Automatic Search of Neural Network Architectures

Jianping Fan
Department of Computer Science
UNC-Charlotte

Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html

Neural Network Architecture Selection

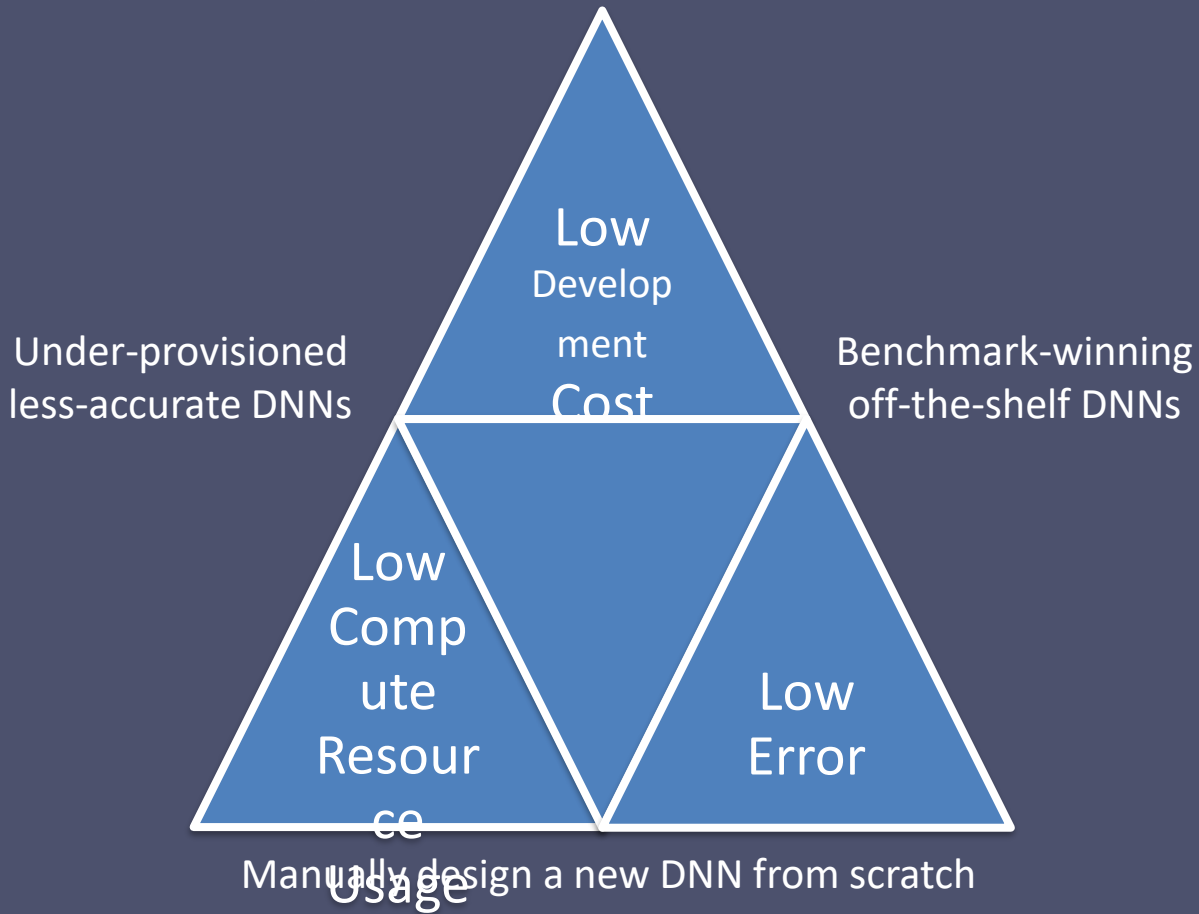
One of the most tedious tasks in the world of Machine Learning is designing and building neural network architectures. Typically humans will spend hours or days trying to iterate through different neural network architecture with different hyper parameters in order to optimize an objective function for a task at hand. This is very time consuming and often prone to errors.

Google introduced the idea of implementing Neural Network Search by employing evolutionary algorithms and reinforcement learning in order to design and find optimal neural network architecture. In essence, what this is doing is that it is training to create a layer and then stacking those layers to create a Deep Neural Network architecture. This area of research has a drawn a lot of attention recently.

NASNet: Learning Transferable Architectures for Scalable Image Recognition, CVPR 2017

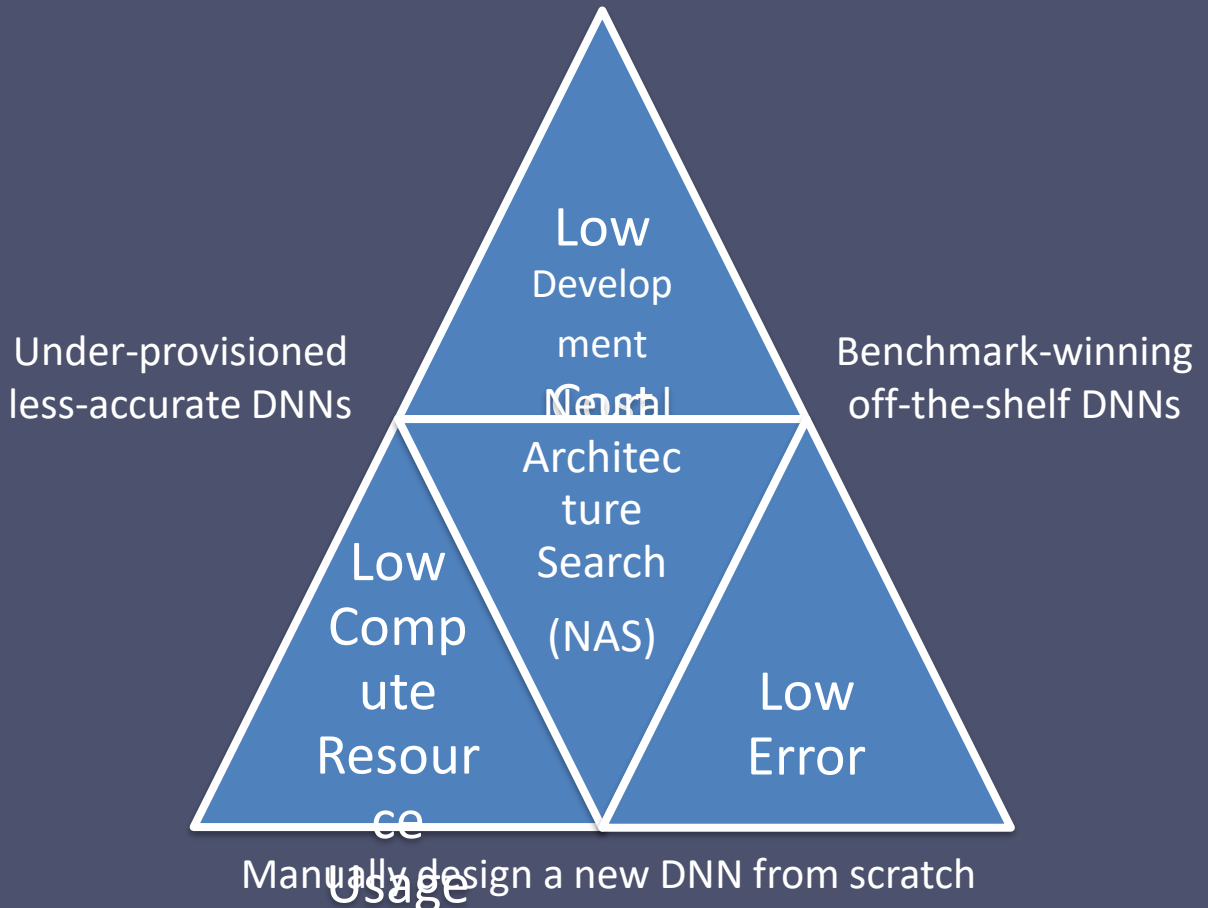
Tradeoffs for deployable DNN models

for automotive deep learning practitioners



Neural Architecture Search (NAS) to the Rescue

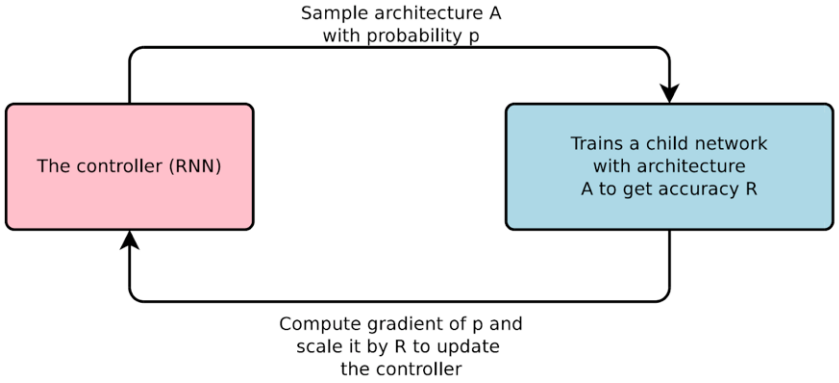
NAS can co-optimize resource-efficiency and accuracy

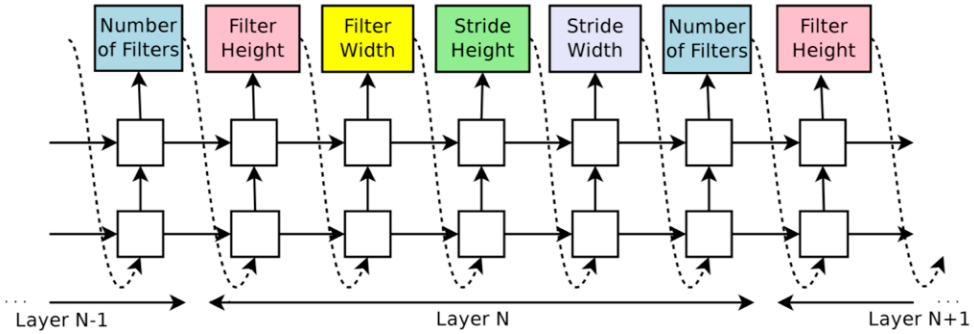


- **Related work on**
- **Neural Architecture Search**

Neural Architecture Search

- We want to find networks that are the best tradeoff of compute and accuracy in some search space
- Within some search space we can vary the number of layers, channels, kernel size, connections, etc…
- It is beyond intractable to train every one
- Can we do better?
  - Random Search
  - Genetic Search
  - **Reinforcement Learning**
  - **Differential Search (Gradient Based)**

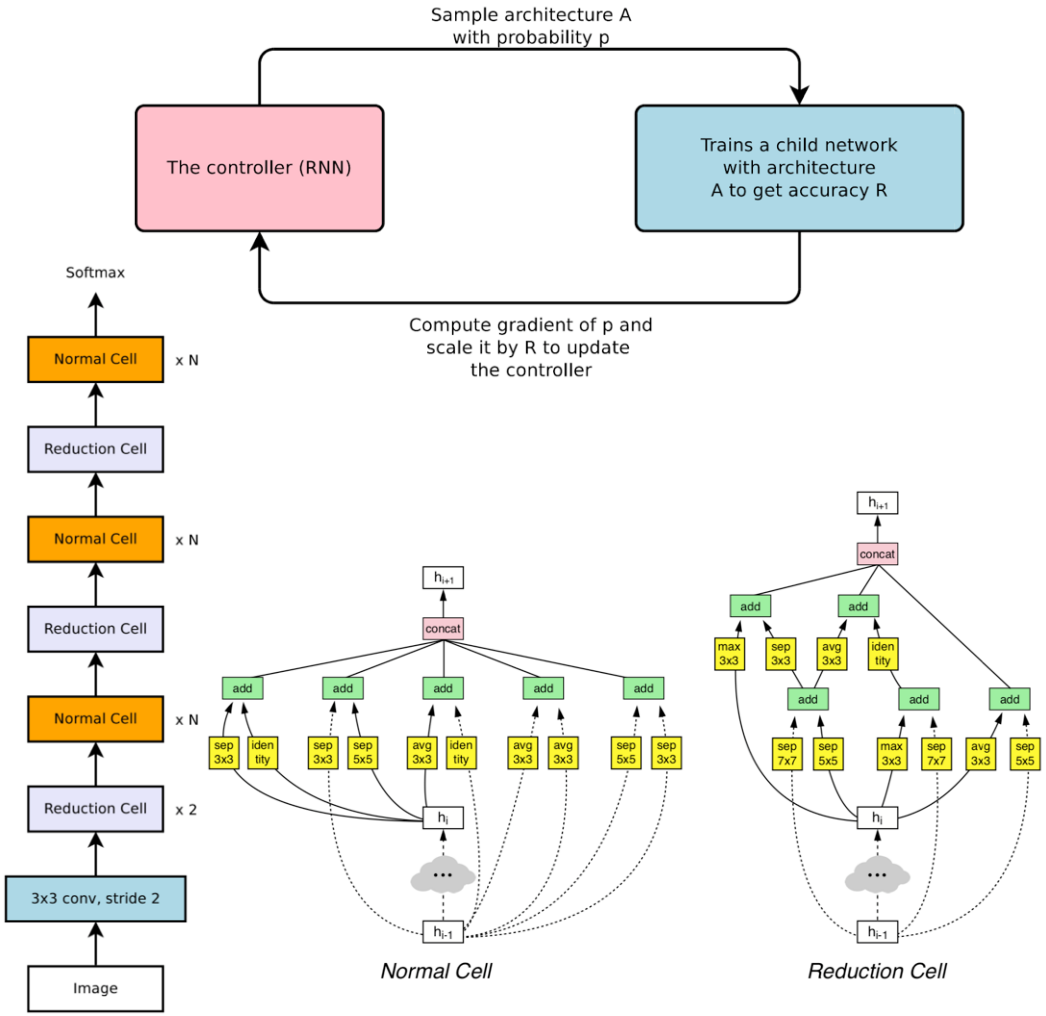




- **NAS with Reinforcement Learning**
- **Block-level search [1]**
- Use a Recurrent Neural Network in a RL loop to generate entire child network for the CIFAR dataset updating after each model has trained
- Achieved 0.09% better accuracy at the time and 1.05x faster on CIFAR-10
- 800 Nvidia K40 GPUs for 28 days = 537,600 GPU Hours
- Search performed on small dataset
- Better than brute force approach but still too much compute too be practical

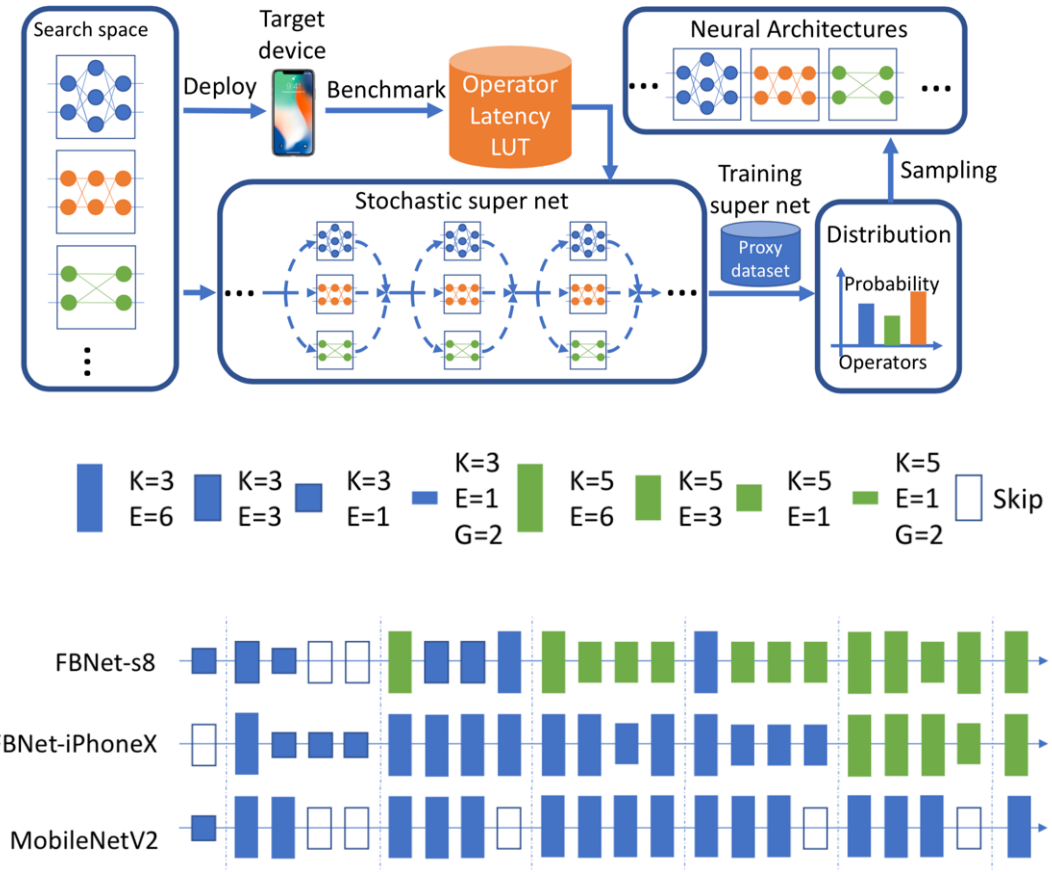
- [1] B. Zoph, Q. Le. Neural Architecture Search with Reinforcement Learning. ICLR, 2018.

DEEPSCALE



- **NAS with Reinforcement**
- **Learning**
- Cell-level search [2]
- Use a Recurrent Neural Network in a RL loop to generate cells using CIFAR-10 as proxy task then adapted to ImageNet
- Achieved 1.20% better accuracy while being 28% faster on ImageNet1000
- 500 Nvidia P100 GPUs for 4 days = 48,000 GPU Hours
- cells are all the same (unlike [1])
- More efficient than previous method but still expensive

- [1] B. Zoph, Q. Le. Neural Architecture Search with Reinforcement Learning. ICLR, 2018.
- [2] B. Zoph et al. Learning Transferable Architectures for Scalable Image Recognition. CVPR, 2018.



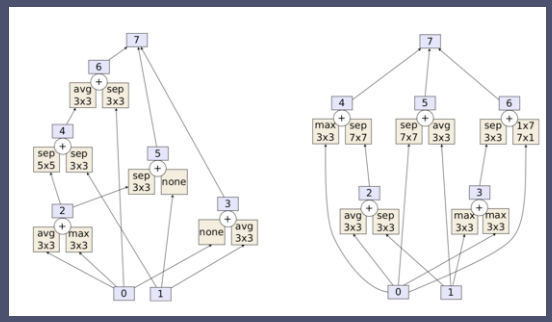
Differential Search

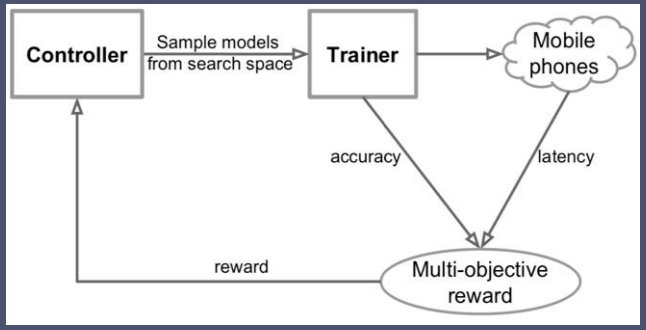
- FBNet [3]
- Use a gradient based stochastic supernet that is dually optimized on a 10% subset of ImageNet which is then trained on ImageNet 1k
- Uses Gumbel-Softmax to sample from categorical distribution for layer choices weighted by learnable parameters
- FBNet-B achieved MobileNetV2-1.3 Accuracy while being 1.5x lower latency
- 216 P100 GPU hours Search Cost
- Search Space inspired by MobilenetV2

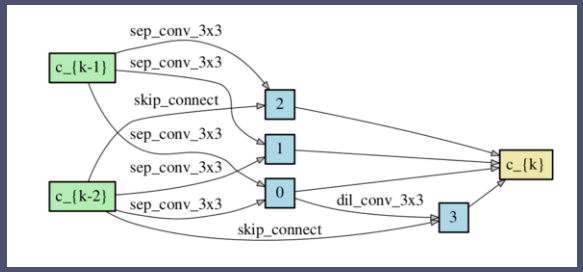
- [3] B. Wu, et al. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. CVPR, 2019.

Other Related Work

- Genetic Search
  - AmoebaNet[1]
    - Tournament Selection Evolutionary on Cell Space
    - 75,600 K40 GPU hours
- Reinforcement Learning
  - MnasNet[2]
    - Latency Aware Block level Search on proxy ImageNet
    - 6912 TPUv2 hours ≈ 48,000 P100 GPU hours
- Differential Search
  - DARTS[3]
    - Gradient Based Cell Search performed on CIFAR-10
    - 96 1080 TI GPU hours







[1] E. Real et al. Regularized Evolution for Image Classifier Architecture Search. AAAI, 2019.
[2] M. Tan et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. CVPR, 2019.
[3] H. Liu et al. DARTS: Differentiable Architecture Search. ICLR, 2019.

- **Applying NAS to design DNNs for semantic segmentation**

Classification vs Semantic Segmentation

Examples of image classification (ImageNet[1])

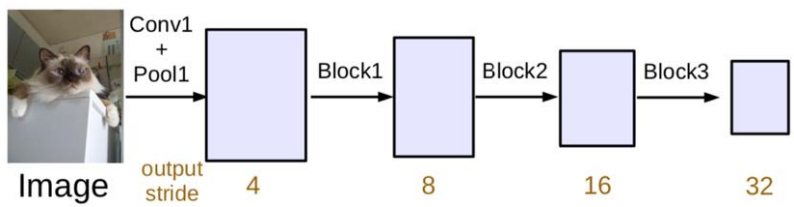
Example of Semantic Segmentation (Cityscapes[2])

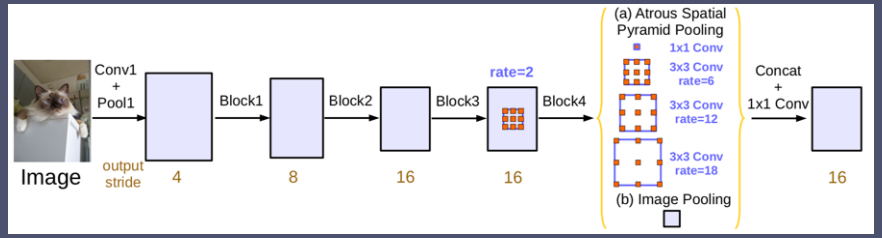
• Image level prediction • Location Invariant • Low Resolution (224x224 input) • SOTA Networks compute: ~10 GFLOPs • Networks designed for task and are trained from scratch	• Pixel level prediction • Location Variant • High Resolution (1024x2048 input) • SOTA Networks range: ~1 **T**FLOPS • SS Networks are adapted from classification networks and then retrained.

[1] O. Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.

[2] M. Cordts et al. The Cityscapes Dataset for Semantic Urban Scene Understanding. CVPR, 2016.

Classification vs Semantic Segmentation (cont.)





Examples of image classification

- Image level prediction
- Location Invariant
- Low Resolution (224x224 input)
- SOTA Networks compute: ~10 GFLOPs

- Networks designed for task and are trained from scratch

Example of Semantic Segmentation (DeepLabV3[1])

- Pixel level prediction
- Location Variant
- High Resolution (1024x2048 input)
- SOTA Networks range: ~1 **T**FLOPS

- SS Networks are adapted from classification networks and then retrained.

[1] LC. Chen et al. Rethinking Atrous Convolution for Semantic Image Segmentation, 2017.

Applying NAS to design DNNs for semantic segmentation

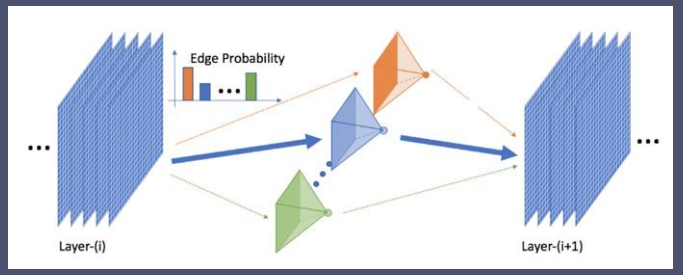
- Goal: advance the frontier of accuracy/efficiency on Semantic Segmentation
- Related work typically starts with ImageNet training, then adapts and fine-tunes the DNN for semantic segmentation
- If we do NAS directly on sematic segmentation (rather than starting with ImageNet), will this give us better results?

Examples of image classification (ImageNet)

Example of Semantic Segmentation (Cityscapes)

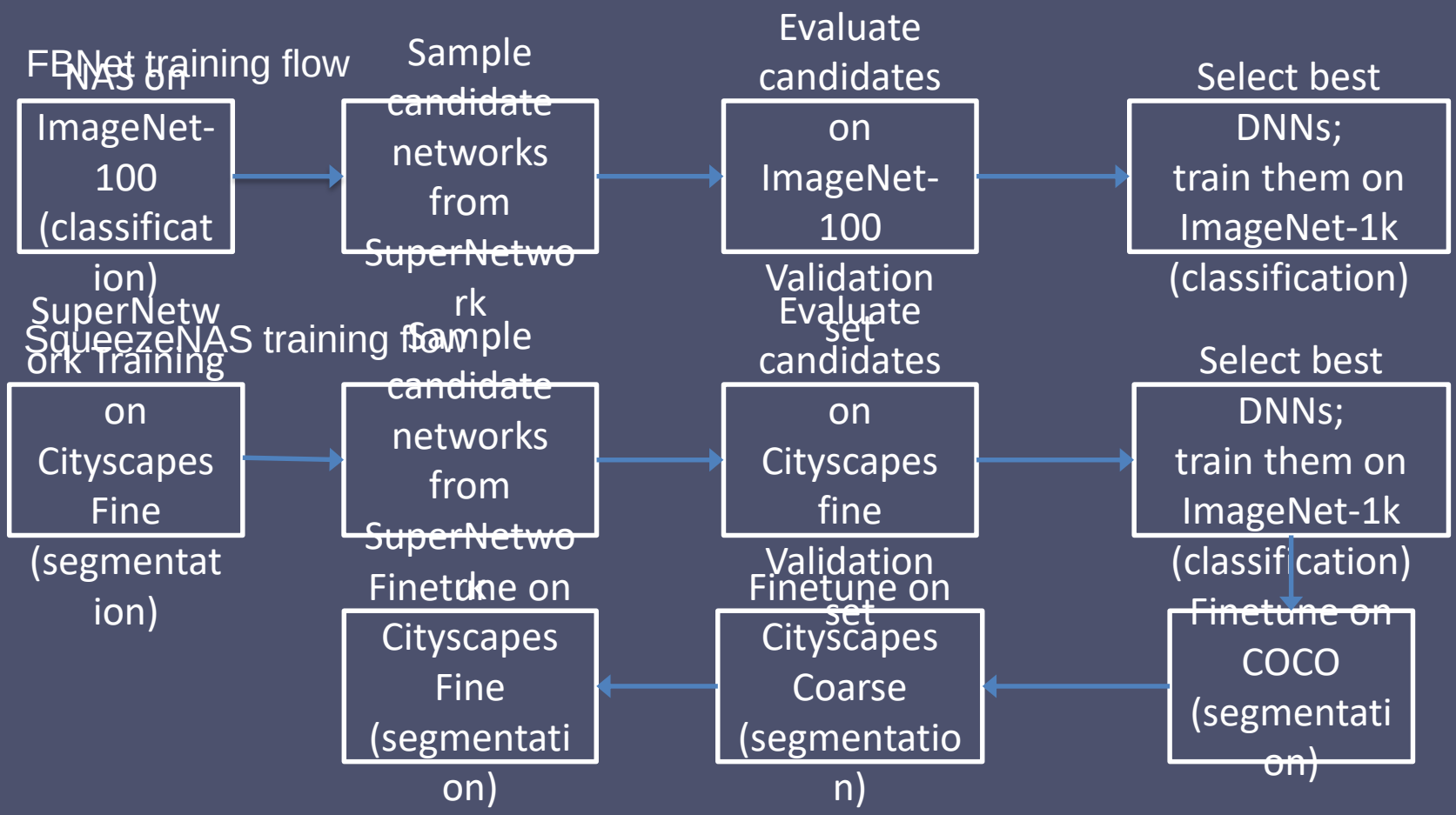
SqueezeNAS: An Adaptation of FBNet for Semantic Segmentation Search

Figure courtesy of Bichen W

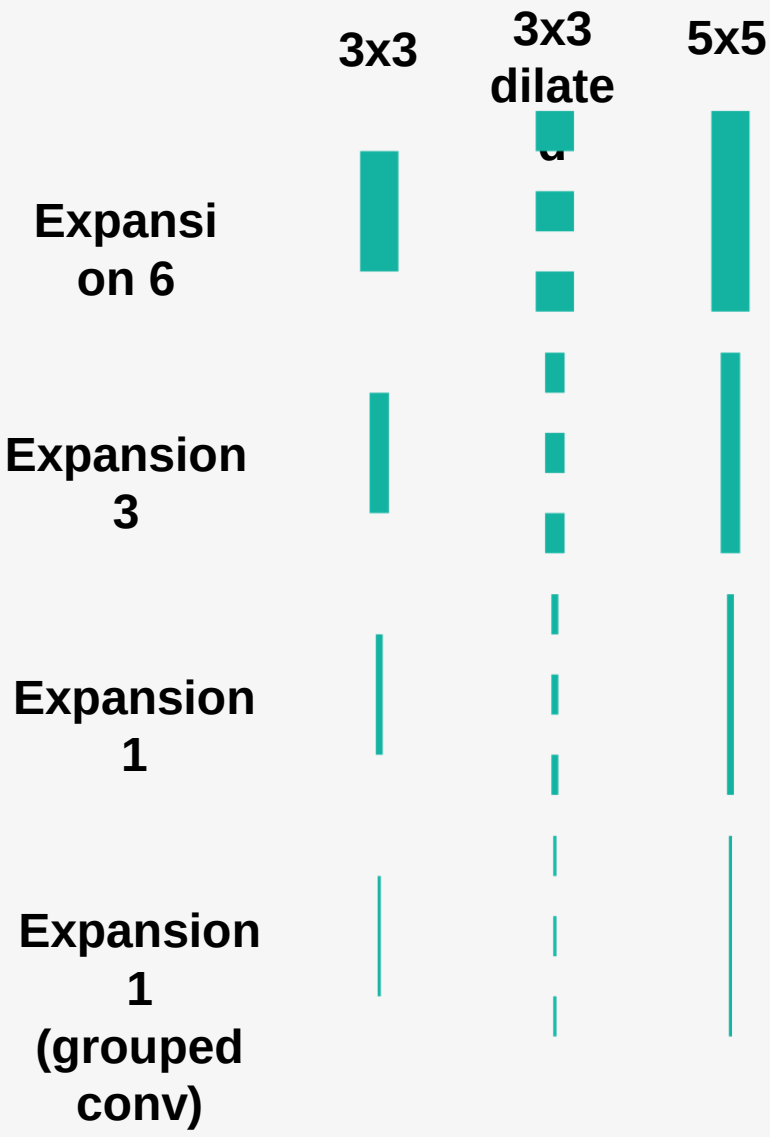


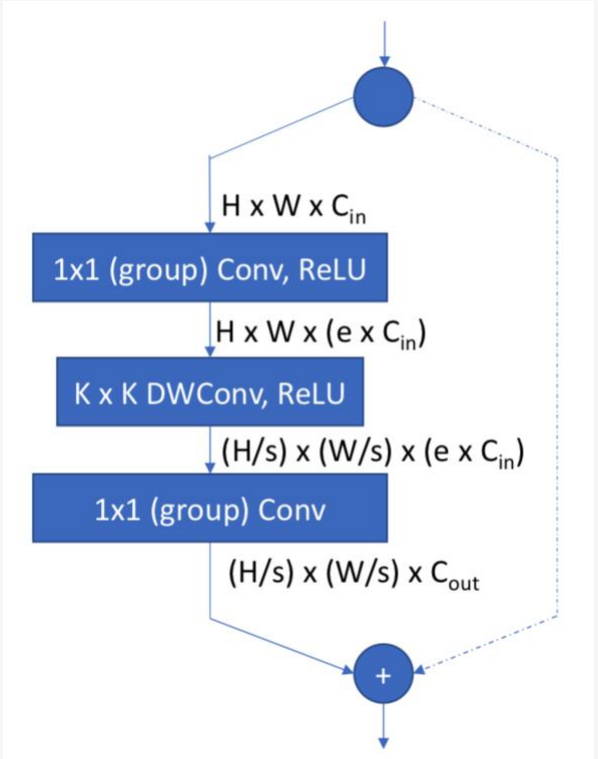
- Stochastic Super Network
  - Run all units in parallel
  - Perform weighted sum of activations where weights are sampled from Gumbel-Softmax
  - 2 types of learned parameters: Convolution parameters and architecture parameters
- Resource aware learned architecture parameter
  - A unit in the meta-network is chosen by its architecture parameter plus a random variable
  - Optimize parameters of network (convolutions) and parameters for unit choices in an alternating fashion
- Proxyless training
  - We train directly on high-resolution cityscapes
  - Training until both convolution parameters and unit parameters converge

- **Training scheme**



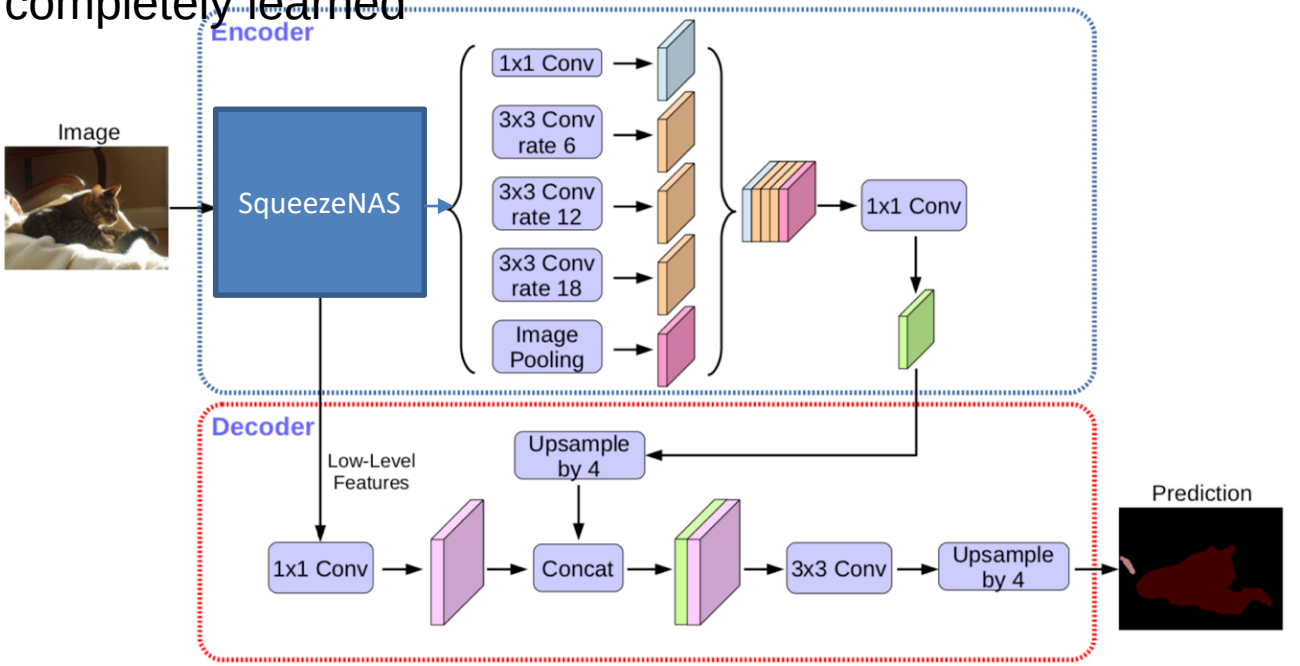
- Search Space





- Search Space
- (cont)

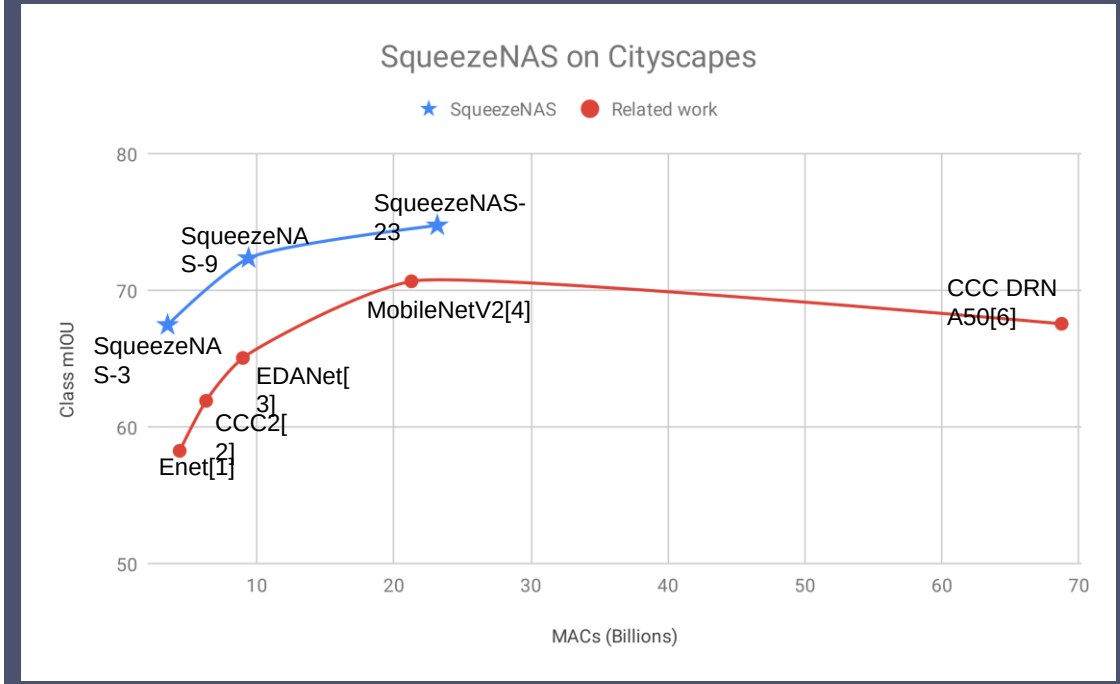
We employ the encoder-decoder depthwise head from DeepLab V3+[1] while allowing the base network to be completely learned



[1] Chen et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, EC

- **SqueezeNAS Results**

SqueezeNAS: CityScapes Results

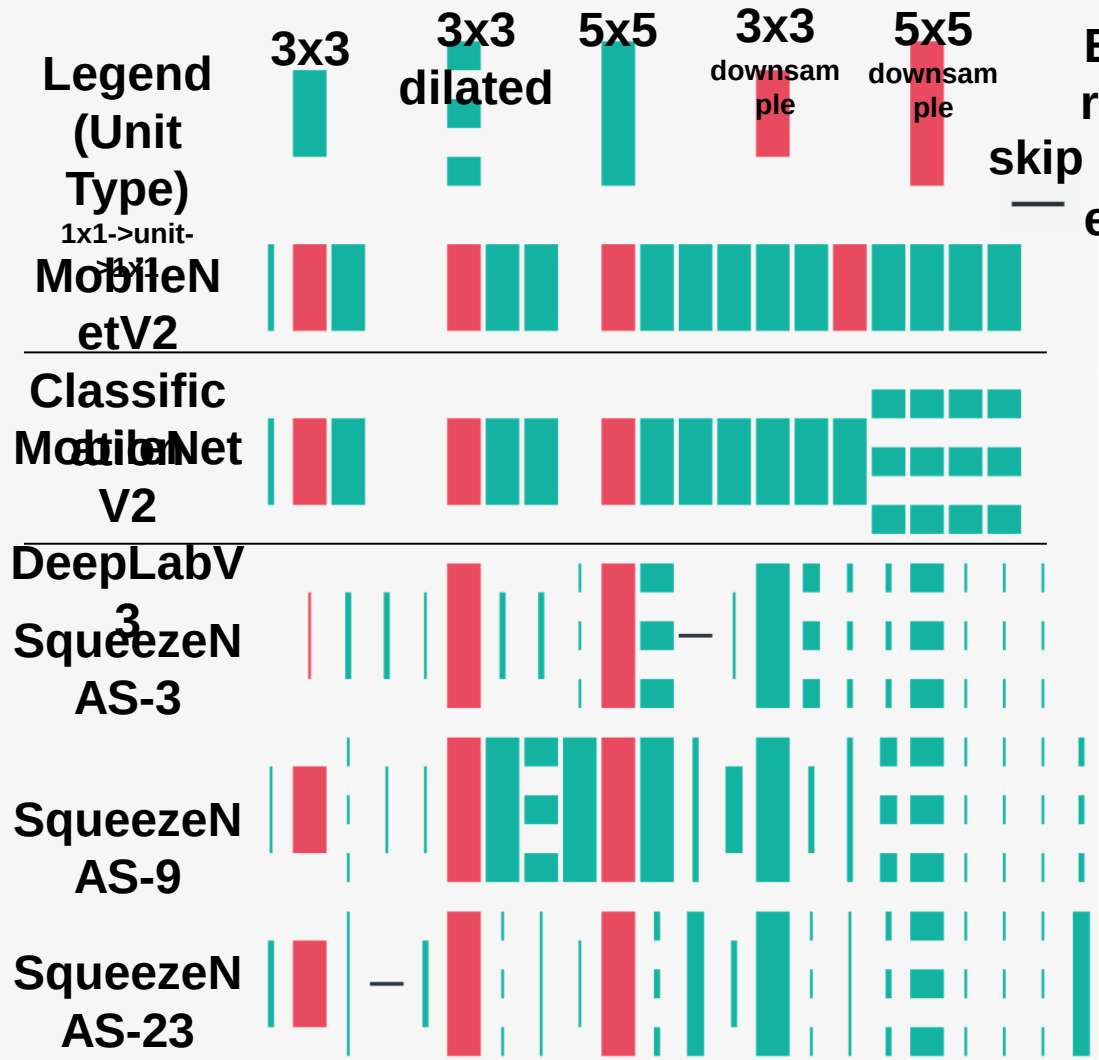


Name	MACs (Billions)	Class mIOU on Cityscapes
SqueezeNAS-3	3.5	67.5
SqueezeNAS-9	9.4	72.4
SqueezeNAS-23	23.2	74.8
Enet[1]	4.4	58.3
CCC2[2]	6.3	62.0
EDANet[3]	9.0	65.1
MobileNetV2 OS=16[4]	21.3 [5]	70.7 [5]
CCC DRN A50[6]	68.7	67.6

[1] Paszke, Adam et al. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation, 2016
[2] Park, Hyojin et al. Concentrated-Comprehensive Convolutions for lightweight semantic segmentation, 2018
[3] Lo, Shao-Yuan et al. Efficient Dense Modules of Asymmetric Convolution for Real-Time Semantic Segmentation, 2018
[4] Sandler, Mark et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks, CVPR 2018.
[5] https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.md
[6] Yu, Fisher et al. Dilated Residual Networks, CVPR 2017.

SqueezeNAS

- **Resulting Networks**



	MACs (Giga)	mIOU %
MobileNetV2		
Classification MobileNetV2	21.3	70.71
DeepLabV3		
SqueezeNAS-3	3.5	67.5
SqueezeNAS-9	9.4	72.4
SqueezeNAS-23	23.2	74.8

Analysis of Networks

- Dilation (Atous) Convolutions use later in the network
  - All of our Searches led to heavily dilated networks towards the head with no priors
  - This follows human intuition from many SOTA segmentation papers (DeepLab V3)

- Downsample Units
  - Our networks also typically picked the most computationally expensive unit when performing a Downsample (strided unit)
  - This is different from typical networks which usually perform a 3x3 kernel when performing a strided convolution

- Non-Uniform Compute
  - Our networks varied the amount of compute each unit used unlike typical networks (ResNet, MobileNet, etc) which use the same of amount of compute per unit throughout the network
  - The low-compute, high-compute alternating pattern could be explained by the network trading off spatial features vs pixel-wise features

SqueezeNAS: Search Time Results

Name	NAS Method	Search Time (GPU Hours)	Dataset Searched on
SqueezeNAS-3	gradient	152	CityScapes
SqueezeNAS-9	gradient	248	CityScapes
SqueezeNAS-23	gradient	316	CityScapes
Neural Architecture Search with Reinforcement Learning	RL	537,600	CIFAR-10
NASNet	RL	48,000	CIFAR-10
mNasNet	RL	48,000*	Proxy ImageNet
AmoebaNet	genetic	75,600	CIFAR-10
FBNet	gradient	216	Proxy ImageNet
DARTS	gradient	96	CIFAR-10

* Approximated from TPUv2 Hours

SqueezeNAS: Implementation details

- 13 candidates per unit for 22 layers (10^{24} possible networks)
- Randomly initialize SuperNetwork and train only convolution weights early on then switch to alternating pattern
- We train in FP16 (half precision) for network and FP32 for the architecture parameters. This also cut GPU memory use in half and allowed us to leverage tensor cores
- We train the SuperNetwork on Cityscapes fine with 768x768 patches with flip, crop and scale augmentations
- After SuperNetwork has converged:
  - Sample candidate networks via the Gumbel-Softmax
  - Run each sampled candidate network on the validation set of Cityscapes fine.
  - Put best of these candidates through the full DeepLabv3+[1] training Regime*
    - ImageNet 90 epochs at 224x224
    - COCO 30 epochs at 768x768
    - CityScapes Coarse (40 epochs) and Fine (100 epochs) with the same regime as the SuperNetwork

[1] Chen et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, ECCV 2018

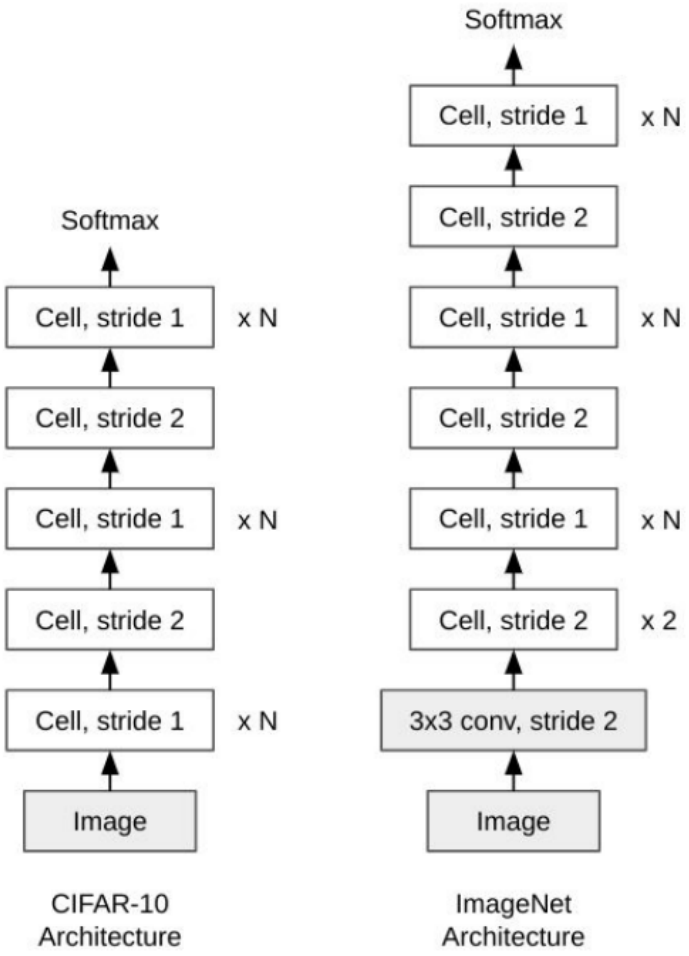
PNASNet: Progressive Neural Architecture Search, ECCV 2018

Taxonomy

- Similar to Zoph et al. (2018)

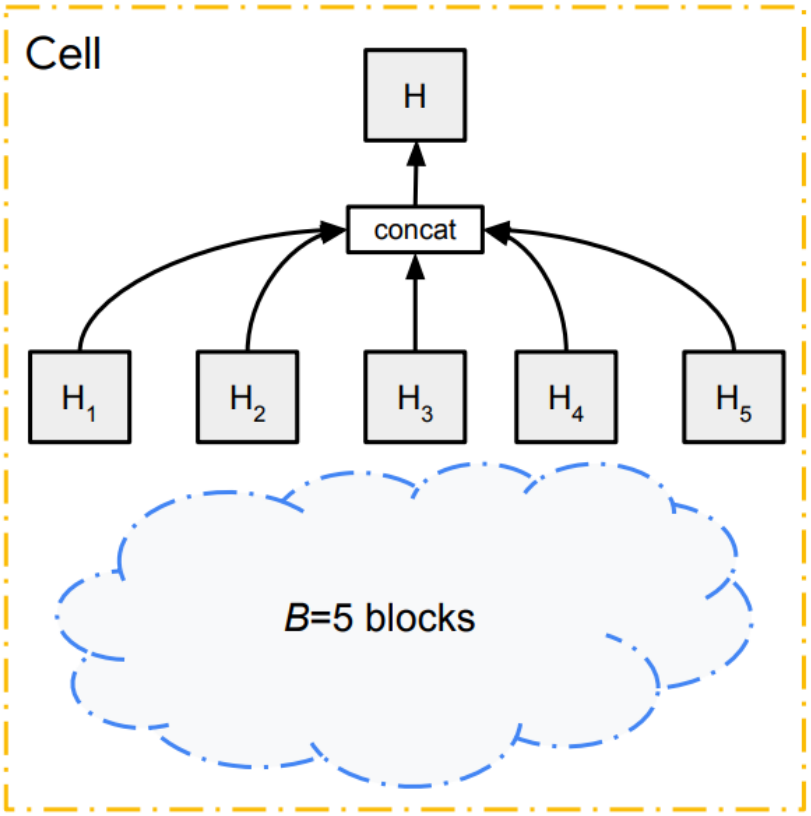
Cell -> Network

- Once we have a cell structure, we stack it up using a predefined pattern
- A network is fully specified with:
  - Cell structure
  - *N* (number of cell repetition)
  - *F* (number of filters in the first cell)
- *N* and *F* are selected by hand to control network complexity



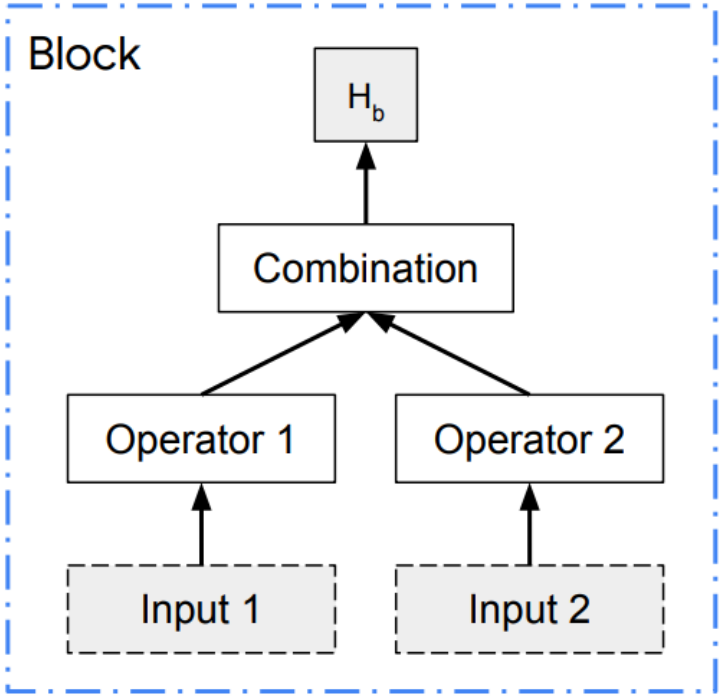
Block -> Cell

- Each cell consists of *B*=5 blocks
- The cell's output is the concatenation of the 5 blocks' outputs



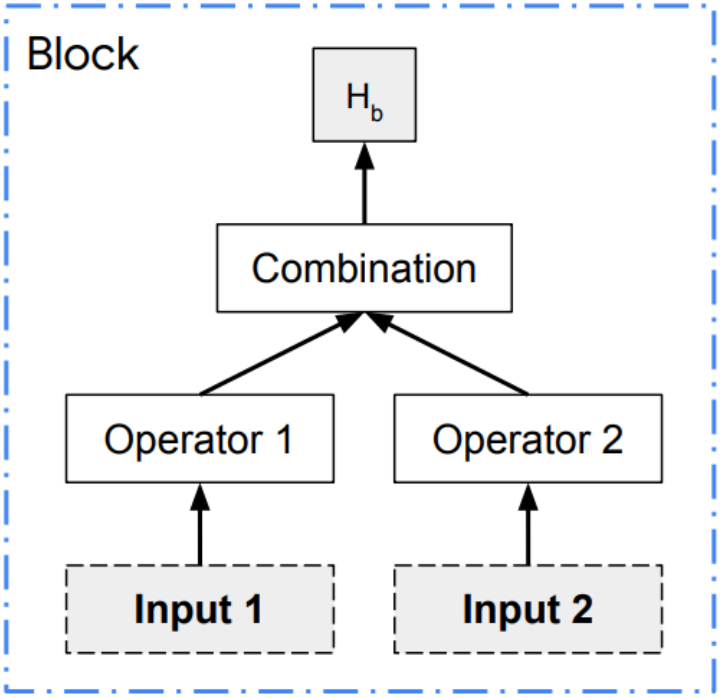
Within a Block

- Input 1 is transformed by Operator 1
- Input 2 is transformed by Operator 2
- Combine to give block's output



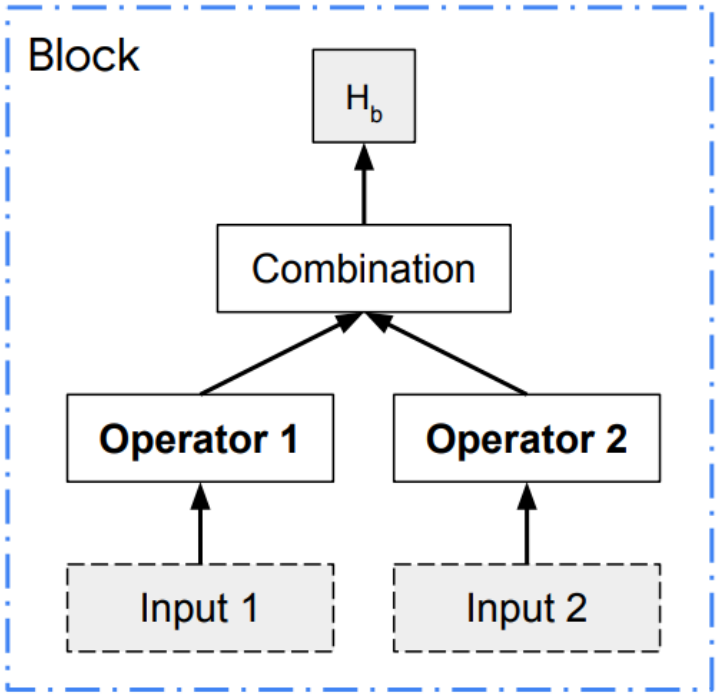
Within a Block

- **Input 1** and **Input 2** may select from:
  - Previous cell's output
  - Previous-previous cell's output
  - Previous blocks' output in current cell



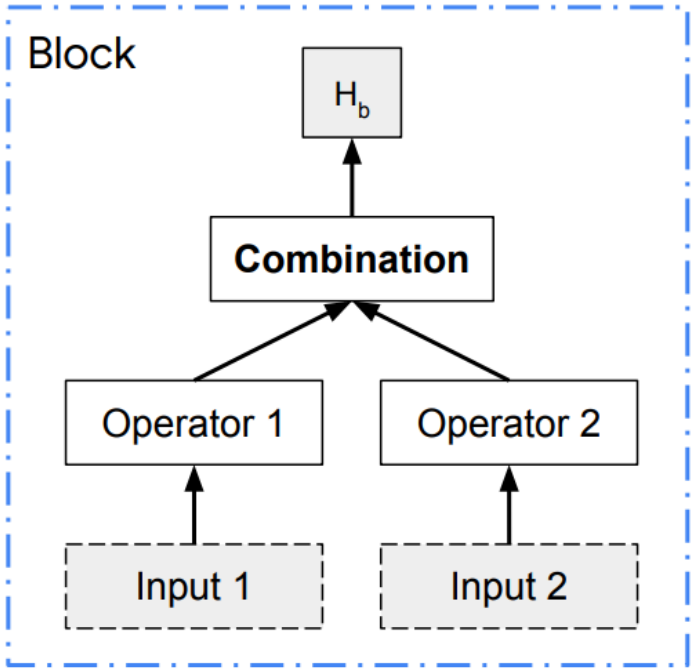
Within a Block

- **Operator 1** and **Operator 2** may select from:
  - 3x3 depth-separable convolution
  - 5x5 depth-separable convolution
  - 7x7 depth-separable convolution
  - 1x7 followed by 7x1 convolution
  - Identity
  - 3x3 average pooling
  - 3x3 max pooling
  - 3x3 dilated convolution



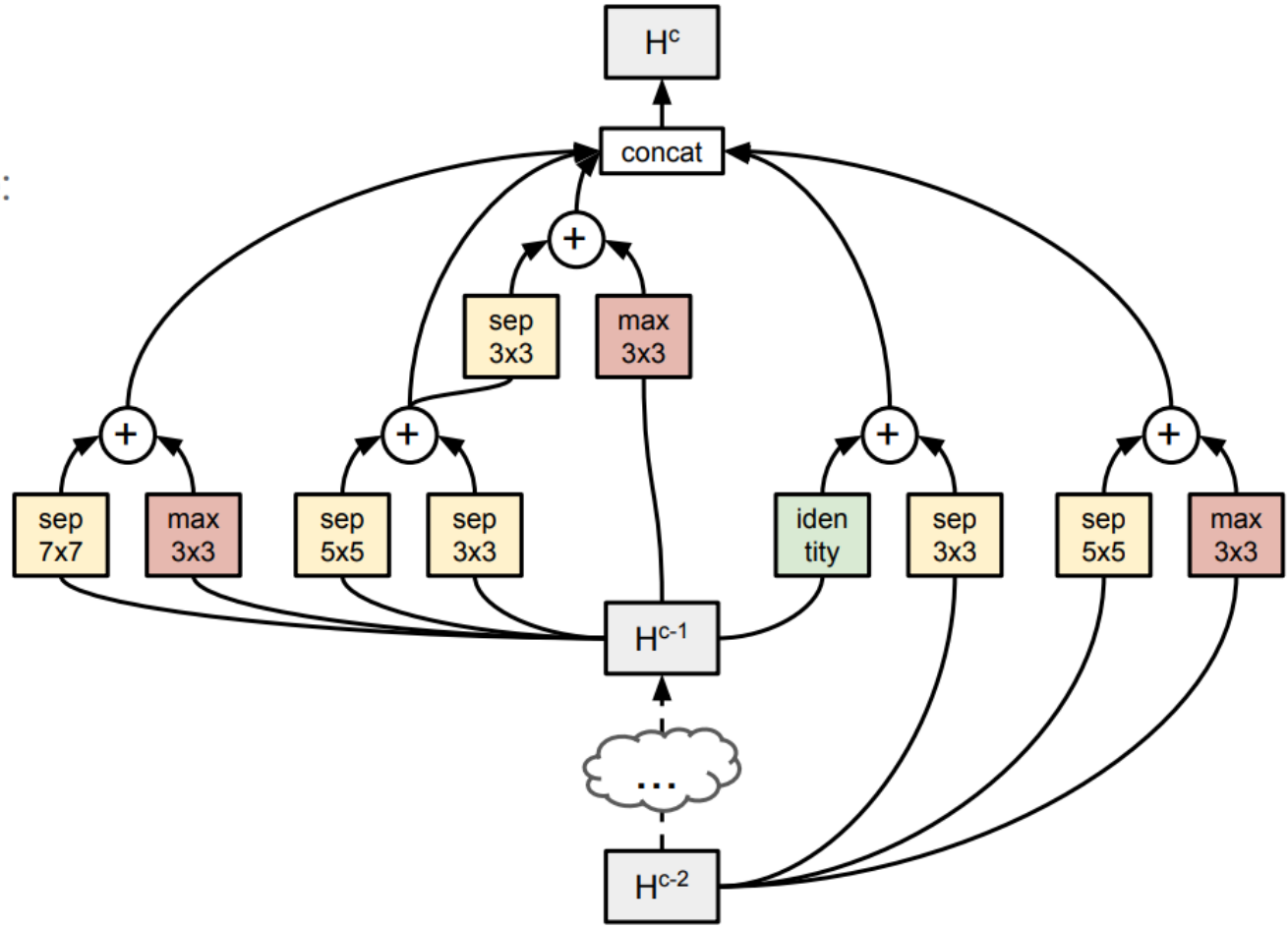
Within a Block

- **Combination** is element-wise addition



Architecture Search Space Summary

- One cell may look like:
- $2^2 * 8^2 * 1 *$
  $3^2 * 8^2 * 1 *$
  $4^2 * 8^2 * 1 *$
  $5^2 * 8^2 * 1 *$
  $6^2 * 8^2 * 1 =$
  10^{14} possible combinations!

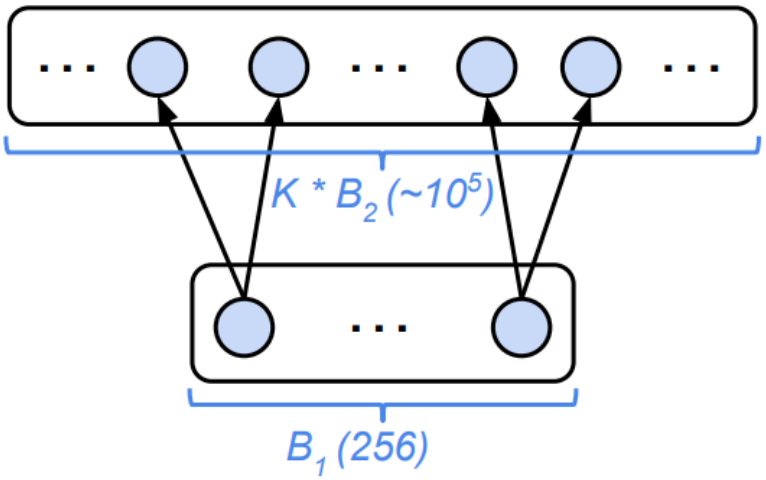


Main Idea: Simple-to-Complex Curriculum

- Previous approaches directly work with the 10^{14} search space
- Instead, what if we progressively work our way in:
  - Begin by training all 1-block cells. There are only 256 of them!
  - Their scores are going to be low, because of they have fewer blocks...
  - But maybe their relative performances are enough to show which cells are promising and which are not.
  - Let the *K* most promising cells expand into 2-block cells, and iterate!

Progressive Neural Architecture Search: First Try

- **Problem:** for a reasonable *K*, too many 2-block candidates to train
  - It is “expensive” to obtain the performance of a cell/string
  - Each one takes hours of training and evaluating
  - Maybe can afford 10^2, but definitely cannot afford 10^5



train these 2-block cells

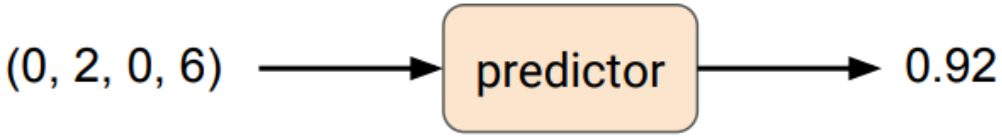
expand promising 2-block cells

enumerate, train, select top *K*

Performance Prediction with Surrogate Model

- **Solution**: train a “cheap” surrogate model that predicts the final performance simply by reading the string
  - The data points collected in the “expensive” way are exactly training data for this “cheap” surrogate model

- The two assessments are in fact used in an alternate fashion:
  - Use “cheap” assessment when candidate pool is large (~10^5)
  - Use “expensive” assessment when it is small (~10^2)

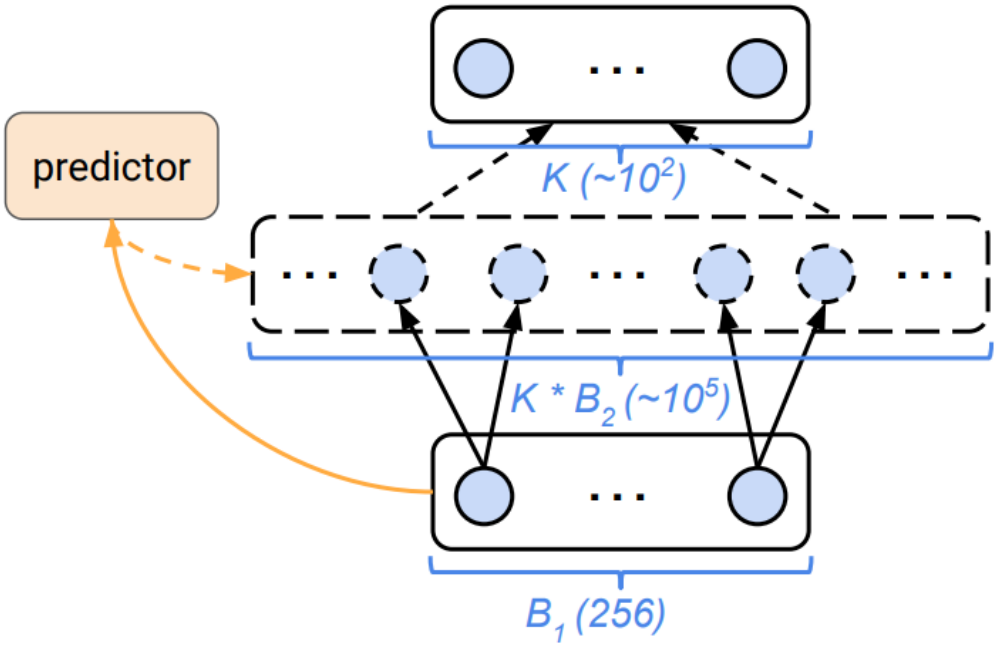


Performance Prediction with Surrogate Model

- Desired properties of this surrogate model/predictor:
  - Handle variable-size input strings
  - Correlate with true performance
  - Sample efficient

- We try both a MLP-ensemble and a RNN-ensemble as predictor
  - MLP-ensemble handles variable-size by mean pooling
  - RNN-ensemble handles variable-size by unrolling a different number of times

Progressive Neural Architecture Search



train the selected 2-block cells

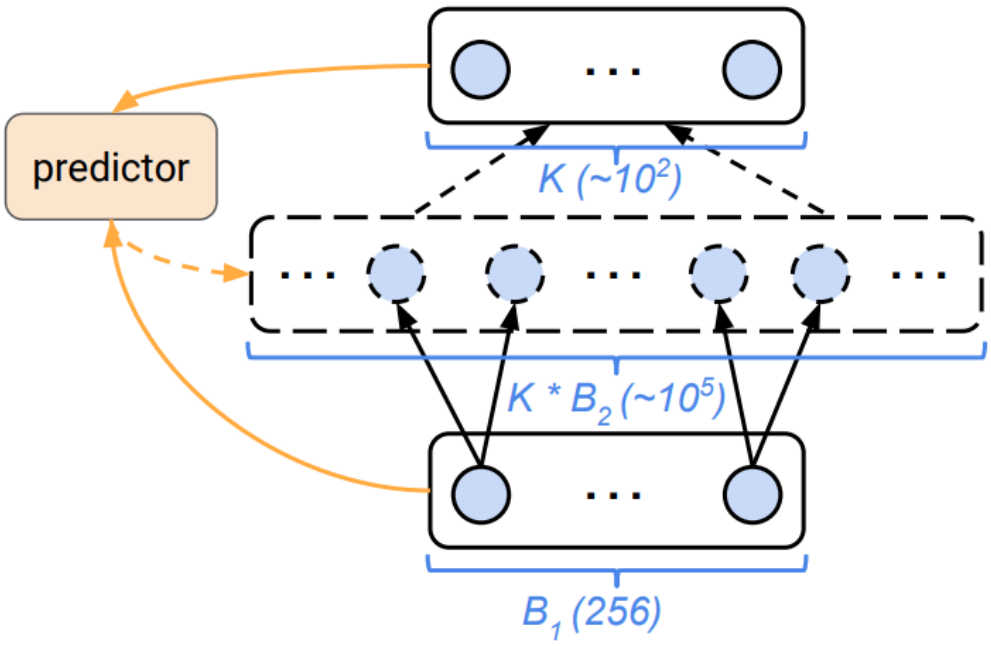
apply predictor to select top *K*

expand promising 2-block cells

train predictor

enumerate and train all 1-block cells

Progressive Neural Architecture Search



finetune predictor

train the selected 2-block cells

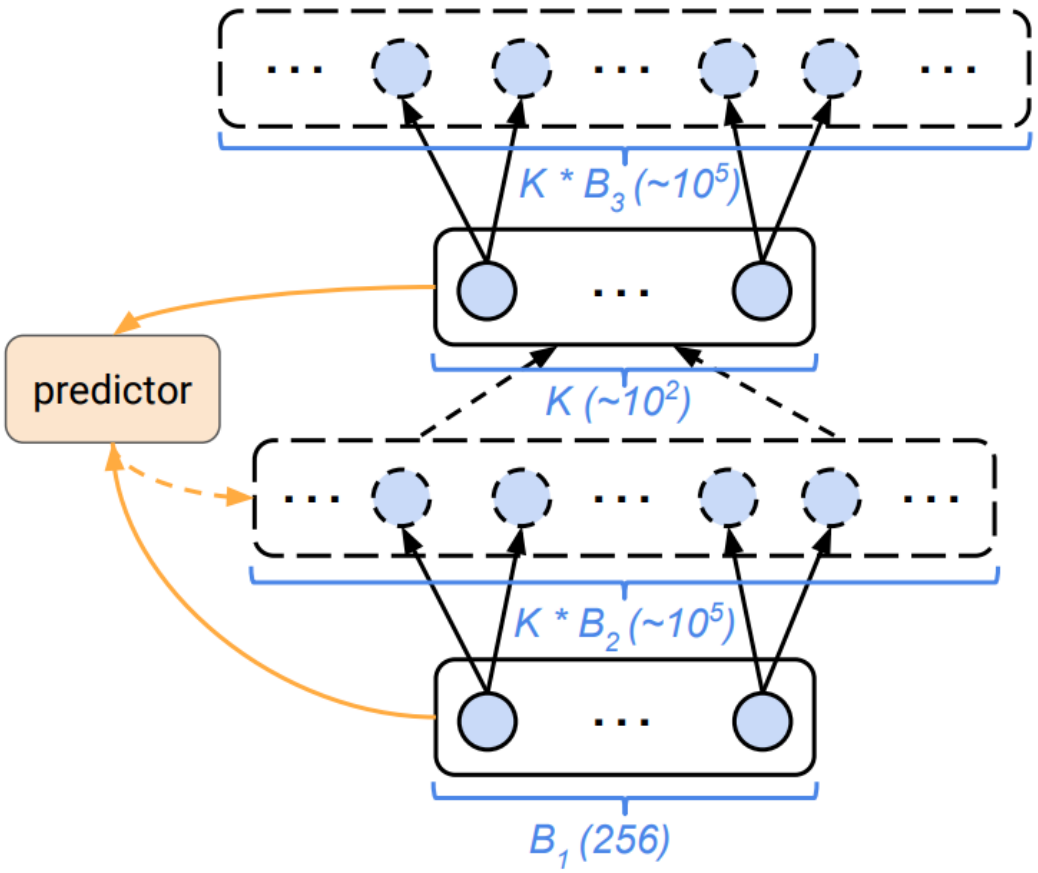
apply predictor to select top *K*

expand promising 2-block cells

train predictor

enumerate and train all 1-block cells

Progressive Neural Architecture Search



expand promising 3-block cells

finetune predictor

train the selected 2-block cells

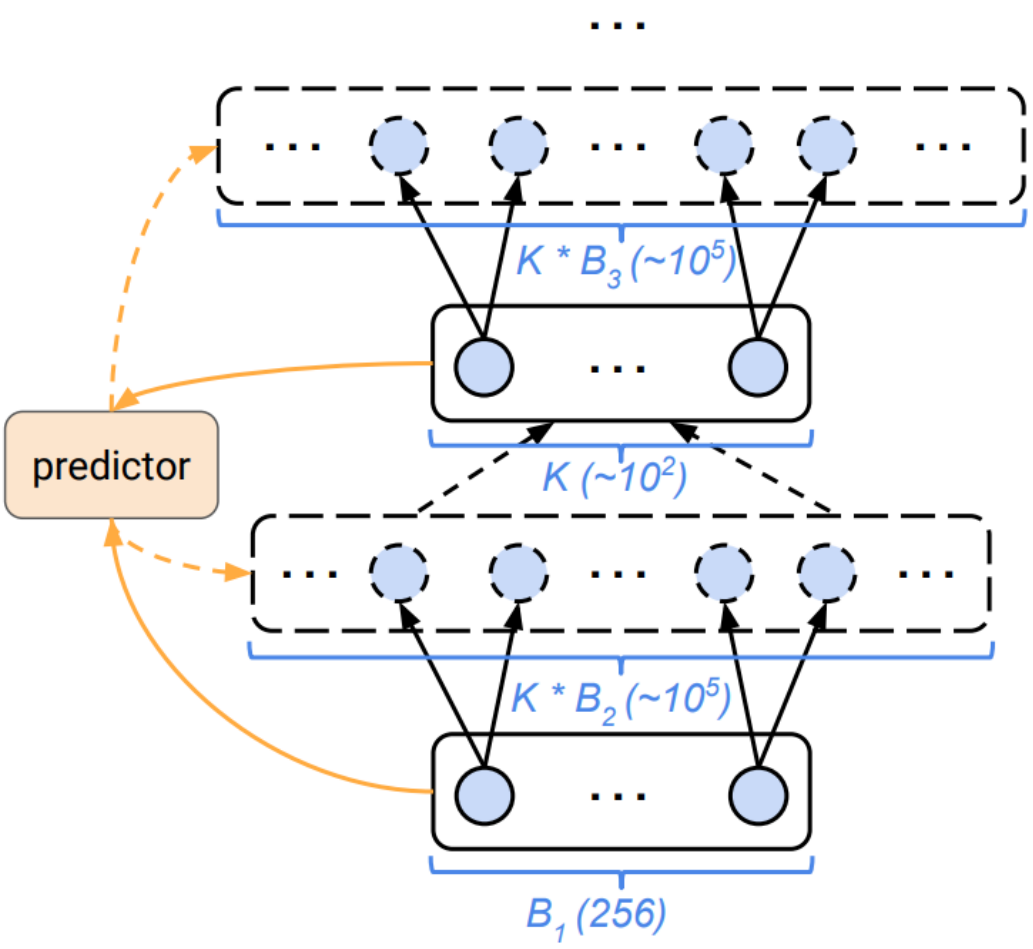
apply predictor to select top *K*

expand promising 2-block cells

train predictor

enumerate and train all 1-block cells

Progressive Neural Architecture Search



apply predictor to select top *K*

expand promising 3-block cells

finetune predictor

train the selected 2-block cells

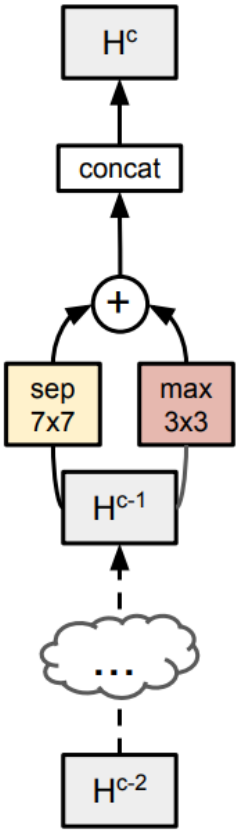
apply predictor to select top *K*

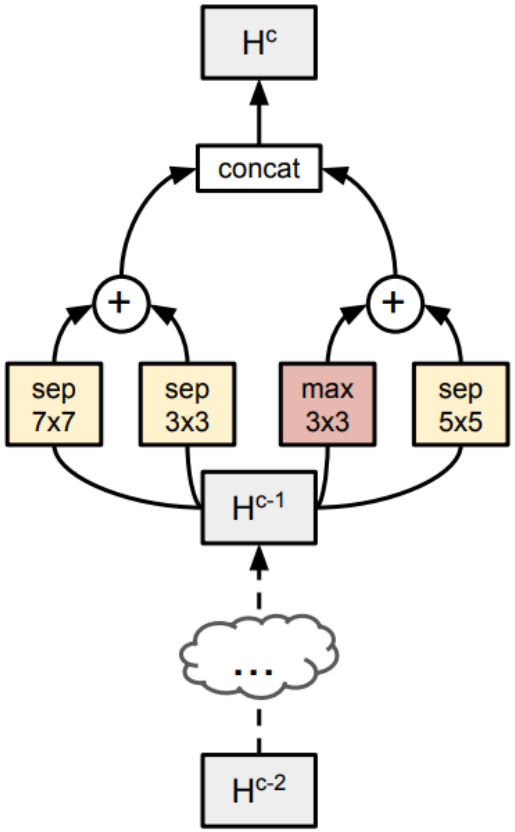
expand promising 2-block cells

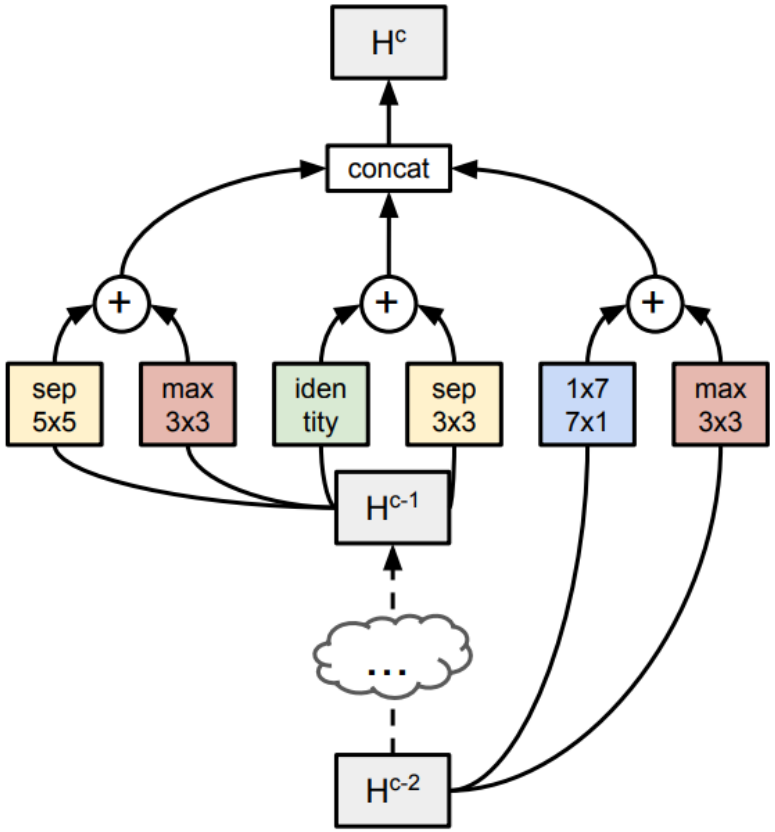
train predictor

enumerate and train all 1-block cells

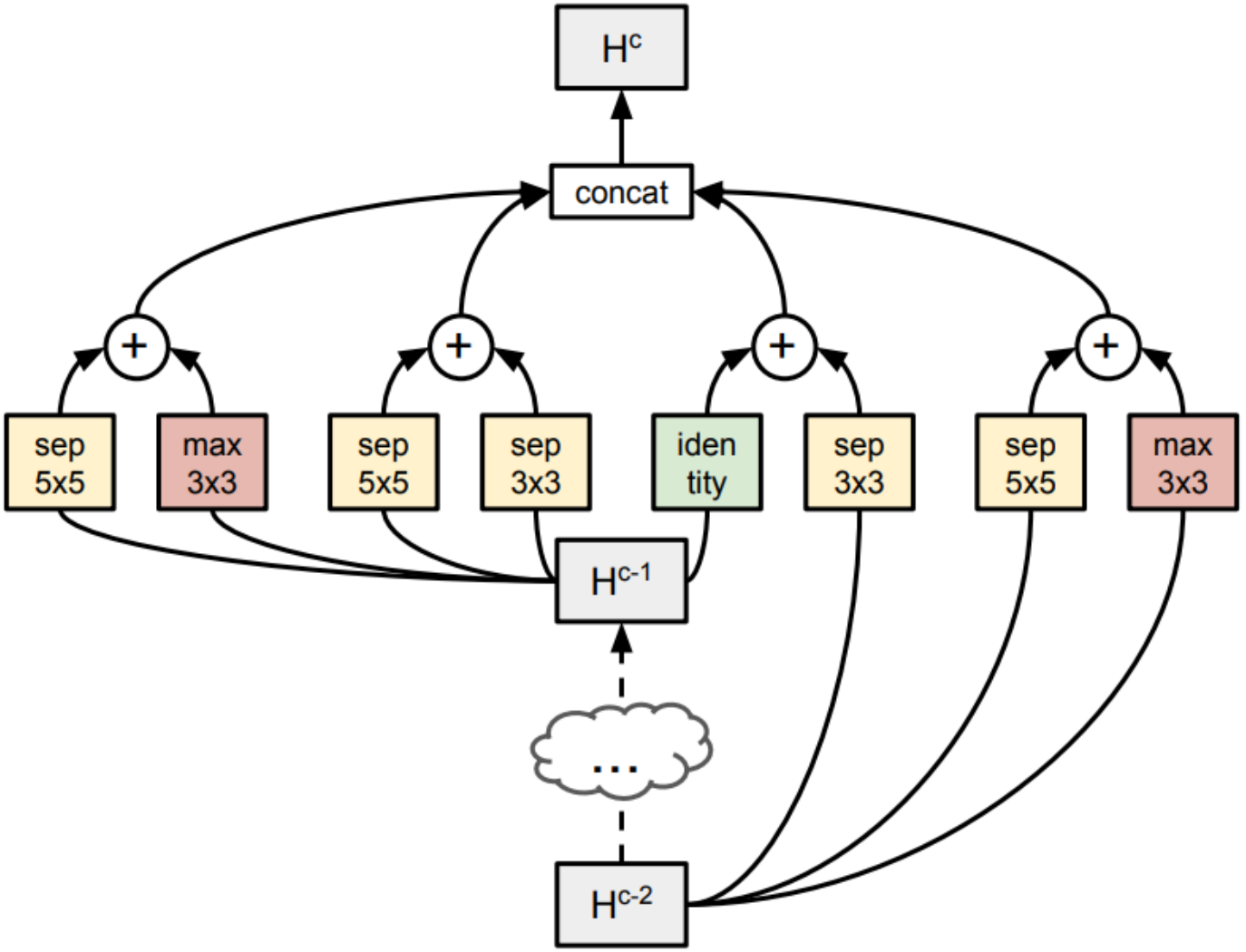
The Search Process: PNASNet-1, 2, 3



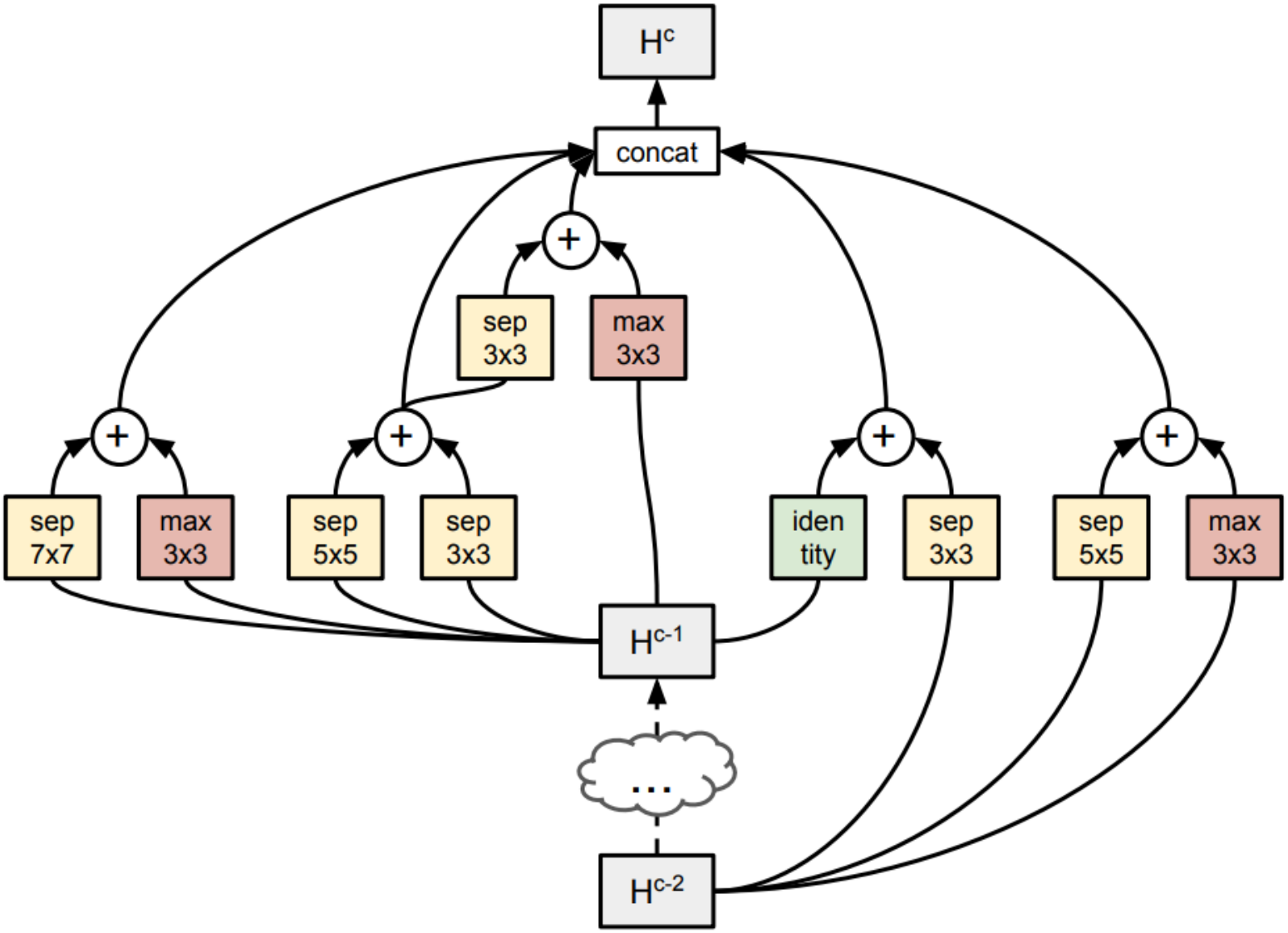




The Search Process: PNASNet-4



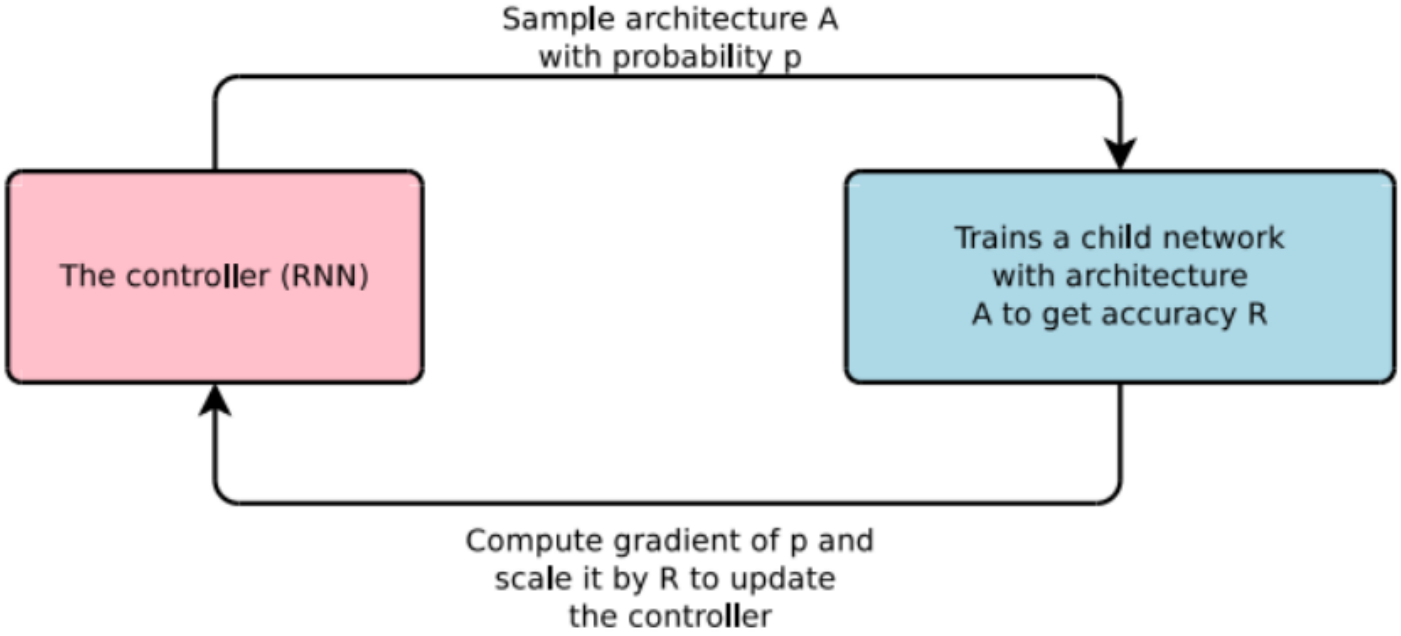
The Search Process: PNASNet-5



ENAS — Efficient Neural Architecture Search

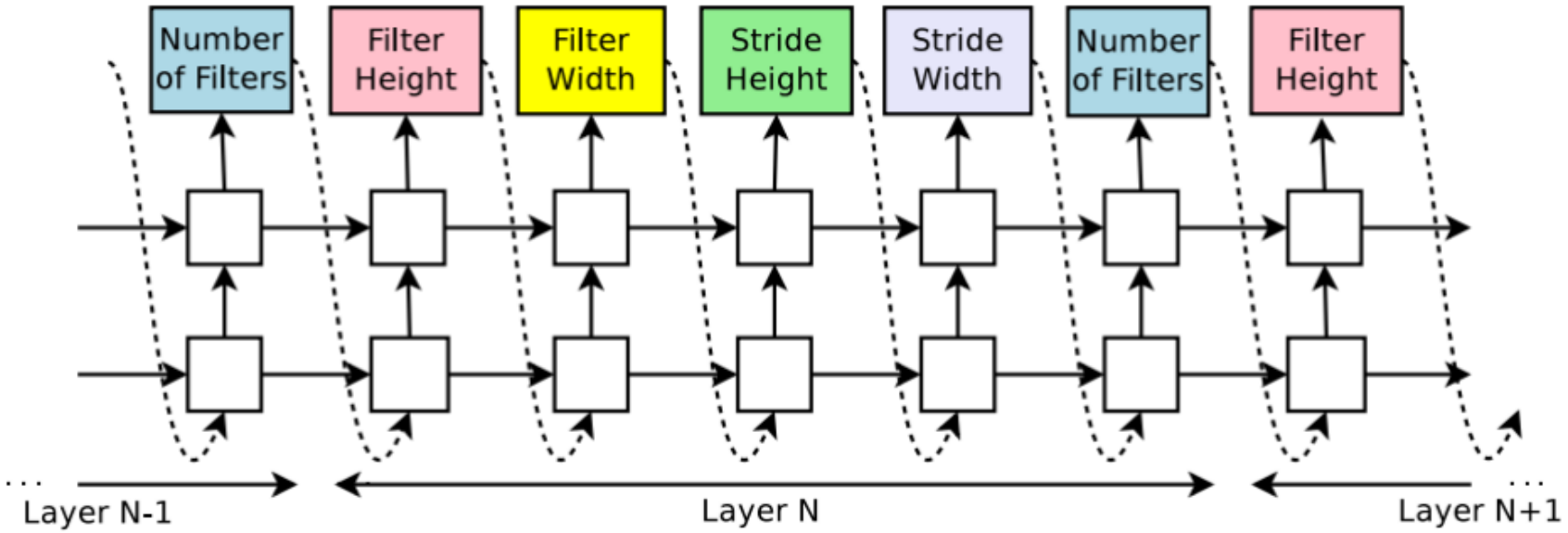
Neural Architecture Search

- B. Zoph and Q. V. Le., "Neural architecture search with reinforcement learning", ICLR-2017



Neural Architecture Search

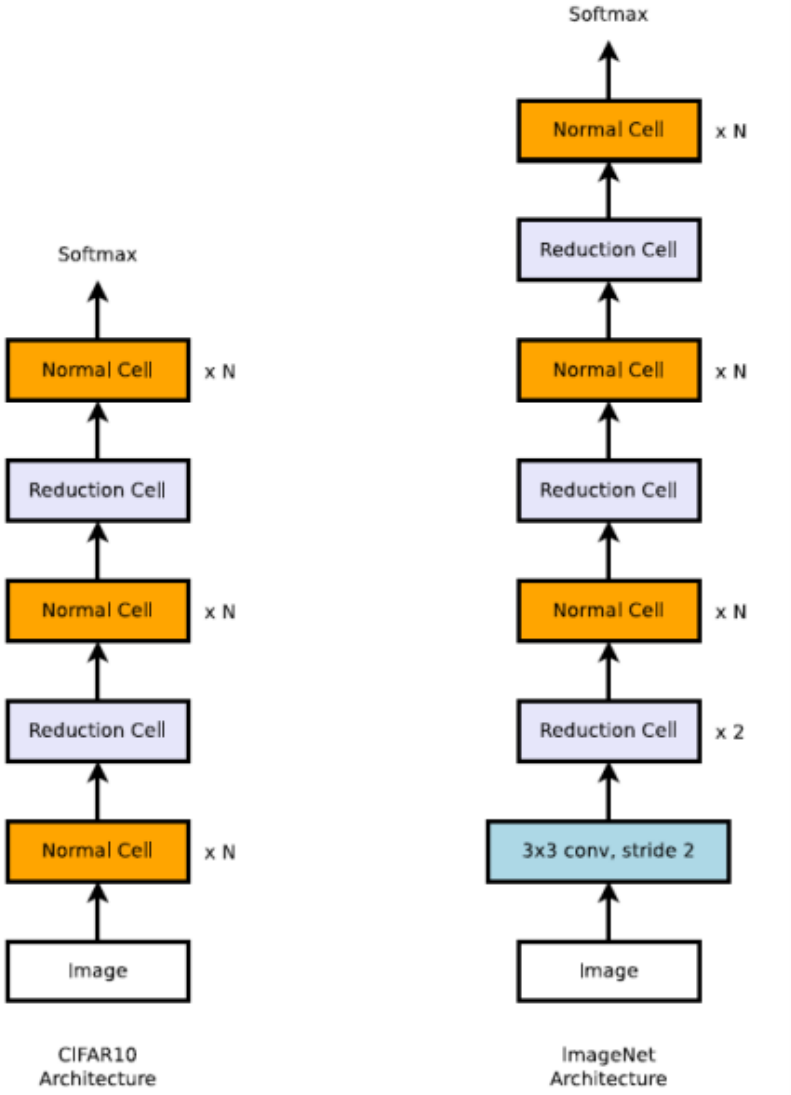
- How controller RNN samples a simple convolutional network



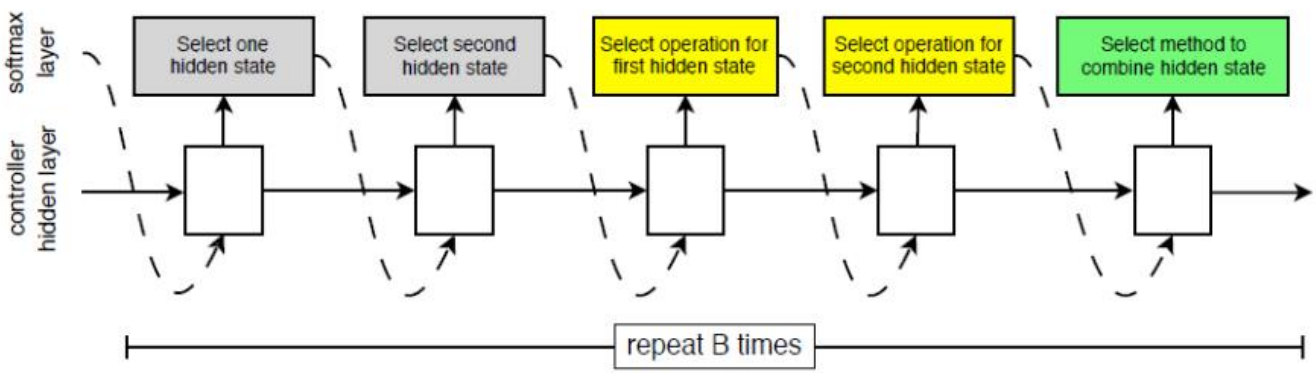
Method

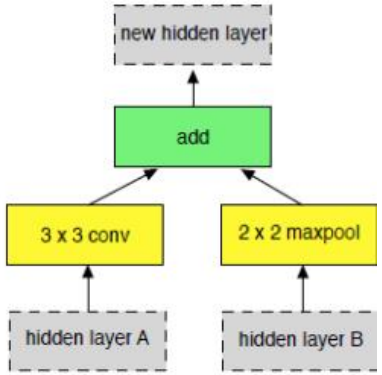
- Overall architectures of the convolutional nets are manually predetermined
  - Normal Cell – convolutional cells that return a feature map of the same dimension
  - Reduction Cell – convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two
- Using common heuristic to double the number of filters in the output whenever the spatial activation size is reduced

Scalable Architectures for Image Classification



Models and Algorithms





Step 1. Select a hidden state from h_i, h_{i-1} or from the set of hidden states created in previous blocks.

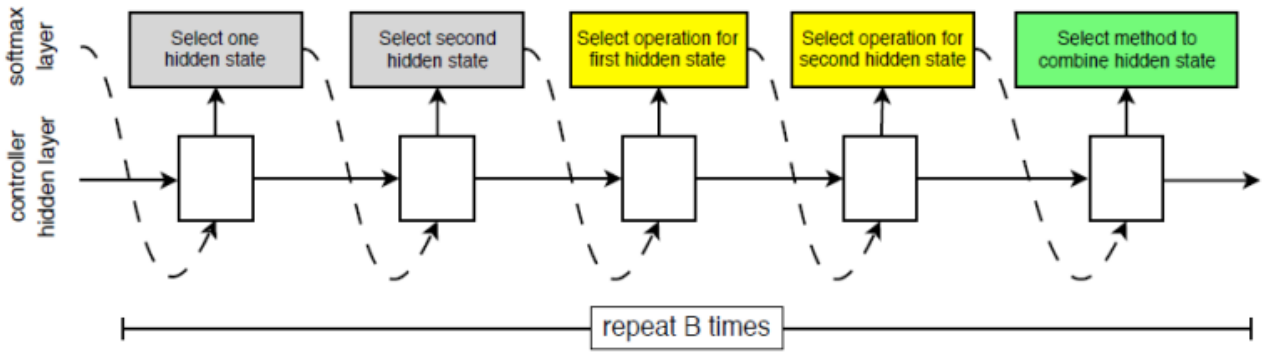
Step 2. Select a second hidden state from the same options as in Step 1.

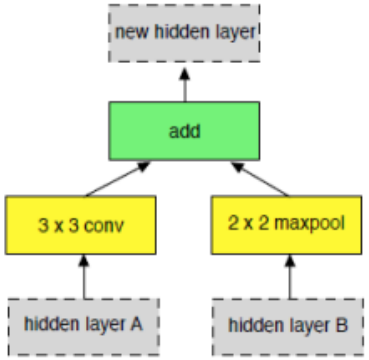
Step 3. Select an operation to apply to the hidden state selected in Step 1.

Step 4. Select an operation to apply to the hidden state selected in Step 2.

Step 5. Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.

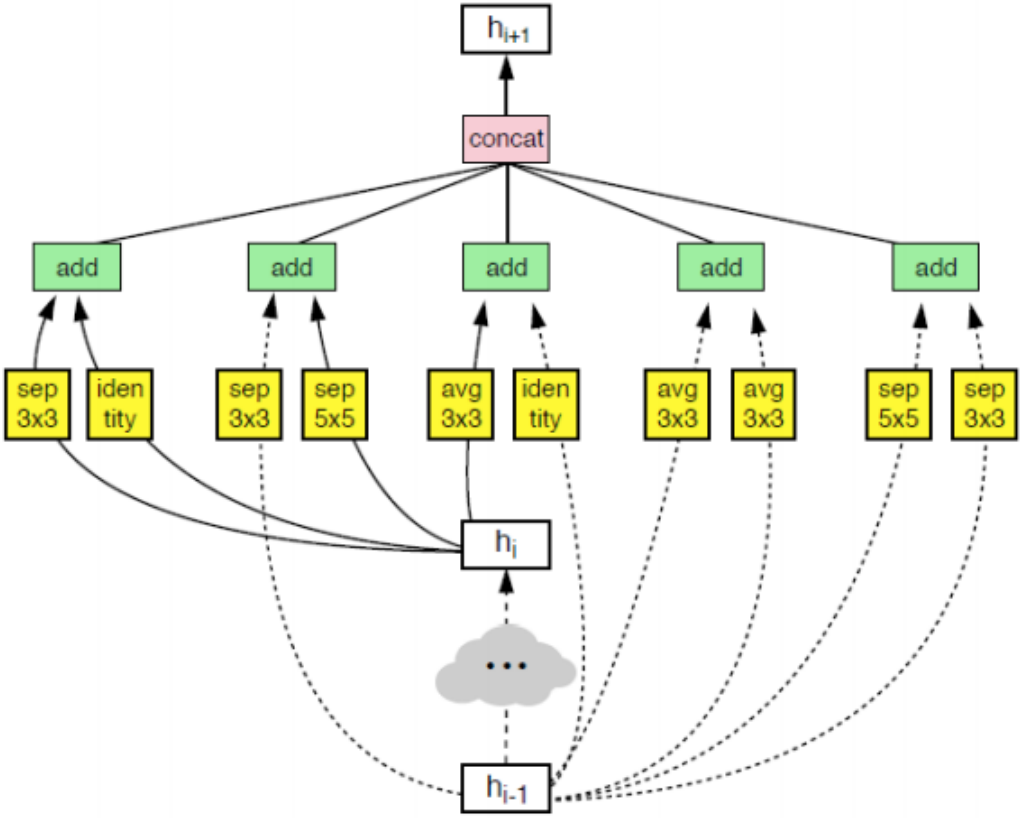
Search Space in a Cell (Step 3 and 4)



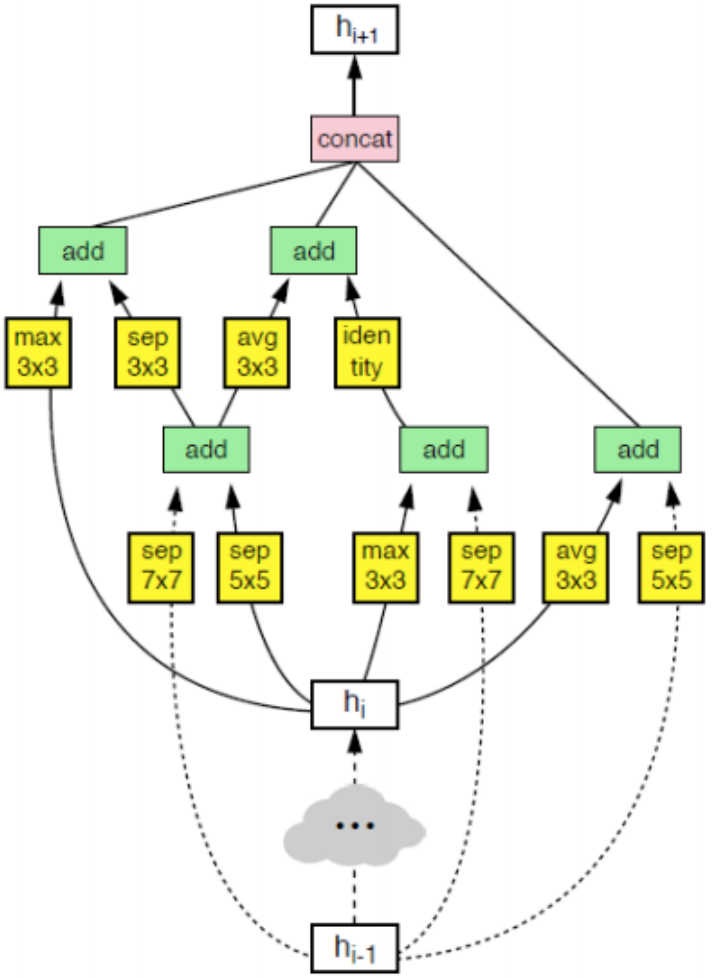


- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

Best Architecture (NASNet-A)

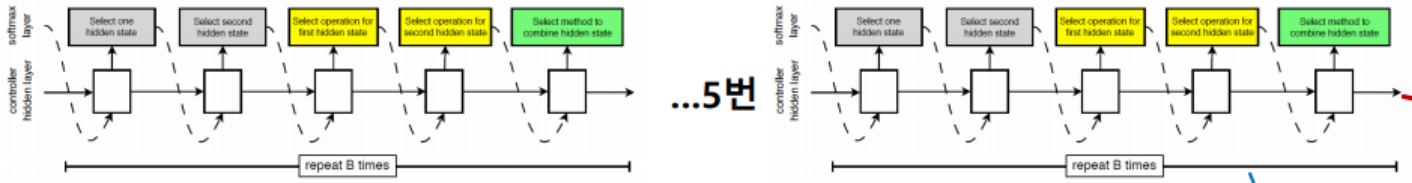


Normal Cell



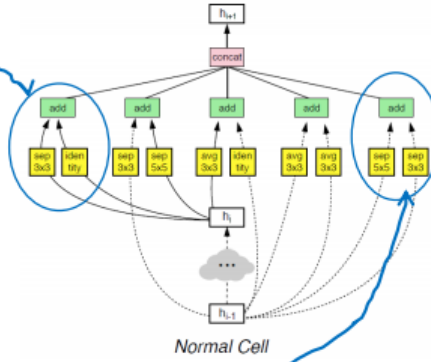
Reduction Cell

Best Architecture (NASNet-A)

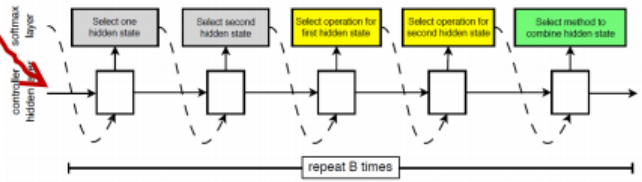


...5번

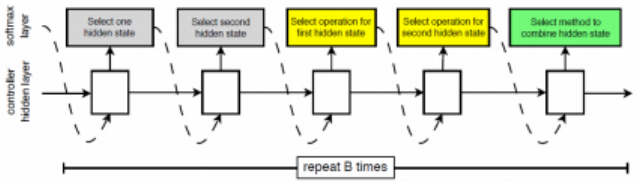
<Normal Cell>



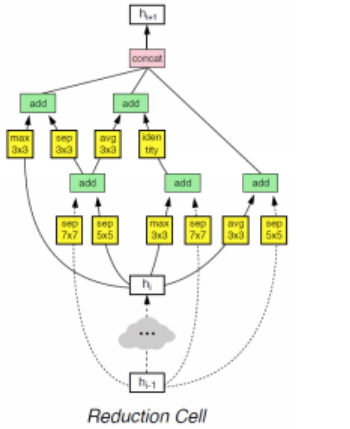
Normal Cell



...5번

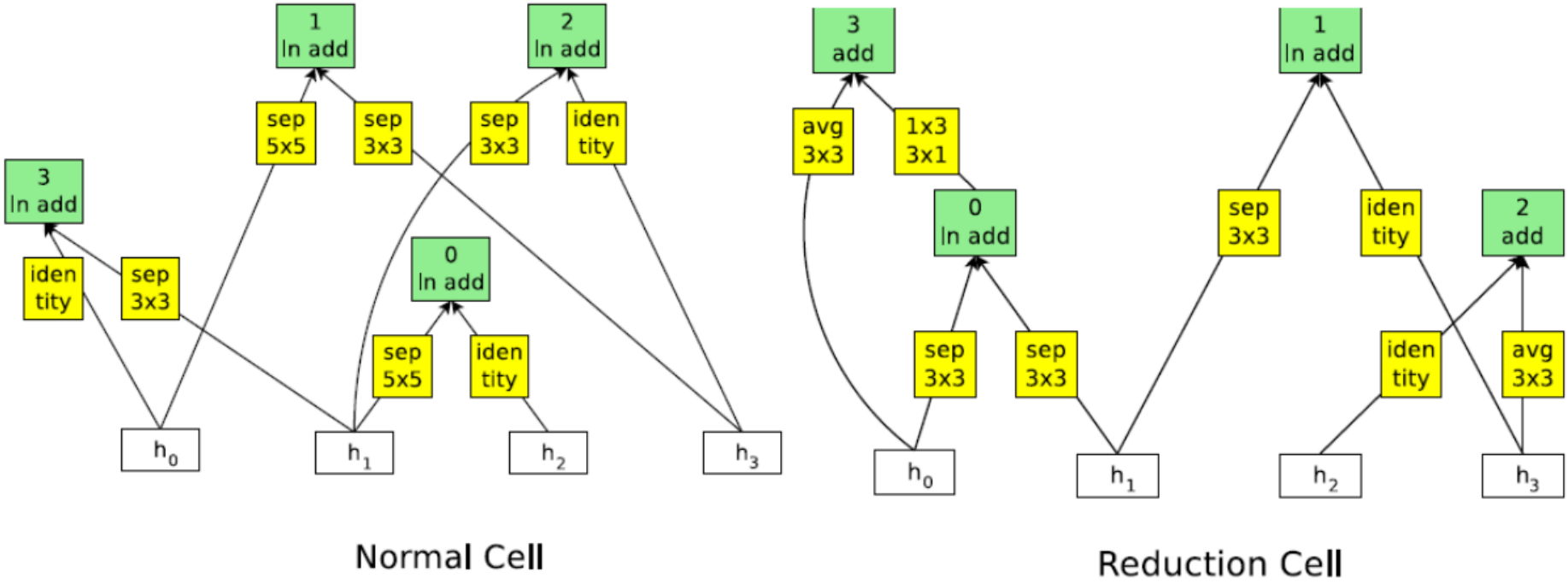


<Reduction Cell>

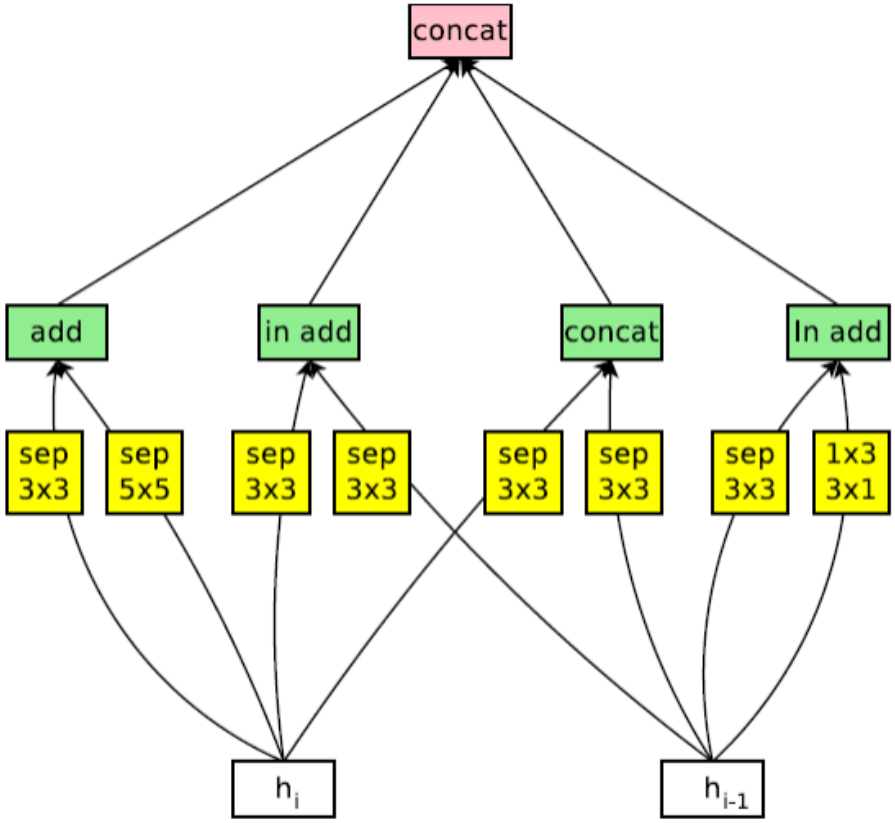


Reduction Cell

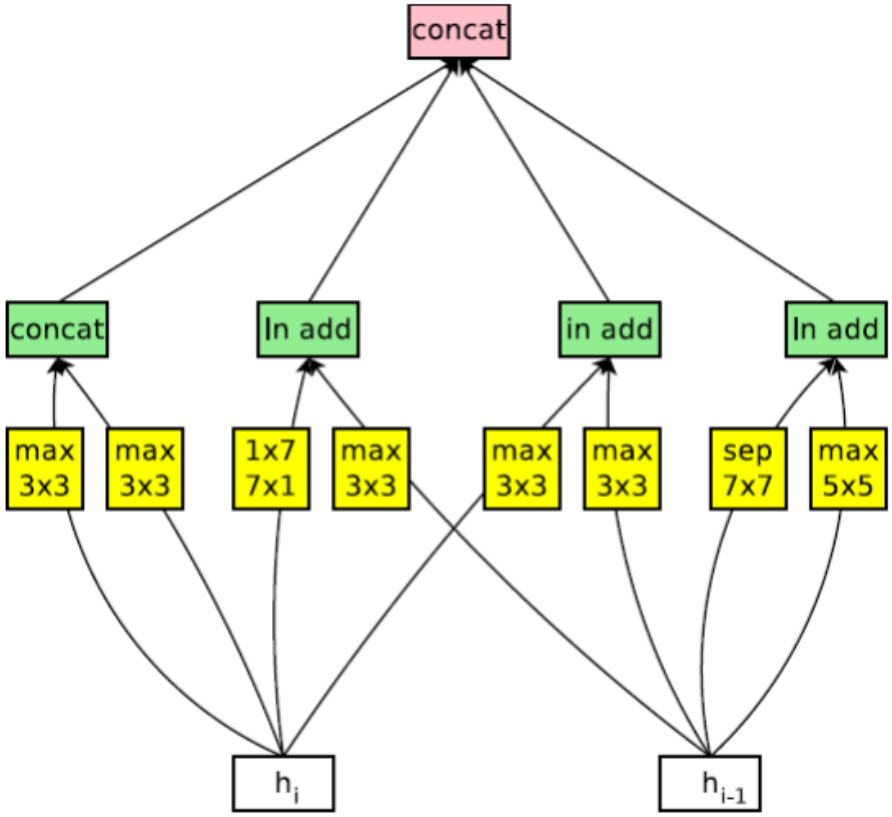
NASNet-B


1
ln add
2
ln add
sep
5x5
sep
3x3
sep
3x3
iden
tity
3
ln add
iden
tity
sep
3x3
0
ln add
sep
5x5
iden
tity
h0
h1
h2
h3
Normal Cell
3
add
avg
3x3
1x3
3x1
0
ln add
sep
3x3
sep
3x3
1
ln add
sep
3x3
iden
tity
2
add
iden
tity
avg
3x3
h0
h1
h2
h3
Reduction Cell

NASNet-C



Normal Cell



Reduction Cell

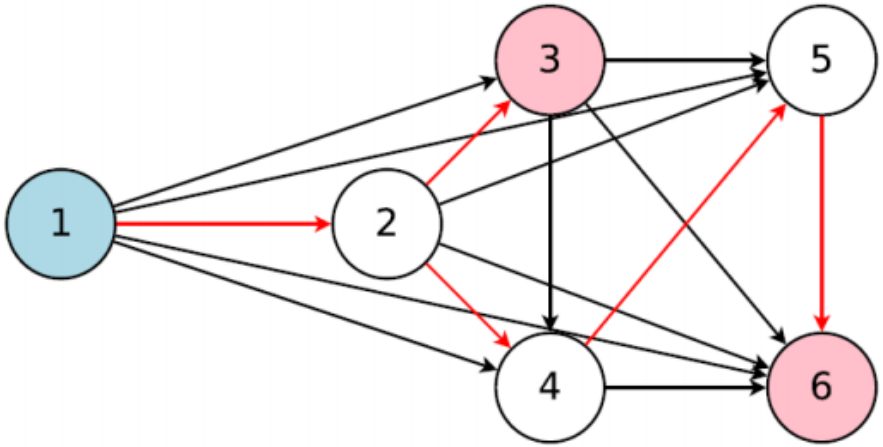
Main Idea

Forcing all child models to share weights

to eschew training each child model from scratch to convergence

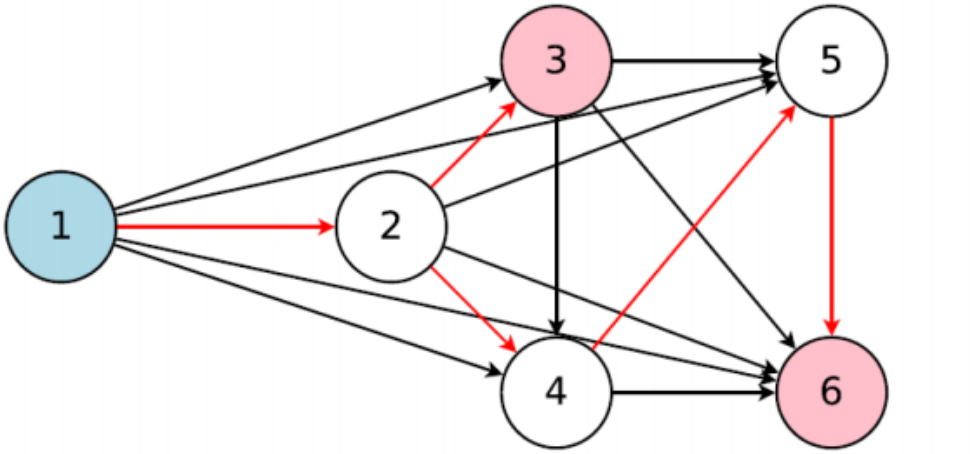
→ Using Single GTX 1080Ti GPU, the search for architectures takes less than 16 hours (compared to NAS, reduction is more than 1000x)

Directed Acyclic Graph(DAG)



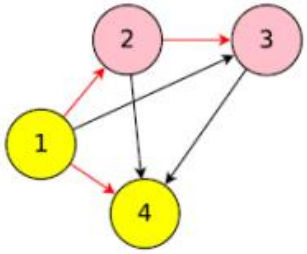
- ENAS's DAG is the superposition of all possible child models in a search space of NAS, where the nodes represent the local computations and the edges represent the flow of information.
- The local computations at each node have their own parameters, which are used only when the particular computation is activated.
- Therefore, ENAS's design allows parameters to be shared among all child models

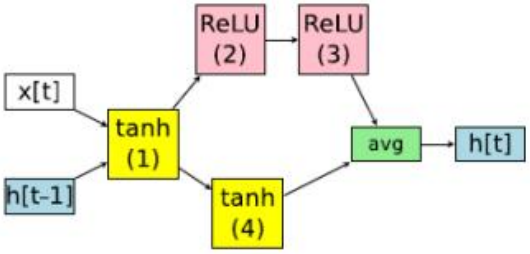
Directed Acyclic Graph(DAG)

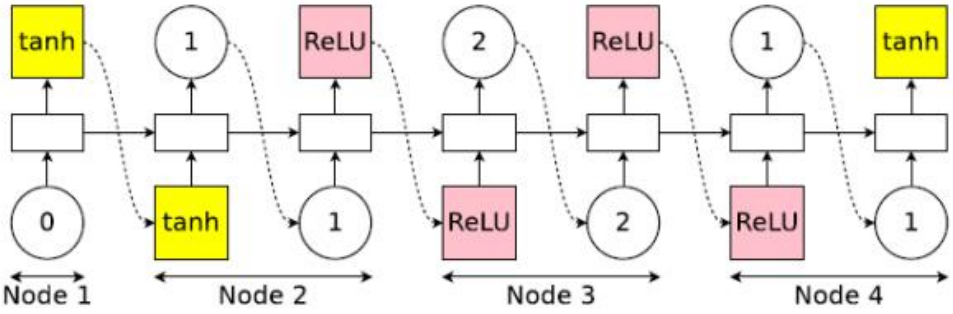


- The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller.
- Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

Designing Recurrent Cells







1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh(\mathbf{x}_t \cdot \mathbf{W}^{(x)} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(h)})$.

2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(h)})$.

3. At node 3: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 2 and the activation function ReLU. Therefore, $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(h)})$.

4. At node 4: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function tanh, leading to $h_4 = \tanh(h_1 \cdot \mathbf{W}_{4,1}^{(h)})$.

5. For the output, we simply average all the loose ends, *i.e.* the nodes that are not selected as inputs to any other nodes. In our example, since the indices 3 and 4 were never sampled to be the input for any node, the recurrent cell uses their average $(h_3 + h_4)/2$ as its output. In other words, $\mathbf{h}_t = (h_3 + h_4)/2$.

Search Space for Recurrent Cells

- 4 activation functions are allowed
  - tanh, ReLU, identity, sigmoid
- If the recurrent cell has N nodes,
  - The search space has $4^N \times N!$ configuration
  - When N = 12, there are approximately 10^{15} models in the search space

Training ENAS

- The controller network is an LSTM with 100 hidden units
- This LSTM samples decisions via softmax classifier, in an autoregressive fashion
- In ENAS, there are two sets of learnable parameters
  - The parameters of the controller LSTM, denoted by θ
  - The shared parameters of child models, denoted by ω
- The first phase trains ω,

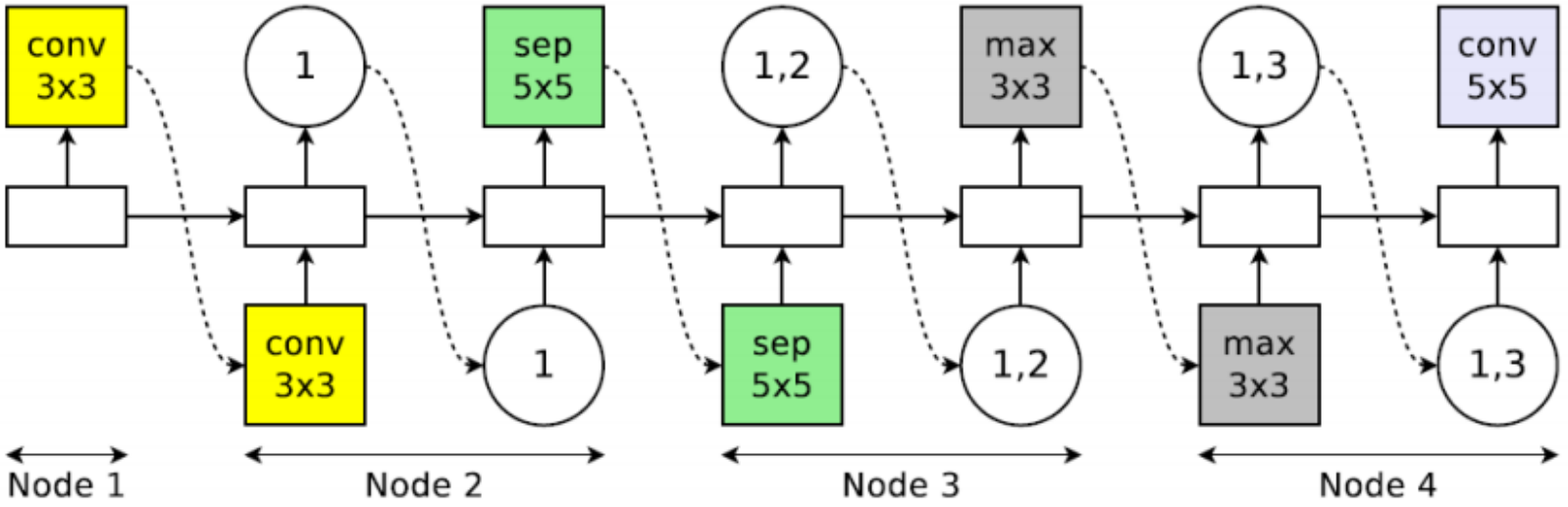
$$\nabla_\omega \mathbb{E}_{\mathbf{m}\sim\pi(\mathbf{m};\theta)}\left[\mathcal{L}(\mathbf{m};\omega)\right] \approx \frac{1}{M}\sum_{i=1}^{M}\nabla_\omega \mathcal{L}(\mathbf{m}_i,\omega),$$

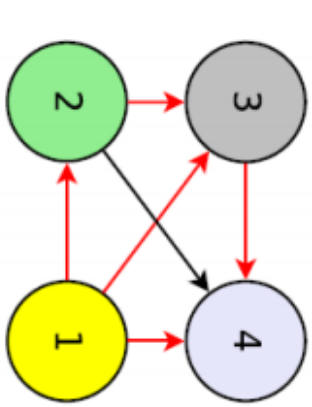
- The second phase trains θ

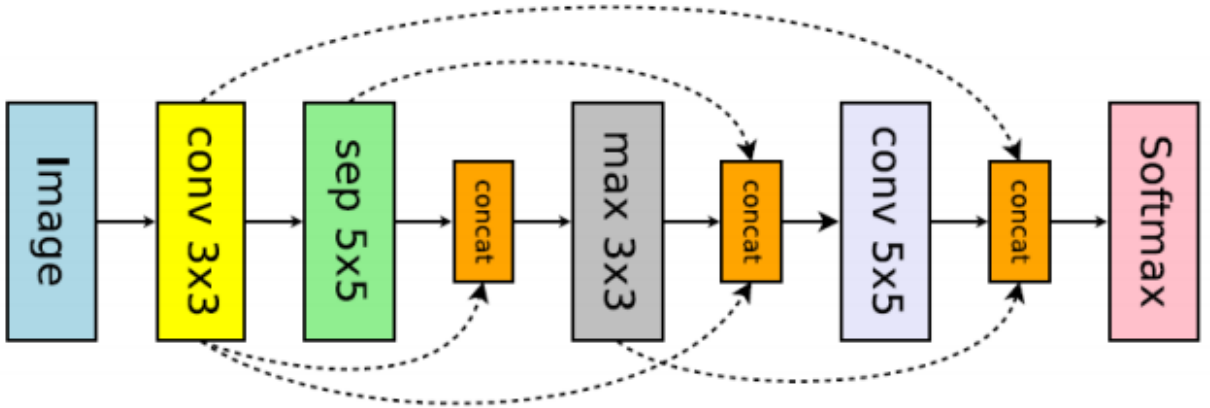
Deriving Architectures

- Sampling several models from the trained policy $\pi(m, \theta)$
- Computing its reward on a single minibatch sampled from the validation set
- The model with the highest reward is taken and retrained from scratch
- training all the sampled models from scratch and selecting the model with the highest performance is possible but it is not economical

Designing CNN



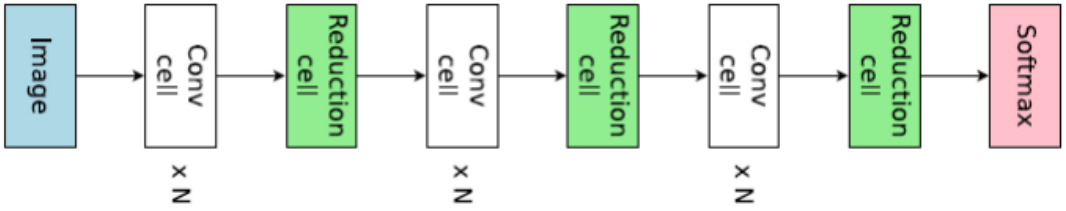


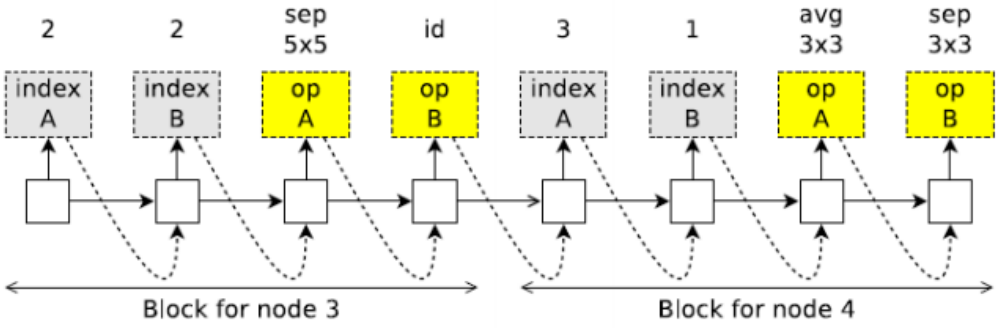


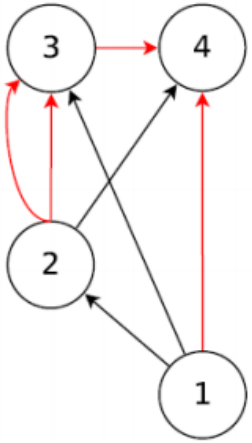
Search Spaces for CNN

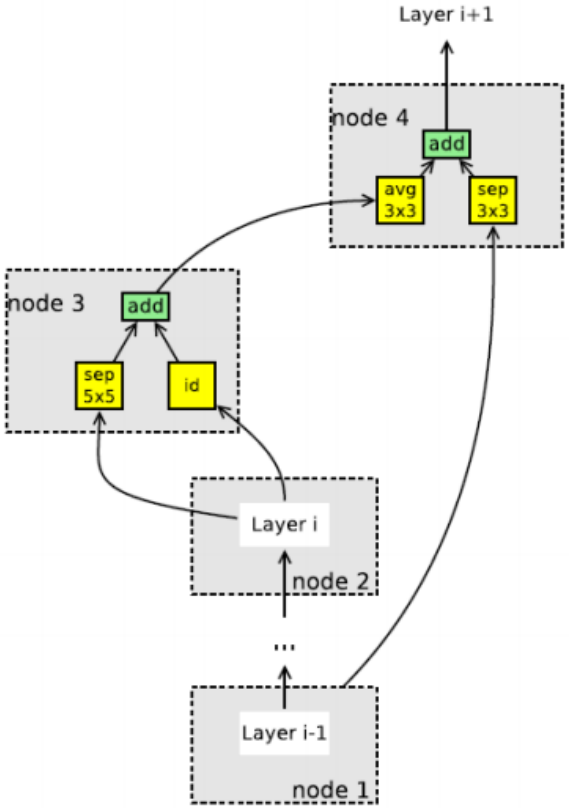
- The 6 operations available for controller are
  - Convolution with filter sizes 3x3 and 5x5
  - Depthwise-separable convolutions with filter sizes 3x3 and 5x5
  - Max pooling and average pooling of kernel size 3x3
- Making the described set of decisions for a total of *L* times,we can sample a network of *L* layers.
- Since all decisions are independent, there are $6^L \times 2^{L(L-1)/2}$
- When L=12, resulting in 1.6×10^{29} possible networks

Designing Convolutional Cells





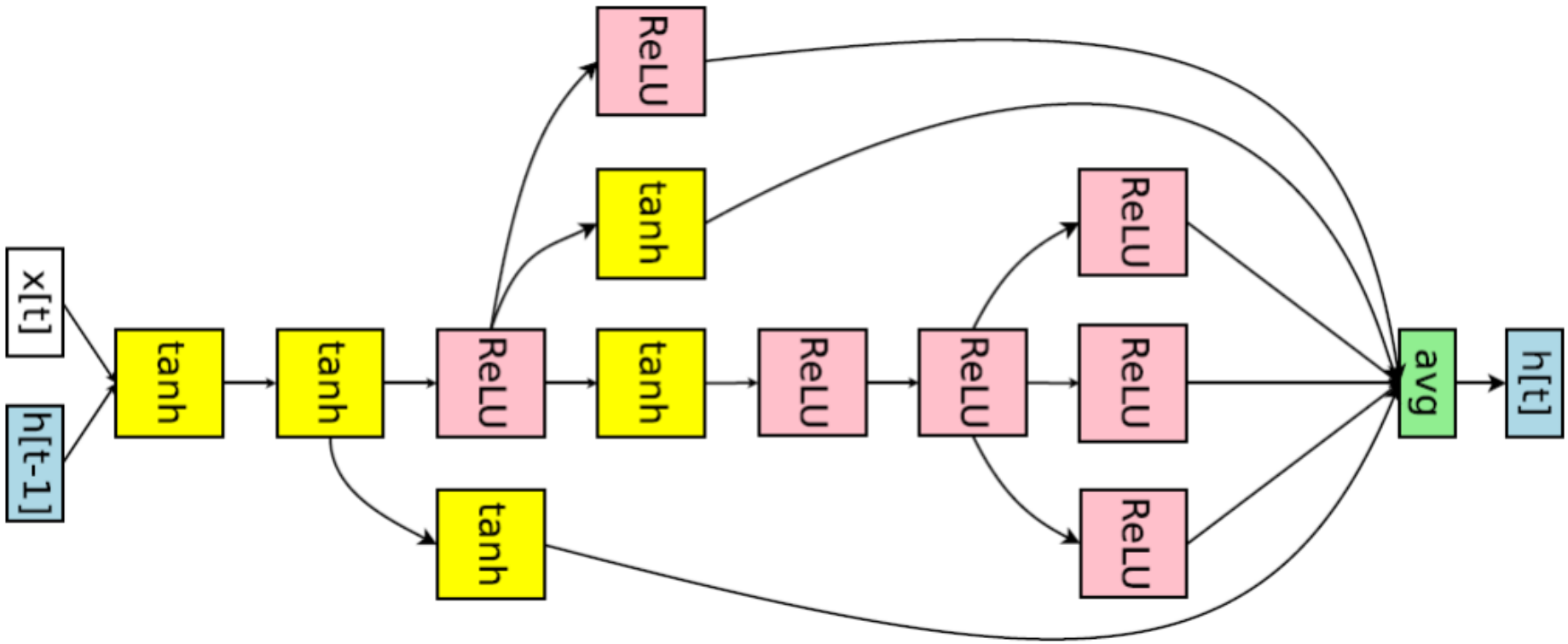




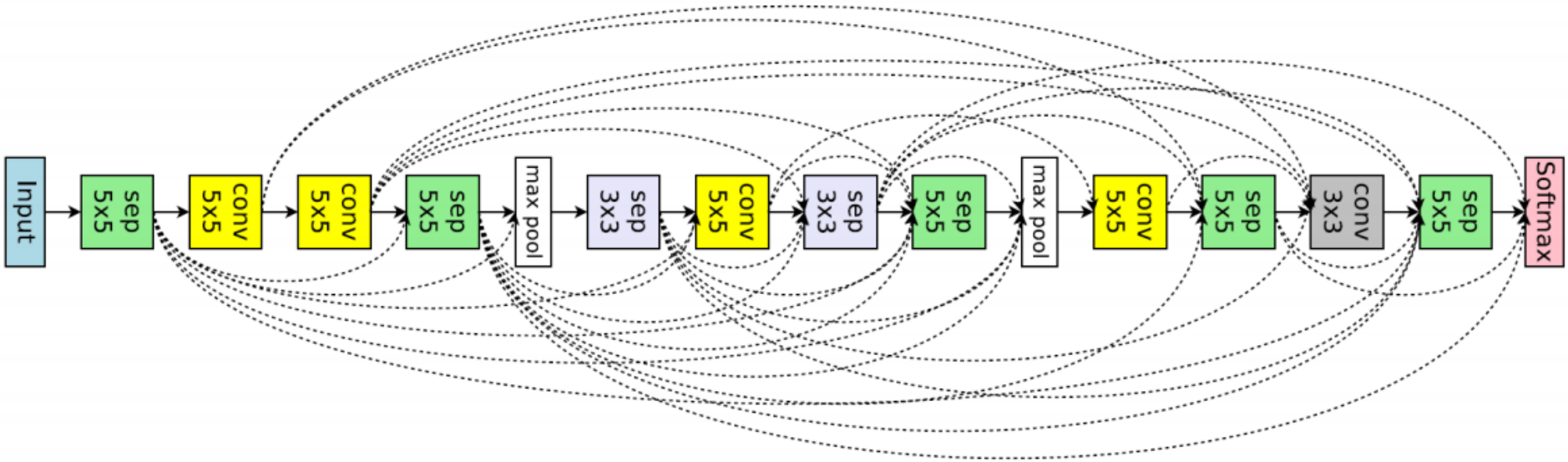
Search Spaces for Convolutional Cells

- The 5 available operations are
  - Identity
  - Separable convolution with kernel size 3x3 and 5x5
  - Average pooling and max pooling with kernel size 3x3
- If there are B nodes, $(5 \times (B-2)!)^4$ cells are possible
- With B = 7, the search space can realize 1.3×10^{11} final networks, making it significantly smaller than the search space for entire convolutional networks

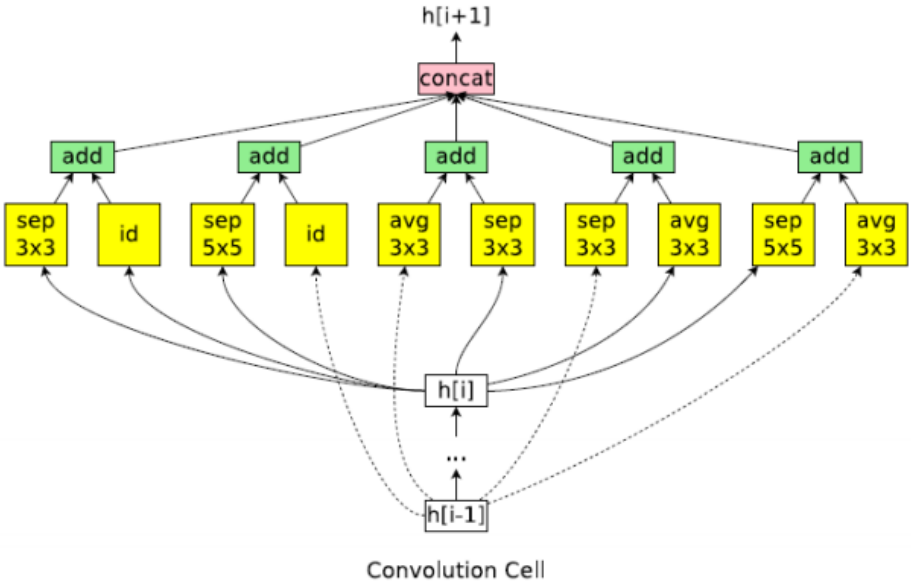
Network Architecture for Penn Treebank



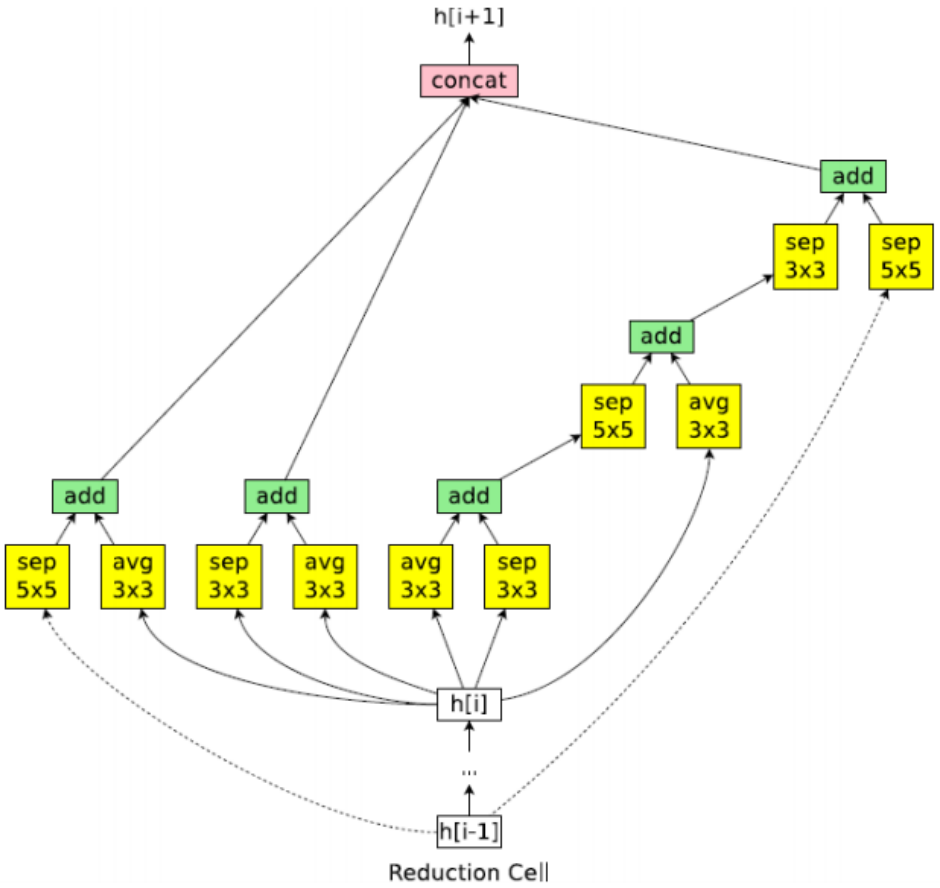
Network Architecture for CIFAR-10(from macro search space)



Network Architecture for CIFAR-10(from micro search space)

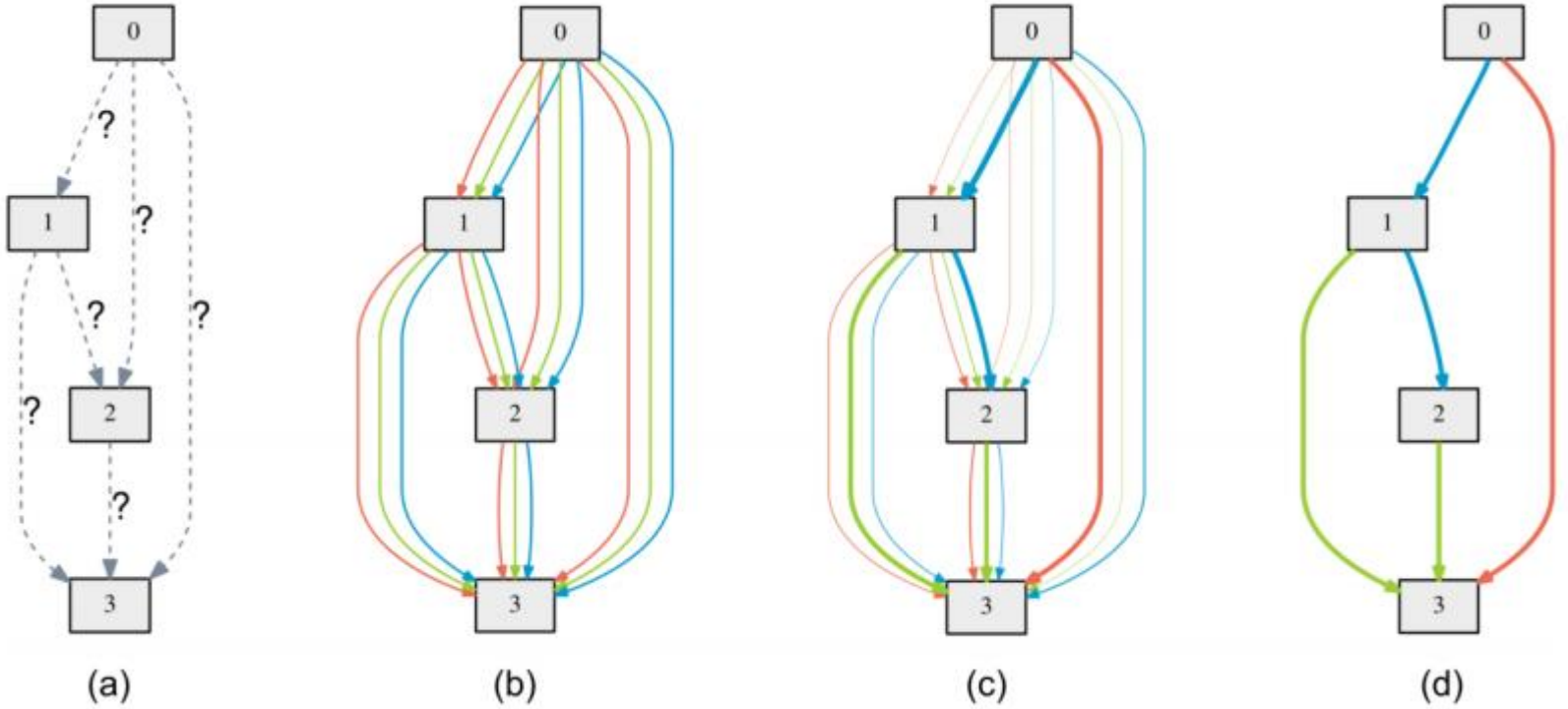


Convolution Cell



Reduction Cell

DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH, ICLR 2019



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 bi-level optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

AmoebaNet — Regularized Evolution for Image Classifier Architecture Search, AAAI 2018