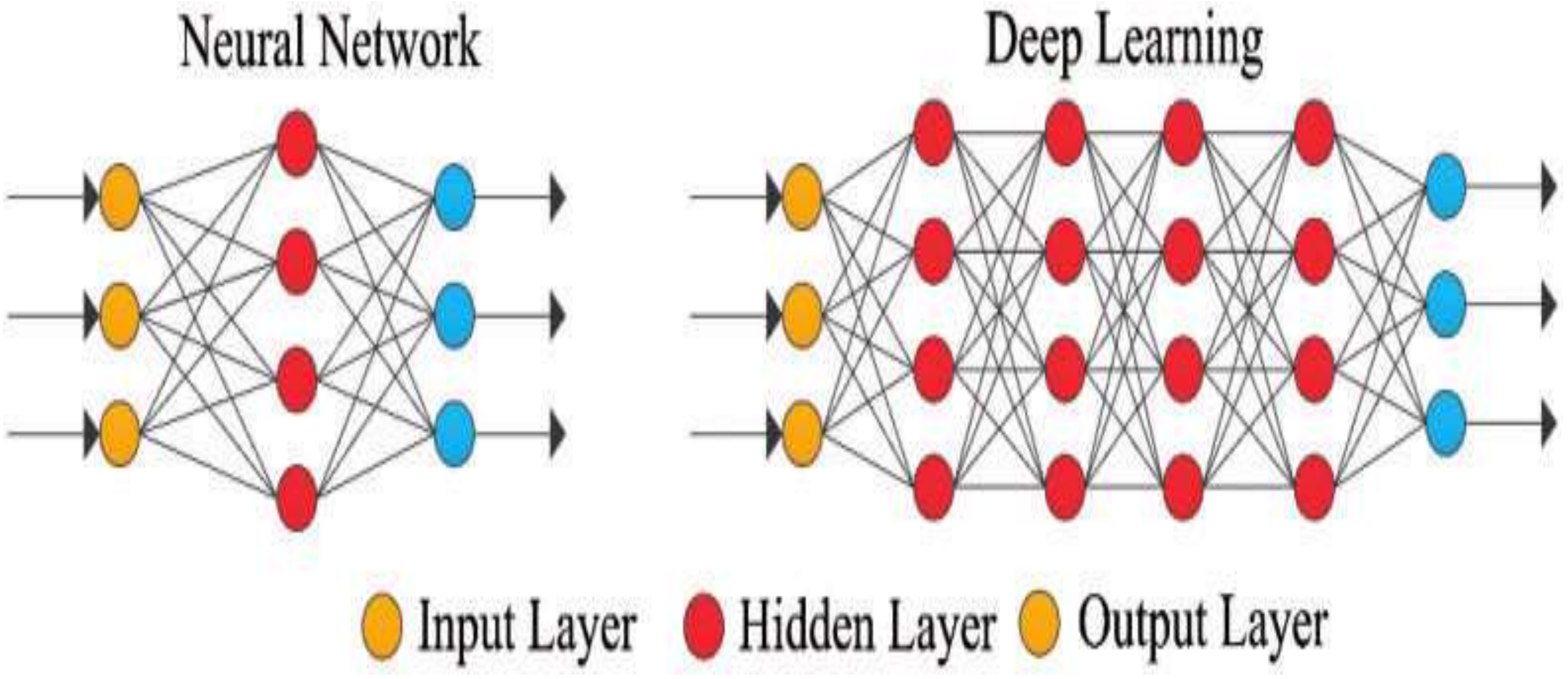
Neural Network Search

(Learning how to learn)

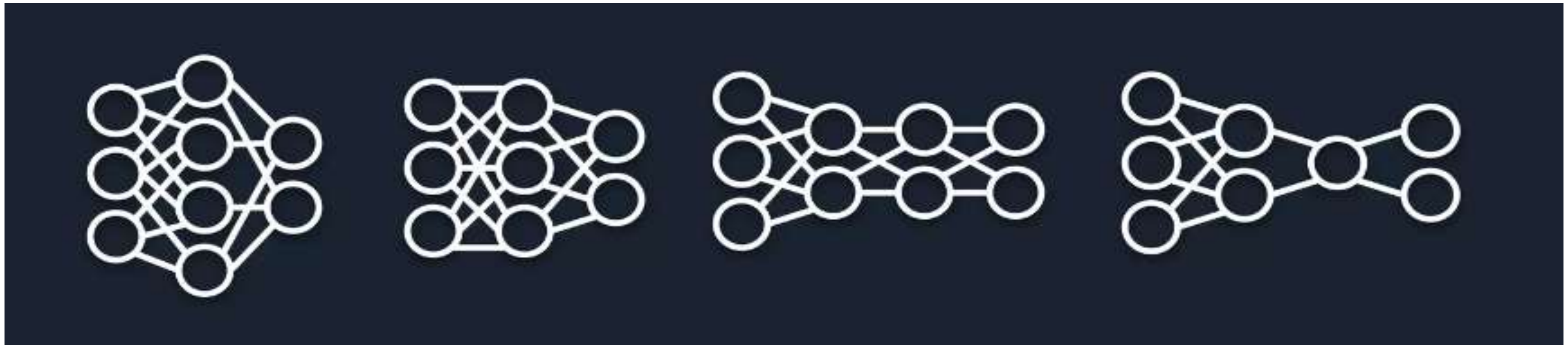
Mounika Marreddy

What is Deep Learning

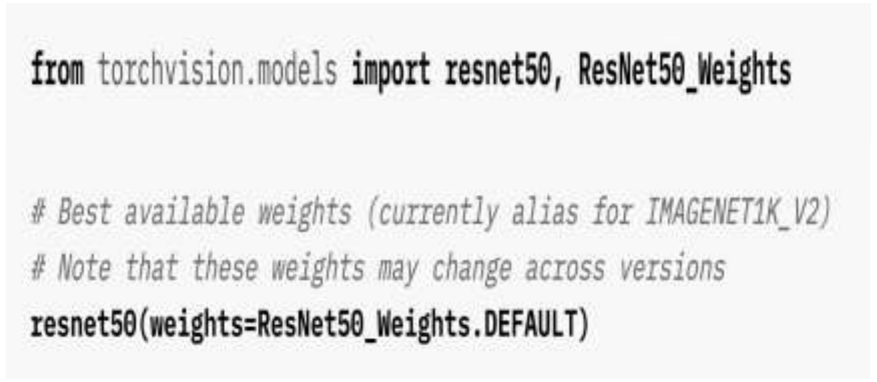


Deep Learning

- Explosion of interest since 2012
- Very powerful machine learning technique
- Huge variety of neural networks for different tasks
- Key ingredients took years to develop
- Algorithms are getting increasingly more specialized and complicated



Machine and Deep Learning: a story of automation

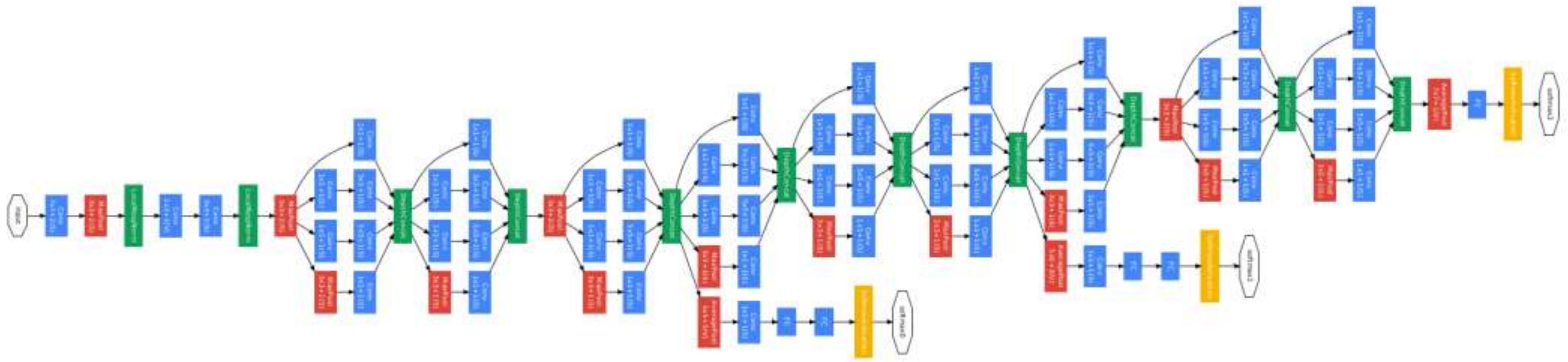


```
from torchvision.models import resnet50, ResNet50_Weights

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

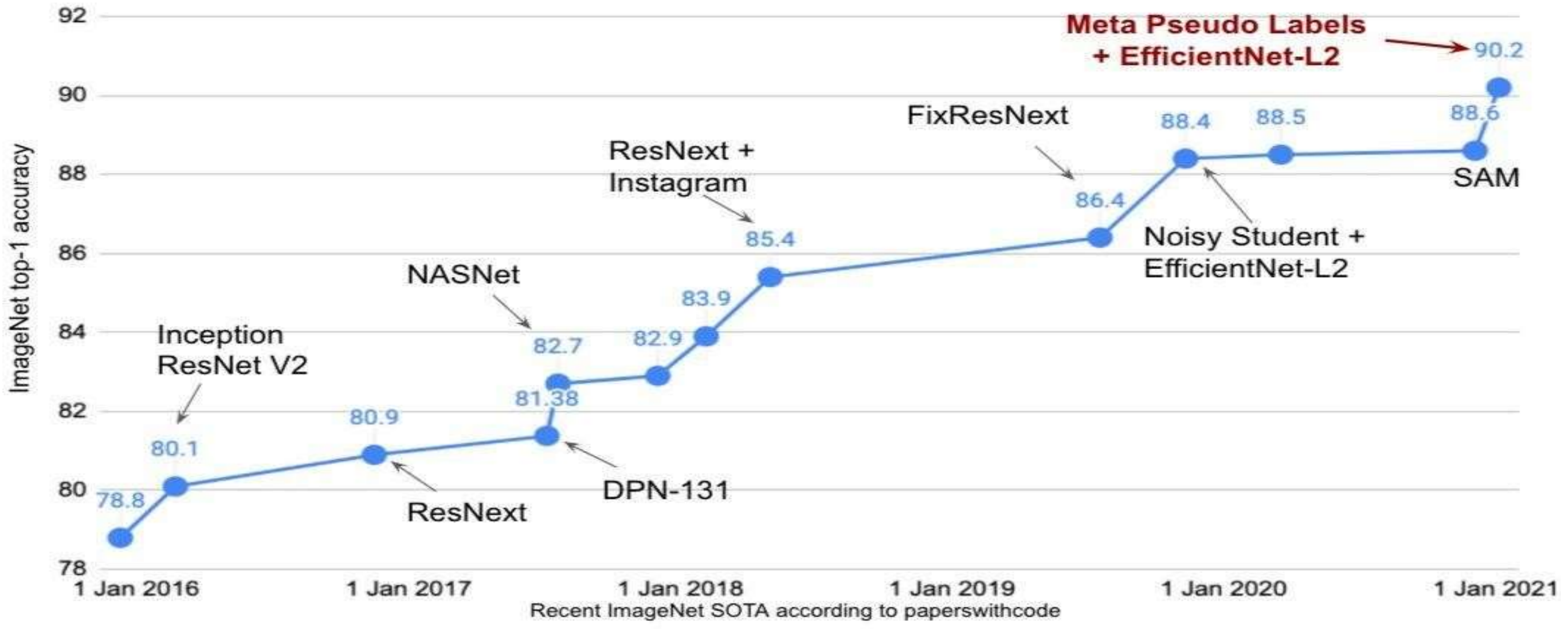
Deep Learning (cont'd)

- Algorithms are getting increasingly more specialized and complicated
- GoogLeNet (2014)



Neural Architectures

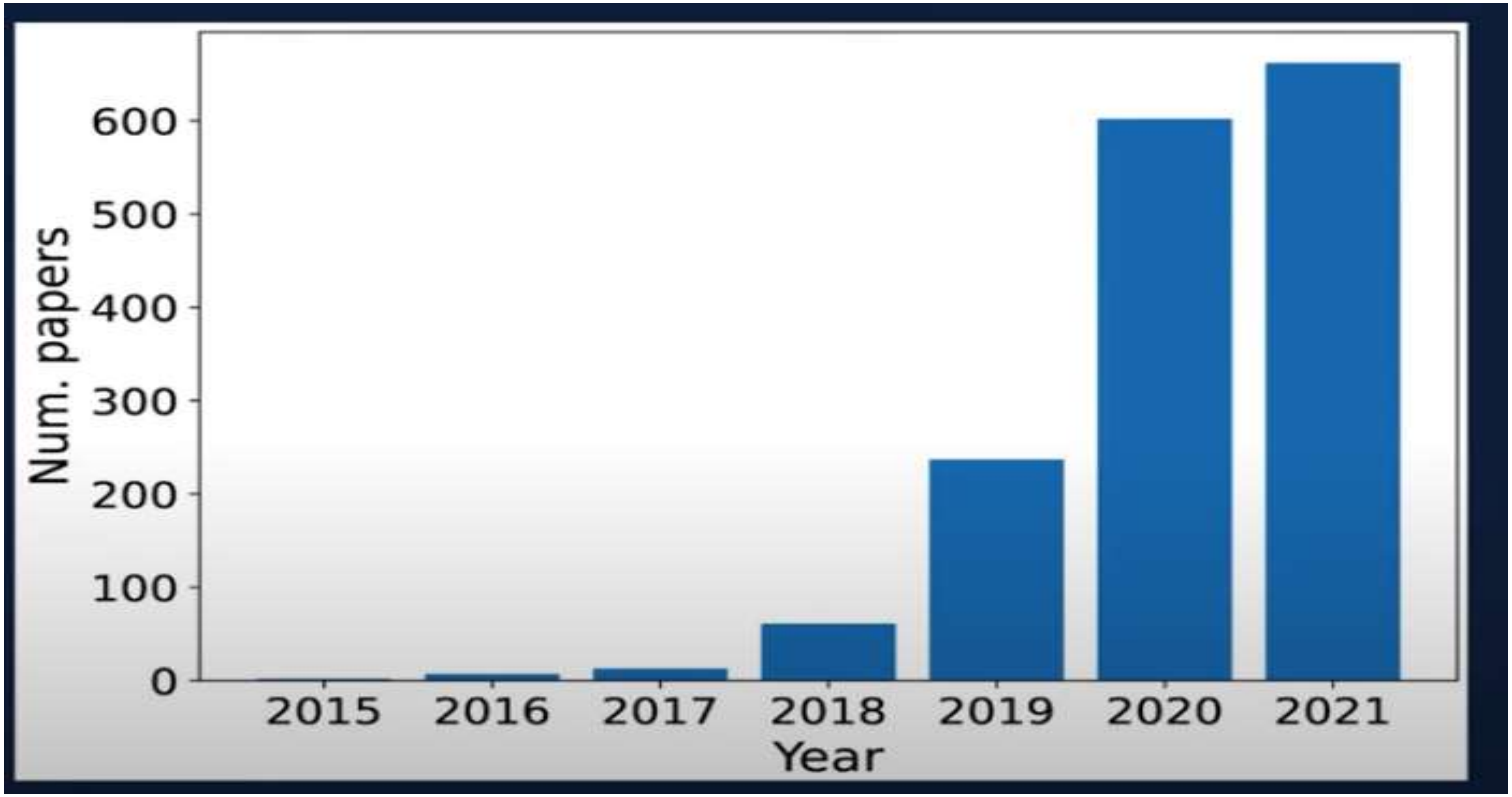
- Search models are replaced by human designed models



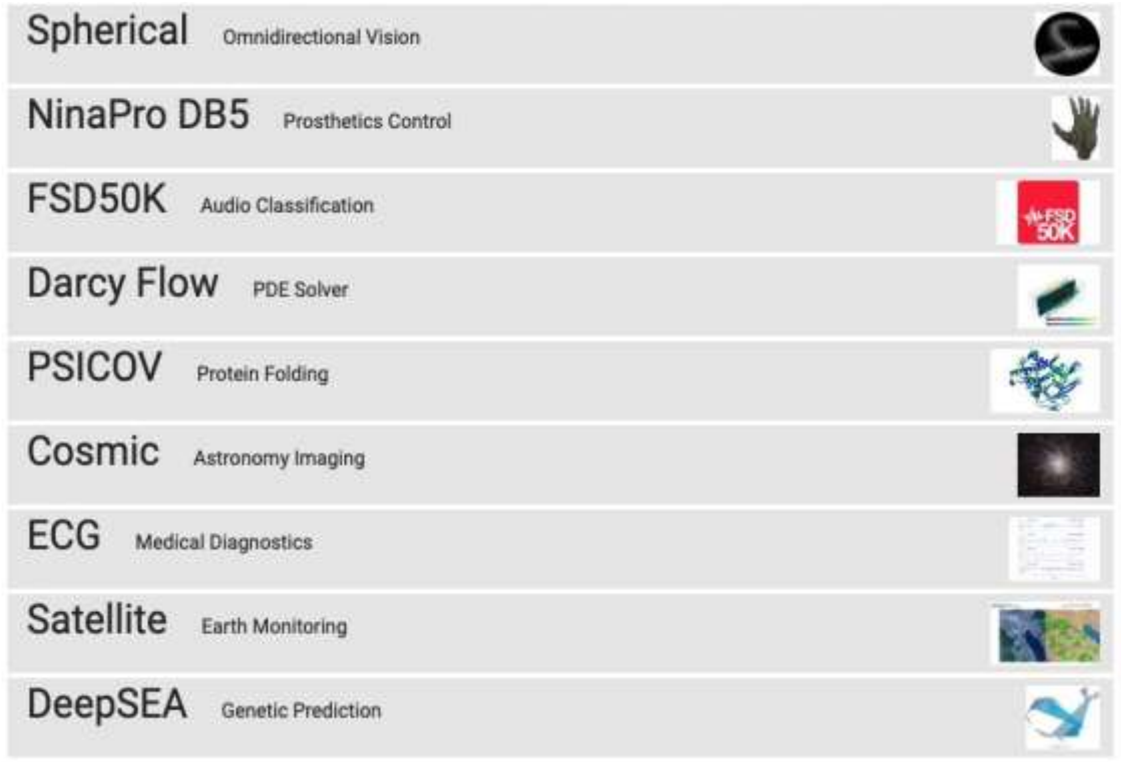
Definition

- Neural network search (NAS) is a technique for automating the design of artificial neural networks (ANNs) for a given dataset.
- NAS automates the process of designing ANNs by searching through a space of possible architectures.
- The search space is defined by the set of possible nodes, connections, and layers that can be used to construct an ANN.
- The NAS algorithm then explores the search space to find an architecture that performs well on a given task

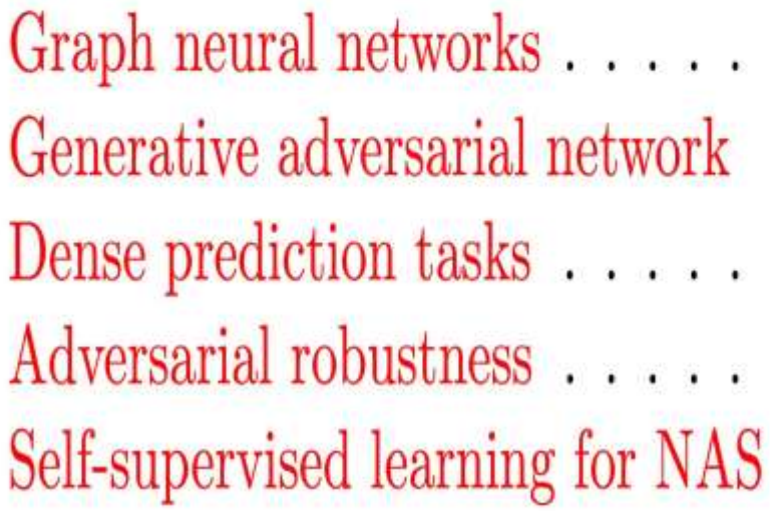
NAS: Growing Field!



NAS can be applied for:



Spherical	Omnidirectional Vision
NinaPro DB5	Prosthetics Control
FSD50K	Audio Classification
Darcy Flow	PDE Solver
PSICOV	Protein Folding
Cosmic	Astronomy Imaging
ECG	Medical Diagnostics
Satellite	Earth Monitoring
DeepSEA	Genetic Prediction



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

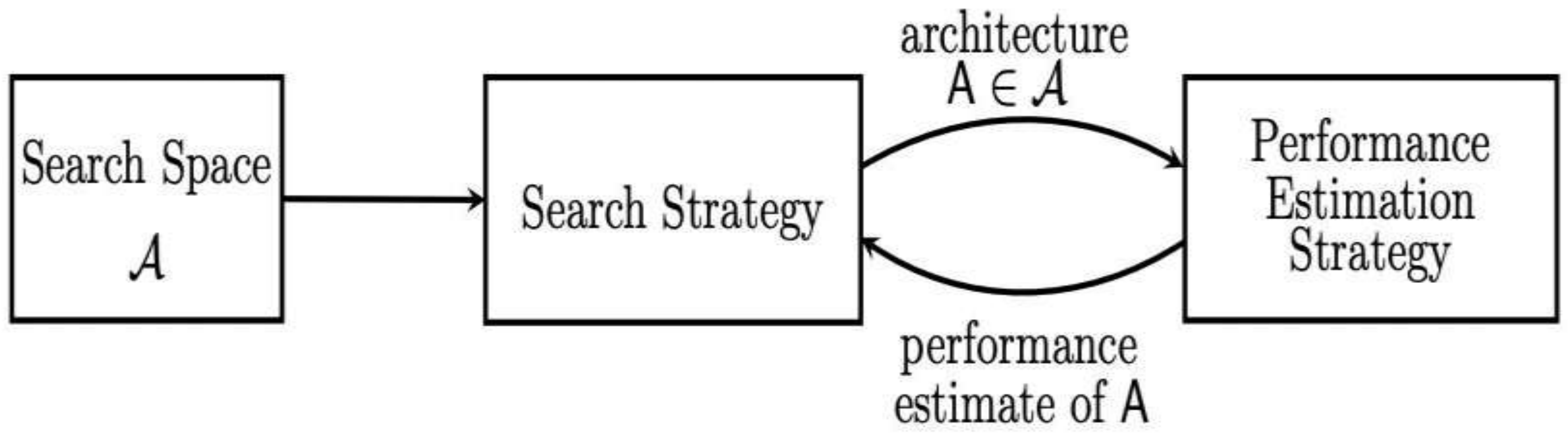
Basic Math Definition

- Define a search space A,

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

Three Pillars of NAS

- Search Space
- Search Strategy
- Performance Estimation Strategy



Search Space

- The search space defines which architectures can be represented in principle.
- Incorporating prior knowledge about typical properties of architectures well-suited for a task can reduce the size of the search space and simplify the search.
- However, this also introduces a human bias, which may prevent finding novel architectural building blocks that go beyond the current human knowledge.

Search Strategy

- The search strategy details how to explore the search space (which is often exponentially large or even unbounded).
- It encompasses the classical exploration-exploitation trade-off since, on the one hand, it is desirable to find well-performing architectures quickly, while on the other hand, premature convergence to a region of suboptimal architectures should be avoided.

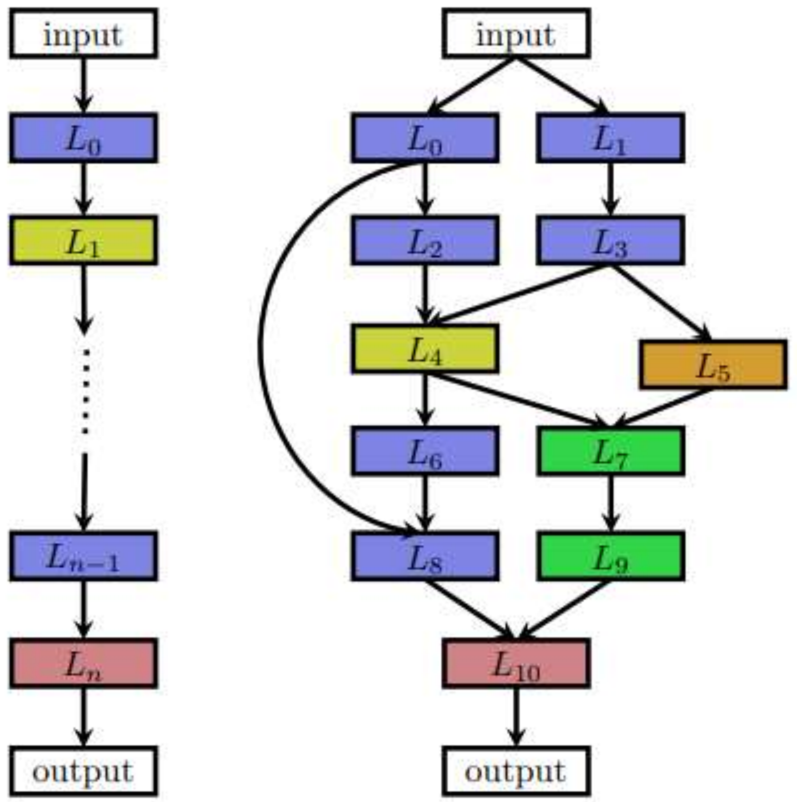
Performance Estimation

- The objective of NAS is typically to find architectures that achieve high predictive performance on unseen data.
- Performance Estimation refers to the process of estimating this performance: the simplest option is to perform a standard training and validation of the architecture on data, but this is unfortunately computationally expensive and limits the number of architectures that can be explored.
- Much recent research therefore focuses on developing methods that reduce the cost of these performance estimations.

Search Space

- The search space defines which neural architectures a NAS approach might discover in principle.
- Some of the recent search spaces are:
  - Chain-structured neural networks
  - Multi-branch networks
  - Cell or block search space

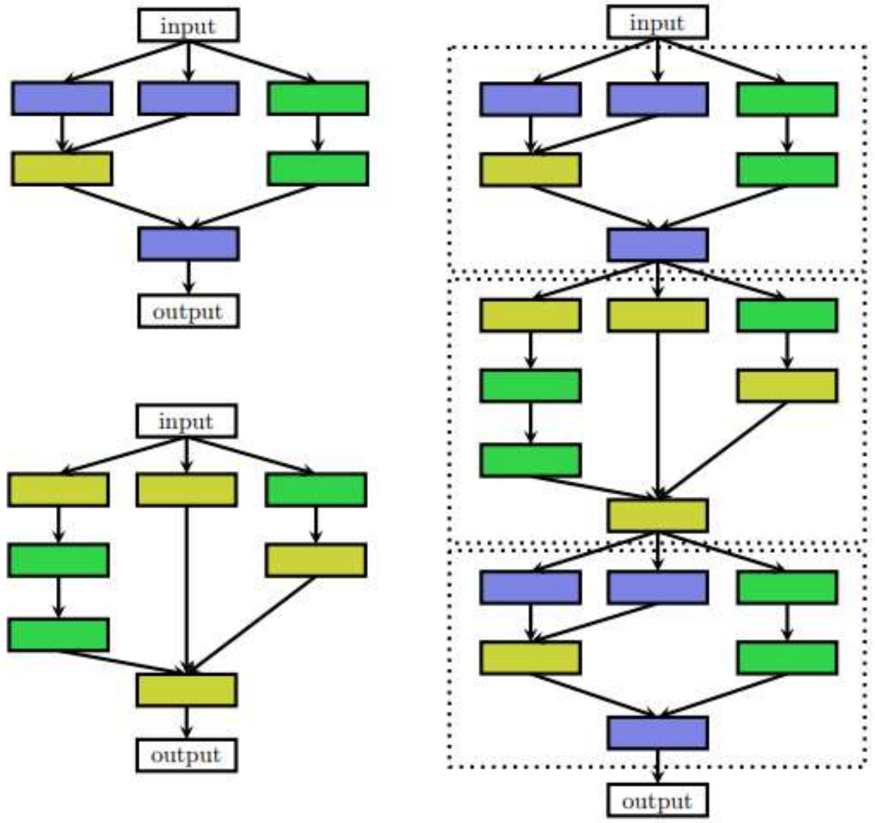
Chain structured and Complex Search Spaces



Advantages of cell or block search

- The size of the search space is drastically reduced since cells usually consist of significantly less layers than whole architectures.
  - For example, Zoph et al. (2018) estimate a seven-times speed-up compared to their previous work (Zoph and Le, 2017) while achieving better performance.
- Architectures built from cells can more easily be transferred or adapted to other data sets by simply varying the number of cells and filters used within a model.
  - Indeed, Zoph et al. (2018) transfer cells optimized on CIFAR-10 to ImageNet and achieve state-of-the-art performance.
- Creating architectures by repeating building blocks has proven a useful design principle in general, such as repeating an LSTM block in RNNs or stacking a residual block.

Search Spaces by Stacking the Cells



Search Strategy

- Many different search strategies can be used to explore the space of neural architectures:
  - Random search
  - Bayesian optimization
  - Evolutionary methods
  - Reinforcement learning (RL), and
  - Gradient-based methods

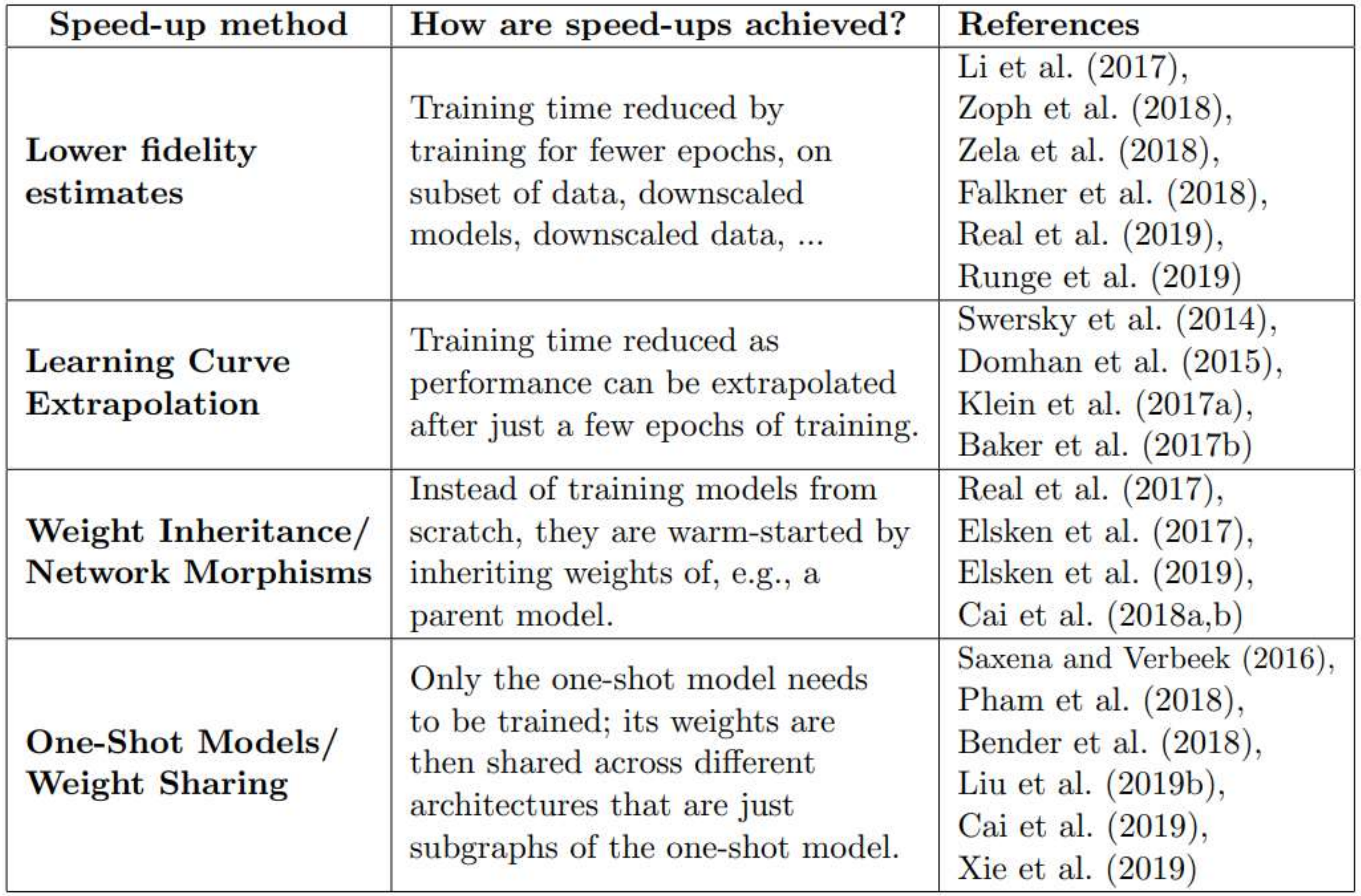
Search Strategy - Types

- Some of the most common algorithms are:
  - Reinforcement learning-based NAS: In reinforcement learning, an agent learns to perform a task by trial and error. The agent receives rewards for taking actions that lead to successful outcomes, and punishments for taking actions that lead to unsuccessful outcomes.
  - Bayesian optimization-based NAS: Bayesian optimization is a technique for finding the best value of a function by iteratively querying the function and using the results of the queries to update the estimate of the function's true value.
  - Genetic algorithm-based NAS: Genetic algorithms are a type of evolutionary algorithm that uses principles of natural selection to search for solutions to problems. NAS algorithms use genetic algorithms to search for ANN architectures that are likely to be fit for a given task.

Performance Estimation - NAS

- The search strategies finds a neural architecture A that maximizes some performance measure, such as accuracy on unseen data.
- To guide their search process, these strategies need to estimate the performance of a given architecture A they consider.
- The simplest way of doing this is to train A on training data and evaluate its performance on validation data.
- However, training each architecture to be evaluated from scratch frequently yields computational demands in the order of thousands of GPU days for NAS.
- This naturally leads to developing methods for speeding up performance estimation.

Performance Estimation - NAS



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)

Performance Estimation – Lower Fidelity

- Performance can be estimated based on lower fidelities of the actual performance after full training (also denoted as proxy metrics).
- Such lower fidelities include shorter training times, training on a subset of the data, on lower-resolution images, or with less filters per layer and less cells.
- While these low-fidelity approximations reduce the computational cost, they also introduce bias in the estimate as performance will typically be underestimated.

Performance Estimation – Learning Curve Extrapolation

- Authors propose to extrapolate initial learning curves and terminate those predicted to perform poorly to speed up the architecture search process.
- Some consider architectural hyperparameters for predicting which partial learning curves are most promising.
- Training a surrogate model for predicting the performance of novel architectures is also proposed, do not employ learning curve extrapolation but support predicting performance based on architectural/cell properties and extrapolate to architectures/cells with larger size than seen during training.

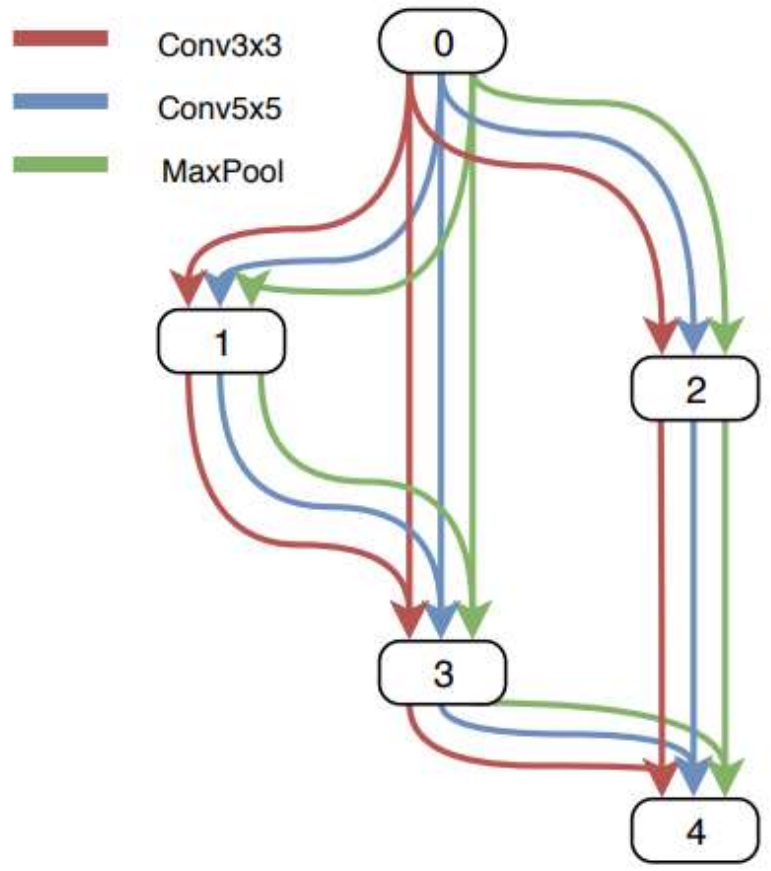
Performance Estimation – Network Morphisms

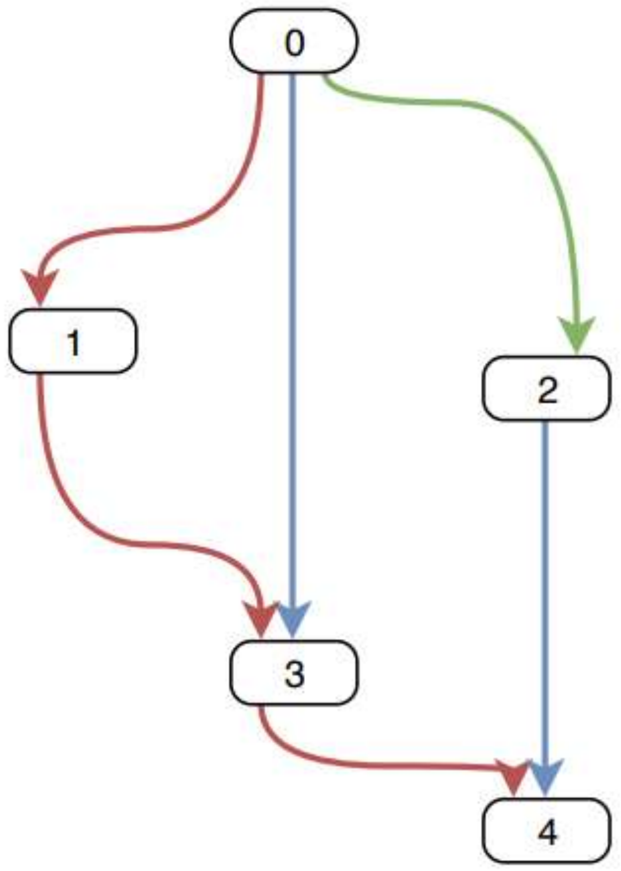
- Another approach to speed up performance estimation is to initialize the weights of novel architectures based on weights of other architectures that have been trained before.
- One way of achieving this, dubbed network morphisms, allows modifying an architecture while leaving the function represented by the network unchanged, resulting in methods that only require a few GPU days.
- This allows increasing the capacity of networks successively and retaining high performance without requiring training from scratch.

Performance Estimation – One-Shot Architecture Search

- One-Shot Architecture Search treats all architectures as different subgraphs of a supergraph (the one-shot model) and shares weights between architectures that have edges of this super graph in common.
- Only the weights of a single one-shot model need to be trained (in one of various ways), and architectures (which are just subgraphs of the one-shot model) can then be evaluated without any separate training by inheriting trained weights from the one-shot model.
- This greatly speeds up performance estimation of architectures, since no training is required (only evaluating performance on validation data), again resulting in methods that only require a few GPU days.

One-Shot Architecture Search Example





References

- Neural Architecture Search: A Survey: https://arxiv.org/abs/1808.06993 by Chen et al.
- Neural Architecture Search with Reinforcement Learning: https://arxiv.org/abs/1606.03657 by Zoph et al.
- Neural Architecture Search with Bayesian Optimization: https://arxiv.org/abs/1802.03268 by Real et al.