Aging Evolution [Real et al. '19]

- Better results compared to the RL and RS baselines on CIFAR-10 (AmoebaNet-A).
- State-of-the-art results on CIFAR-10 when ran on larger search space (AmoebaNet-B/C)



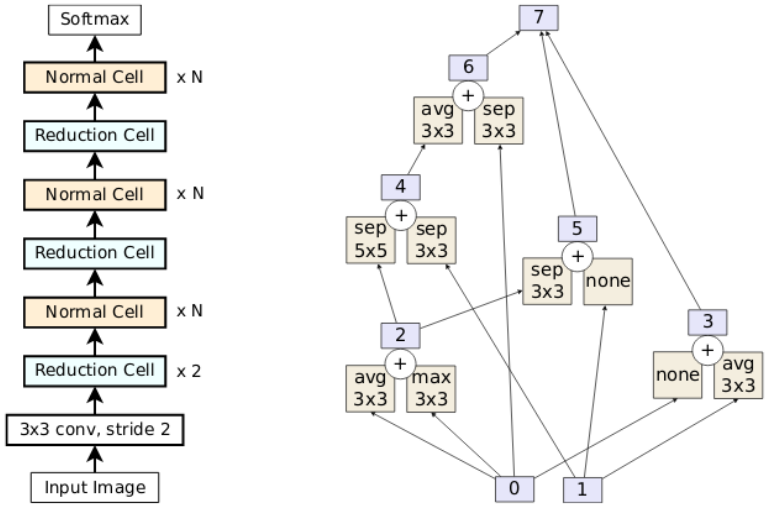


Figure 5: AmoebaNet-A architecture. The overall model [54] (LEFT) and the AmoebaNet-A normal cell (MIDDLE) and reduction cell (RIGHT).

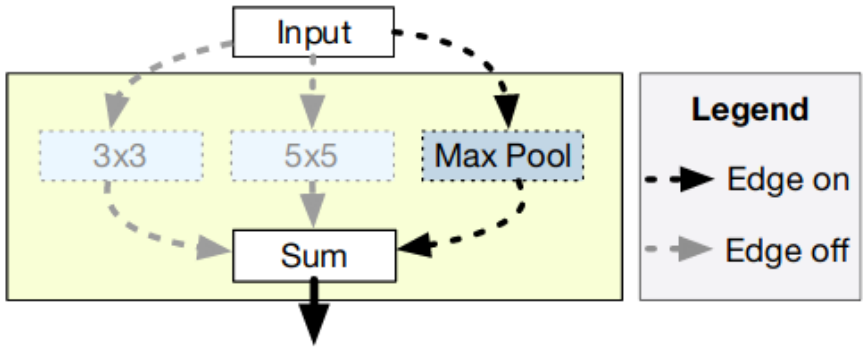
Lecture Overview

Efficient NAS by performance estimation speedup

Speed-up method	How are speed-ups achieved?	References
Lower fidelity estimates	Training time reduced by training for fewer epochs, on subset of data, downscaled models, downscaled data, ...	Li et al. (2017), Zoph et al. (2018), Zela et al. (2018), Falkner et al. (2018), Real et al. (2019), Runge et al. (2019)
Learning Curve Extrapolation	Training time reduced as performance can be extrapolated after just a few epochs of training.	Swersky et al. (2014), Domhan et al. (2015), Klein et al. (2017a), Baker et al. (2017b)
Weight Inheritance/ Network Morphisms	Instead of training models from scratch, they are warm-started by inheriting weights of, e.g., a parent model.	Real et al. (2017), Elsken et al. (2017), Elsken et al. (2019), Cai et al. (2018a.b)
One-Shot Models/ Weight Sharing	Only the one-shot model needs to be trained; its weights are then shared across different architectures that are just subgraphs of the one-shot model.	Saxena and Verbeek (2016), Pham et al. (2018), Bender et al. (2018), Liu et al. (2019b), Cai et al. (2019), Xie et al. (2019)

Weight Sharing and One-shot Models

- All possible architectures are subgraphs of a large supergraph (the one-shot model)
- Weights are shared between different architectures with common edges/nodes in the supergraph
- Search costs reduced drastically since one only has to train one model (the one-shot model).

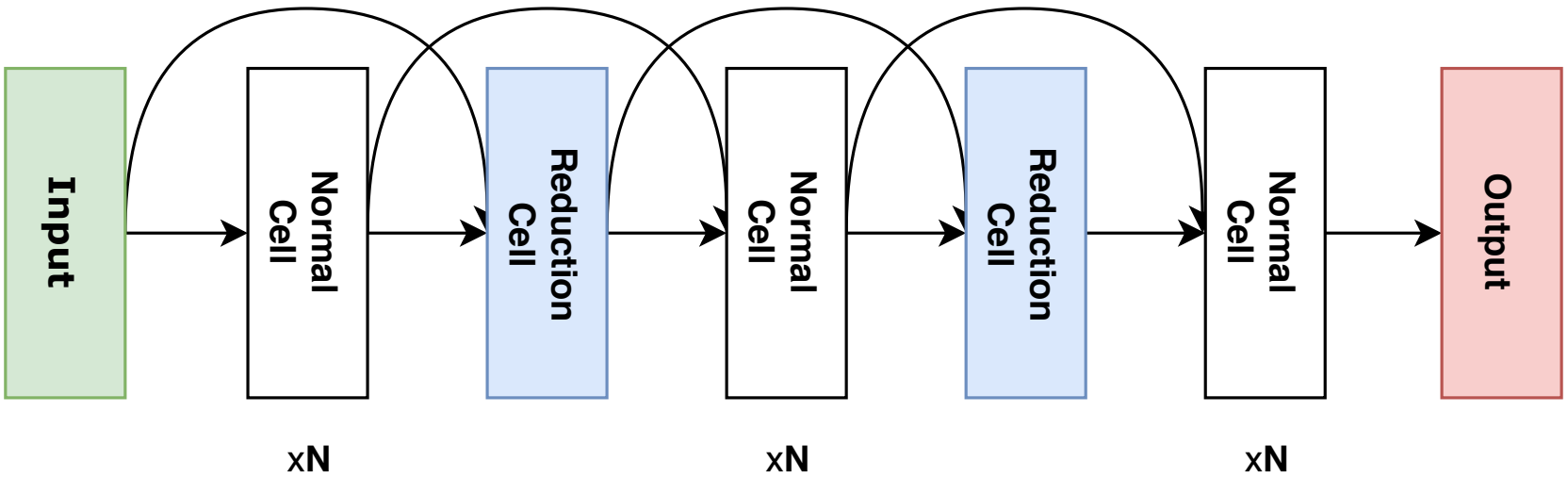


Weight Sharing and One-shot Models

- All possible architectures are subgraphs of a large supergraph (the one-shot model)
- Weights are shared between different architectures with common edges/nodes in the supergraph
- Search costs reduced drastically since one only has to train one model (the one-shot model).
- Different one-shot NAS methods differ in how the train the one-shot model and sample from it:
  - $\Rightarrow$ ENAS [Pham et al. '18] uses an RL controller to sample architectures from the supergraph and trains the one-shot model using on approximate gradients obtained through REINFORCE
  - $\Rightarrow$ Bender et al. ('18) only train the one-shot model and sample architectures in the end
  - $\Rightarrow$ DARTS (Differentiable Architecture Search; [Liu et al. '18]) creates a continuous relaxation of the discrete architecture space and optimizes the architecture distribution with gradient descent

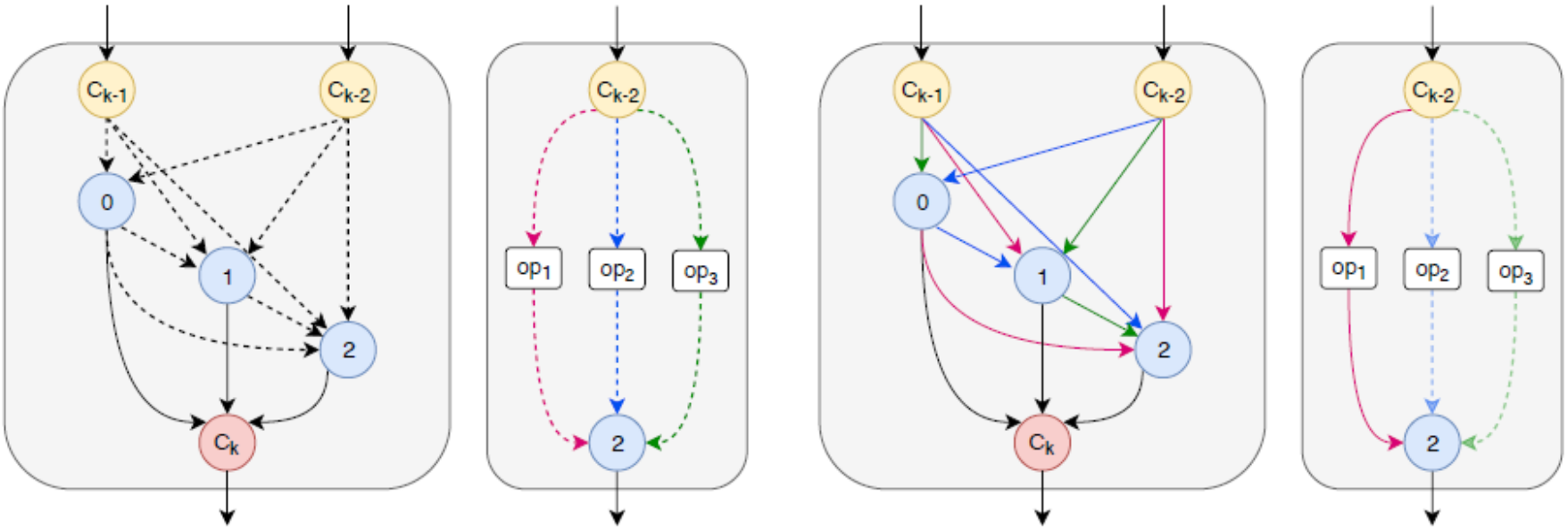
DARTS: Cell Search Space

- 2 cell types
  - $\Rightarrow$ Normal - keep spatial resolution unchanged.
  - $\Rightarrow$ Reduction - reduce spatial resolution by a factor of 2.
- Cell C_k gets as input the output of cells C_{k-1} and C_{k-2}



DARTS: Cell Search Space

- DAG (Multigraph) representation
  - ⇒ Nodes - lattent representations.
  - ⇒ Edges (dashed) - operations.
- Architecture optimization problem: Find optimal path from the input to the output cell

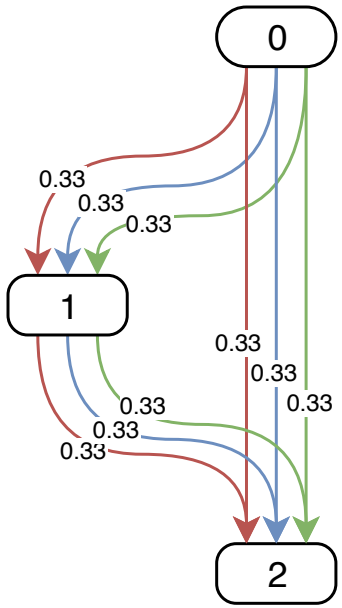


(a) One-shot search

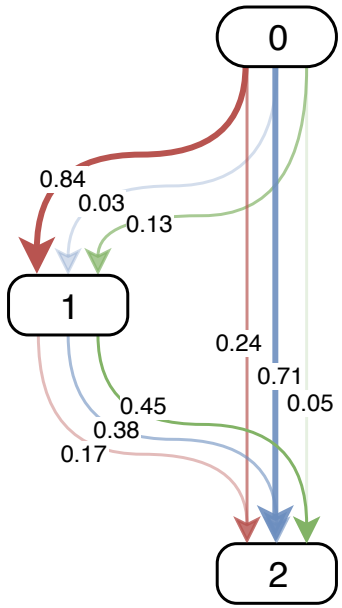
(b) Final evaluation

DARTS: Continuous Relaxation

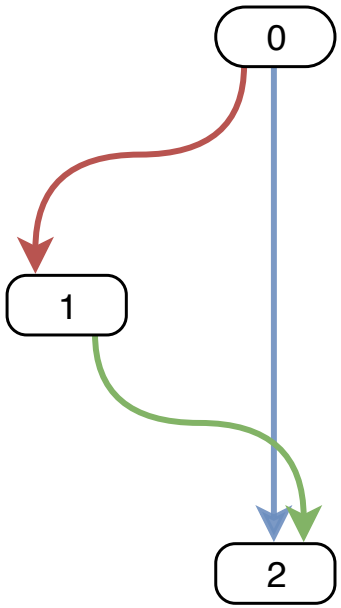
- $x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{e^{\alpha_o^{(i,j)}}}{\sum_{o' \in \mathcal{O}} e^{\alpha_{o'}^{(i,j)}}} o(x^{(i)})$
- $o^{(i,j)} = \arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$
- 8 operations: $sep\_conv\_3\times3$, $sep\_conv\_5\times5$, $dil\_conv\_3\times3$, $dil\_conv\_5\times5$, $max\_pool\_3\times3$, $avg\_pool\_3\times3$, *identity* and *zero*



(a) Initialization



(b) Search end



(c) Final cell

DARTS: Architecture Optimization

Algorithm 1: DARTS

Input Operations $\mathcal{O}$, #nodes n, #cells K, #channels C, ξ, learning rates η_1, η_2

`model` $\leftarrow$ `stack`$\big(DAG(\mathcal{O}, n), K, C\big)$

while *not converged* **do**

Update one-shot weights $\mathbf{w}$ by $\mathcal{L}_{train}(\mathbf{w}, \alpha)$

Update architectural parameters α by $\nabla_\alpha \mathcal{L}_{valid}(\mathbf{w} - \xi \nabla_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \alpha), \alpha)$

end while

Output α

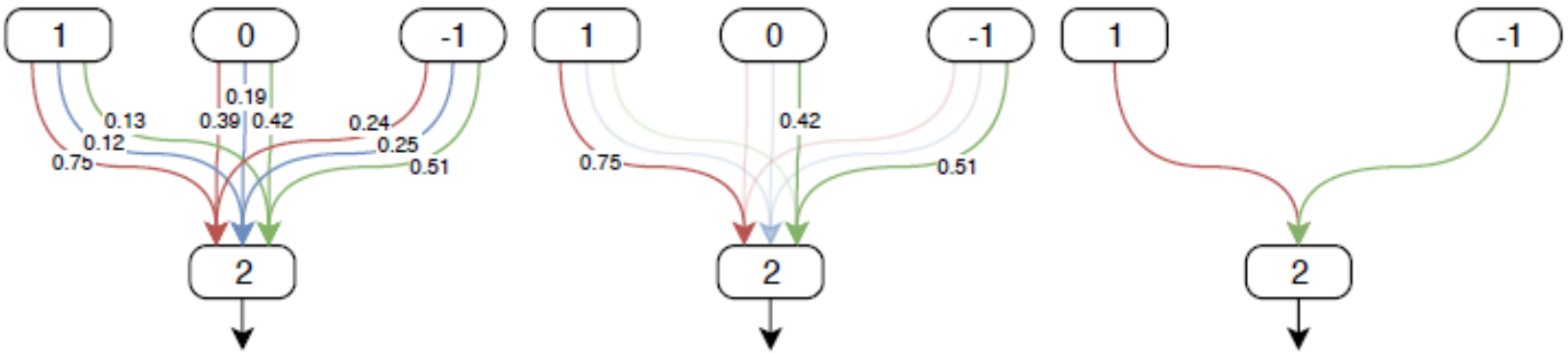
- Optimizing both $\mathcal{L}_{train}$ and $\mathcal{L}_{valid}$ corresponds to a bilevel optimization problem:

$$\min_\alpha \{ f(\alpha) \triangleq \mathcal{L}_{valid}(w^*(\alpha), \alpha) \}$$
$$s.t. \quad w^*(\alpha) = \arg\min_w \mathcal{L}_{train}(w, \alpha)$$

- Approximation: $w^*(\alpha) \approx w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha)$
  1. First-order approximation: $\xi = 0$
  2. Second-order approximation: $\xi > 0$

DARTS: Deriving the final cell

- After search finishes keep $\arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ as the optimal (non-zero) operation connecting node *i* to *j*
- Following Zoph et al, '18, keep top-2 incoming operations in each node



(a) $Softmax(\alpha)$ values per operation after optimization

(b) Select strongest operation between two nodes

(c) Select top-2 from 3 incoming edges based on α

DARTS: Results on CIFAR-10

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10 (lower error rate is better). Note the search cost for DARTS does not include the selection cost (1 GPU day) or the final evaluation cost by training the selected architecture from scratch (1.5 GPU days).

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	8	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	6	RL
ENAS + cutout (Pham et al., 2018b)*	2.91	4.2	4	6	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	7	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	7	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	7	gradient-based

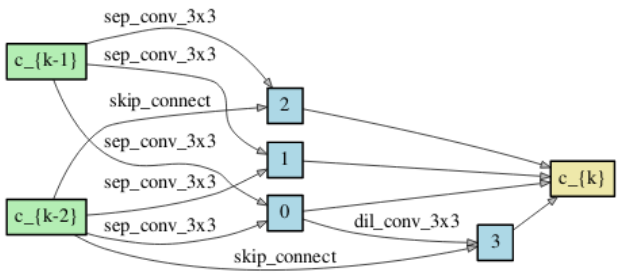


Figure 4: Normal cell learned on CIFAR-10.

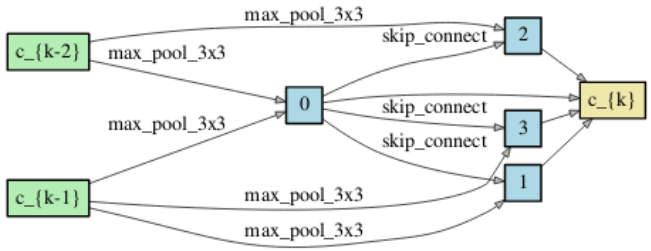
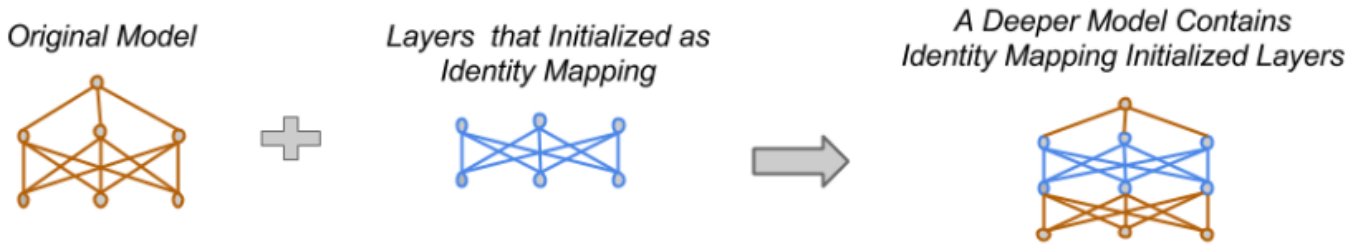


Figure 5: Reduction cell learned on CIFAR-10.

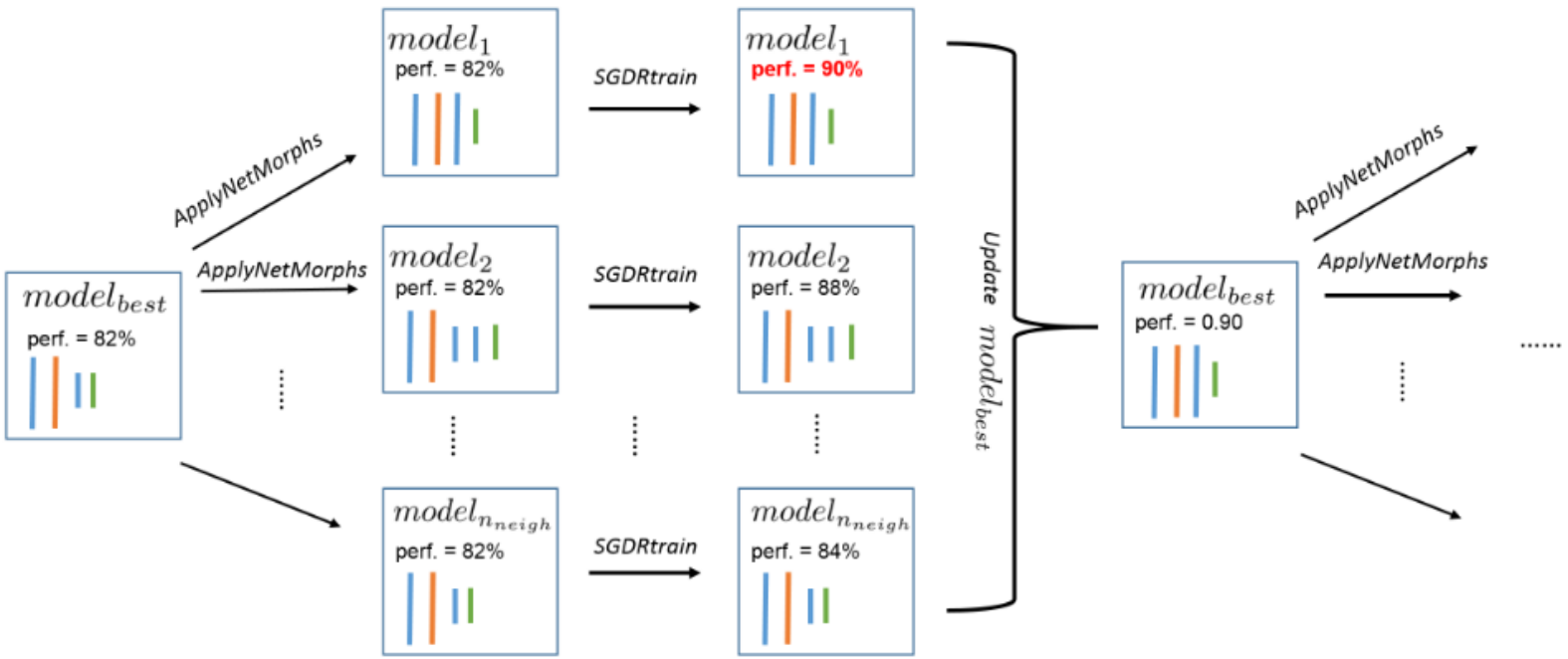
Network Morphisms and Weight Inferitance

- Network Morphisms [Chen et al. '16; Wei et al. '16; Cai et al. '17]
  - Change the network structure, but not the modelled function
  - i.e., for every input the network yields the same output as before applying some network morphisms operations, such as "Net2DeeperNet", "Net2WiderNet", etc.
  - Allows efficient moves in architecture space



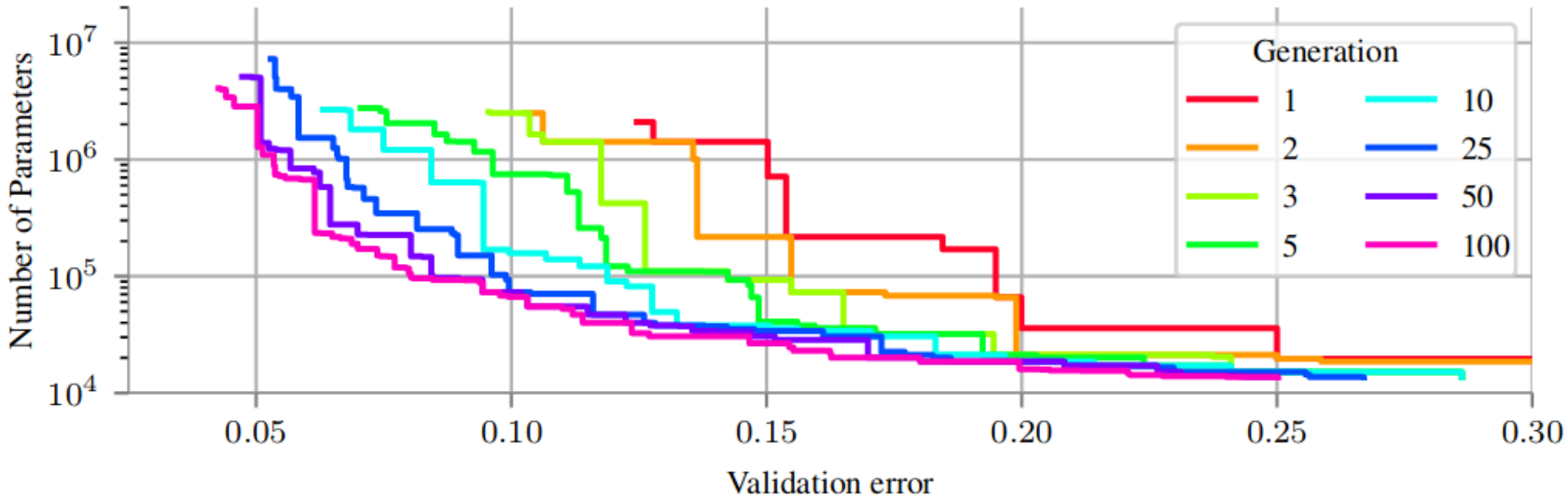
Network Morphisms and Weight Inferitance

- Weight Inheritance [Real et al. '17; Cai et al. '18; Elsken et al. '19]
  - Inherit already trained weights from parent architectures to child ones
  - Do not necessary require preserving the function as in Network Morphisms
  - Network Morphisms intrinsically fulfill this property



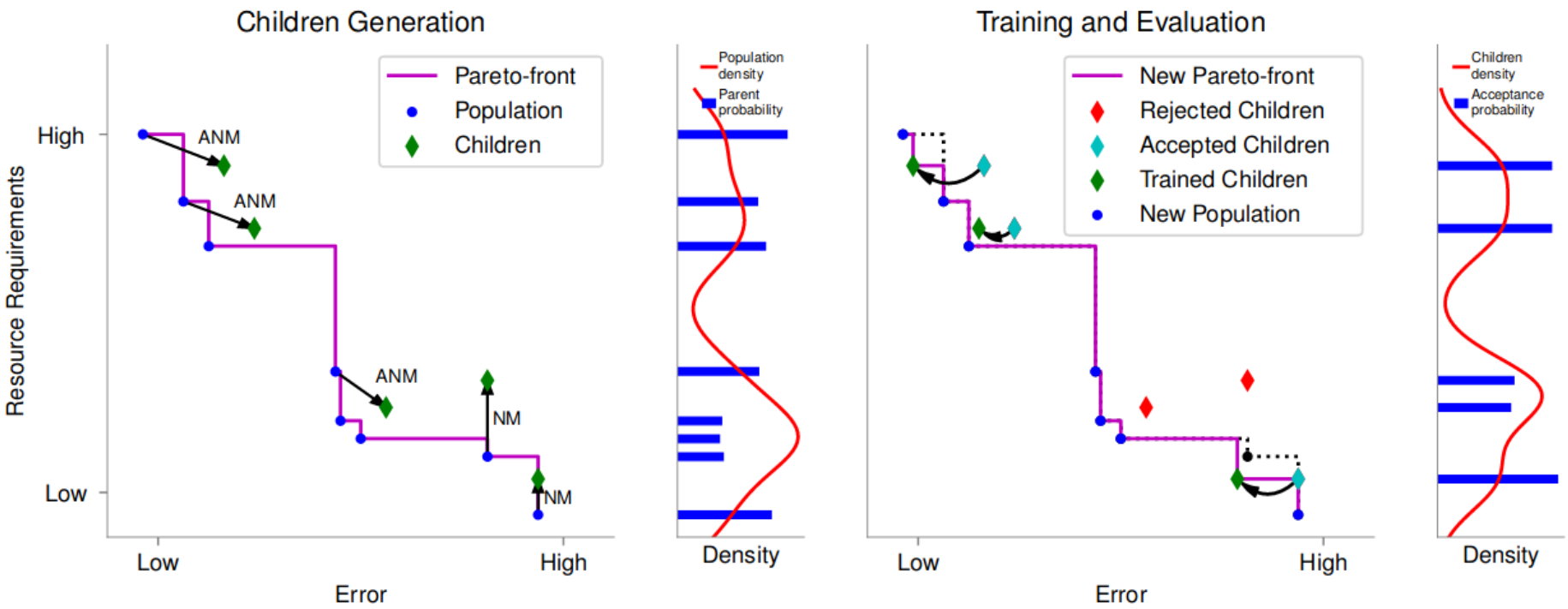
Efficient Multi-objective NAS [Elsken et al. '19]

- Trades-off network size vs. performance; maintain a Pareto front of the 2 objectives
- Evolve a population of Pareto-optimal architectures over time
- LEMONADE: Lamarckian Evolution for Multi-Objective Neural Architecture Design
  - Weight inheritance through network morphisms
  - Still cheap: 1 week on 8 GPUs



Efficient Multi-objective NAS [Elsken et al. '19]

- Children generation and training/evaluation of architectures



Other works that aim for efficiency

- DARTS follow-up works:
  - SNAS [Xie et al. '19], optimize the architecture by sampling discrete paths from the one-shot model
  - ProxylessNAS [Cai et al. '19]: memory-efficient variant of DARTS
- Graph Hypernetworks for NAS [Zhang et al. '19]
  - Use hypernetwork to generate the weights of the sampled architectures
- Approaches based on Random Search:
  - RandomNAS [Li and Talwalkar. '19]: Randomly samples architectures from the one-shot model.
  - Xie et al. (2019) exploit random connectivity in a large neural networks, showing that it is comparable to other architectures found by NAS methods on ImageNet.

Learning Goals

Now you are able to ...

- describe different search spaces for neural architecture search (NAS)
- describe various optimization methods for NAS
- explain various speedup techniques for NAS