Course Notes: Deep Learning for Visual Computing

Peter Wonka

October 24, 2021

Contents

1 Neural Architecture Search

1.1 Neural Architecture Search Components

- The search space that defines what type of architecture is being searched
- The search strategy / algorithm
- The performance estimation strategy
  - to evaluate the performance of a possible network from its design, e.g. without constructing and training it or by constructing and training it.

1.2 Search Strategy

- Reinforcement learning
- Evolutionary algorithms
- Genetic algorithms
- Sequential model-based optimization (SMBO)
- Bayesian optimisation
- Hill-climbing
- Multi-objective search
- Supernetwork search
- Gradient-based optimization

1.3 Search Strategy

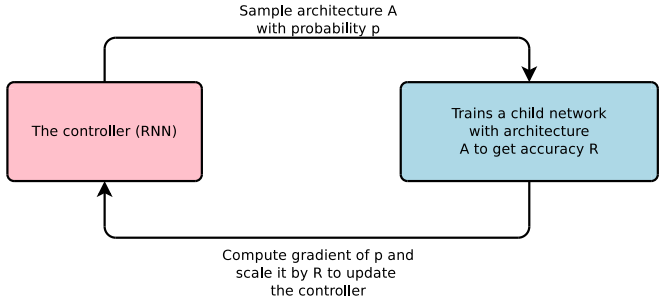
- Searching complete architecture vs. searching for cells
  - macro: searching for the complete architecture. More difficult to define the search space
  - micro: search only for one or multiple cell architectures. Combine cells into hand crafted overall architecture

1.4 Literature

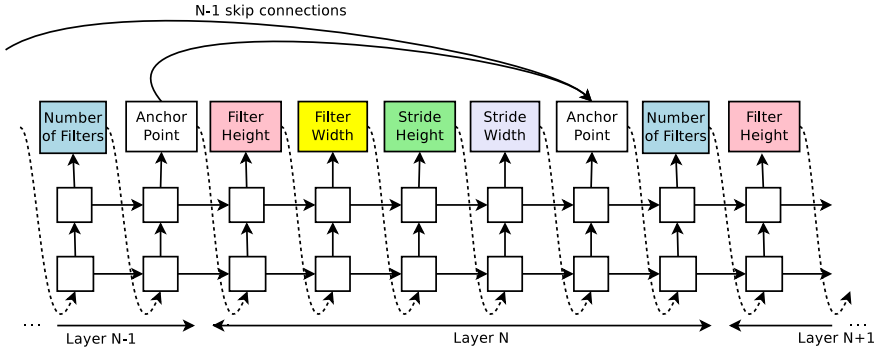
- Literature List

1.5 Neural Architecture Search using RL

- Literature:
  - Barret Zoph, Quoc V. Le, Neural Architecture Search with Reinforcement Learning, ICLR 2017
  - youtube talk
- Warning: uses 800 GPUs for training



- Overview of Reinforcement Learning for Neural Architecture Search:
  - Controller network proposes child architectures (sampled)
  - Child architectures are trained to convergence and evaluated on val set
  - Performance on val set is used to give rewards to the controller network using the REINFORCE algorithm
  - Many child architectures trained in parallel with asynchronous optimization



- Controller network generates network one layer at a time
  - softmax to choose between discrete choices
  - filter height $\in [1, 3, 5, 7]$
  - filter width $\in [1, 3, 5, 7]$
  - number of filters $\in [24, 36, 48, 64]$

- stride $\in [1,2,3]$ (or always 1)
- skip connections
- Controller RNN is a two-layer LSTM with 35 hidden units each layer

1.6 RL details

The list of tokens that the controller predicts can be viewed as a list of actions $a_{1:T}$ to design an architecture for a child network. At convergence, this child network will achieve an accuracy R on a held-out dataset. We can use this accuracy R as the reward signal and use reinforcement learning to train the controller. More concretely, to find the optimal architecture, we ask our controller to maximize its expected reward, represented by $J(\theta_c)$:

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R] \tag{1.1}$$

Since the reward signal R is non-differentiable, we need to use a policy gradient method to iteratively update θ_c. In this work, we use the REINFORCE rule from [**Williams92simplestatistica**

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\big[\nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R\big] \tag{1.2}$$

An empirical approximation of the above quantity is:

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)R_k \tag{1.3}$$

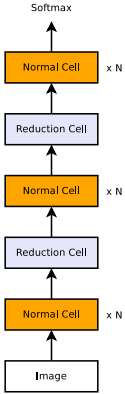
Where m is the number of different architectures that the controller samples in one batch and T is the number of hyperparameters our controller has to predict to design a neural network architecture. The validation accuracy that the k-th neural network architecture achieves after being trained on a training dataset is R_k.
The above update is an unbiased estimate for our gradient, but has a very high variance. In order to reduce the variance of this estimate we employ a baseline function:

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k-b) \tag{1.4}$$

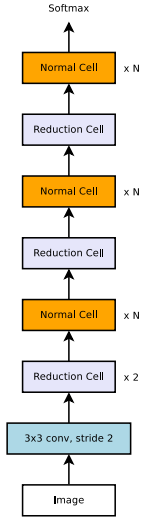
As long as the baseline function b does not depend on the on the current action, then this is still an unbiased gradient estimate. In this work, our baseline b is an exponential moving average of the previous architecture accuracies.

1.7 NAS-NET

- Literature:
  - Zoph et al., Learning Transferable Architectures for Scalable Image Recognition, CVPR 2018
- Still hardware intensive: only 500 GPUs
- Main ideas:
  - run architecture search for cells on CIFAR-10 and transfer to imagenet
  - create a better search space
  - realization that random search is similar to RL in performance

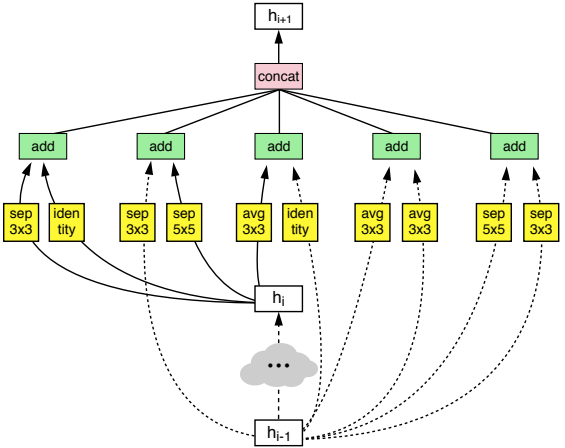


CIFAR10
Architecture

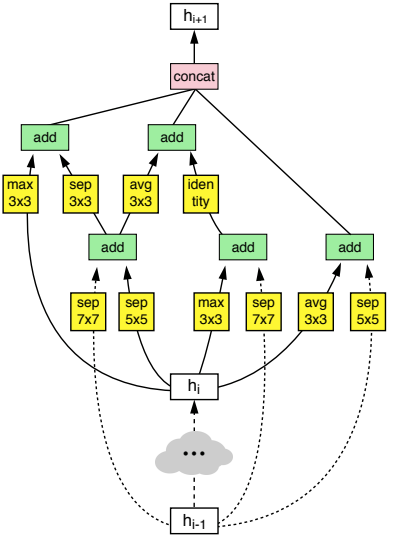


ImageNet
Architecture

- NASNET architecture for CIFAR and ImageNet. N can vary.



Normal Cell



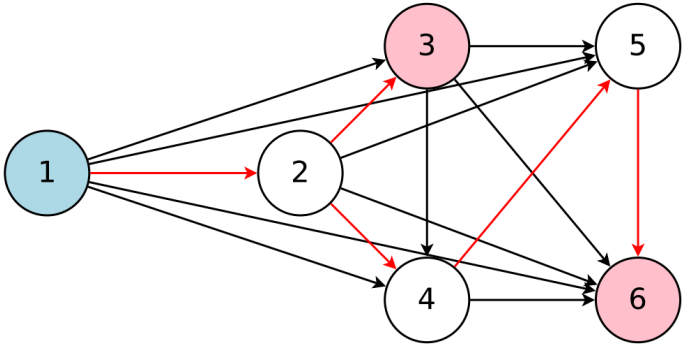
Reduction Cell

- Cells Found for NASNET-A
  - Normal Cells (output has same resolution as input)
  - Reduction Cells (resolution is divided by two in each dimension)
  - input to a cell are the outputs of two lower cells
- Controller Choices:
  - Select a hidden state from h_i, h_{i-1} or from the set of hidden states created in previous blocks.
  - Select a second hidden state from the same options as in Step 1.
  - Select an operation to apply to the hidden state selected in Step 1.
  - Select an operation to apply to the hidden state selected in Step 2.
  - Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.
- Operations

- identity
- 1x3 then 3x1 convolution
- 1x7 then 7x1 convolution
- 3x3 dilated convolution
- 3x3 average pooling
- 3x3 max pooling
- 5x5 max pooling
- 7x7 max pooling
- 1x1 convolution
- 3x3 convolution
- 3x3 depthwise-separable conv
- 5x5 depthwise-seperable conv
- 7x7 depthwise-separable conv

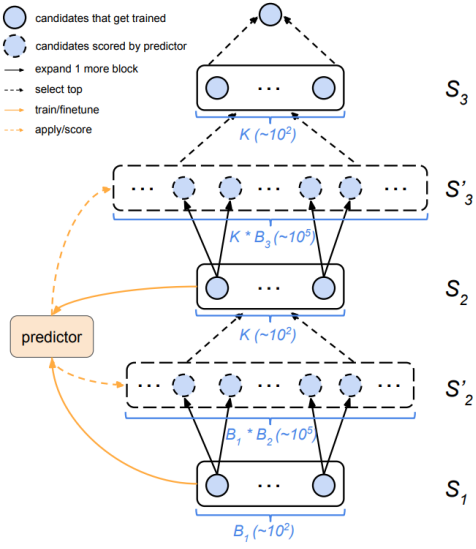
1.8 ENAS

- Literature: Pham et al., Efficient Neural Architecture Search via Parameter Sharing, 2018
- Main Idea:
  - We observe that the computational bottleneck of NAS is the training of each child model to convergence, only to measure its accuracy whilst throwing away all the trained weights
  - Weight sharing between different architectures
  - Setup a superarchitecture and train networks that are sub-graphs of the superarchitecture
  - All computations can be done on a single GPU
  - Results comparable to previous work


1
2
3
4
5
6

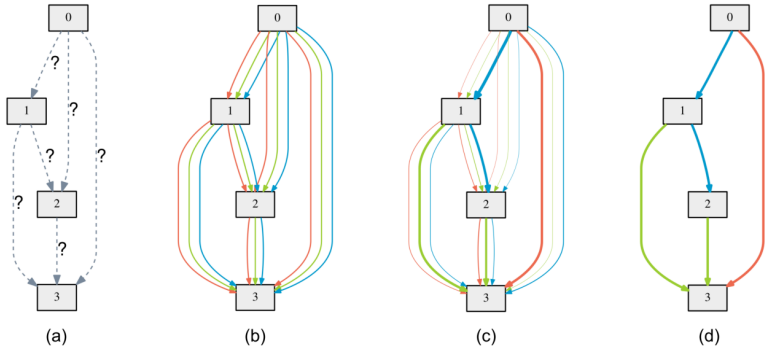
1.9 PNAS

- Literature: Liu et al., Progressive Neural Architecture Search, 2017
- Algorithm Outline:
  - Start with a set of small 1 block networks
  - Pick the well performing ones and make modifications (add blocks)
  - Repeat
- Use a predictor to predict performance without fully training the networks


candidates that get trained
candidates scored by predictor
expand 1 more block
select top
train/finetune
apply/score
predictor
S_3
$K\ (\sim 10^2)$
S'_3
$K * B_3\ (\sim 10^5)$
S_2
$K\ (\sim 10^2)$
S'_2
$B_1 * B_2\ (\sim 10^5)$
S_1
$B_1\ (\sim 10^2)$

1.10 Darts

- Literature: Liu et al., DARTS: Differentiable Architecture Search, 2018



- Each node $x^{(i)}$ is a latent representation
  - e.g. a feature map in convolutional networks

- Each directed edge (i, j) is some operation $o^{(i,j)}$ that transforms node $x(i)$
  - Each edge has multiple candidate operations $o^{(i,j,k)}$, e.g., convolution, max pooling, zero (for no connection)
- A cell has two input nodes and a single output node
  - For convolutional cells, the input nodes are the cell outputs in the previous two layers
- Each intermediate node is computed based on all of its predecessors:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \tag{1.5}$$

- All k candidate operations on an edge have a weight $\alpha^{(i,j,k)}$
- During training: all candidate operations are added using softmax on the alphas
- For the final model: softmax is replaced with the most likely candidate operation
- The bi-level optimization is described in the paper (skipped here)