COMS E6998 010

Practical Deep Learning Systems Performance

Lecture 12 12/03/20

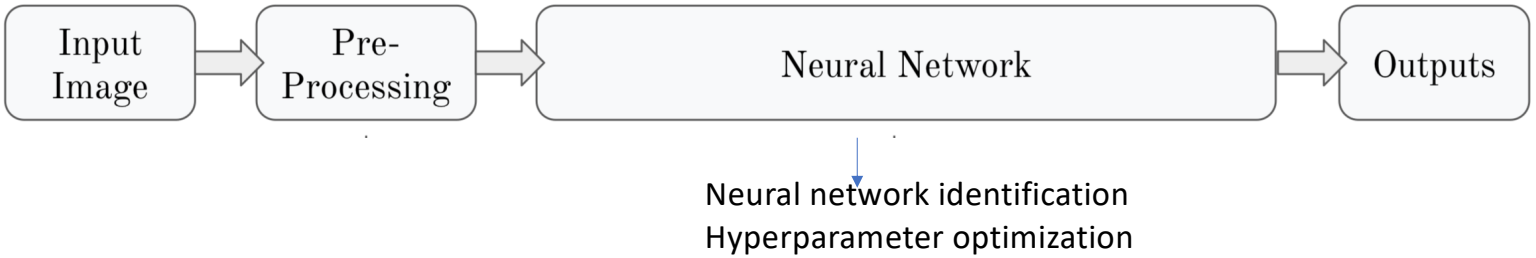
Logistics

- Homework 5 due Dec 9 11:59 PM
- Requirements for final project presentation and rubric are posted. Posted a template for final presentation.
- Project presentation recording due by 3 PM one day before the scheduled presentation**. 5% penalty for not submitting the recording by the deadline**.
- Dec 14, 16, 17: Final project presentation. Please select a slot in the shared sheet available under Collaborations on Canvas.
- Dec 19: Final project github repo and report due.

Recall from last lecture

- Containers, Docker, Kubernetes
- Fabric for Deep Learning, IBM open source distributed DL platform
- Kubeflow, a machine learning toolkit for Kubernetes
- Amazon Sagemaker, a fully managed machine learning service

Automated Machine Learning

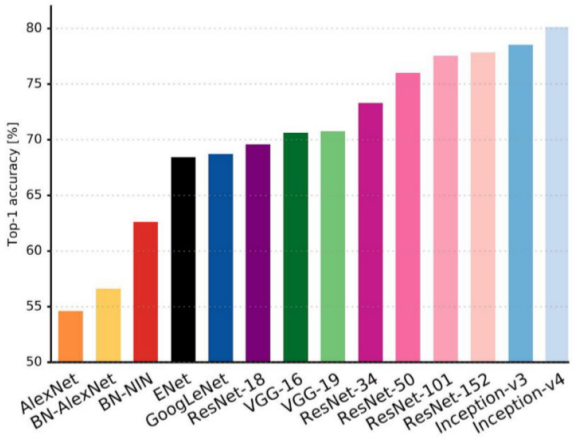


- Data Preprocessing: normalization, data-augmentation
- Neural architecture search (NAS)
  - Standard, off the shelf
  - Synthesize a new
    - Types of layers (conv, maxpool), number of different layers, how to stack the layers
    - Convolution layer parameters (filter dim, stride dim)
- Hyperparameter optimization
  - Batch size, learning rate, momentum
- NAS and hyperparameter optimization can be done jointly or sequentially

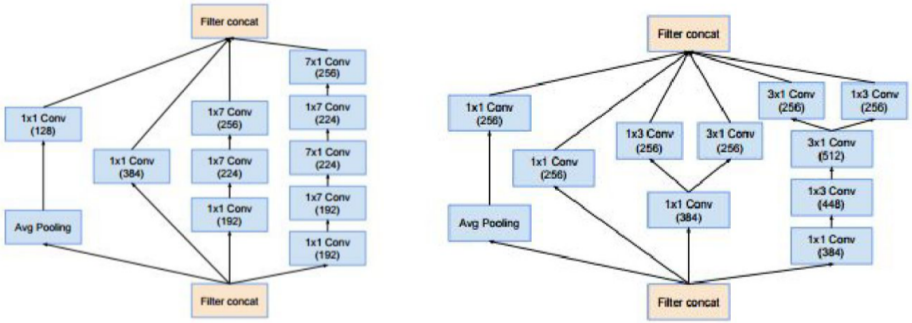
Automated Machine Learning

- IBM Auto AI
- H20 AutoML

Importance of Neural Architectures in Vision



Canziani et al (2017)



Complex hand-engineered layers from Inception-V4 (Szegedy et al., 2017)

Design Innovations (2012 - Present): Deeper networks, stacked modules, skip connections, squeeze-excitation block, ...

Can we try and learn good architectures automatically?

https://metalearning-cvpr2019.github.io/assets/CVPR_2019_Metalearning_Tutorial_Nikhil_Naik.pdf

Neural Architecture Search

Defines the neural architectures a NAS approach may discover

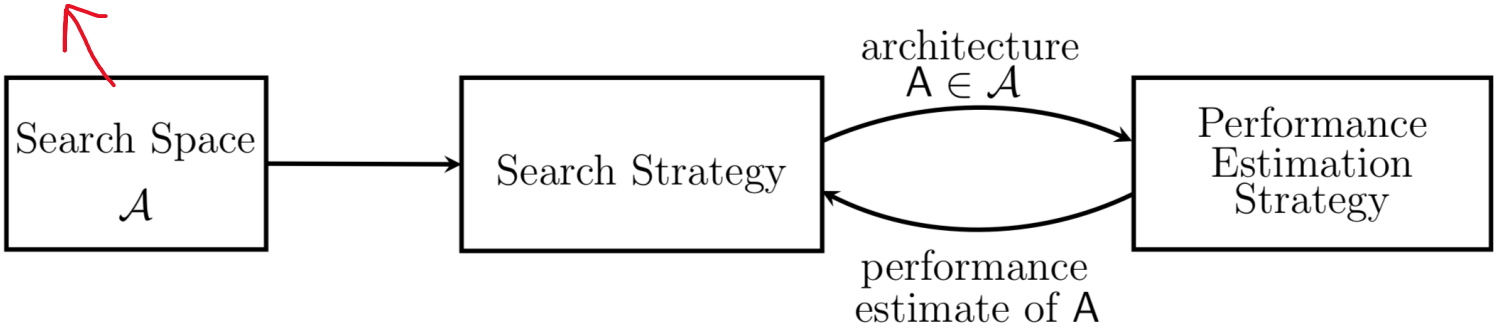
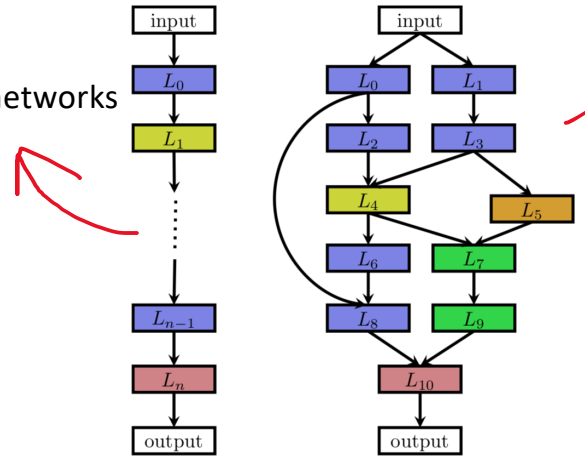


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

Elsken et al. Neural Architecture Search: A Survey. JMLR 2019

NAS Search Space: Layer Based

Chain structured networks



Mutli branch networks with skip connections

Problem with layer based NAS search space

- Search space is of the order of layer types and associated parameters
  - Conv layer: filter size, number of filters, stride length, padding

Figure 2: An illustration of different architecture spaces. Each node in the graphs corresponds to a layer in a neural network, e.g., a convolutional or pooling layer. Different layer types are visualized by different colors. An edge from layer L_i to layer L_j denotes that L_j receives the output of L_i as input. Left: an element of a chain-structured space. Right: an element of a more complex search space with additional layer types and multiple branches and skip connections.

Elsken et al. Neural Architecture Search: A Survey. JMLR 2019

NAS Search Space: Cell Based

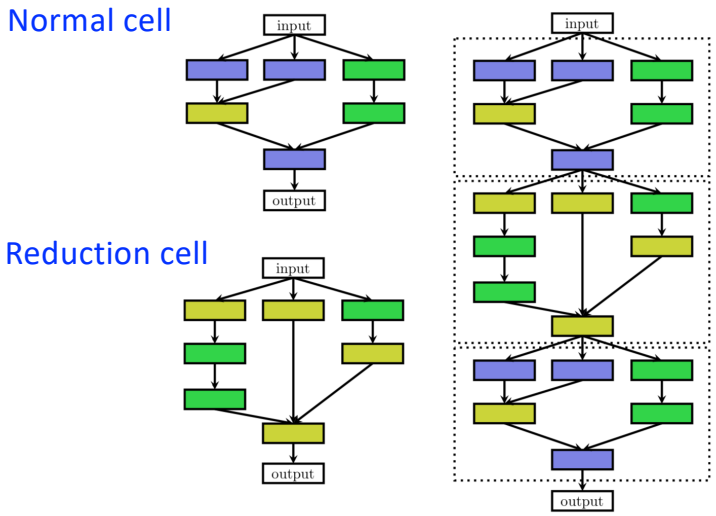


Figure 3: Illustration of the cell search space. Left: Two different cells, e.g., a normal cell (top) and a reduction cell (bottom) (Zoph et al., 2018). Right: an architecture built by stacking the cells sequentially. Note that cells can also be combined in a more complex manner, such as in multi-branch spaces, by simply replacing layers with cells.

Motivated by hand-crafted architectures consisting of repeated motifs (cells/blocks)

- Inception blocks: normal cell, reduction cell
- Residual blocks: normal cell reduction cell

Elsken et al. Neural Architecture Search: A Survey. JMLR 2019

NAS Search Space: Cell Based

Advantages of cell based NAS search space

- Drastically reduced search space compared to layer based
- Easy adaptability of a network for across datasets of varying size and complexity a given domain
- Applicability across different domains, LSTM blocks in RNN, inception and residual blocks in CNNs

- Macro-architecture search problem: how many cells shall be used and how should they be connected to build the actual model?

NAS Search Strategy

- Random search
- Bayesian optimization
- **Evolutionary algorithms (EA)**
- **Reinforcement learning (RL)**
- **Gradient based methods**

EA based NAS search

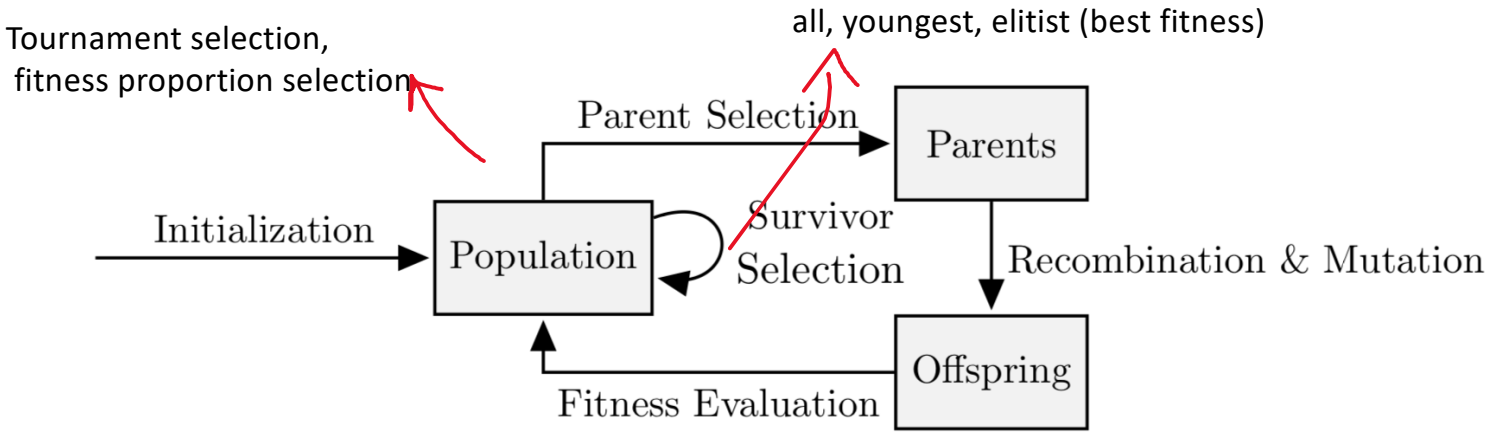


Figure 11: A general framework for evolutionary algorithms.

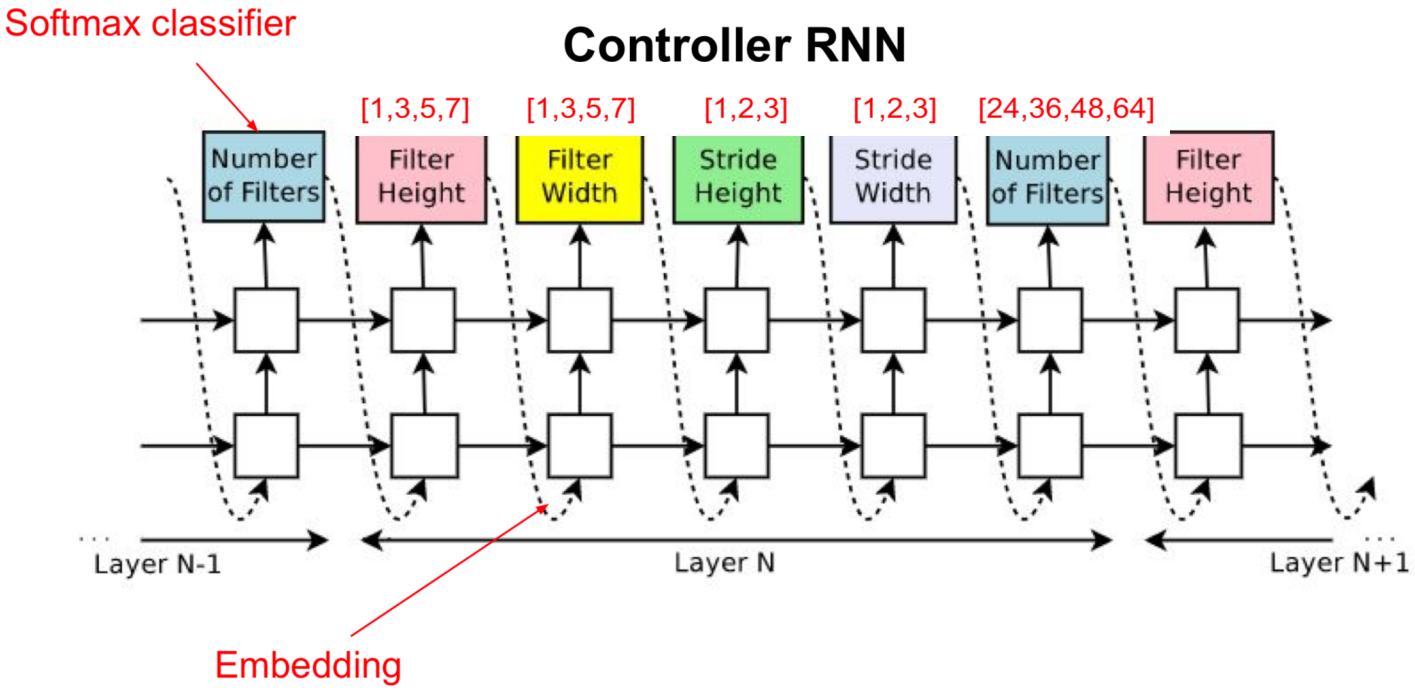
- Earlier work used genetic algorithms to optimize both the neural architecture and its weights
- Recent work use evolutionary algorithms only for optimizing the neural architecture whereas the weights of each candidate architecture are optimized using gradient-based methods like SGD or its variants

Witsuba et al. A Survey on Neural Architecture Search. 2019

RL based NAS search

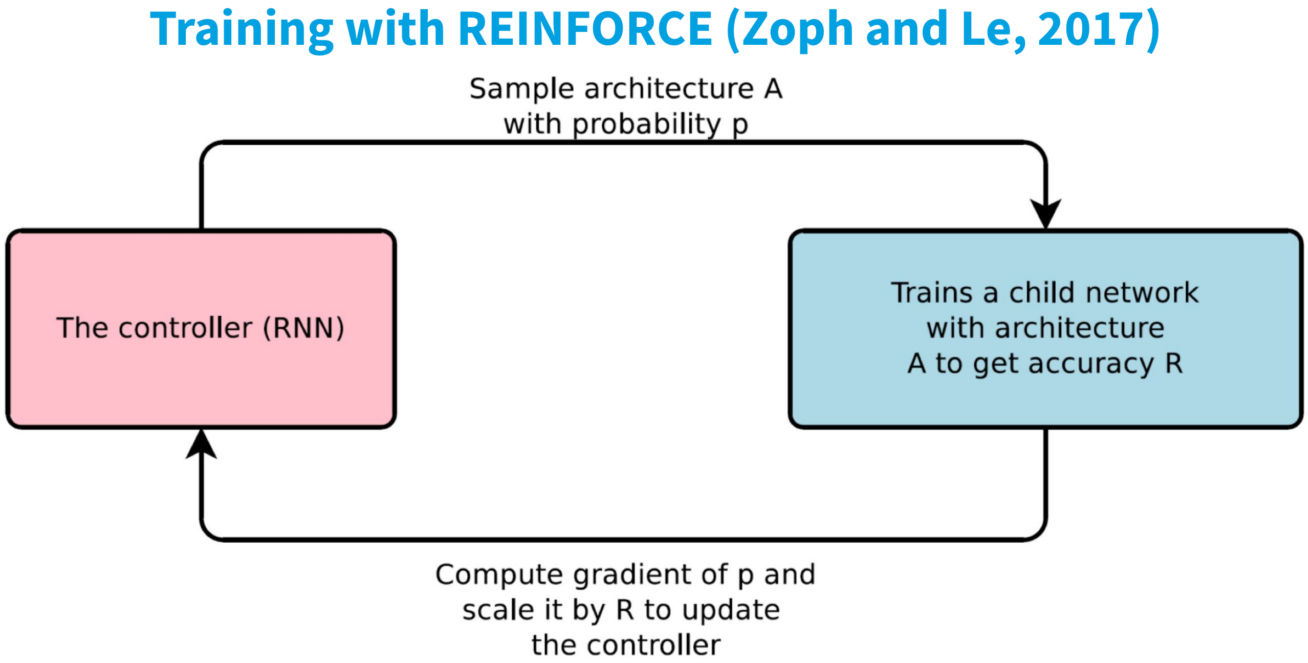
- Specify the structure and connectivity of a neural network by using a configuration string (e.g., *[“Filter Width: 5”, “Filter Height: 3”, “Num Filters: 24”]*)
- Zoph and Le (2017): Use a RNN (“Controller”) to generate this string that specifies a neural network architecture
- Train this architecture (“Child Network”) to see how well it performs on a validation set
- Use reinforcement learning to update the parameters of the Controller model based on the accuracy of the child model

Neural Architecture Search for Convolutional Networks



Zoph et al. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING. ICLR 2017

RL based NAS Search



Zoph et al. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING. ICLR 2017

Training with REINFORCE

Parameters of Controller RNN

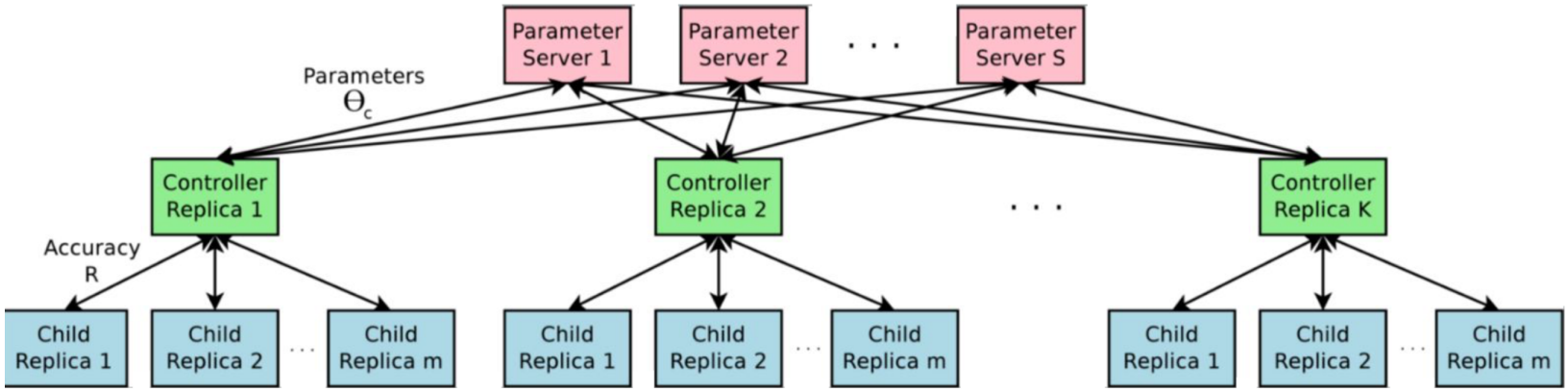
Accuracy of architecture on held-out dataset

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Architecture predicted by the controller RNN viewed as a sequence of actions

Scaling NAS

Distributed Training



Zoph et al. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING. ICLR 2017

Computational Cost of NAS with RL and EA

CIFAR10 dataset

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

Thousands of GPU hours

Witsuba et al. A Survey on Neural Architecture Search. 2019

Differentiable Architecture Search

Continuous relaxation of the architecture representation, allowing efficient search of the architecture using gradient descent

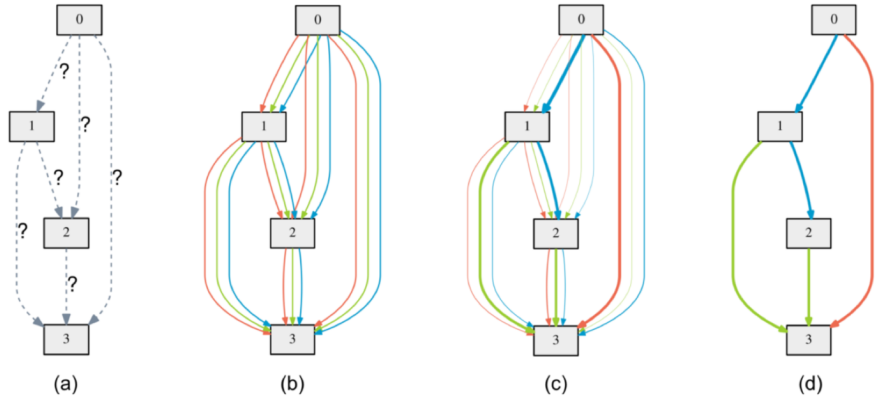


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Liu et al. DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH. ICLR 2019

DARTS

- Goal is to jointly learn the architecture α and the weights w within all the mixed operations (e.g. weights of the convolution filters)
- The continuous variable α determines the operation mixing weights for different pair of nodes in the network

$\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ the training and the validation loss,

- Both losses are determined not only by the architecture α, but also the weights w in the network.

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

Bi-level optimization problem

α as the upper-level variable

w as the lower-level variable

NAS Performance Estimation

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)

Elsken et al. Neural Architecture Search: A Survey. JMLR 2019

TAPAS

- Train-less accuracy predictor for architecture search (TAPAS)
- Predicts accuracy of CNN based classifiers on unseen data without training
- Achieved by adapting past predictions to the “difficulty” level of the dataset
- Characterization of the dataset-difficulty
  - Reliable estimation a CNN performance is dependent on the complexity of dataset

Istrate et al. TAPAS: Train-less Accuracy Predictor for Architecture Search. 2018

TAPAS Framework

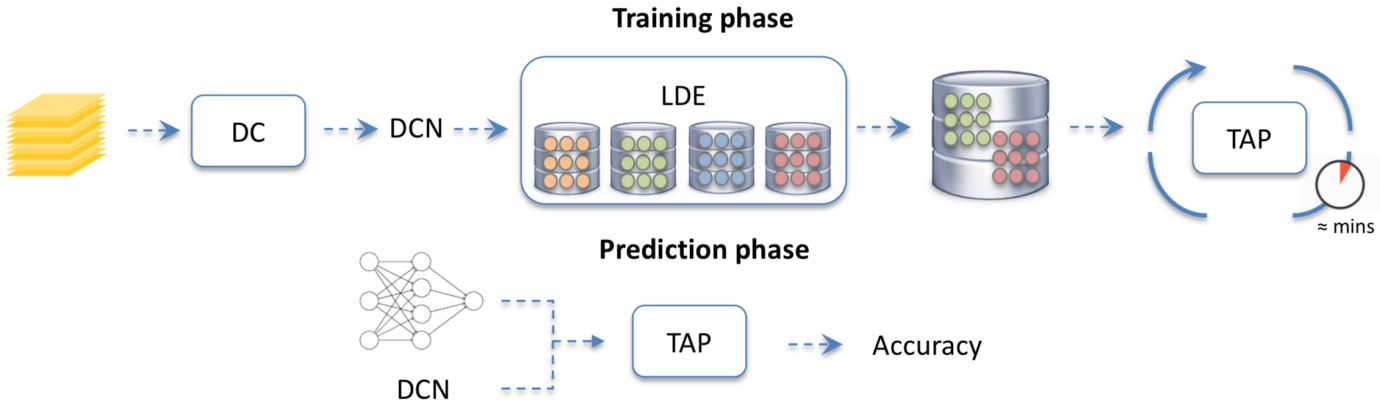


Figure 1: Schematic TAPAS workflow. First row: the Dataset Characterization (DC) takes a new, unseen dataset and characterizes its difficulty by computing the Dataset Characterization Number (DCN). This number is then used to select a subset of experiments executed on similarly difficult datasets from the Lifelong Database of Experiments (LDE). Subsequently, the filtered experiments are used to train the Train-less Accuracy Predictor (TAP), an operation that takes up to a few minutes. Second row: the trained TAP takes the network architecture structure and the dataset DCN and predict the peak accuracy reachable after training. This phase scales very efficiently in a few seconds over a large number of networks.

Istrate et al. TAPAS: Train-less Accuracy Predictor for Architecture Search. 2018

TAPAS Components

1. Dataset Characterization (DC)
   - Receives an unseen dataset and computes a scalar score, namely the Dataset Characterization Number (DCN)
   - DCN is calculated by training Deep Normalized ProbeNet (modest size network) for few epochs
   - DCN is a rough estimation of dataset difficulty, tolerant to approximations
2. Lifelong Database of Experiments (LDE)
   - Ingests training experiments of NNs on a variety of image classification datasets executed inside the TAPAS framework
   - Experiment: CNN architecture description, the training hyper-parameters, the employed dataset (with its DCN), the achieved accuracy
   - A continuously growing DB
   - Returns all experiments performed with datasets with DCN close to the target
3. Train-less Accuracy Predictor (TAP)
   - Given an NN architecture and a DCN, it predicts the accuracy without training the network
   - Employs a layer-by-layer encoding vector of NN architecture
   - 2 stacked LSTM

DCN and LDE

- DCN is a rough estimation of the dataset difficulty
  - Training a *probe net* to obtain a dataset difficulty estimation
- LDE experiments selection

Let us consider an LDE populated with experiments from N_d different datasets D_j, with $j = 1, \ldots, N_d$. Given a new input dataset $\hat{D}$ and its corresponding characterization $\mathrm{DCN}(\hat{D})$, the LDE block returns all experiments performed with datasets that satisfy the following relation

$$\|\mathrm{DCN}(\hat{D}) - \mathrm{DCN}(D_j)\| \leq \tau \qquad j \in [1, N_d], \tag{1}$$

where τ is a predefined threshold that, in our experiments, is set to 0.05.

Train-less accuracy predictor (TAP)

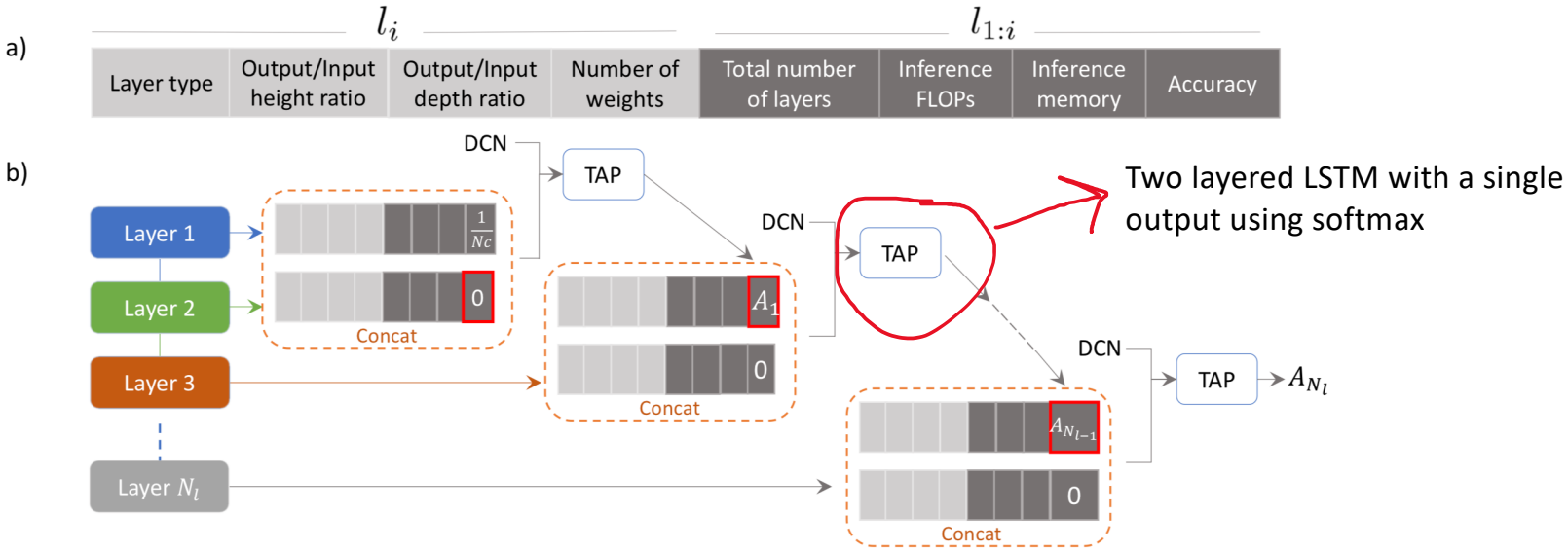


Figure 2: Encoding vector structure and its usage in the iterative prediction. a) The encoding vector contains two blocks: i-layer information and from input to i-layer sub-network information. b) The encoding vector is used by the TAP following an iterative scheme. Starting from Layer 1 (input) we encode and concatenate two layers at a time and feed them to the TAP. In the concatenated vector, the *Accuracy* field A_i of l_i is set to the predicted accuracy obtained from the previous TAP evaluation, whereas the one of A_{i+1} corresponding to l_{i+1} is always set to zero. For the input layer, we set A_0 to $1/N_c$, where N_c is the number of classes, assuming a random distribution. The final predicted accuracy A_{N_l} is the accuracy of the complete network.

Istrate et al. TAPAS: Train-less Accuracy Predictor for Architecture Search. 2018

TAPAS Performance

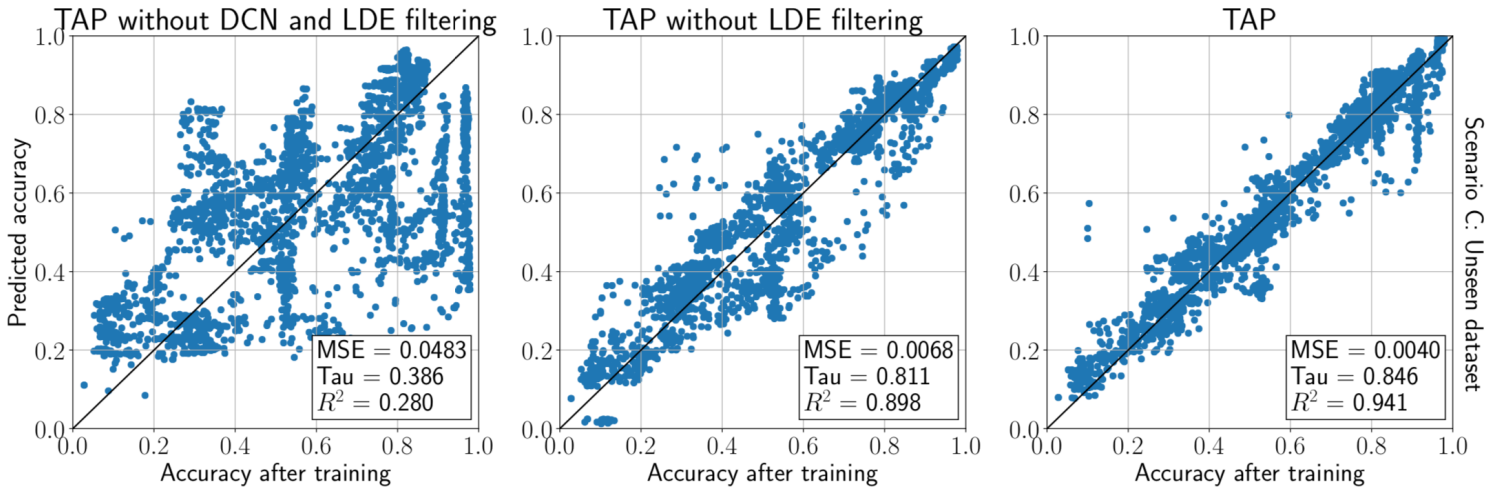


Figure 5: Predicted vs real performance (i.e., after training) for Scenario C. Left plot: TAP trained without DCN or LDE pre-filtering. Middle plot: TAP trained with DCN, but LDE is not pre-filtered. Right plot: TAP trained only on LDE experiments with similar dataset difficulty, according to (1).

Istrate et al. TAPAS: Train-less Accuracy Predictor for Architecture Search. 2018

NeuNetS: Neural Network Synthesis

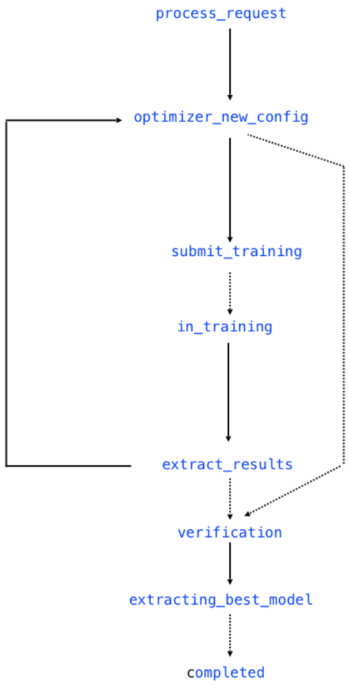
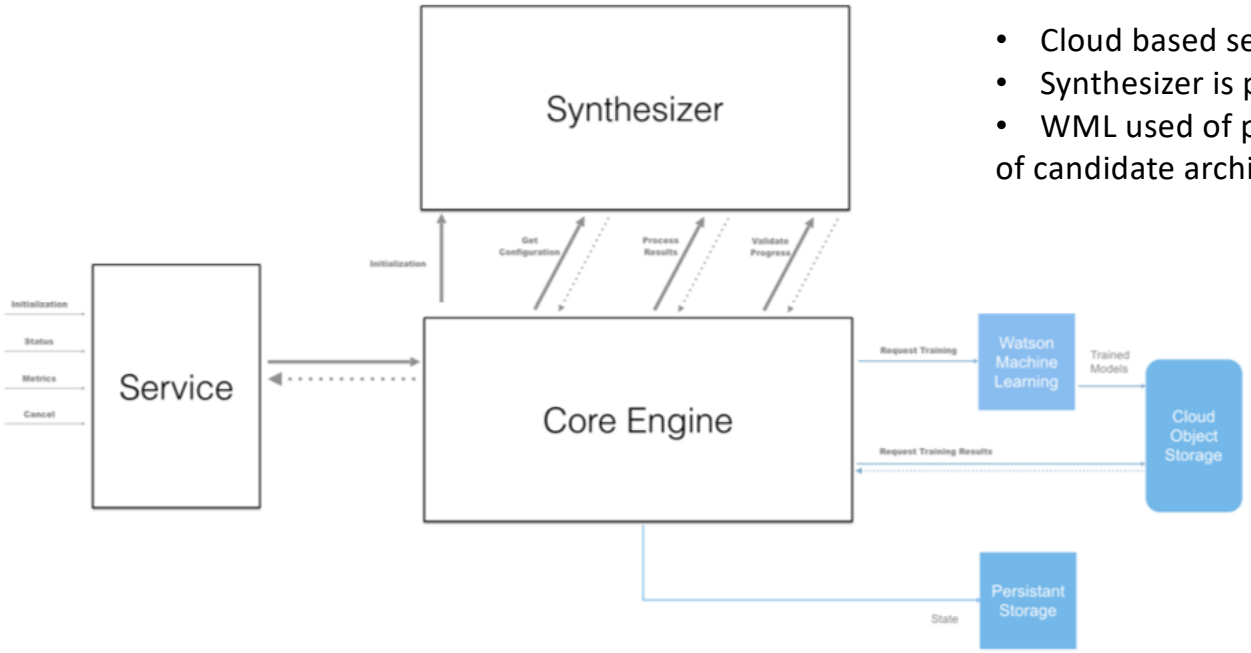


Figure 1: Execution Pipeline Operational States

The lifecycle of a NeuNetS project consists of a series of states or stages, as detailed in Fig. (1). During this lifecycle, the synthesis states are executed multiple times to explore/evolve, train, and evalu- ate different networks. Once stopping conditions, whether budgetary or algorithmic, are reached, the synthesis loop ends and final results are extracted to the user's storage instance.

Sood et al. NEUNETS: AN AUTOMATED SYNTHESIS ENGINE FOR NEURAL NETWORK DESIGN. 2019

NeuNetS Component Architecture



- Cloud based service
- Synthesizer is pluggable
- WML used of performance estimation of candidate architecture

Sood et al. NEUNETS: AN AUTOMATED SYNTHESIS ENGINE FOR NEURAL NETWORK DESIGN. 2019

NeuNetS Visualization

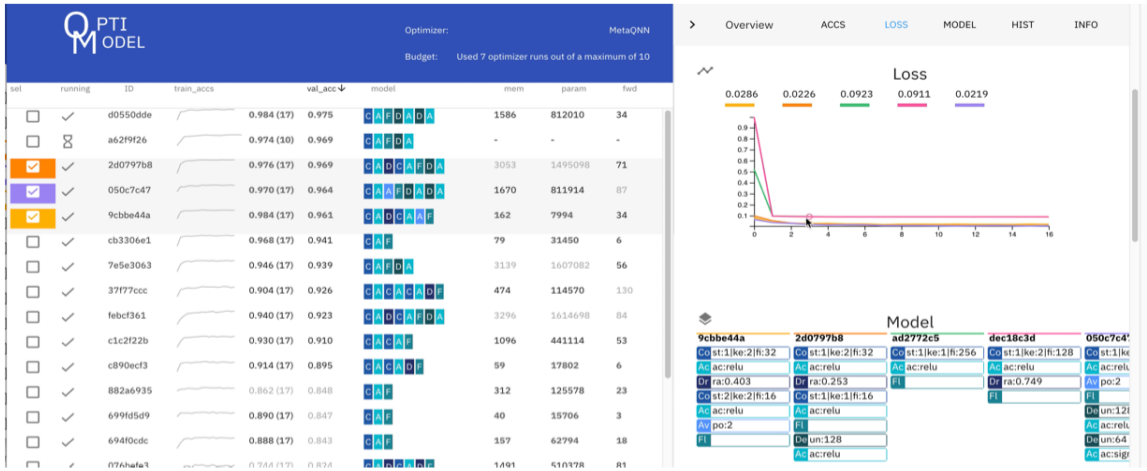


Figure 3: NeuNetS visual interface to manually pick a best model. Left: list of models in training and trained models with measures of performance and architecture diagram. Right: Details to compare selected models in depth.

Sood et al. NEUNETS: AN AUTOMATED SYNTHESIS ENGINE FOR NEURAL NETWORK DESIGN. 2019

Hyperparameter optimization

- Random search
- Bayesian optimization methods
  - Snoek et al., 2012; Hutter et al., 2011; Bergstra et al., 2011; Thornton et al., 2013; Eggensperger et al., 2013; Snoek et al., 2015b
- Hyperband

Kubeflow Katib

- K8S based system for hyperparameter optimization and NAS
- https://github.com/kubeflow/katib
- https://www.kubeflow.org/docs/components/hyperparameter-tuning/overview/
- https://www.kubeflow.org/docs/components/hyperparameter-tuning/hyperparameter/

Reference Papers

- Zoph et al. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING. ICLR 2017
- Elsken et al. Neural Architecture Search: A Survey. JMLR 2019
- Witsuba et al. A Survey on Neural Architecture Search. 2019
- Liu et al. DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH. ICLR 2019
- Istrate et al. TAPAS: Train-less Accuracy Predictor for Architecture Search. 2018
- Sood et al. NEUNETS: AN AUTOMATED SYNTHESIS ENGINE FOR NEURAL NETWORK DESIGN. 2019
- Li et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2018

Reference Materials

- AutoML.org. Literature on Neural Architecture Search. Compilation of papers in NAS. Regularly updated.
- Zoph et al. Neural Architecture Search with Reinforcement Learning. Lecture slides.
- Nikhil Naik. Neural Architecture Search Tutorial. CVPR 2019