NAS: Neural Architecture Search

Marius Lindauer

Leibniz Universität Hannover
Germany

/ LindauerMarius
m.lindauer@ai.uni-hannover.de

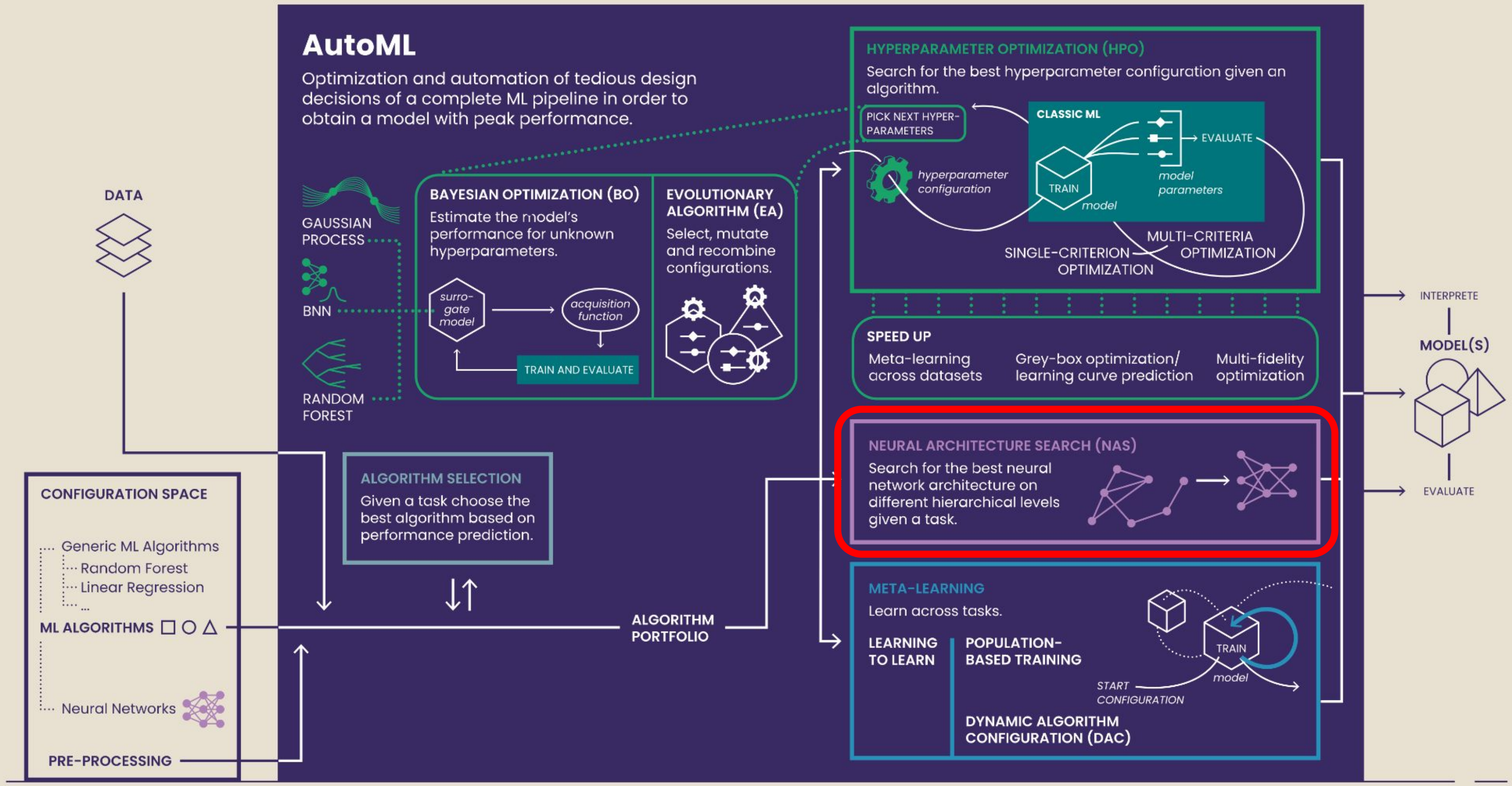
Katharina Eggensperger

Eberhard Karls Universität Tübingen
Germany

/ KEggensperger
katharina.eggensperger@uni-tuebingen.de

Questions?

→ Day 3!


AutoML

Optimization and automation of tedious design decisions of a complete ML pipeline in order to obtain a model with peak performance.

HYPERPARAMETER OPTIMIZATION (HPO)
Search for the best hyperparameter configuration given an algorithm.

BAYESIAN OPTIMIZATION (BO)
Estimate the model's performance for unknown hyperparameters.

EVOLUTIONARY ALGORITHM (EA)
Select, mutate and recombine configurations.

SPEED UP
Meta-learning across datasets
Grey-box optimization/ learning curve prediction
Multi-fidelity optimization

NEURAL ARCHITECTURE SEARCH (NAS)
Search for the best neural network architecture on different hierarchical levels given a task.

META-LEARNING
Learn across tasks.
LEARNING TO LEARN
POPULATION-BASED TRAINING
DYNAMIC ALGORITHM CONFIGURATION (DAC)

ALGORITHM SELECTION
Given a task choose the best algorithm based on performance prediction.

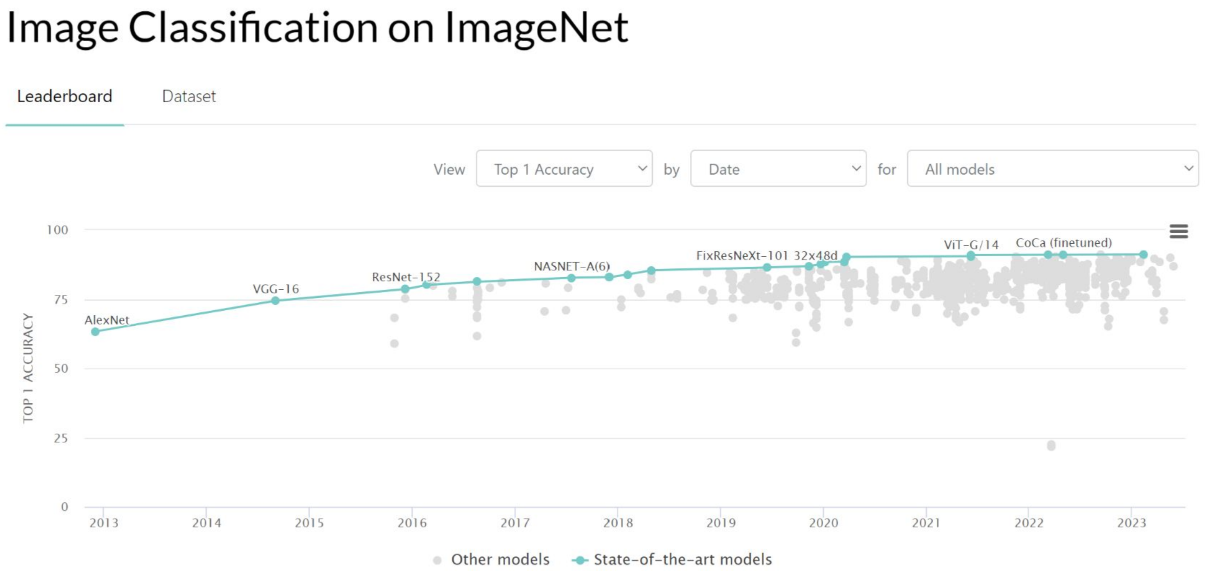
CONFIGURATION SPACE
- Generic ML Algorithms
  - Random Forest
  - Linear Regression
  - ...
- ML ALGORITHMS □ ○ △
- Neural Networks
- PRE-PROCESSING

Story Line Today

- Can we do more than simply configuring hyperparameters by also taking the architectures of DNNs into consideration?
- How can we model architecture search spaces efficiently?
- How to optimize architectures efficiently?
- Can we combine model training and architecture search?
- Can we even approximate the quality of an architecture without training at all?

Development of new Neural Architectures

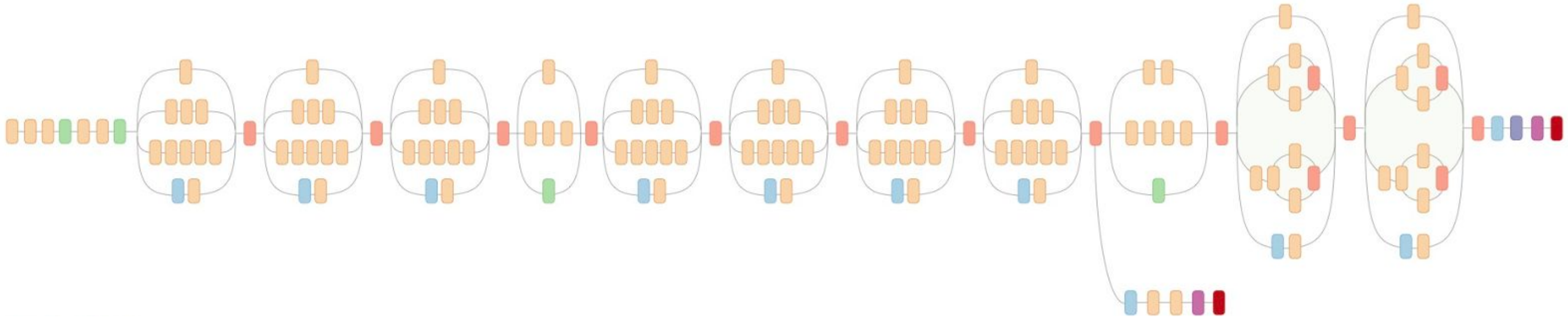
- Performance improvements on various tasks due to novel architectures
- Can we automate this design process, potentially discovering new components/topologies?



Source: Paperswithcode

Manual Development of Architectures

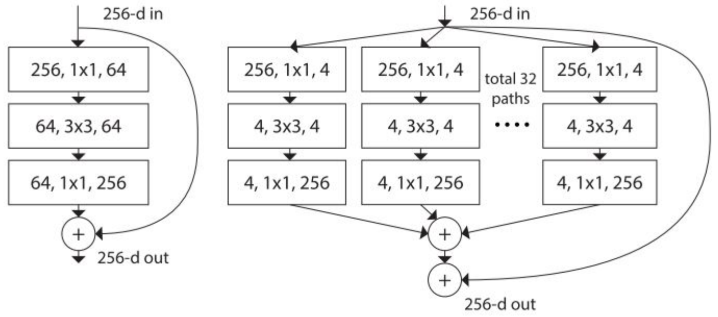
- Manual design of architectures is time consuming
- Complex state-of-the-art architectures are a result of years of trial and errors by experts



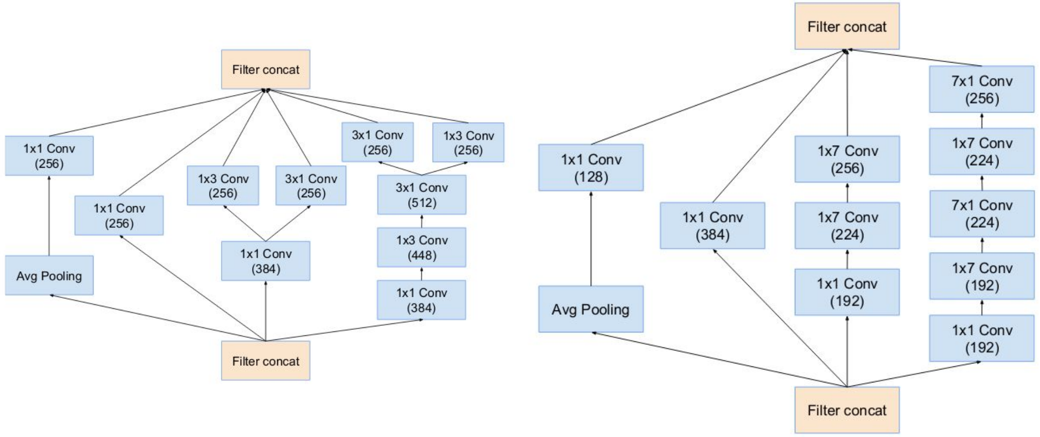
Inception-v3 [Szegedy et al. 2015]

Manual Development of Architectures

- Manual design of architectures is time consuming
- Complex state-of-the-art architectures are a result of years of trial and errors by experts



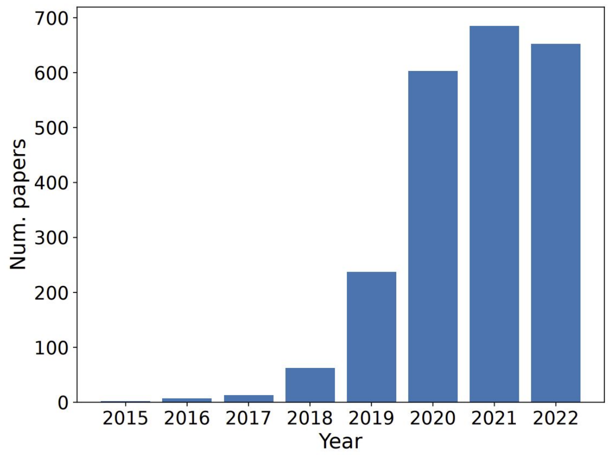
ResNet/ResNeXt blocks
[He et al. 2016; Xie et al. 2016]



Inception-v4 blocks [Szegedy et al. 2016]

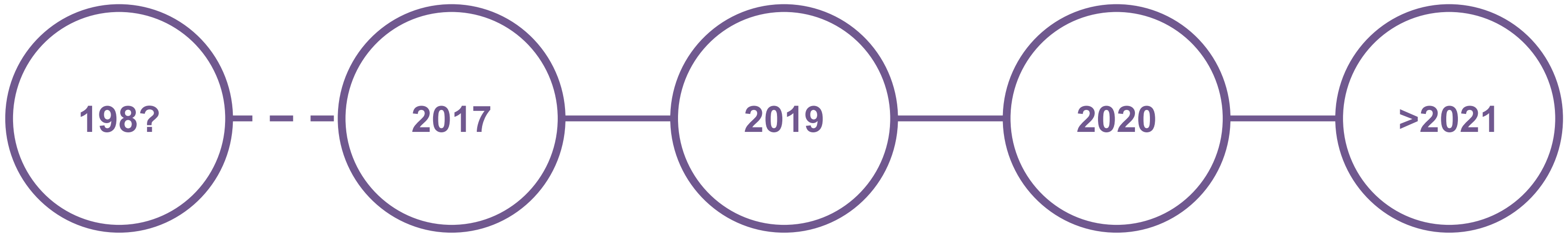
Neural Architecture Search

- **Goal**: automatically find neural architectures with strong performance
  - Optionally, subject to a resource constraint
- A decade-old problem, but main stream since 2017 and now intensely researched
- One of the main problems AutoML is known for
- Initially extremely expensive
- By now several methods promise low overhead over a single model training



Literature collection:
Deng & Lindauer

History of NAS



First NAS ideas, e.g. Tenorio and Wei-Tsih 1988

NAS with RL Zoph et al.

800 GPUs for 2 weeks for CIFAR-10

Differentiable NAS (DARTS) Liu et al.

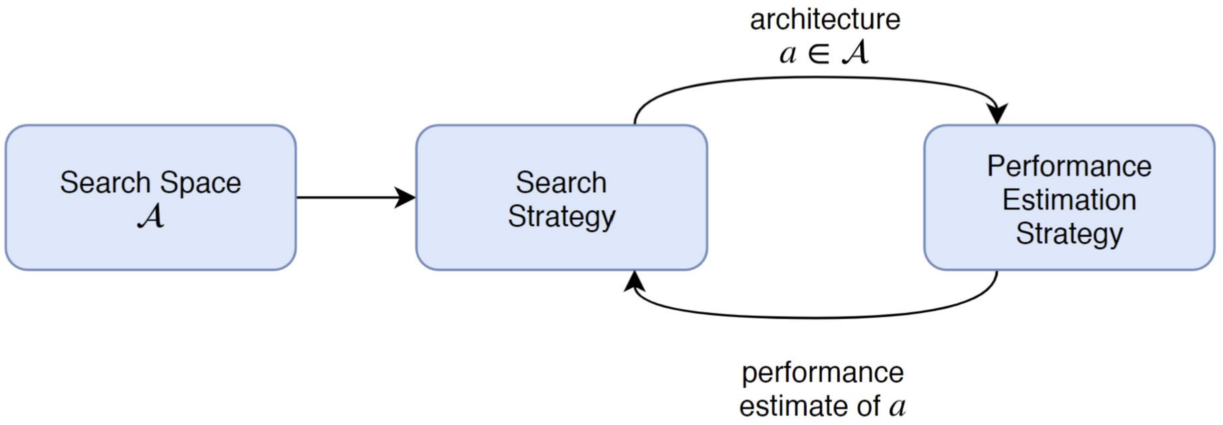
4 GPU days for CIFAR-10

Failure Modes of DARTS, e.g. Zela et al.

Zero-Cost Proxies, e.g. Abdellfatah et al.

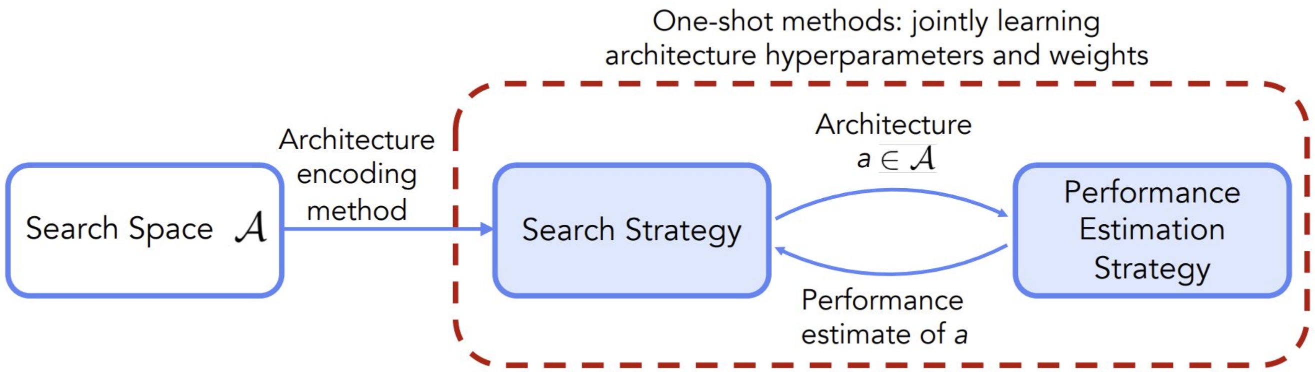
NAS in minutes?

NAS Components [Elsken et al. 2019]



- **Search Space**: the types of architectures we consider; micro, macro, hierarchical, etc.
- **Search Strategy:** Reinforcement learning, evolutionary strategies, Bayesian optimization, gradient-based, etc.
- **Performance Estimation Strategy**: validation performance, lower fidelity estimates, one-shot model performance, etc

NAS Components cont'd [White et al. 2023]


One-shot methods: jointly learning architecture hyperparameters and weights
Search Space $\mathcal{A}$
Architecture encoding method
Search Strategy
Architecture $a \in \mathcal{A}$
Performance Estimation Strategy
Performance estimate of a

- For **one-shot methods**, the search strategy and performance estimation strategy are inherently coupled.

NAS (Core) Definition

Given

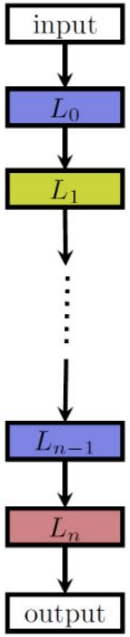
- a search space A ,
- a dataset D,
- a training pipeline P,
- and a time or computation budget t,

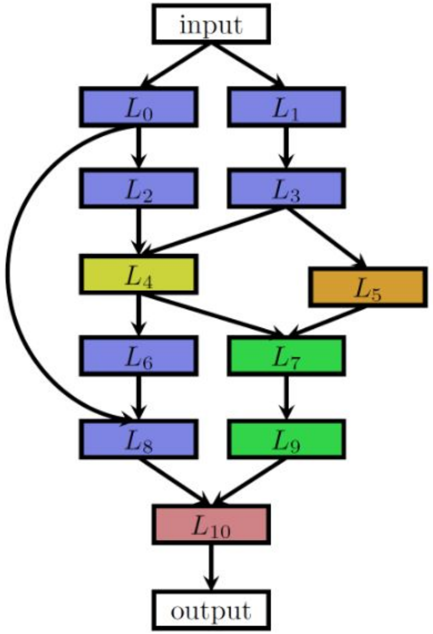
the goal is to find an architecture a ∈ A within budget t which has the highest possible validation accuracy when trained using dataset D and training pipeline P

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

NAS Search Spaces

Macro Search Space





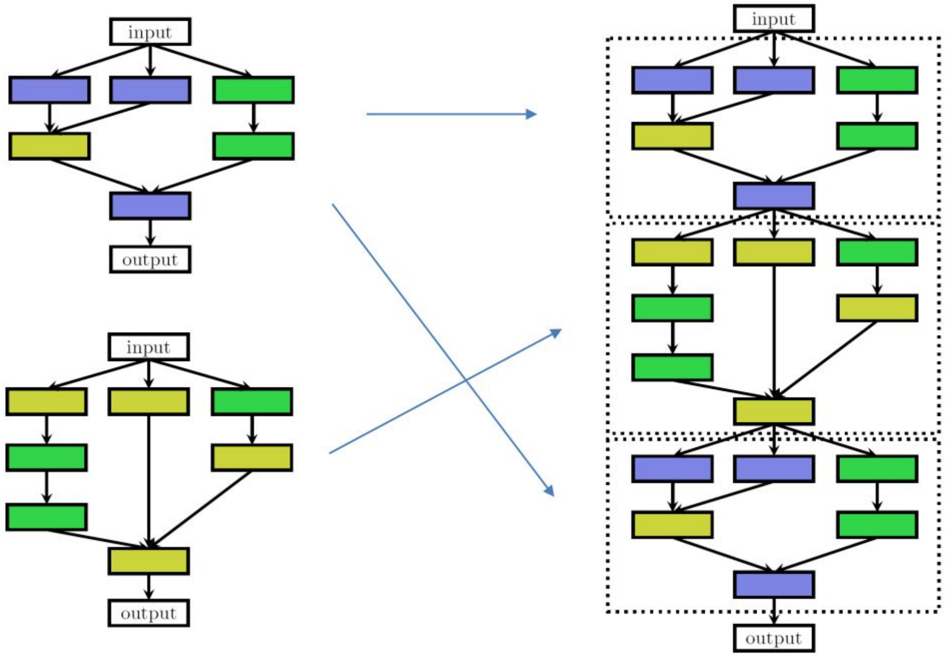
Chain-structured space (different colours: different layer types)

More complex space with multiple branches and skip connections

Cell Search Spaces [Zoph et al. 2018]

Normal cell: preserves spatial resolution

Reduction cell: reduces spatial resolution

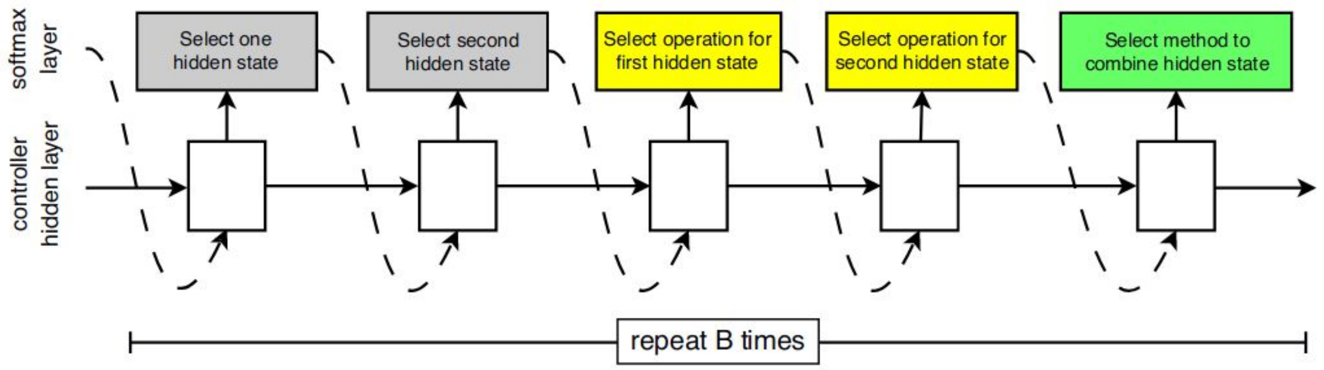


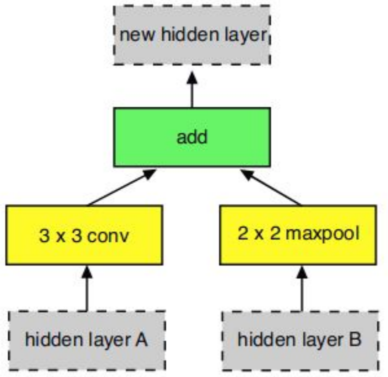
Two possible cells

Architecture composed of stacking together individual cells

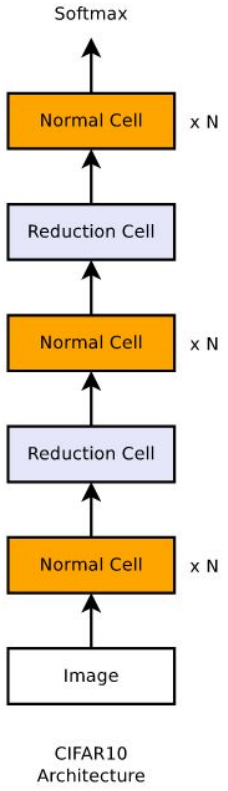
Details on Cell Search Spaces

- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
  - Choose two previous feature maps (from this cell)
  - For each of these, choose an operation (3×3 conv, max-pool, etc.)
  - Choose a merge operation to combine the two results (concat or add)

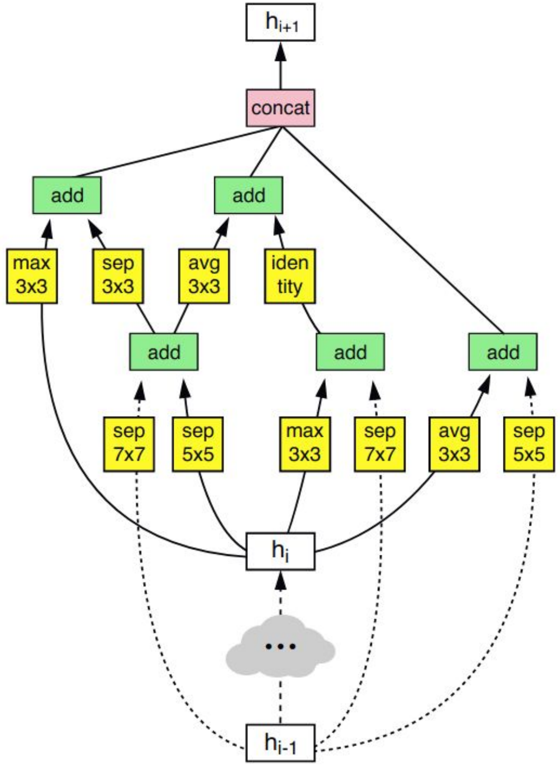

softmax layer
controller hidden layer
Select one hidden state
Select second hidden state
Select operation for first hidden state
Select operation for second hidden state
Select method to combine hidden state
repeat B times



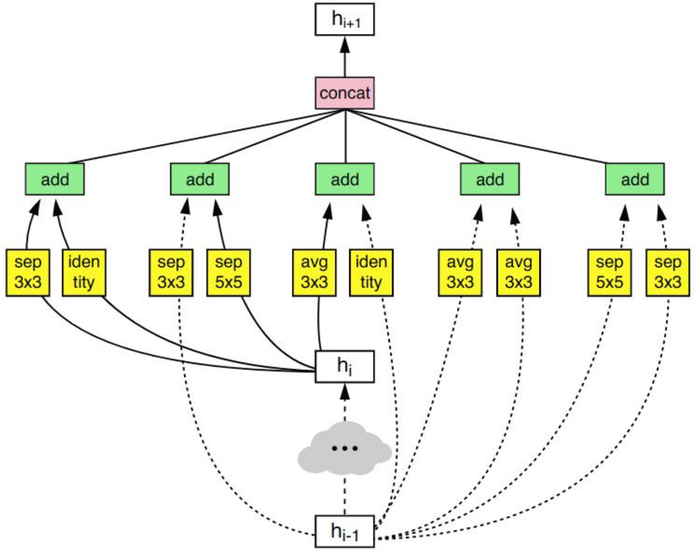
Example of an architecture sample with B=5



CIFAR10 Architecture



Reduction Cell



Normal Cell

What are pros and cons of cell search spaces?

Pros of Cell Search Space

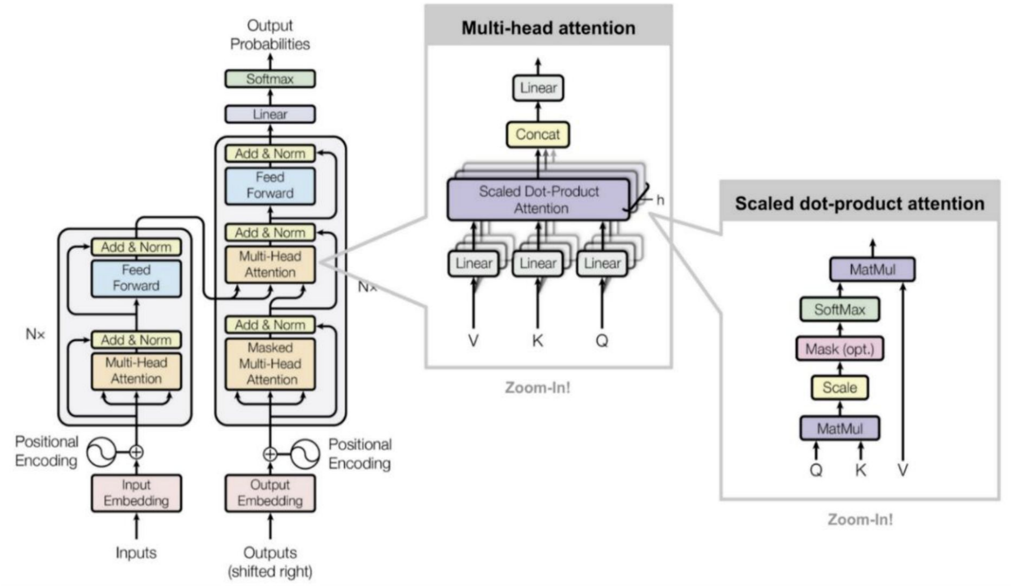
- Reduced search space size
- speed-ups in terms of search time
- Transferability to other datasets
  (e.g., cells found on CIFAR-10 transfer to ImageNet)
  - be careful with that!
- Stacking repeating patterns is proven to be a useful design principle
  (ResNet, Inception, etc.)

Cons of Cell Search Spaces

- Still need to (manually) determine the macro architecture, i.e., the way in which cells are connected.
- Limiting if different cells work better in different parts of the network
  - E.g., different spatial resolutions may favour different convolutions

Hierarchical Search Spaces

- Reuse of substructures, like in cell search spaces
- But choices on many different levels
- Some examples in the literature, but understudied
- Potential example for an element of a hierarchical space: **Transformers**



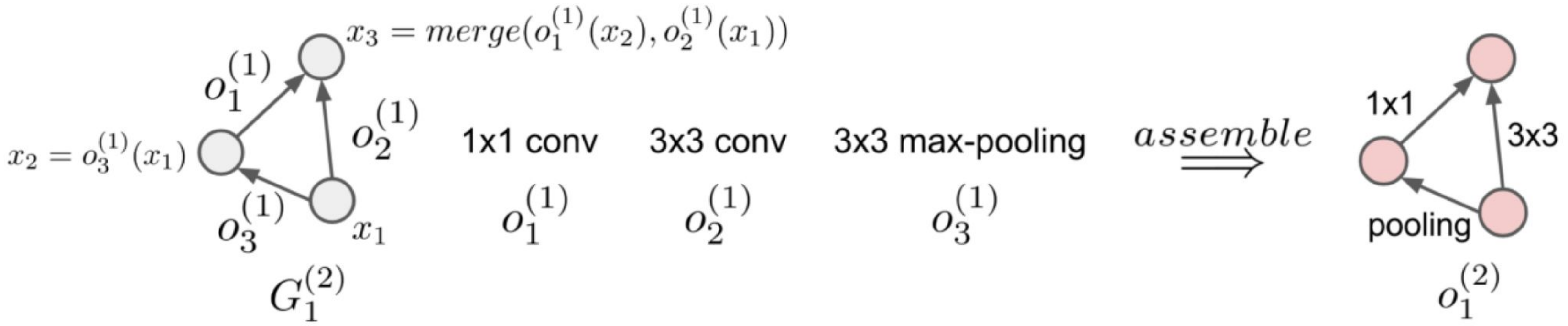
Hierarchical representation of search space [Liu et al. 2017]

Directed Acyclic Graph (DAG) representation of architectures

Each node is a latent representation; each edge is an operation/motif

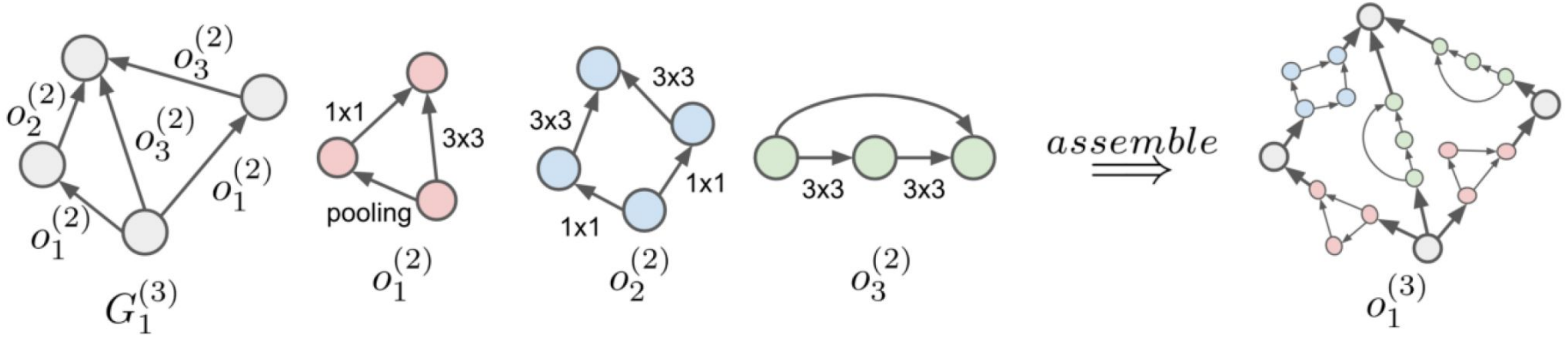
There are different levels of motivs

- Level-1 primitives: standard operators; e.g., 3x3 conv, max pooling, . . .
- Level-2 motivs: combinations of level-1 primitives



Hierarchical representation of search space [Liu et al. 2017]

- Level-3 motivs: combinations of level-2 motivs



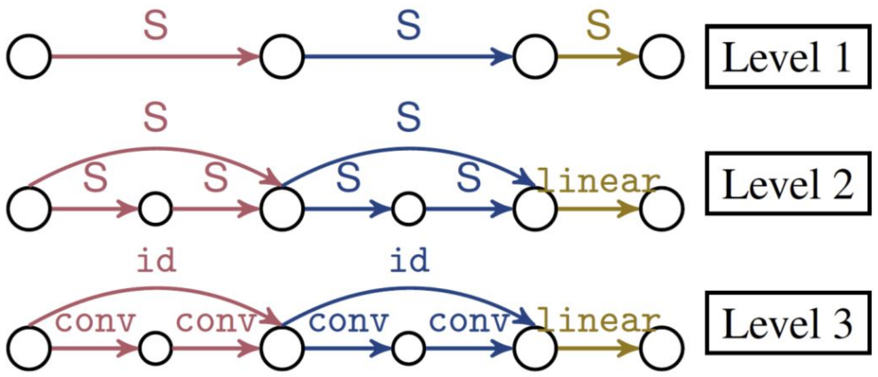
Formulation of Search Spaces by Context-free Grammars

- Choices on multiple levels of the architecture
- Can be described by context-free grammars [Schrodi et al. 2022]
- At each level, we apply another production rule to add more detail
- Allows for **compact and powerful representation** of architectures

S → Linear(S, S, S)

→ Linear(Residual(S, S, S),
Residual(S, S, S), linear)

→ Linear(Residual(conv, id, conv),
Residual(conv, id, conv), linear)

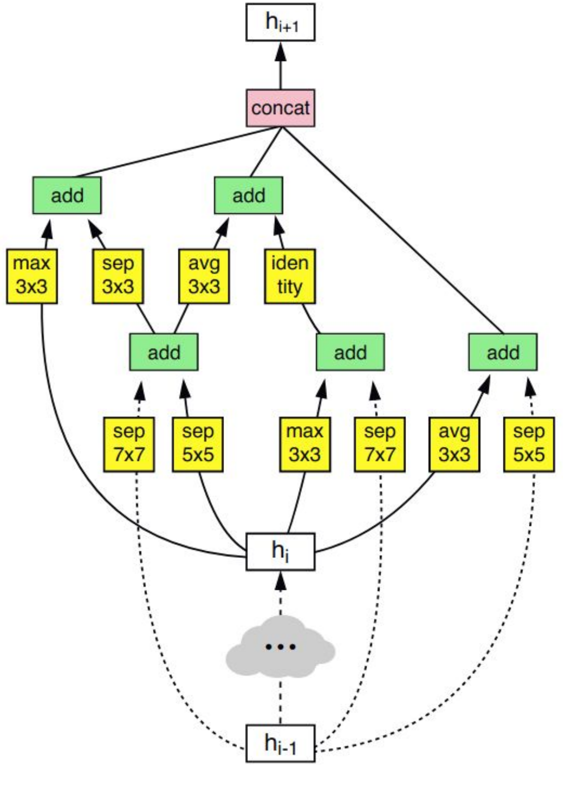


Search Methods for NAS: **Black-Box**

NAS as an HPO-Problem

NAS can be formulated as a HPO problem

- E.g., cell search space by [Zoph et al. 2018] has 5 categorical choices per block
  - 2 categorical choices of hidden states
  - 2 categorical variables choosing between operations
  - 1 categorical variable choosing the combination method
- Total number of hyperparameters for the cell: 5B (with B=5 by default)
- In general: one may require conditional hyperparameters
  E.g., chain-structured search space
  - Top-level hyperparameter: number of layers L
  - Hyperparameters of layer k conditional on L ≥ k



Reduction Cell

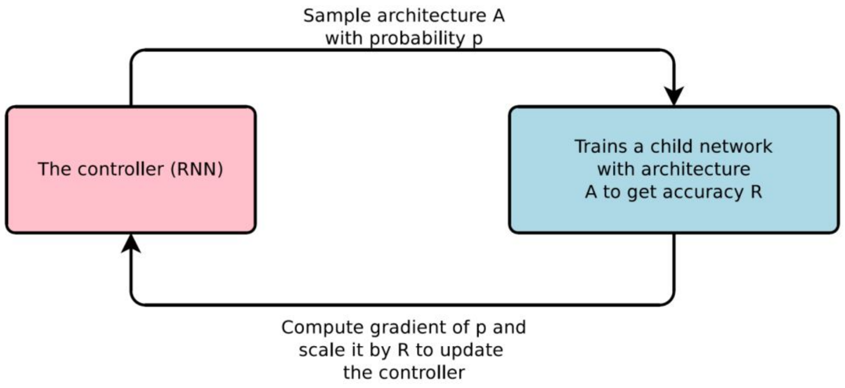
Early Work on Neuroevolution (already since the 1990s)

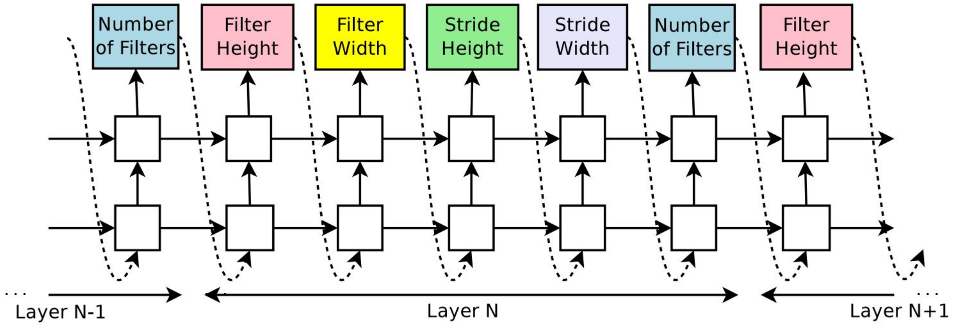
Evolves architectures & often also their weights

- Typical approach:
  - Initialize a population of N random architectures
  - Sample N individuals from that population (with replacement) according to their fitness
  - Apply mutations to those N individuals to produce the next generation’s population
  - Optionally: elitism to keep best individuals in the population
- Mutations include adding, changing or removing a layer

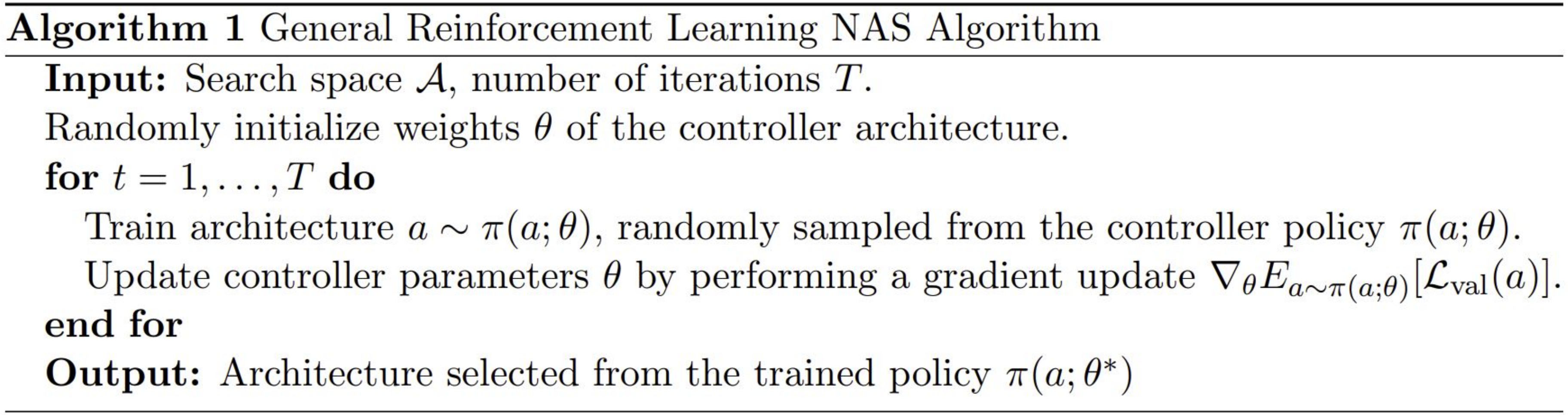
NAS by Reinforcement Learning [Zoph et al. 2017]

- Use RNN (“Controller”) to generate a NN architecture **piece-by-piece**
- Train this NN (”Child Network”) and evaluate it on a validation set
- Use Reinforcement Learning (RL) to update the parameters of the Controller RNN to optimize the performance of the child models
- **⇒ Outdated and suboptimal use of Reinforcement Learning**


Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller



NAS by Reinforcement Learning [White et al. 2023]

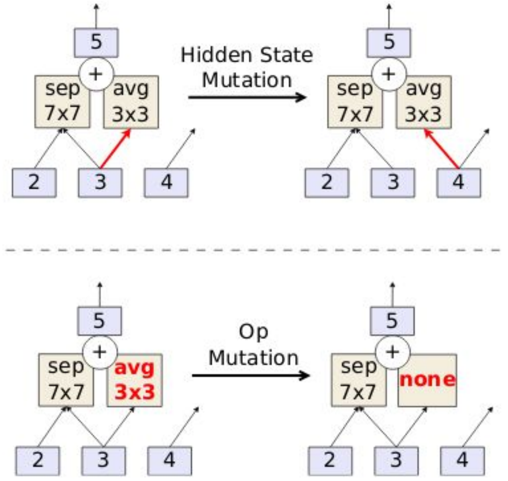


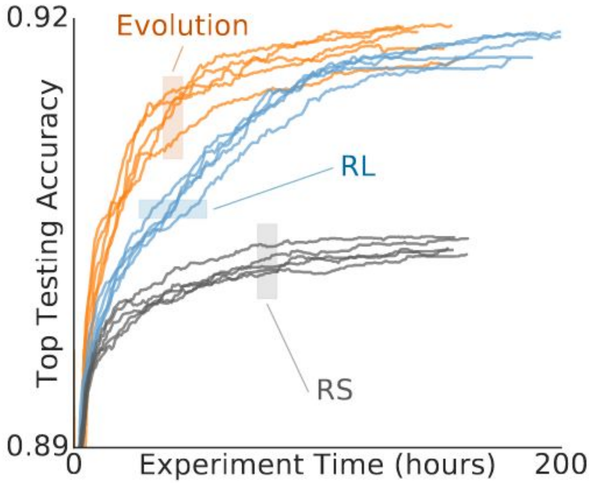
Algorithm 1 General Reinforcement Learning NAS Algorithm

```
Input: Search space $\mathcal{A}$, number of iterations $T$.
Randomly initialize weights $\theta$ of the controller architecture.
for $t = 1, \ldots, T$ do
    Train architecture $a \sim \pi(a;\theta)$, randomly sampled from the controller policy $\pi(a;\theta)$.
    Update controller parameters $\theta$ by performing a gradient update $\nabla_\theta E_{a\sim\pi(a;\theta)}[\mathcal{L}_{\text{val}}(a)]$.
end for
Output: Architecture selected from the trained policy $\pi(a;\theta^*)$
```

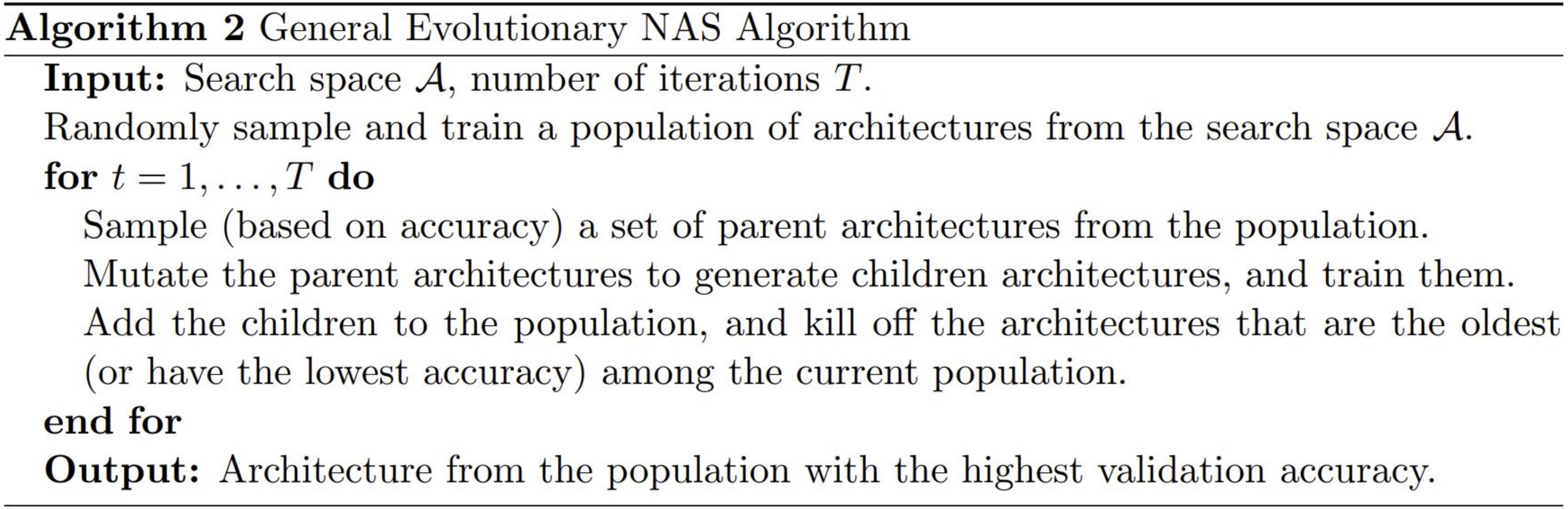
Regularized Evolution [Real et al. 2018]

- Quite standard evolutionary algorithm
  - But oldest solutions are dropped from population, instead of the worst (a.k.a. “regularization”)
- Standard SGD for training weights; fixed-length (HPO) search space





NAS by Evolutionary Algorithms [White et al. 2023]

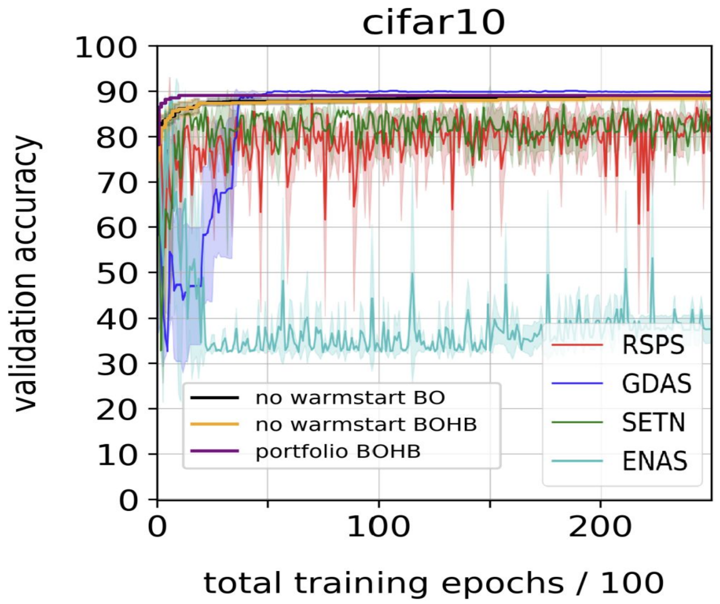


Algorithm 2 General Evolutionary NAS Algorithm

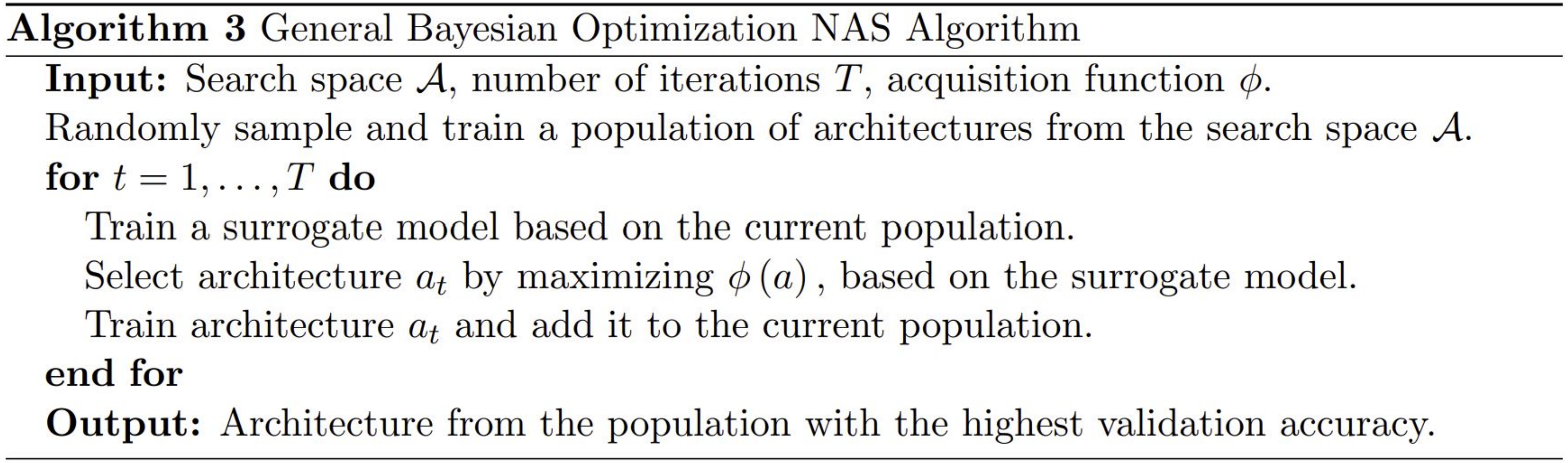
```
Input: Search space $\mathcal{A}$, number of iterations $T$.
Randomly sample and train a population of architectures from the search space $\mathcal{A}$.
for $t = 1, \dots, T$ do
    Sample (based on accuracy) a set of parent architectures from the population.
    Mutate the parent architectures to generate children architectures, and train them.
    Add the children to the population, and kill off the architectures that are the oldest
    (or have the lowest accuracy) among the current population.
end for
Output: Architecture from the population with the highest validation accuracy.
```

Bayesian Optimization

- Encode the architecture space by categorical hyperparameters
- Strong performance with tree-based models
  - TPE [Bergstra et al. 2013]
  - SMAC3 [Zimmer et al. 2021, Lindauer et al. 2022]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al. 2013]
  - NASBOT [Kandasamy et al. 2018]
- There are also several promising BO approaches based on NN
  - BANANAS [White et al. 2019]
  - Local search seems to be more important than encoding of architecture encoding [Schneider et al. 2022]
- BO is very competitive [Zimmer et al. 2021]
  - has also outperformed RL [Ying et al. 2019]



NAS by Bayesian Optimization [White et al. 2023]

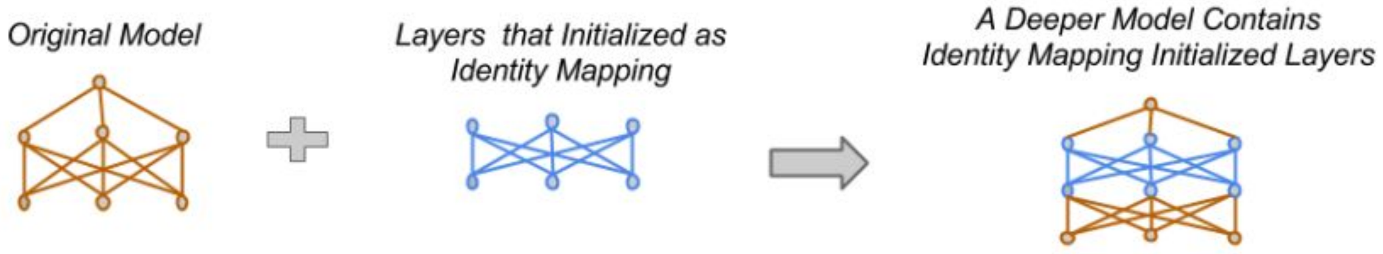


Algorithm 3 General Bayesian Optimization NAS Algorithm

```
Input: Search space $\mathcal{A}$, number of iterations $T$, acquisition function $\phi$.
Randomly sample and train a population of architectures from the search space $\mathcal{A}$.
for $t = 1, \ldots, T$ do
    Train a surrogate model based on the current population.
    Select architecture $a_t$ by maximizing $\phi(a)$, based on the surrogate model.
    Train architecture $a_t$ and add it to the current population.
end for
Output: Architecture from the population with the highest validation accuracy.
```

Speeding Up Black-Box NAS

1. Multi-fidelity optimization
   - see HPO lecture
2. Learning curve prediction
   - Predict how the learning curve will extend with further training and continue only the promising architectures
3. Meta-learning across datasets
   - e.g., learning promising architectures as portfolio [Zimmer et al. 2021]
4. Network morphisms & weight inheritance
   - learn how to grow network size while training by using network morphisms [Chen et al. 2016; Wei et al. 2016; Cai et al. 2017]


Original Model
Layers that Initialized as Identity Mapping
A Deeper Model Contains Identity Mapping Initialized Layers

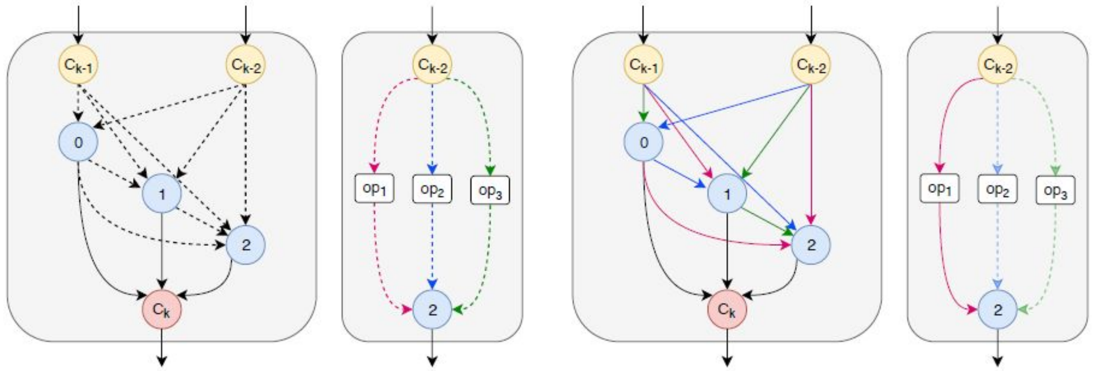
Kahoot Quiz I

One-Shot NAS

One-Shot: Idea [Pham et al. 2018; Bender et al. 2018]

- The one-shot model can be seen as a directed acyclic multigraph
  - Nodes - latent representations.
  - Edges (dashed) - operations.
- Architecture optimization problem:

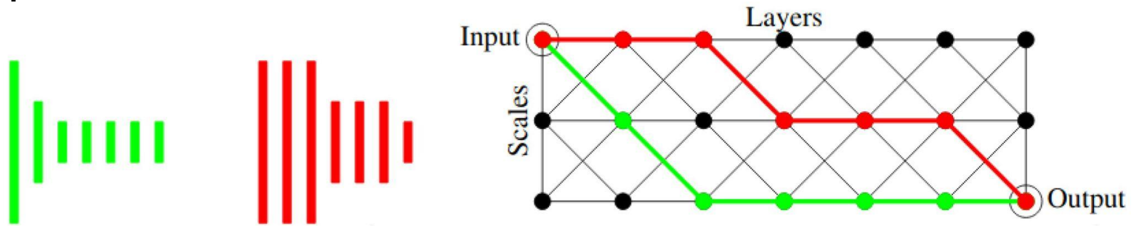
  Find optimal path from the input to the output



(a) One-shot search (b) Final evaluation

Neural Fabrics [Saxena and Verbeek. 2017]

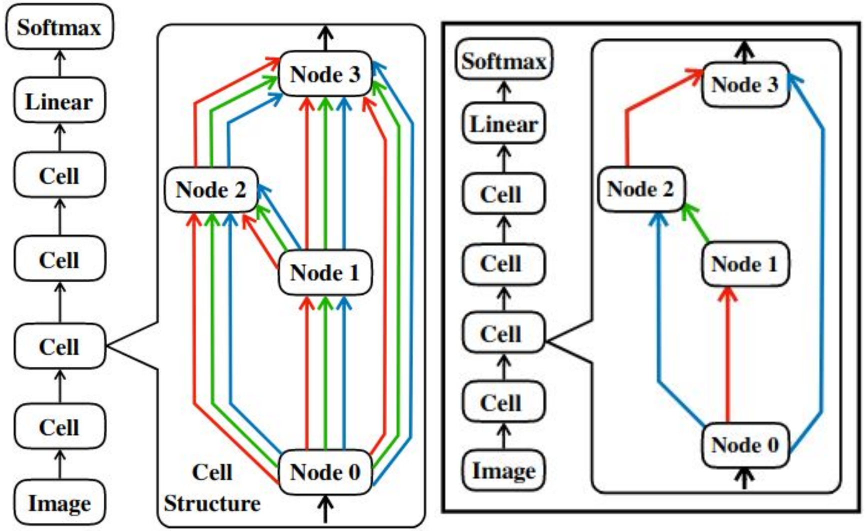
- A one-shot model is a big model that has **all architectures in a search space as submodels**
  - This allows weights sharing across architectures
  - One only needs to train the single one-shot model,
    and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: convolutional neural fabrics



  - Each path from the input to the output represents an architecture
  - The nodes represent tensors
  - The edges represent computations (e.g., convolution / strided convolution)
  - Weights for the operation on an edge are shared
  - Across all (exponentially many) architectures that have that edge

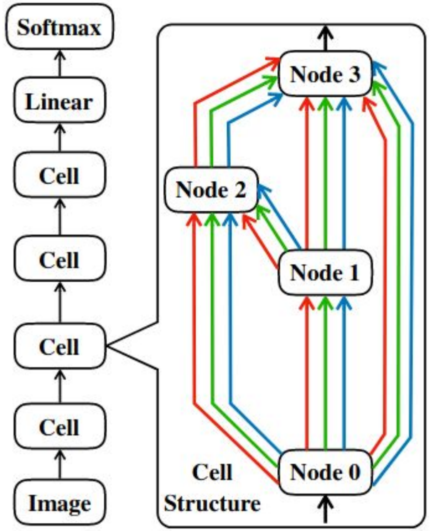
One-shot models for cell search spaces

- Directed acyclic multigraph to capture all (exponentially many) cell architectures
  - The nodes represent tensors
  - The edges represent computations
  - The results of operations on multiple edges between two nodes are combined
- Individual architectures are subgraphs of this multigraph
  - Weights for the operation on an edge are shared across all (exponentially many) architectures that have that edge
- **Goal**: Extract the best performing architecture from one-shot model



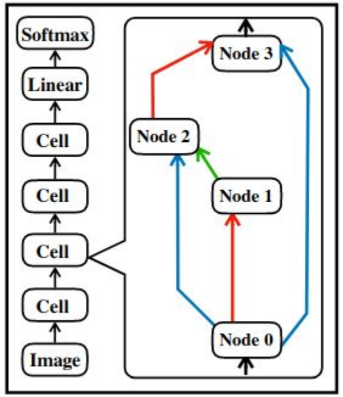
Why can't we simply train the one-shot model?

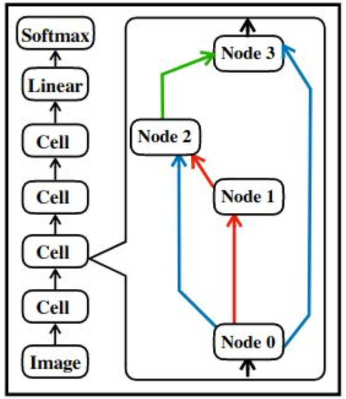
- One-shot model is an acyclic graph; thus, backpropagation applies
  - Simplest method: standard training with SGD
  - This implicitly trains an exponential number of architectures
- Potential issue: co-adaptation of weights
  - Weights are implicitly optimized to work well on average across all architectures
  - They are not optimized specifically for the top-performing architecture



Training One-Shot Models

- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space
  - Random Search with Weight Sharing [Li and Talwalkar. 2020]
    → sample from uniform distribution
  - ENAS [Pham et al. 2018] → sample from the learned policy of a RNN controller
- Update the parameters of the one-shot model corresponding to only that architecture





How to utilize the trained one-shot model?

- After training the one-shot model we have to select the best individual architecture from it
- There are multiple ways we can approach this. Some of these are:
  - 1. Sample uniformly at random M architectures and rank them based on their validation error using the one-shot model parameters
  - 1b. (Optional) Select top K (K < M) and retrain them from scratch for a couple of epochs
  - 2. Return the top performing architecture to retrain from scratch for longer

- **Pitfall**: the correlation between architectures evaluated with the one-shot weights and retrained from scratch (stand-alone models) should be high
  - If not, selecting the best architecture based on the one-shot weights is sub-optimal.

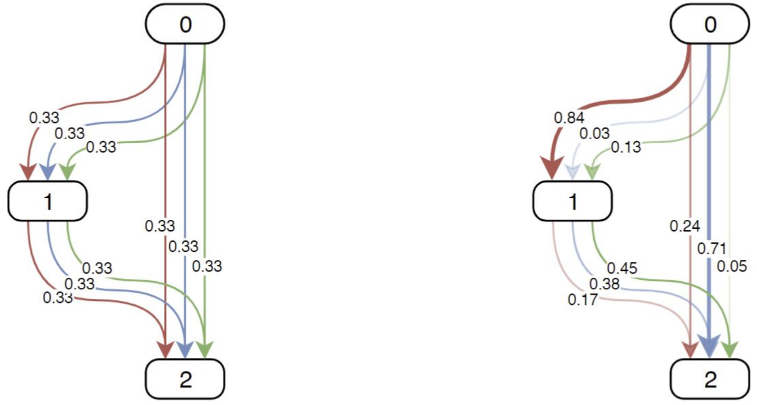
DARTS: Differentiable Architecture Search

DARTS: Differentiable Architecture Search [Liu et al, 2018]

- Use one-shot model with **continuous architecture weight α** for each operator

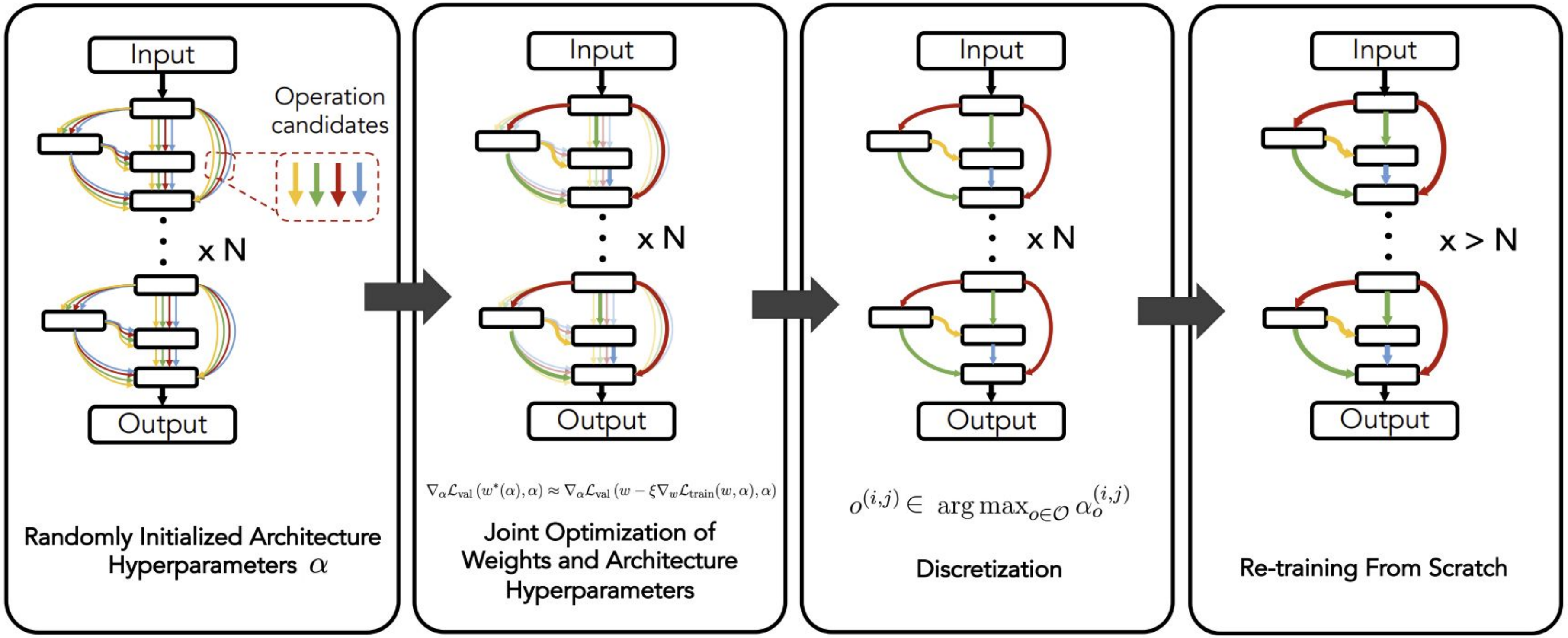
$$x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})} o(x^{(i)})$$

- By optimizing the architecture weights α, DARTS assigns importance to each operation
  - Since the α are continuous, we can optimize them with gradient descent



(a) Initialization (b) Search end

DARTS: Differentiable Architecture Search [Liu et al. 2018]


Input
Operation candidates
x N
Output
Randomly Initialized Architecture Hyperparameters α
Input
x N
Output
$\nabla_\alpha \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{\text{val}}(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha)$
Joint Optimization of Weights and Architecture Hyperparameters
Input
x N
Output
$o^{(i,j)} \in \arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$
Discretization
Input
x > N
Output
Re-training From Scratch

From [White et al. 2023]

DARTS: Training

The optimization problem (a → b) is a bi-level optimization problem:

$$\begin{aligned} &\min_\alpha \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \\ s.t.\ &w^*(\alpha) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \alpha) \end{aligned}$$

This is solved using alternating SGD steps on architectural parameters α and weights w

Algorithm 4 DARTS - Differentiable Architecture Search

```
Input: Search space $\mathcal{A}$, number of iterations $T$, hyperparameter $\xi$.
Randomly initialize a one-shot model based on $\mathcal{A}$ with weights $w$ and architecture hy-
perparameters $\alpha$.
for $t = 1, \dots, T$ do
    Perform a gradient update on the architecture weights $\alpha$
    Perform a gradient update on $w$ according to $\nabla_w \mathcal{L}_{train}(w, \alpha)$.
end for
Output: Derive the final architecture by taking the argmax of $\alpha$, across all operation
choices, and then retrain this architecture from scratch.
```

Performance of DARTS

- E.g., original CNN search space
  - 8 operations on each MixedOp
  - 28 MixedOps in total
- $>10^{23}$ possible architectures
- Performance: <3% error on CIFAR-10 in less than 1 GPU day of search

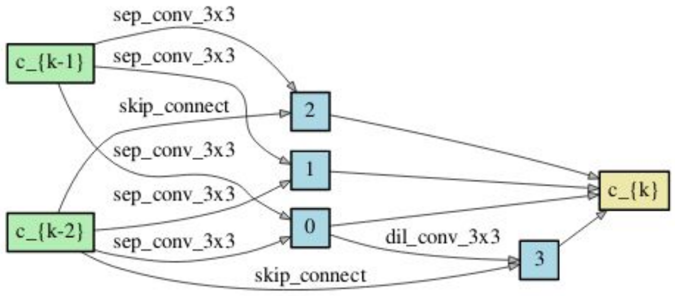


Figure 4: Normal cell learned on CIFAR-10.

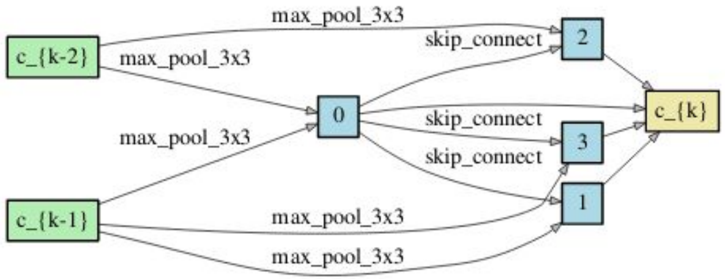
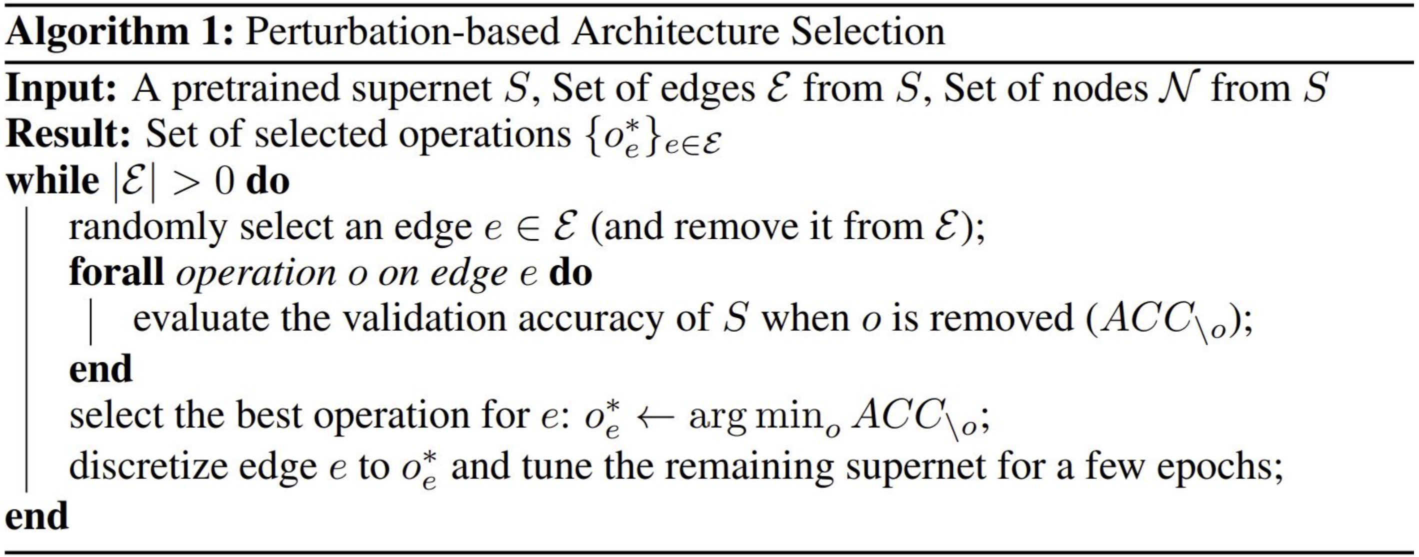


Figure 5: Reduction cell learned on CIFAR-10.

- BUT: DARTS can be very brittle; e.g. convergence to only skip connections

Selecting the final Architecture from DARTS

- Problem: Selecting according to highest weights might not be optimal
  - Weights are not indicative about overall importance of operator
- Iterative selection of final architecture [Wang et al. 2021]



Algorithm 1: Perturbation-based Architecture Selection

```
Input: A pretrained supernet $S$, Set of edges $\mathcal{E}$ from $S$, Set of nodes $\mathcal{N}$ from $S$
Result: Set of selected operations $\{o_e^*\}_{e \in \mathcal{E}}$
while $|\mathcal{E}| > 0$ do
    randomly select an edge $e \in \mathcal{E}$ (and remove it from $\mathcal{E}$);
    forall operation $o$ on edge $e$ do
        evaluate the validation accuracy of $S$ when $o$ is removed ($ACC_{\backslash o}$);
    end
    select the best operation for $e$: $o_e^* \leftarrow \arg\min_o ACC_{\backslash o}$;
    discretize edge $e$ to $o_e^*$ and tune the remaining supernet for a few epochs;
end
```

Memory Issue of DARTS

- DARTS keeps the entire one-shot model in memory, together with its computed tensors
  - This constrains the search space size and the fidelity used to train the one-shot model
  - Impossible to run on large datasets as ImageNet
- A lot of research aims to fix this issue:
- GDAS [Dong et al, 2019]
  - samples from a Gumbel-Softmax distribution to keep only a 1 architecture in RAM
- ProxylessNAS [Cai et al, 2019]
  - computes approximate gradients on α keeping only 2 edges between two nodes in memory at a time
- PC-DARTS [Xu et al, 2020]
  - performs the search on a subset of the channels in the one-shot model

Kahoot Quiz II

Zero-Cost Proxies

From expensive to cheap NAS?

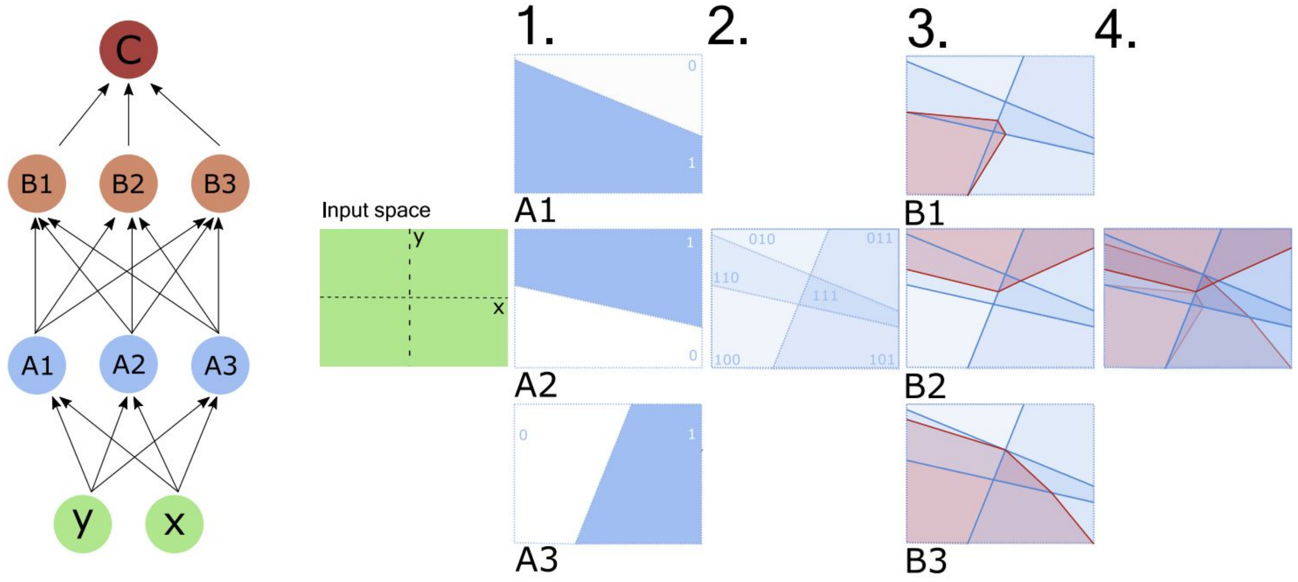
- All NAS methods depend on evaluating several different architectures (both in black-box methods and differentiable architectures)

- **Vision**: Can we estimate the performance of an architecture without training it
  - intermediate step: multi-fidelity techniques or one-epoch training
  - ultimate goal: use few statistics from an architecture to estimate their performance
    - e.g., only a few forward passes

- ⇒ Zero-Cost (ZC) proxies
  - can be computed in a few seconds
  - instead of training for minutes, hours or days

ZC-Proxy: Linear Seperable at Initialization [Mellor et al. 2021]

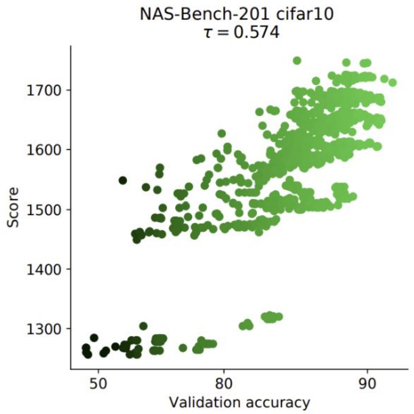
- Exemplary ZC-Proxy: For mini-batch of data points, check whether points are well seperable in a DNN with **ReLUs**

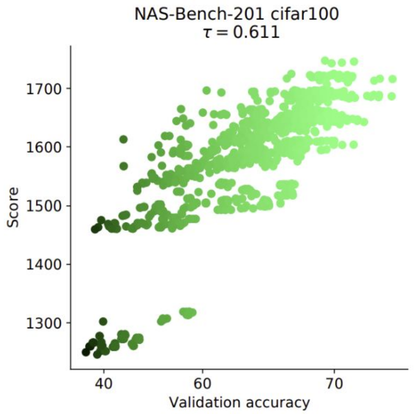


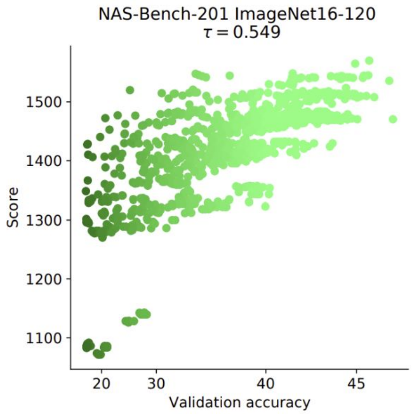
Data-Dependent and Data-Agnostic ZC-Proxies

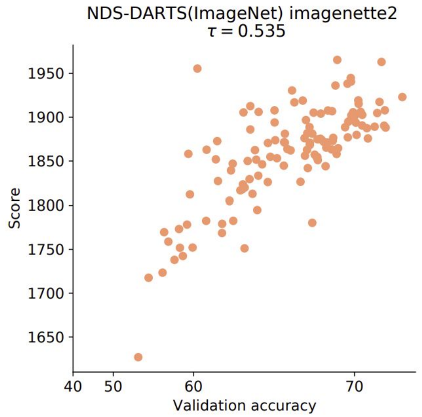
- Data-dependent ZC-Proxies
  - (few) Forward passes with mini-batches
    - typically, not using the entire dataset
  - No gradient updates to the network weights
  - For example, the intra- and inter-class correlations of the prediction Jacobian matrices [Lopes et al. 2021]
- Data-agnostic ZC-Proxies
  - ignores dataset at hand
  - assumes existence of (near) universal architecture(s)
  - *Zen-Score* approximates the neural network by piecewise linear functions conditioned on activation patterns and computes the Frobenius norm [Lin et al. 2021]

Correlation between Zero-Cost Proxy and Validation Loss









- To ensure high quality of zero-cost proxy and NAS performance, correlation should be high
  - e.g., Kendall-Tau should be close to 1
  - In practice τ is often between 0.2 and 0.6 according to [Mellor et al. 2021] or between -0.1 and 0.8 according to [White et al. 2022]

From [Mellor et al. 2021]

Problems of ZC-Proxies [White et al. 2022]

1. Best ZC-proxy depends on the task at hand
   - there is no single best
2. #FLOPS and number of parameters are competitive baselines
   - → larger networks perform better
3. data-agnostic ZC-proxies perform inconsistently
   - → Assumption of NAS (/AutoML): Datasets need different architectures
   - → data-agnostic ZC-proxies imply that there are universal architectures
   - → data-agnostic ZC-proxies can only exclude poor architectures?
4. Even if the NAS part can be sped up by ZC-Proxies, the training of the final architecture can still be expensive — limiting the overall possible speed up

Augmenting NAS with ZC-Proxies

- Pre-Filtering and Initialization
  - use ZC-Proxies to start with a decent set of architectures, e.g., using an evolutionary algorithm [Mellor et al. 2021]
- Additional context-information for surrogate-based optimizers
  - surrogate model of Bayesian Optimization can additionally get ZC-proxy information to better predict the performance of an architecture [White et al. 2021]
- Faster selection of final architecture in DARTS
  - Pertubation-based filtering can be augmented by good ranking of architectures [Xiang et al. 2021]
- Combination of ZC-Proxies can be more powerful [Chen et al. 2022]

Conclusion

Neural Architecture Search: Final Thoughts

- While first NAS approaches required 100s of GPU hours, good architectures with NAS within few hours these days are feasible – **tremendous speedup**
- Gradient-based NAS is the fastest, but not the most stable
  - Bayesian Optimization with some tricks can be quite competitive
- Zero-cost proxies can provide additional information how well an architecture can perform

BUT:

- NAS is still quite limited in its expressive capabilities;
  e.g., will NAS be able to rediscover the Transformer architecture?
- NAS is not easily applicable to very expensive models (e.g., LLMs)
  [Tornede et al. 2023]

Recommendations

- Surveys:
  - Neural Architecture Search: Insights from 1000 Papers by White et al. 2023
  - Neural Architecture Search: A Survey by Elsken et al. 2019
- Software Packages:
  - Auto-Keras
  - NNI
  - Auto-PyTorch

Kahoot Quiz III

Thanks.
See you tomorrow!
👋