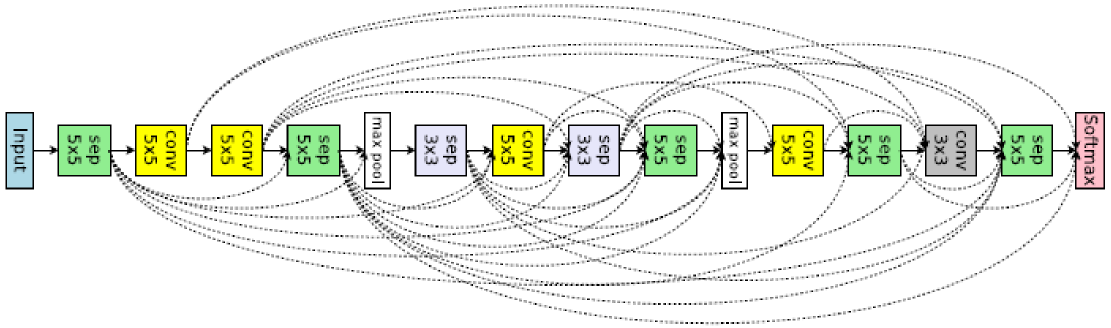
Efficient Neural Architecture Search via Parameter Sharing



Hieu Pham, et al. "Efficient Neural Architecture Search via Parameter Sharing", arXiv:1802.03268

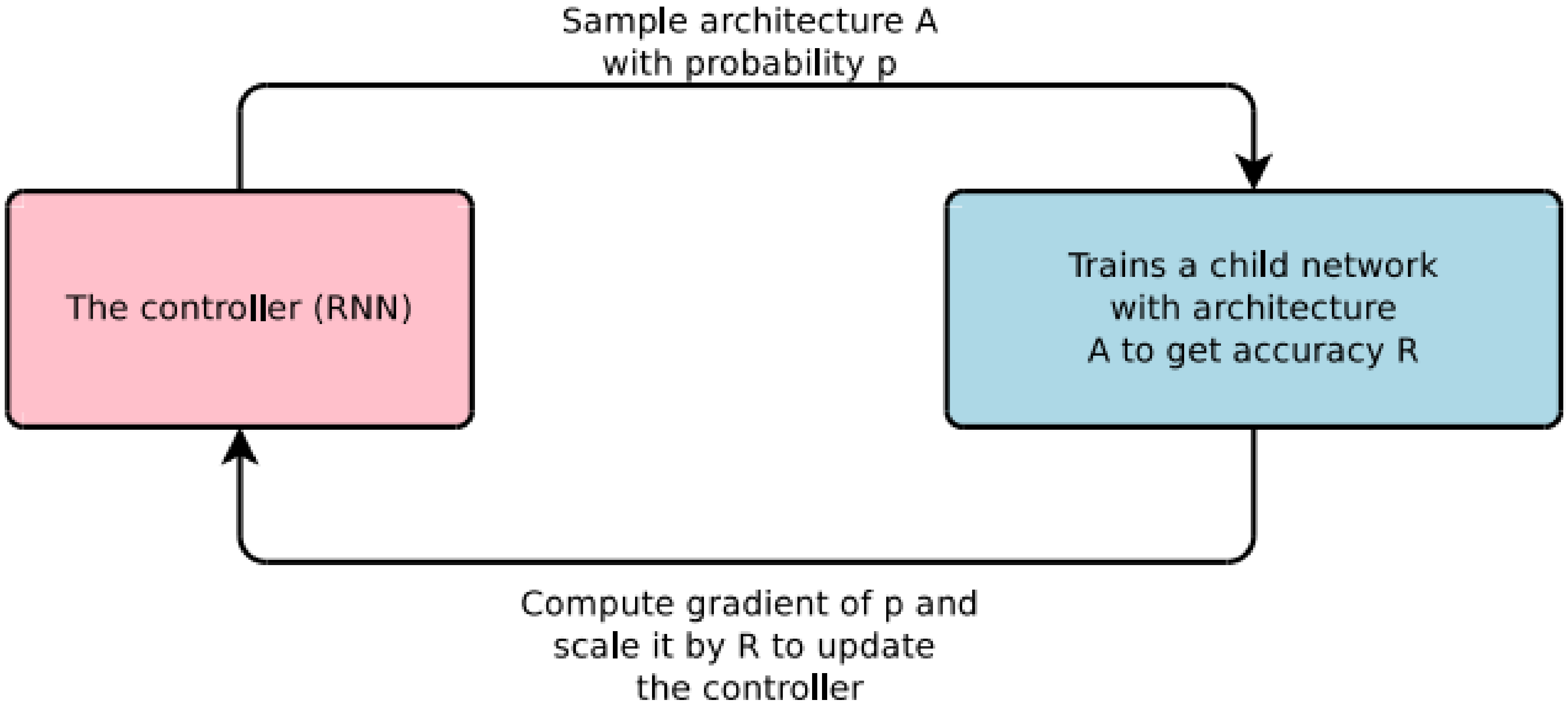
Jinwon Lee

Reference Papers

- Barret Zoph, et al. “Neural Architecture Search with Reinforcement Learning”, In ICLR, 2017
- Irwan Bello, et al. “Neural Optimizer Search with Reinforcement Learning”, In ICML, 2017
- Barret Zoph, et al. “Learning Transferable Architectures for Scalable Image Recognition”, In CVPR, 2018

Neural Architecture Search

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



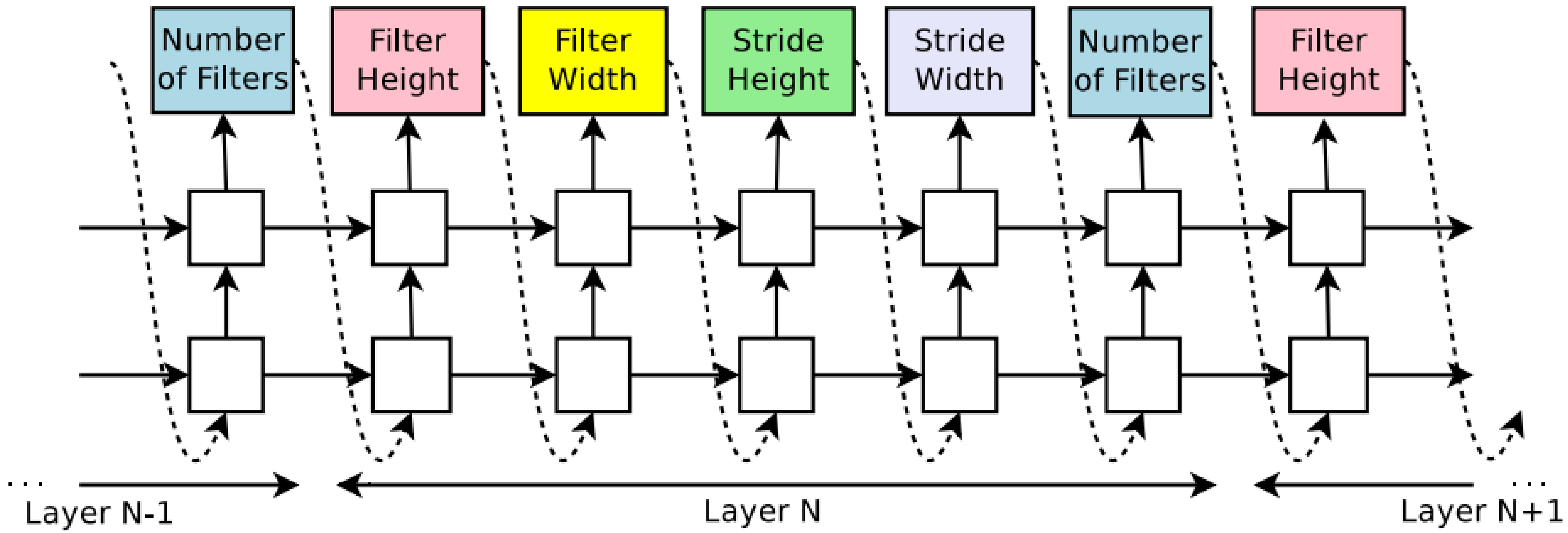
Tesorflow Pre-trained Models

- https://github.com/tensorflow/models/tree/master/research/slim

Model	TF-Slim File	Checkpoint	Top-1 Accuracy	Top-5 Accuracy
Inception V1	Code	inception_v1_2016_08_28.tar.gz	69.8	89.6
Inception V2	Code	inception_v2_2016_08_28.tar.gz	73.9	91.8
Inception V3	Code	inception_v3_2016_08_28.tar.gz	78.0	93.9
Inception V4	Code	inception_v4_2016_09_09.tar.gz	80.2	95.2
Inception-ResNet-v2	Code	inception_resnet_v2_2016_08_30.tar.gz	80.4	95.3
ResNet V1 50	Code	resnet_v1_50_2016_08_28.tar.gz	75.2	92.2
ResNet V1 101	Code	resnet_v1_101_2016_08_28.tar.gz	76.4	92.9
ResNet V1 152	Code	resnet_v1_152_2016_08_28.tar.gz	76.8	93.2
ResNet V2 50^	Code	resnet_v2_50_2017_04_14.tar.gz	75.6	92.8
ResNet V2 101^	Code	resnet_v2_101_2017_04_14.tar.gz	77.0	93.7
ResNet V2 152^	Code	resnet_v2_152_2017_04_14.tar.gz	77.8	94.1
ResNet V2 200	Code	TBA	79.9*	95.2*
VGG 16	Code	vgg_16_2016_08_28.tar.gz	71.5	89.8
VGG 19	Code	vgg_19_2016_08_28.tar.gz	71.1	89.8
MobileNet_v1_1.0_224	Code	mobilenet_v1_1.0_224_2017_06_14.tar.gz	70.7	89.5
MobileNet_v1_0.50_160	Code	mobilenet_v1_0.50_160_2017_06_14.tar.gz	59.9	82.5
MobileNet_v1_0.25_128	Code	mobilenet_v1_0.25_128_2017_06_14.tar.gz	41.3	66.2
NASNet-A_Mobile_224#	Code	nasnet-a_mobile_04_10_2017.tar.gz	74.0	91.6
NASNet-A_Large_331#	Code	nasnet-a_large_04_10_2017.tar.gz	82.7	96.2

Neural Architecture Search

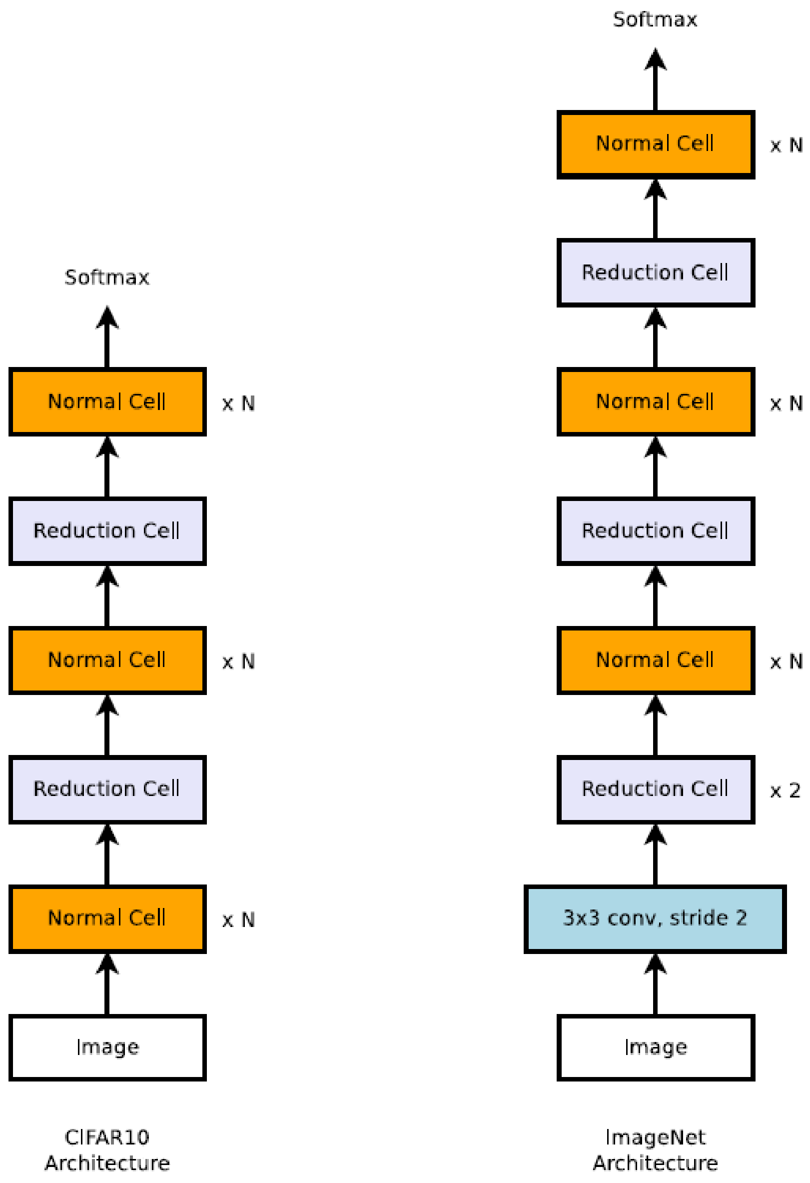
- How controller RNN samples a simple convolutional network



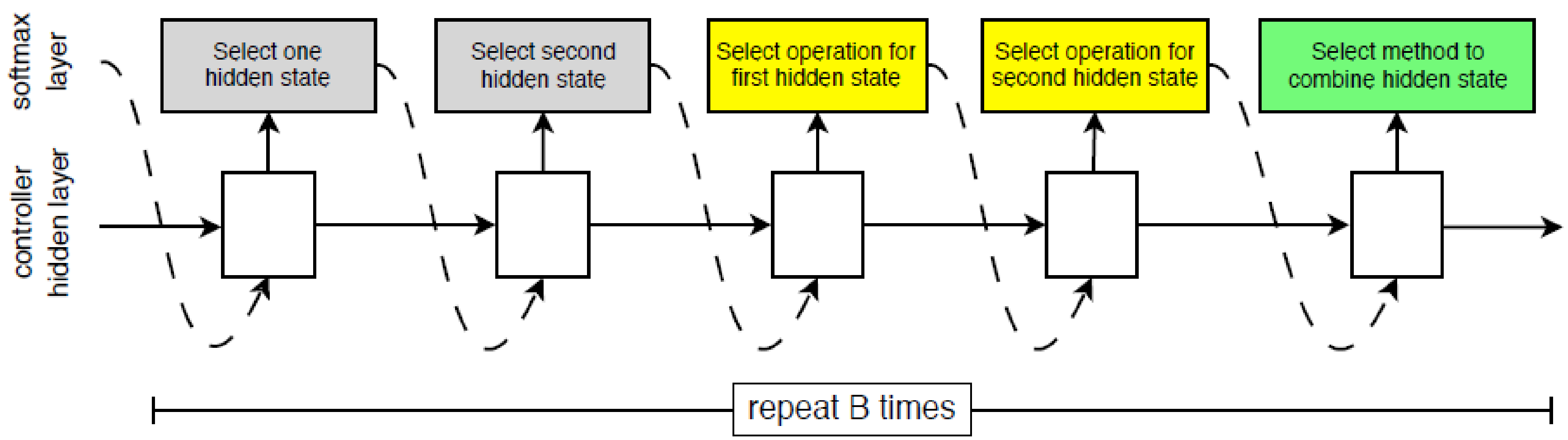
Method

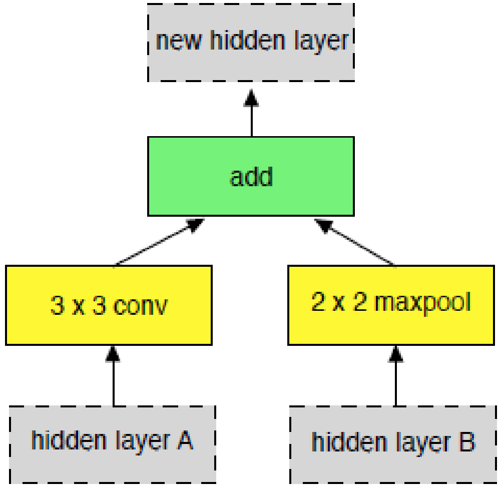
- Overall architectures of the convolutional nets are manually predetermined
  - Normal Cell – convolutional cells that return a feature map of the same dimension
  - Reduction Cell – convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two
- 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.

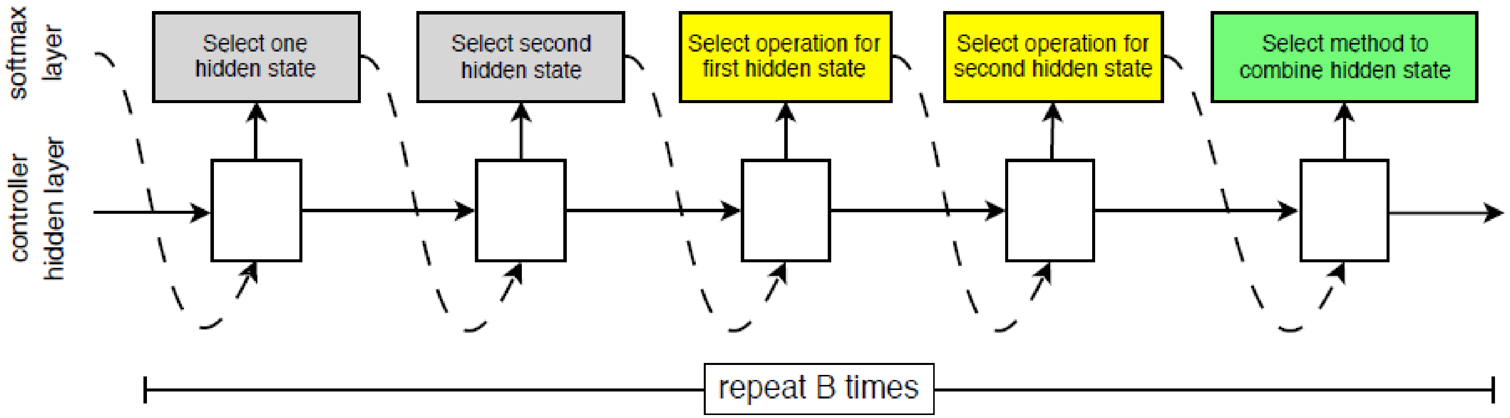
Step 2. Select a second hidden state from the same options as in Step 1.

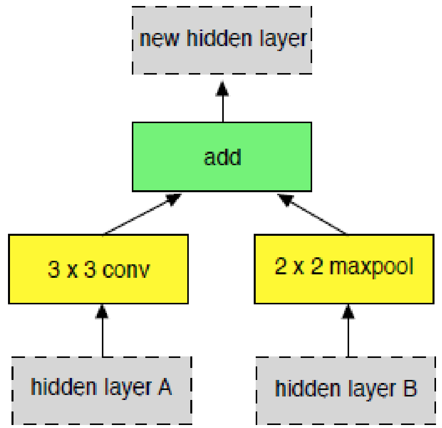
Step 3. Select an operation to apply to the hidden state selected in Step 1.

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.

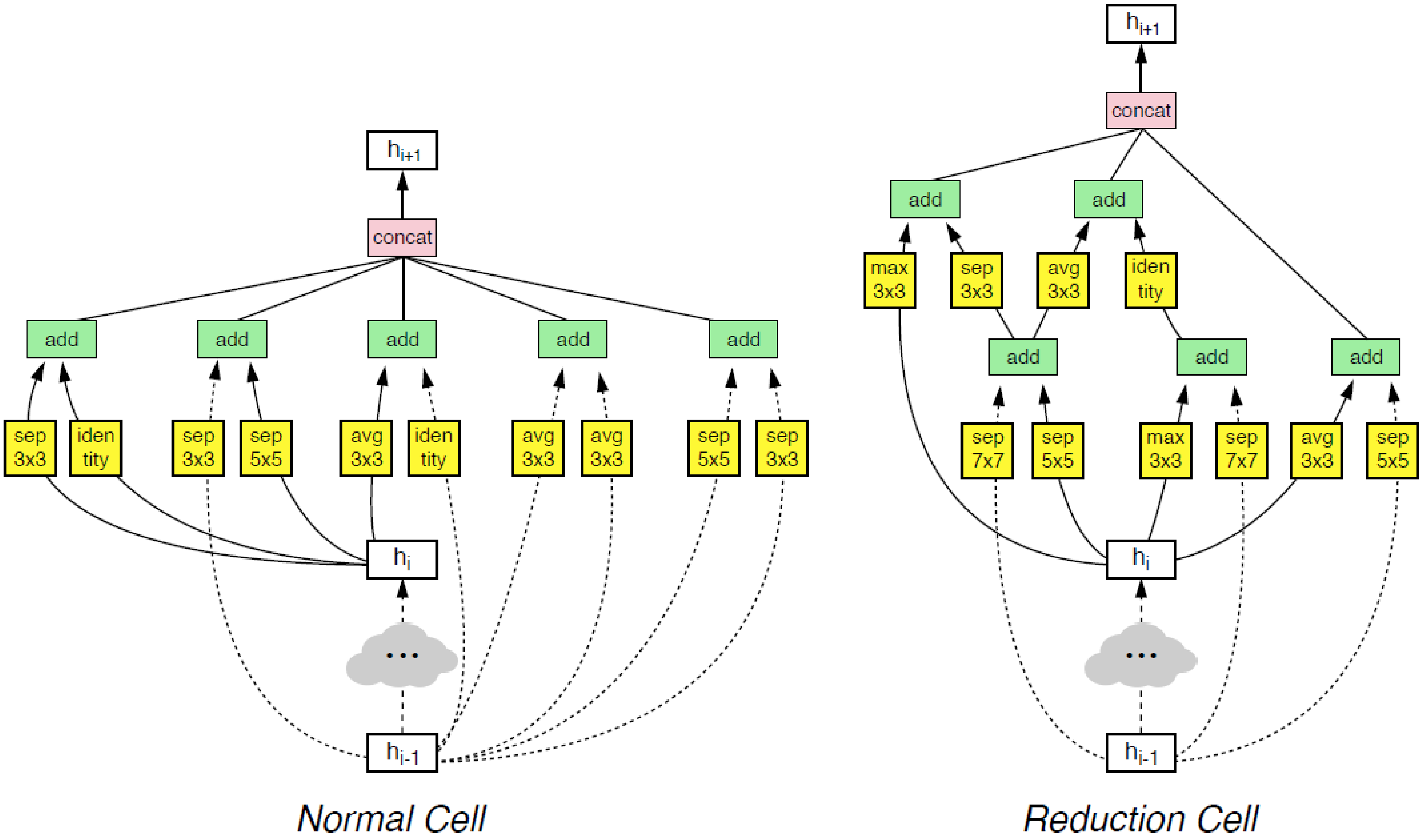
Search Space in a Cell (Step 3 and 4)



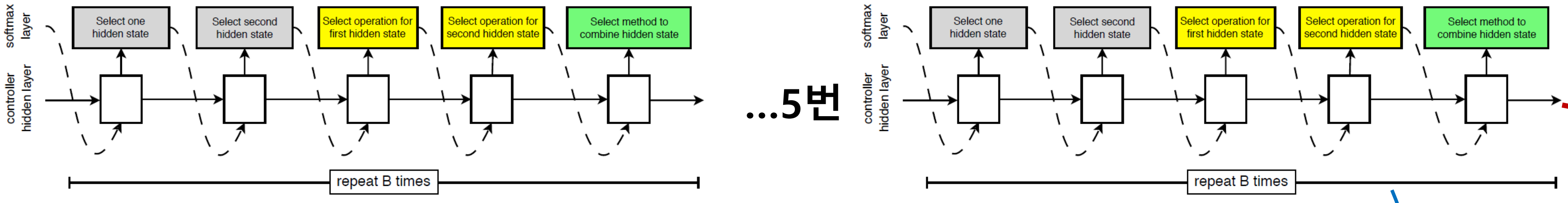


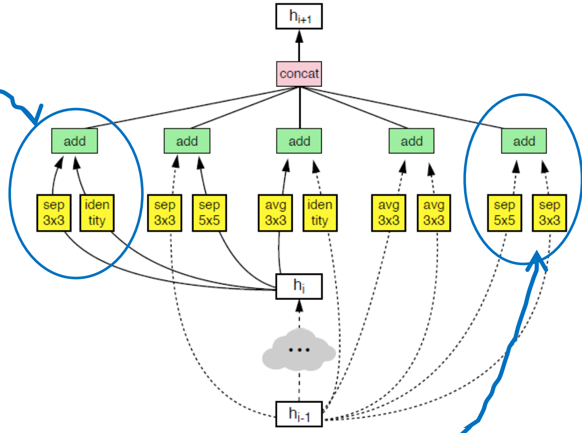
- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

Best Architecture (NASNet-A)



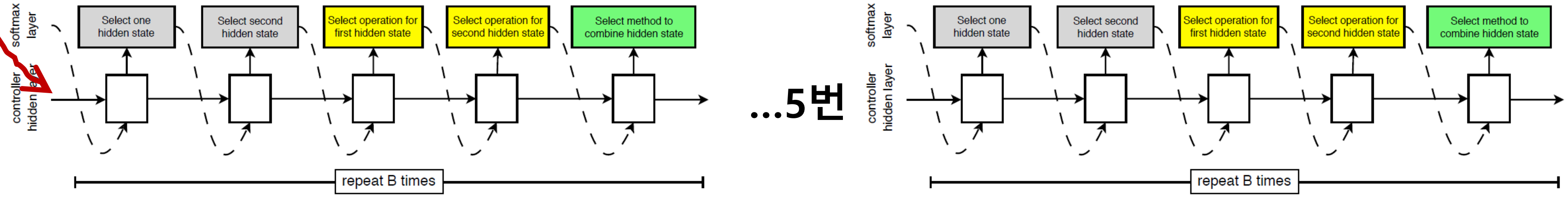
Best Architecture (NASNet-A)

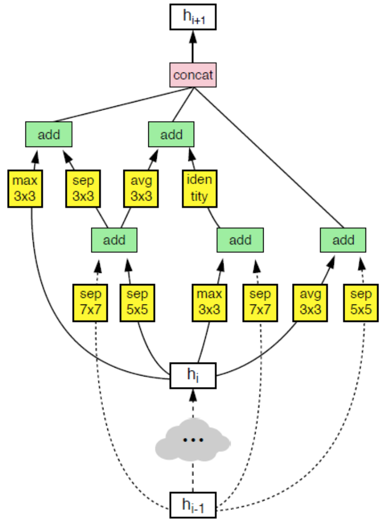




Normal Cell

<Normal Cell>

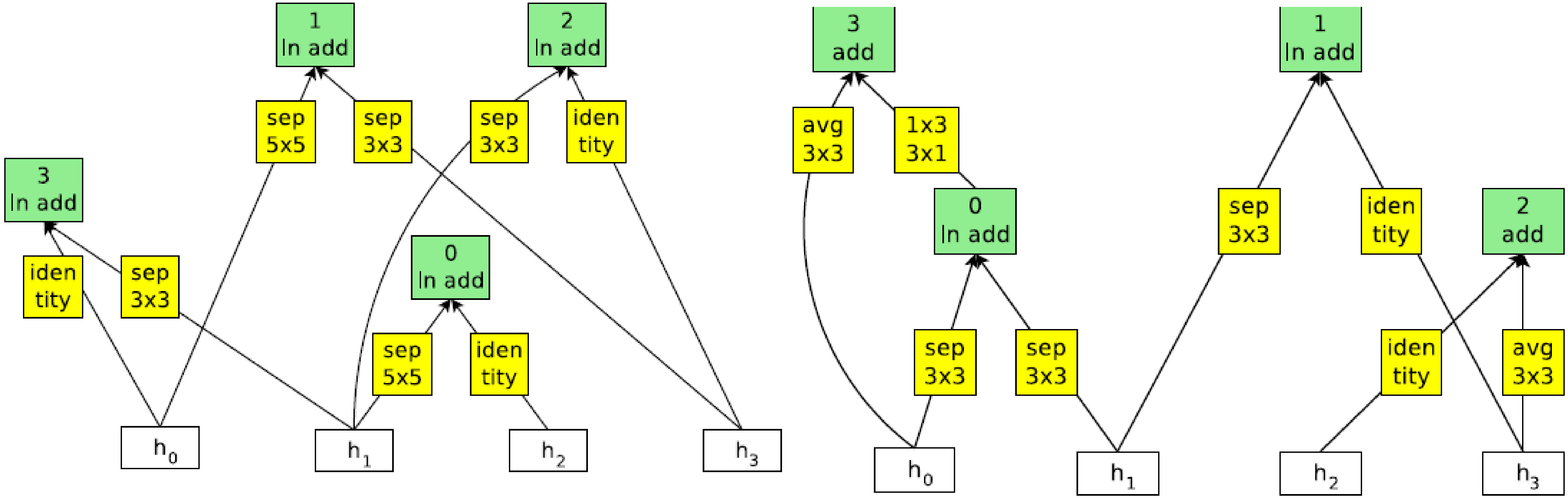




Reduction Cell

<Reduction Cell>

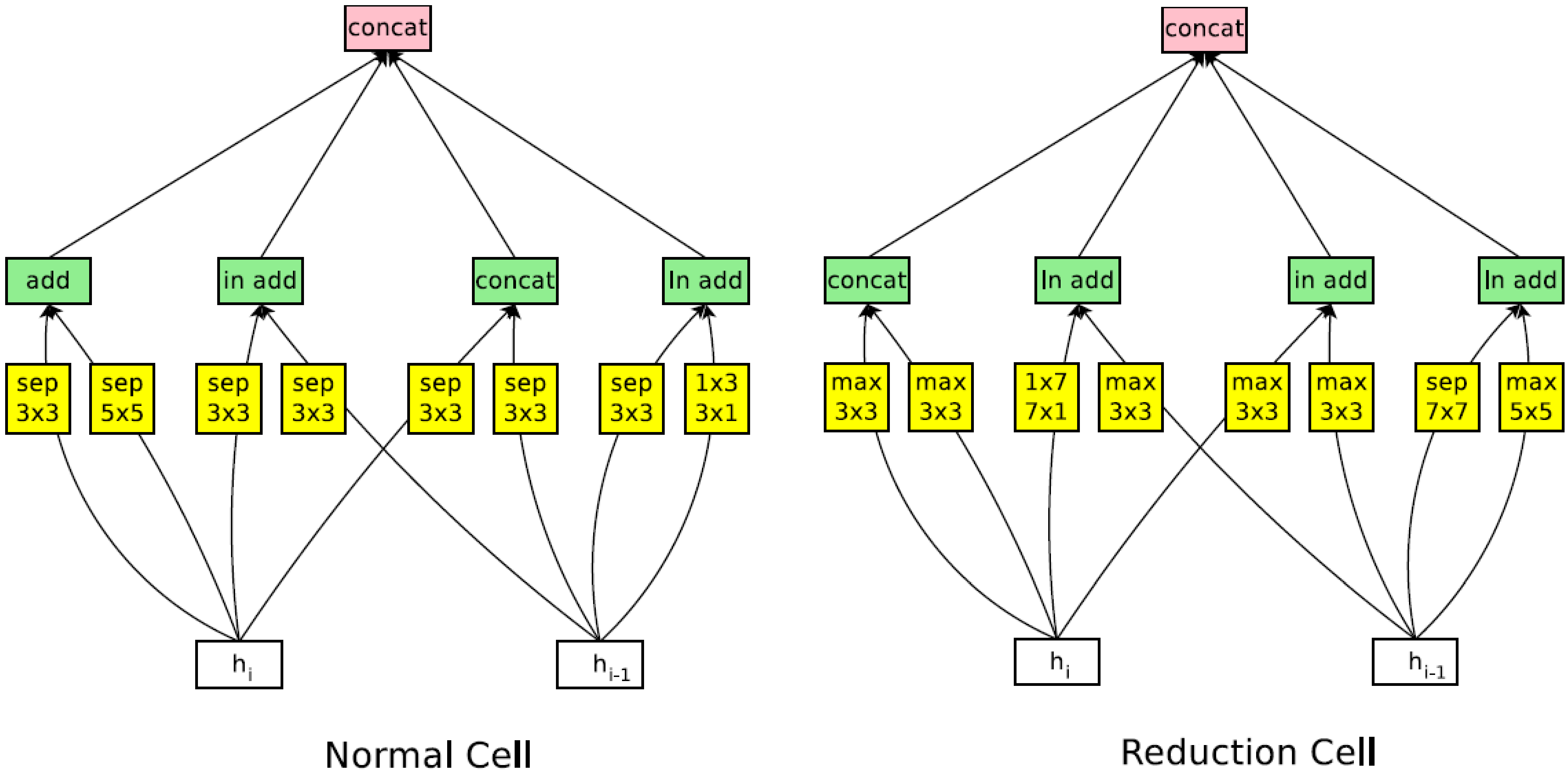
NASNet-B



Normal Cell

Reduction Cell

NASNet-C


concat
add
in add
concat
In add
sep 3x3
sep 5x5
sep 3x3
sep 3x3
sep 3x3
sep 3x3
sep 3x3
1x3 3x1
h_i
h_{i-1}
Normal Cell
concat
concat
In add
in add
In add
max 3x3
max 3x3
1x7 7x1
max 3x3
max 3x3
max 3x3
sep 7x7
max 5x5
h_i
h_{i-1}
Reduction Cell

Motivation

- NAS used 800 GPUs for 28days and NASNet used 450 GPUs for 3-4 days (i.e. 32,400-43,200 GPU hours)
- Meanwhile, using less resources tends to produce less compelling results
- 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

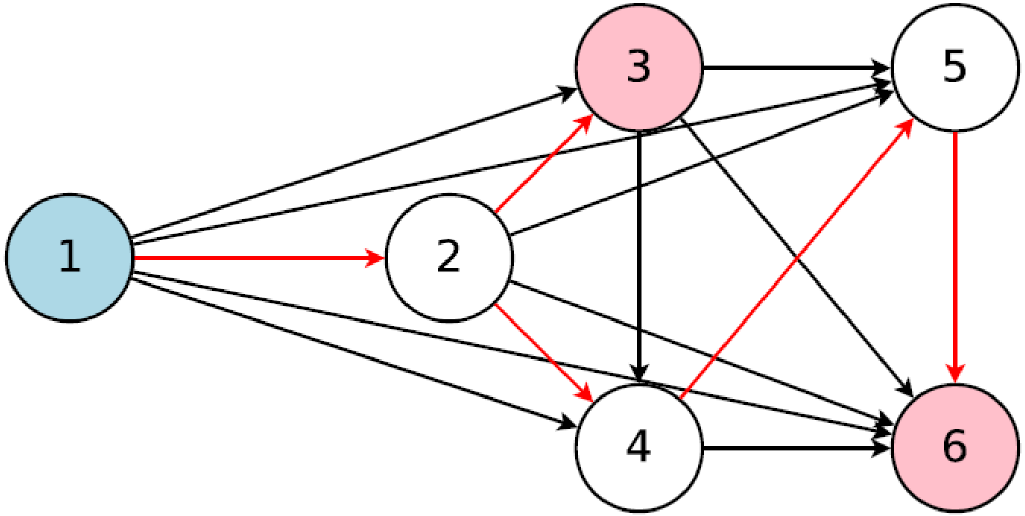
Main Idea

Forcing all child models to share weights

to eschew training each child model from scratch to convergence

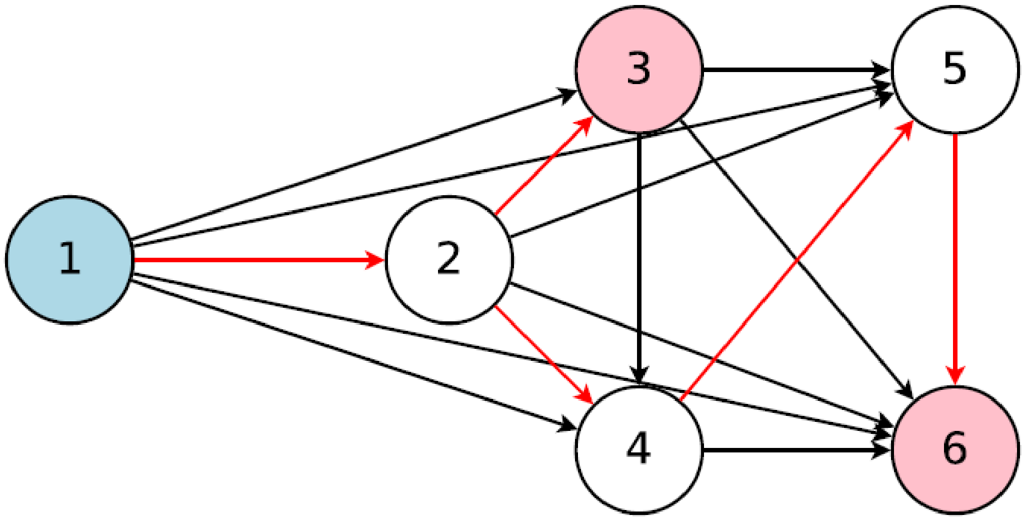
→ Using Single GTX 1080Ti GPU, the search for architectures takes less than 16 hours (compared to NAS, reduction is more than 1000x)

Directed Acyclic Graph(DAG)



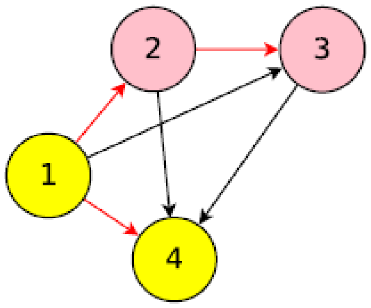
- ENAS's DAG is the superposition of all possible child models in a search space of NAS, where the nodes represent the local computations and the edges represent the flow of information.
- The local computations at each node have their own parameters, which are used only when the particular computation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

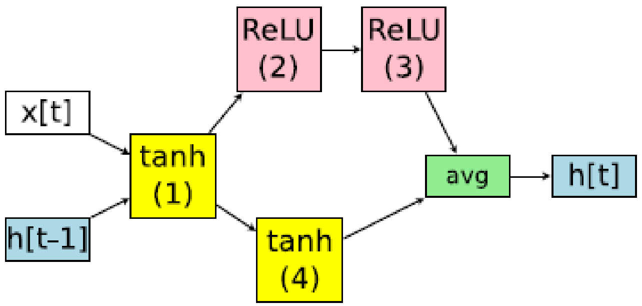
Directed Acyclic Graph(DAG)

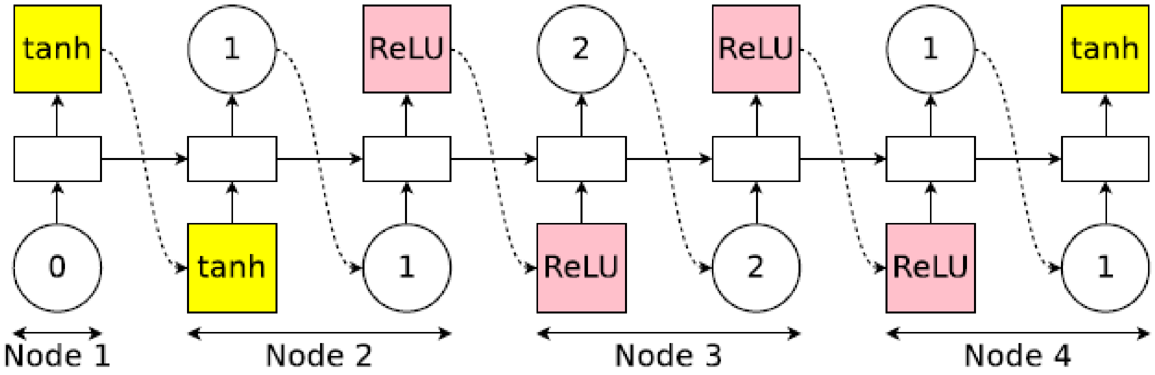


- The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller.
- Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

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 $h_1 = \tanh\left(\mathbf{x}_t \cdot \boxed{\mathbf{W}^{(\mathrm{x})}} + \mathbf{h}_{t-1} \cdot \boxed{\mathbf{W}_1^{(\mathrm{h})}}\right)$.

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 $h_2 = \mathrm{ReLU}(h_1 \cdot \boxed{\mathbf{W}_{2,1}^{(\mathrm{h})}})$.

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, $h_3 = \mathrm{ReLU}(h_2 \cdot \boxed{\mathbf{W}_{3,2}^{(\mathrm{h})}})$.

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 $h_4 = \tanh\left(h_1 \cdot \boxed{\mathbf{W}_{4,1}^{(\mathrm{h})}}\right)$.

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,

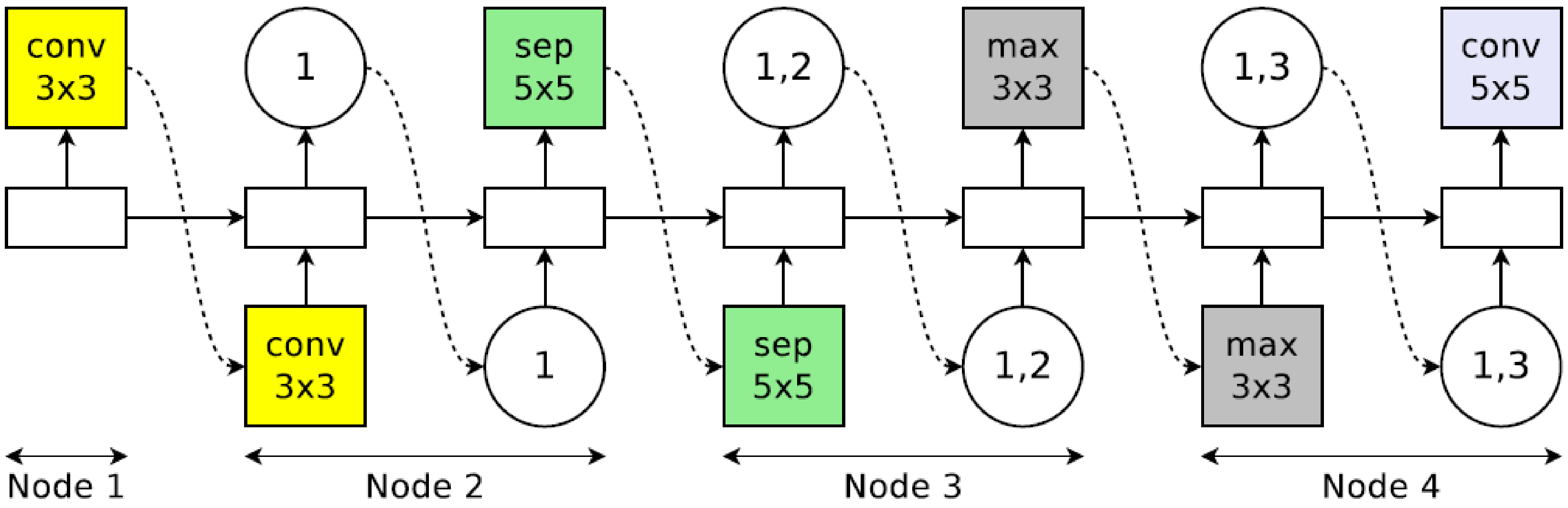
$$\nabla_\omega \mathbb{E}_{\mathbf{m}\sim\pi(\mathbf{m};\theta)}\left[\mathcal{L}(\mathbf{m};\omega)\right] \approx \frac{1}{M}\sum_{i=1}^{M}\nabla_\omega \mathcal{L}(\mathbf{m}_i,\omega),$$

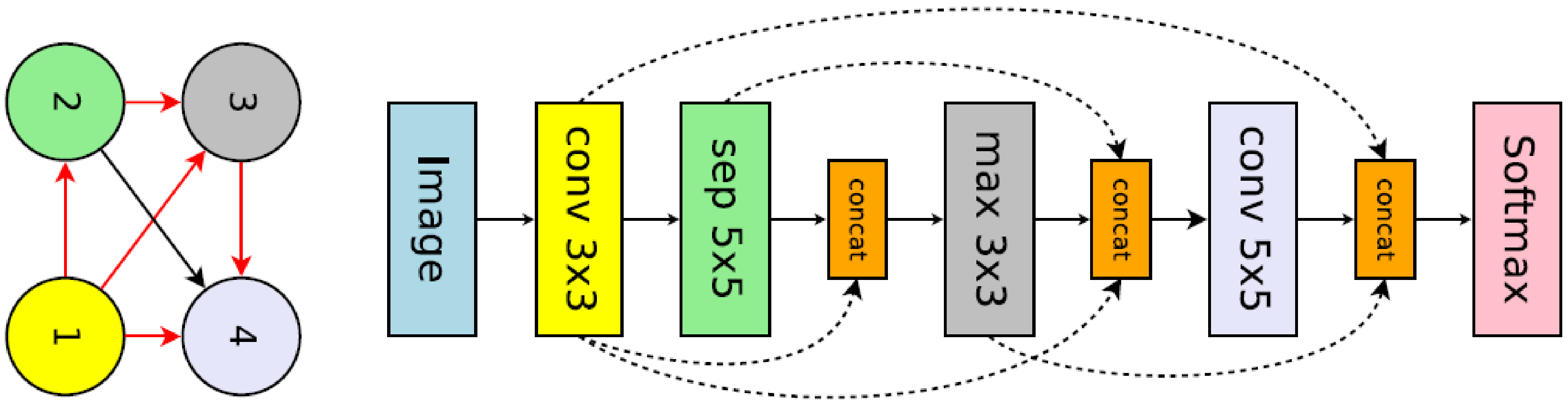
- The second phase trains θ

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

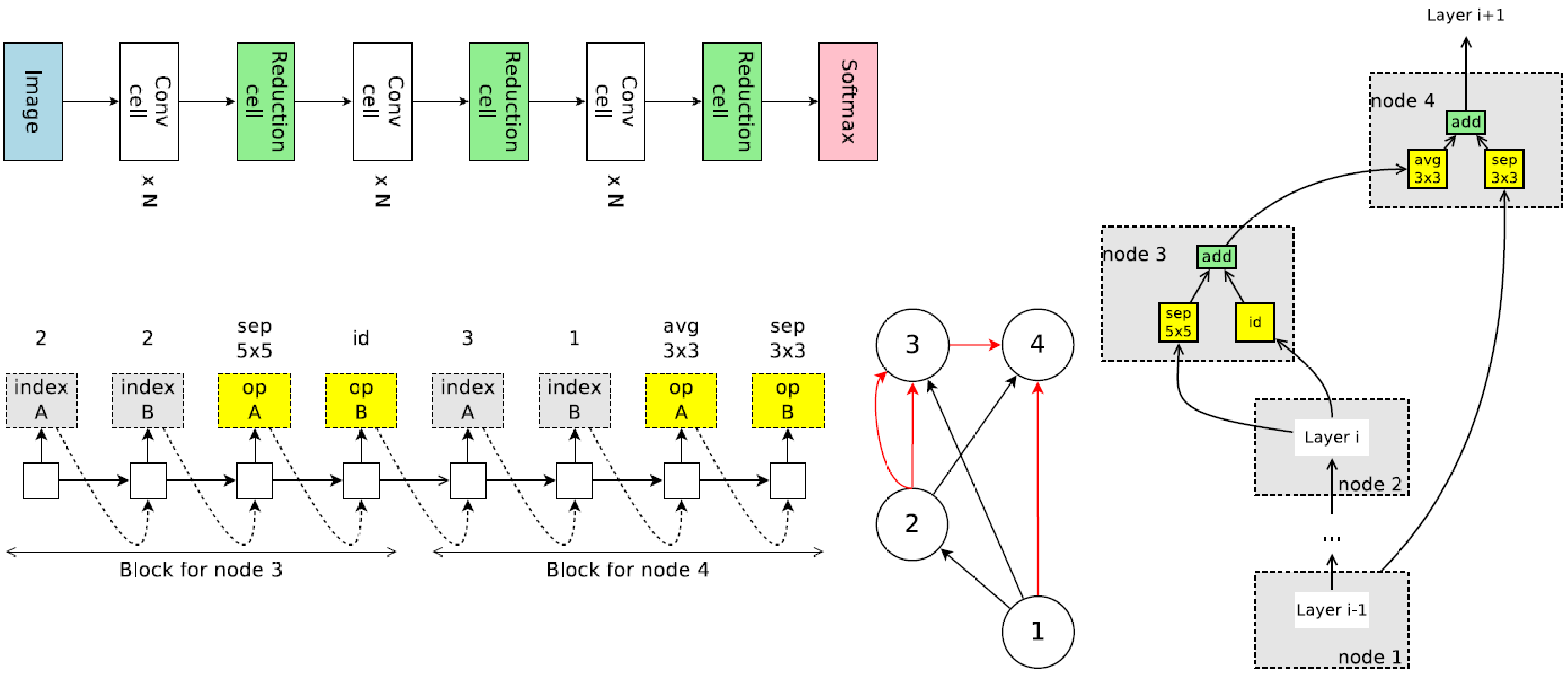




Search Spaces for CNN

- 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

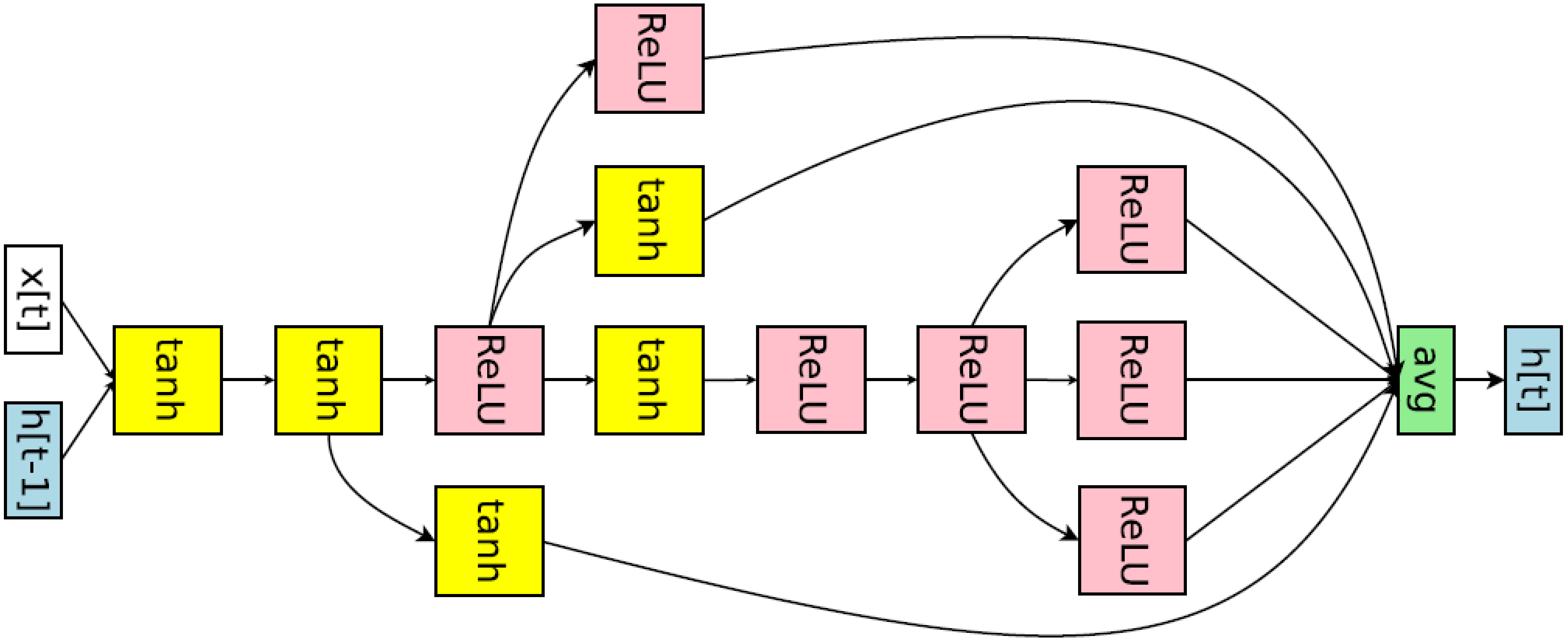
Designing Convolutional Cells



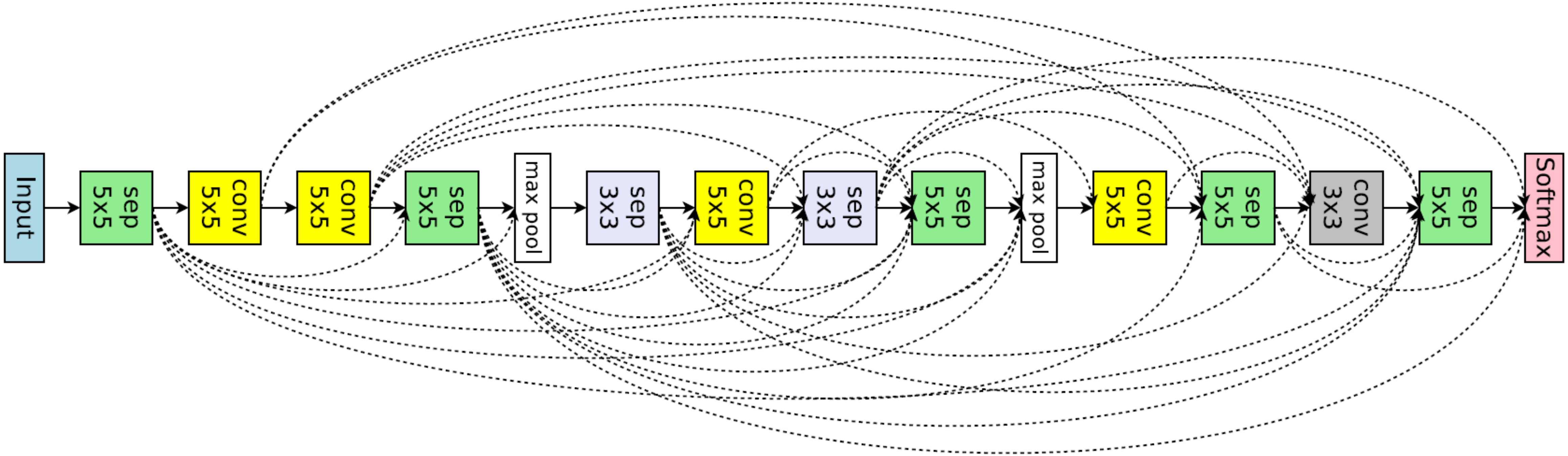
Search Spaces for Convolutional Cells

- 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

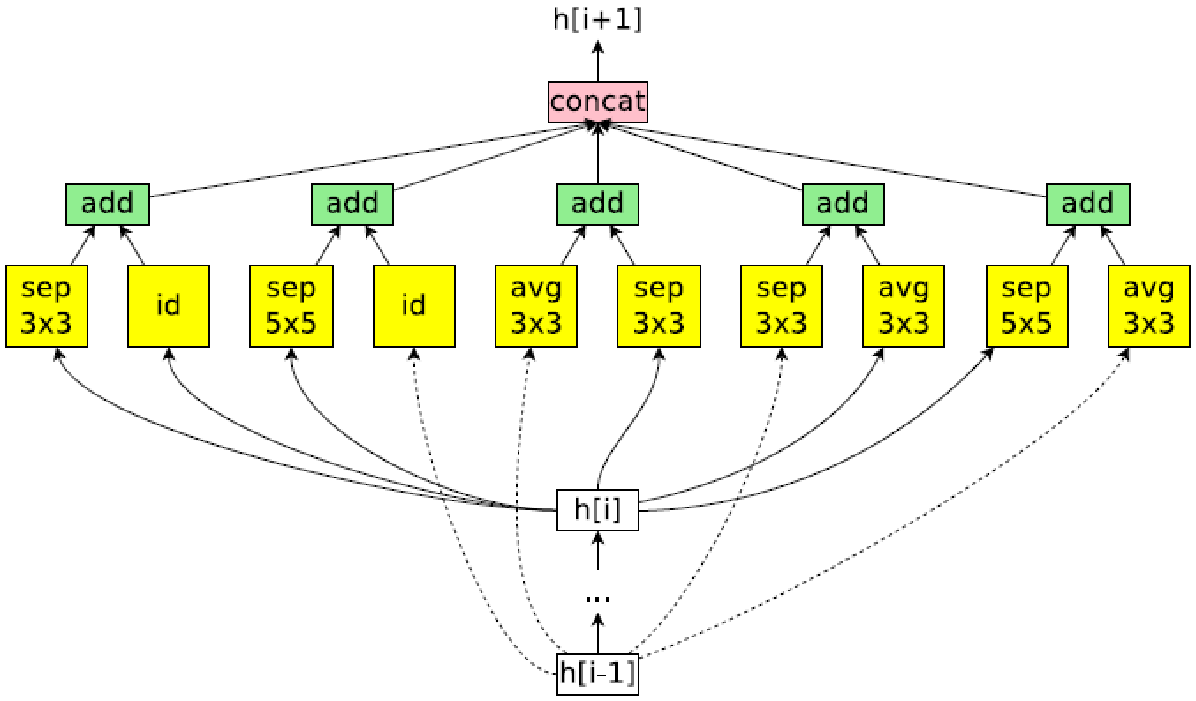
Network Architecture for Penn Treebank



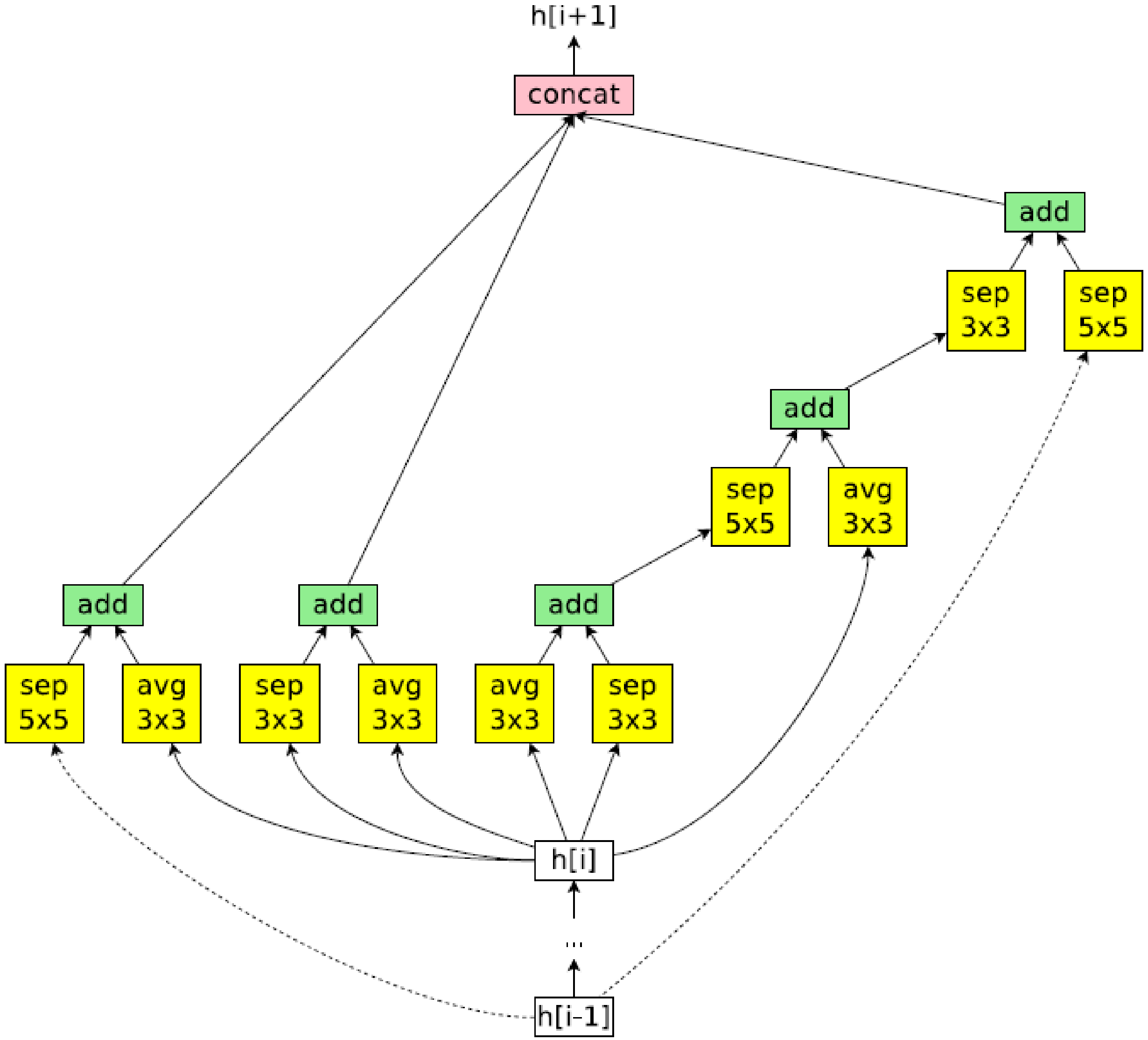
Network Architecture for CIFAR-10(from macro search space)



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



Convolution Cell



Reduction Cell

Experimental Results

- Test perplexity on Penn Treebank of ENAS and other baselines

Architecture	Additional Techniques	Params (million)	Test PPL
LSTM (Zaremba et al., 2014)	Vanilla Dropout	66	78.4
LSTM (Gal & Ghahramani, 2016)	VD	66	75.2
LSTM (Inan et al., 2017)	VD, WT	51	68.5
LSTM (Melis et al., 2017)	Hyper-parameters Search	24	59.5
LSTM (Yang et al., 2018)	VD, WT, ℓ_2, AWD, MoC	22	57.6
LSTM (Merity et al., 2017)	VD, WT, ℓ_2, AWD	24	57.3
LSTM (Yang et al., 2018)	VD, WT, ℓ_2, AWD, MoS	**22**	**56.0**
RHN (Zilly et al., 2017)	VD, WT	24	66.0
NAS (Zoph & Le, 2017)	VD, WT	54	62.4
ENAS	VD, WT, ℓ_2	**24**	**55.8**

Experimental Results

- Classification errors of ENAS and baselines on CIFAR-10

Method	**GPUs**	**Times** (days)	**Params** (million)	**Error** (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**