Deep Learning Optimization - Neural Architecture Search

April 17, 2023

Eunhyeok Park

Policy Gradient based RL

Policy Gradient based RL

- Previous methods are based on the value or action-value function

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrise the policy

$$\pi_\theta(s,a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- We will focus again on model-free reinforcement learning

Policy Optimization

- Policy based reinforcement learning is an optimisation problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

One-Step MDPs

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \mathcal{R}_{s,a} \\
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]
\end{aligned}
$$

- Likelihood ratios exploit the following identity

$$
\begin{aligned}
\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\
&= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)
\end{aligned}
$$

Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^{\pi}(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, *or* $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

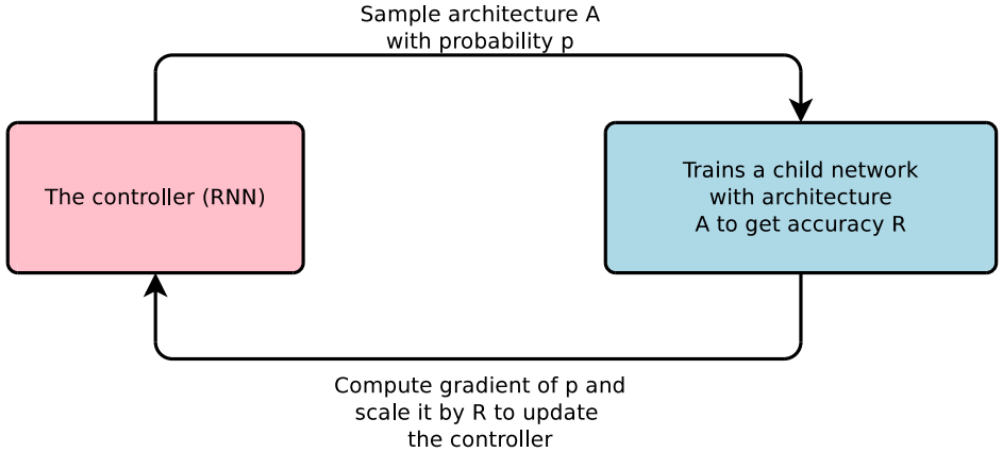
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \; Q^{\pi_\theta}(s, a) \right]$$

Neural Architecture Search with REINFORCEMENT Learning

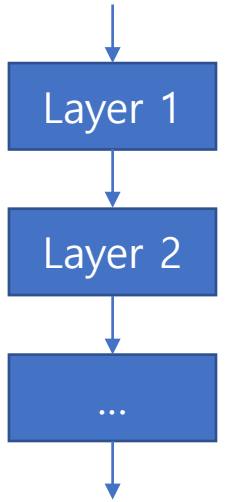
Neural Architecture Search

- Automate architecture design based on RL algorithm
  - Action = architecture design
  - Reward = classification accuracy

  - Try to find out the best network configuration based on policy gradient
  - Use the final accuracy as a reward
    - Update the policy to maximize the final reward

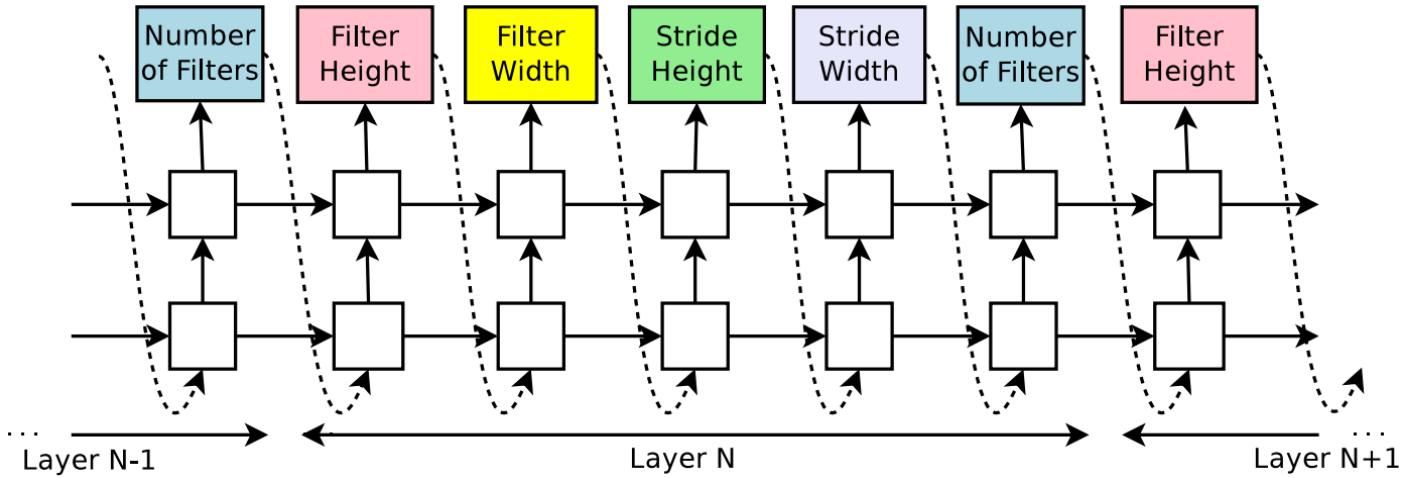


Neural Architecture Search - 2



A filter height [1,3,5,7]
A filter width [1,3,5,7]
A number of filters [24, 36, 48, 64]
Stride or max-pooling [1,2,3]

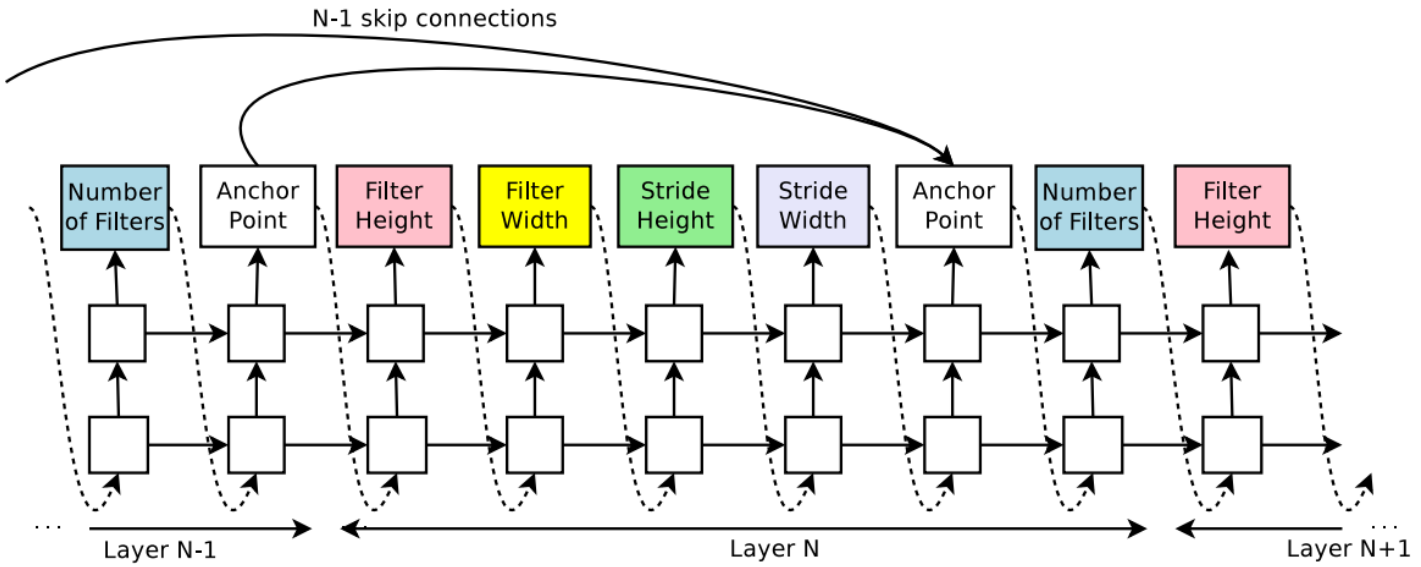
- Set the network configurations as an action
  - Make a decision in discrete space
  - Sequence decision making equal to # of layers
  - Use LSTM to integrate the effort of actions
  - LSTM sequentially generate action probability (network configuration)



Attach Skip Connection

- Adopt additional neural network to predict the connection probability of
- Calculate probability of skip connection based on hidden state of anchor point
  - h_i is the hidden state from the i^{th} anchor point
  - v, W_{prev}, W_{curr} are trainable parameters

$$\mathrm{P}(\text{Layer j is an input to layer i}) = \mathrm{sigmoid}(v^{\mathrm{T}}\tanh(W_{prev} * h_j + W_{curr} * h_i))$$

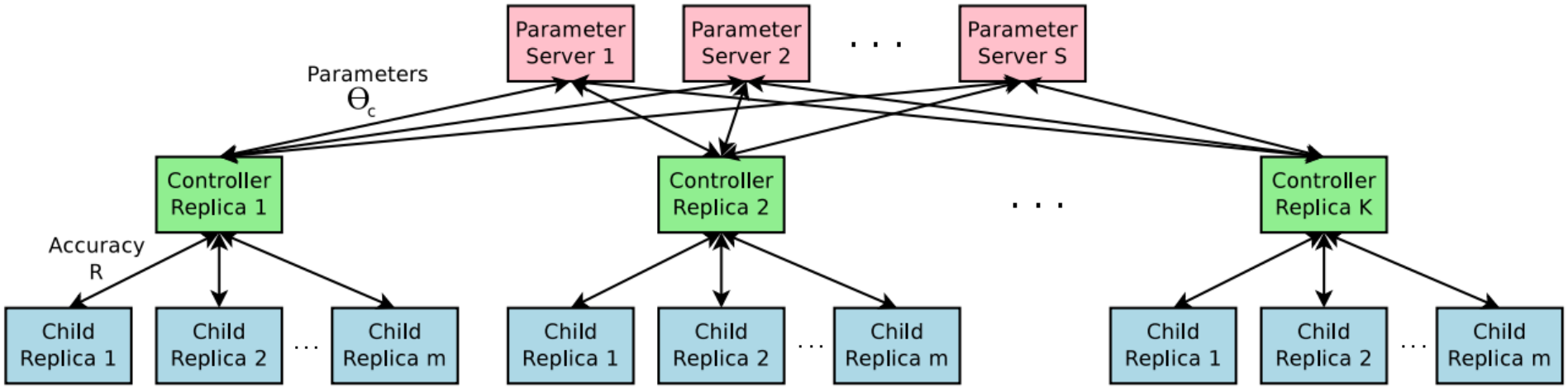


Search Pipeline

- 1. Controller RNN generates a probability of a given action space
- 2. Sample a network from the action and train the proposal from scratch
- 3. Record the validation accuracy when the network converges
- 4. Update RNN generator parameters using policy gradient
- 5. Repeat 1~4 until the validation accuracy is not improved
- 6. Select the architecture having the highest validation accuracy

$$\begin{aligned}\nabla_{\theta_c} J(\theta_c) &= \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\left[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c) R\right] \\ &\approx \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c) R_k \\ &\approx \frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)(R_k - b)\end{aligned}$$

Acceleration NAS with Distributed Training



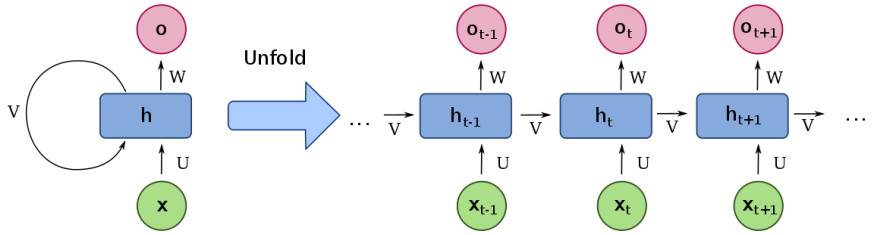
- 12800 architecture search for Cifar-10 dataset
- 800 GPUs are parallelized to accelerate architecture evaluation
- S parameter servers, K controller replicas, m architecture per controller

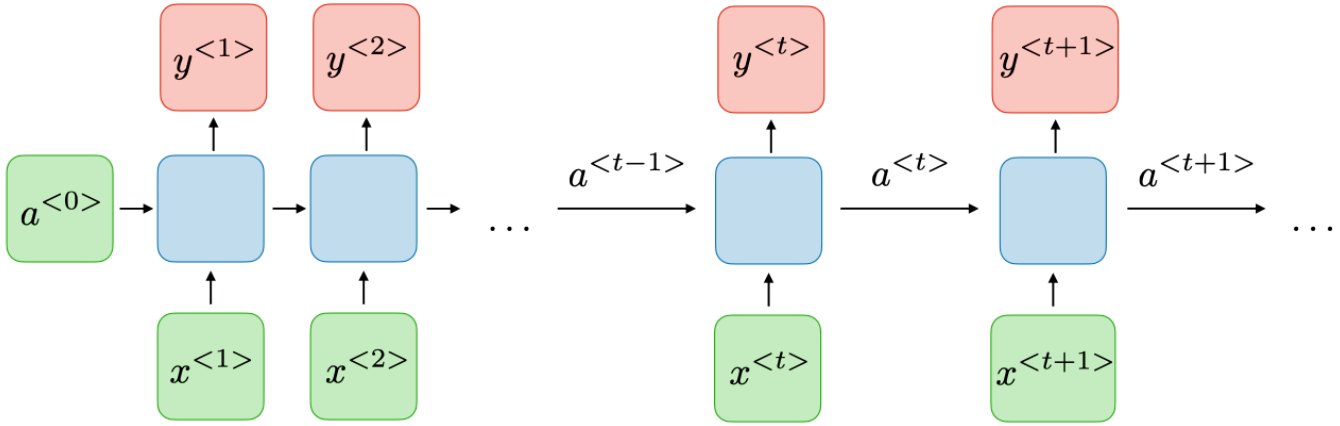
Results

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet $(L = 40, k = 12)$ Huang et al. (2016a)	40	1.0M	5.24
DenseNet$(L = 100, k = 12)$ Huang et al. (2016a)	100	7.0M	4.10
DenseNet $(L = 100, k = 24)$ Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC $(L = 100, k = 40)$ Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

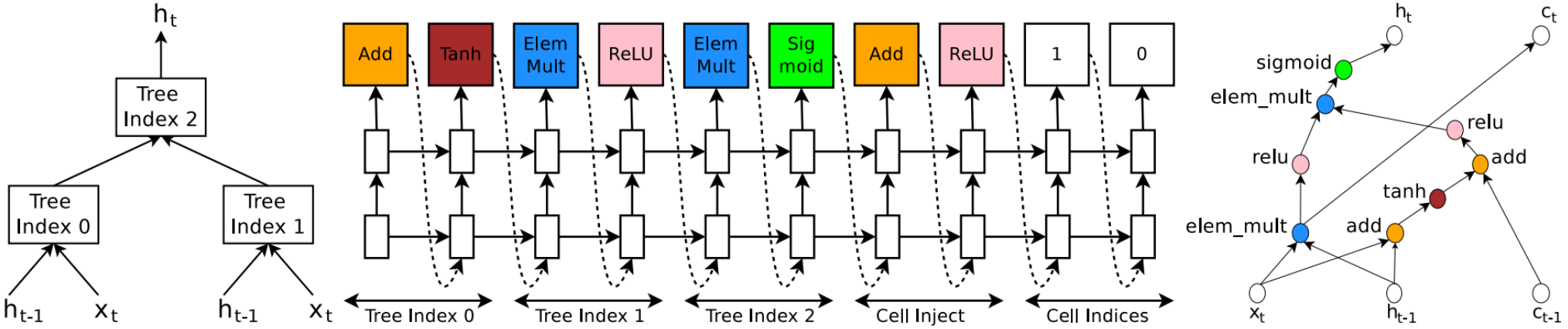
Recurrent Neural Network

- Recurrent Neural Network, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.
  - Use the same cell repeatedly to handle sequence input data
  - At each cell, there are two input data
    - Input data(embedding) from current time step
    - Hidden embedding from previous time step
  - There are two output data
    - Output embedding of current time step
    - Hidden embedding of current time step





RNN Cell Generation



- The controller predicts Add and $Tanh$ for tree index 0, this means we need to compute $a_0 = \tanh(W_1 * x_t + W_2 * h_{t-1})$.
- The controller predicts $ElemMult$ and $ReLU$ for tree index 1, this means we need to compute $a_1 = \mathrm{ReLU}\big((W_3 * x_t) \odot (W_4 * h_{t-1})\big)$.
- The controller predicts 0 for the second element of the "Cell Index", Add and $ReLU$ for elements in "Cell Inject", which means we need to compute $a_0^{new} = \mathrm{ReLU}(a_0 + c_{t-1})$. Notice that we don't have any learnable parameters for the internal nodes of the tree.
- The controller predicts $ElemMult$ and $Sigmoid$ for tree index 2, this means we need to compute $a_2 = \mathrm{sigmoid}(a_0^{new} \odot a_1)$. Since the maximum index in the tree is 2, h_t is set to a_2.
- The controller RNN predicts 1 for the first element of the "Cell Index", this means that we should set c_t to the output of the tree at index 1 before the activation, i.e., $c_t = (W_3 * x_t) \odot (W_4 * h_{t-1})$.

Results

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

Remaining Question

- Cifar-10 is too simple to use in real life
  - However, finding architecture in ImageNet is prohibitively expensive
  - Can we transfer the knowledge from cifar to imagenet?

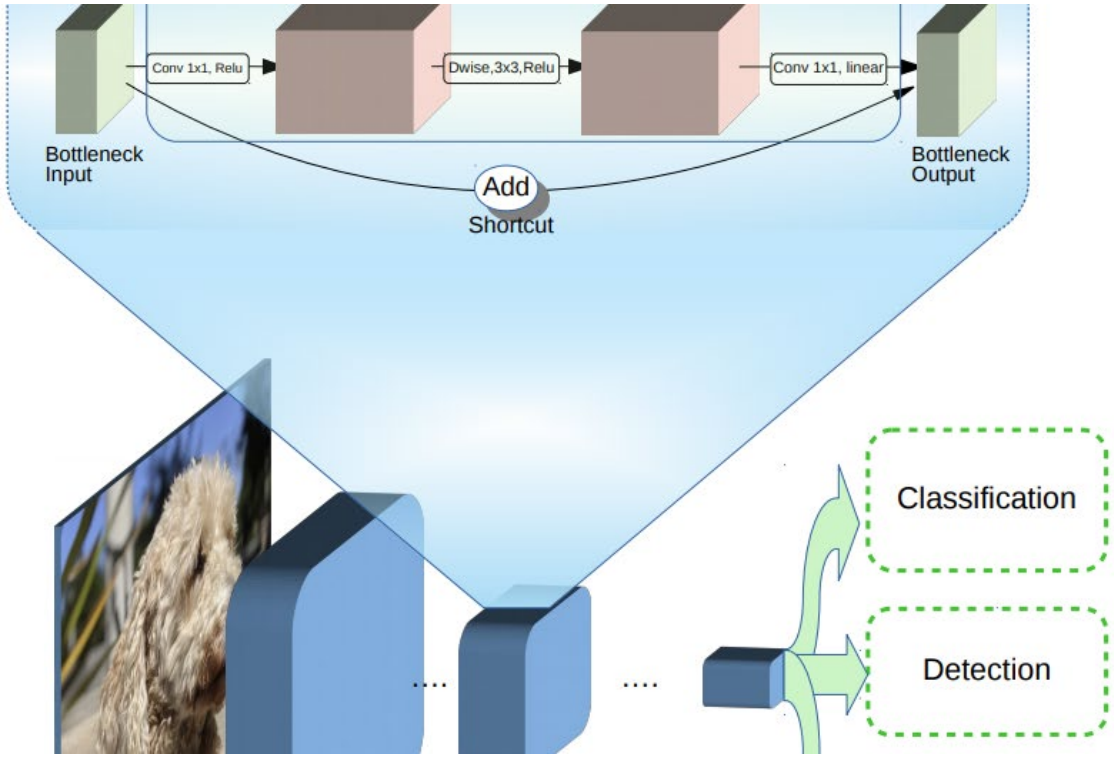
- Search space is enormous
  - Too expensive to evaluate network candidates
  - Takes 800 GPU x 28 days for 12800 candidates evaluation
  - Good initialization, reuse intermediate results, or performance prediction?

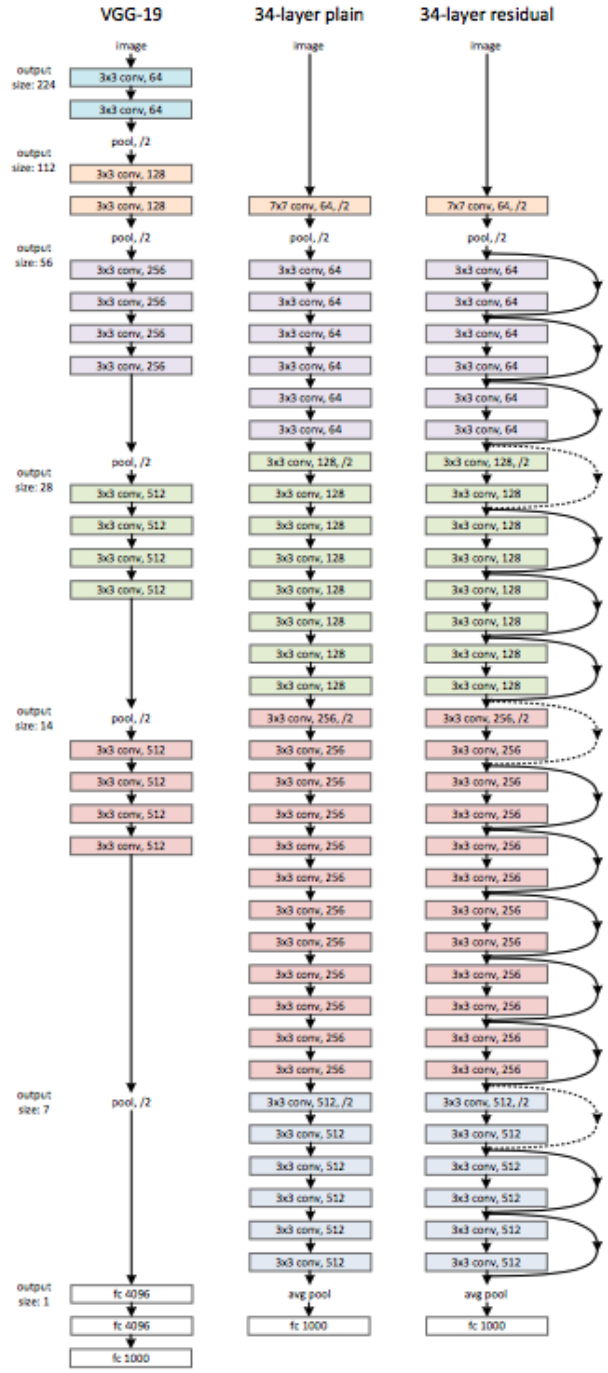
- Other approaches for network evaluation?

Learning Transferable Architectures for Scalable Image Recognition

Cell-based CNNs

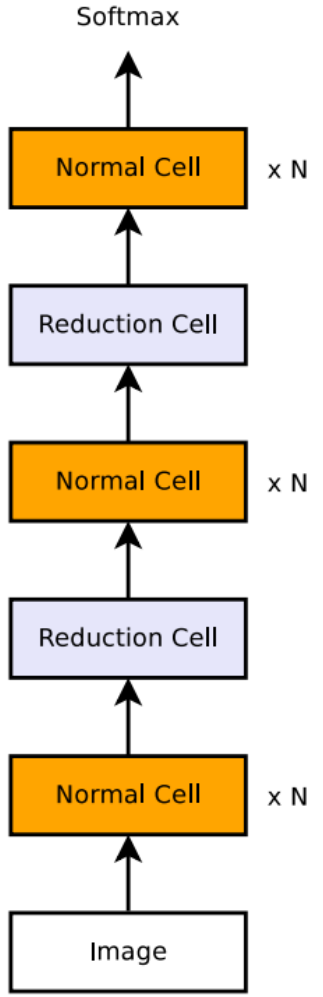
- Stack the predefined cell repeatedly
- Pooling/stride for spatial dimension reduction



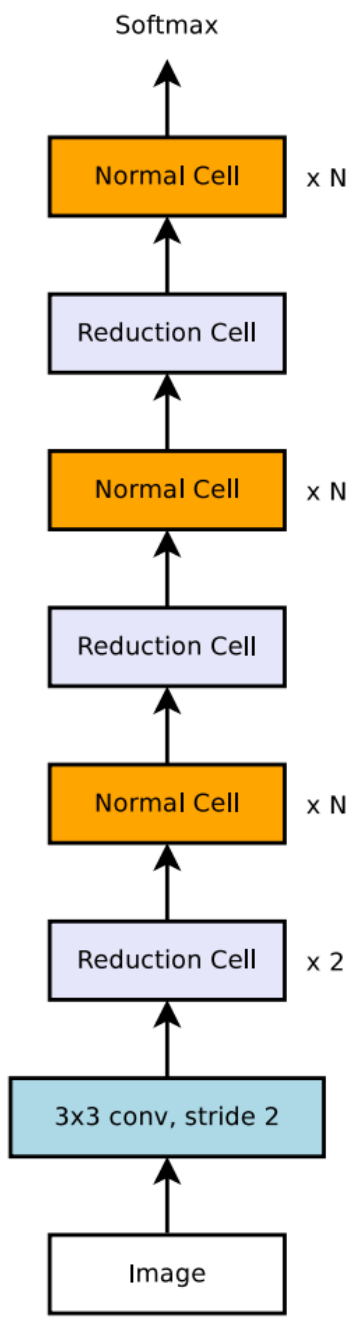


Properties of Cell-based CNN

- Search normal/reduction cell instead of entire network
  - Reduce search space drastically
  - Enable architecture transfer for different dataset
    - Use the same cell design
    - Basic assumption: Cells that can extract good features from images share similar forms.
    - Change channel width / cell repeat number based on the difficulty of the dataset



CIFAR10 Architecture



ImageNet Architecture

CNN Space Design

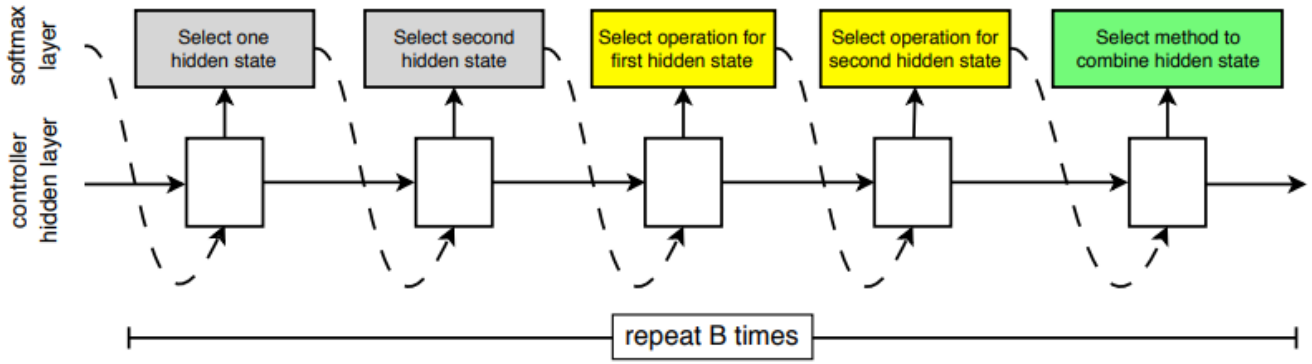
- Cell-based approach with 2 to 1 node design
  - Each cell has a strict restrictions but has fluent expression capability.

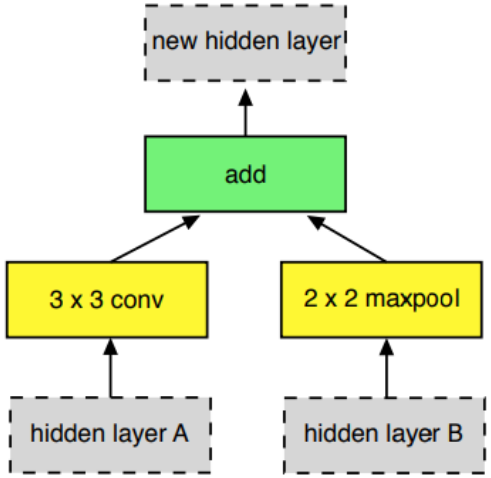
  - 1) two previous nodes to be used as inputs to the current node
  - 2) two operations to apply to the two sampled nodes.

  - Node 1 and 2 are the outputs of the two previous cells in the final network

  - The 13 available operations

NASNet Search





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

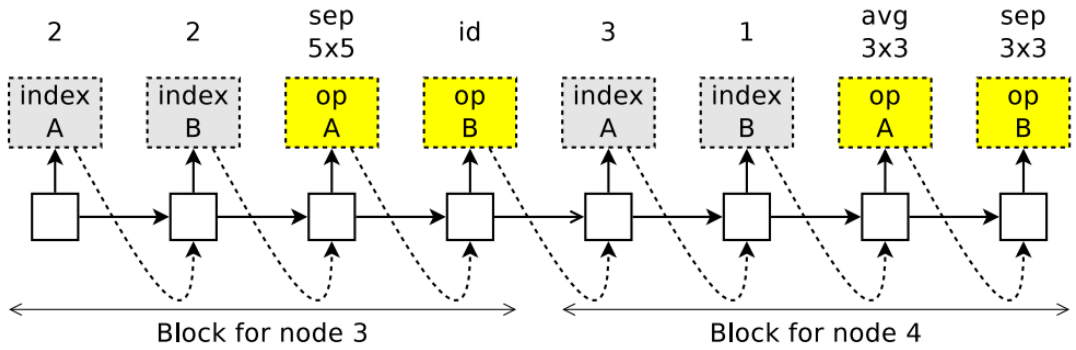
Step 2. Select a second hidden state from the same options as in Step 1.

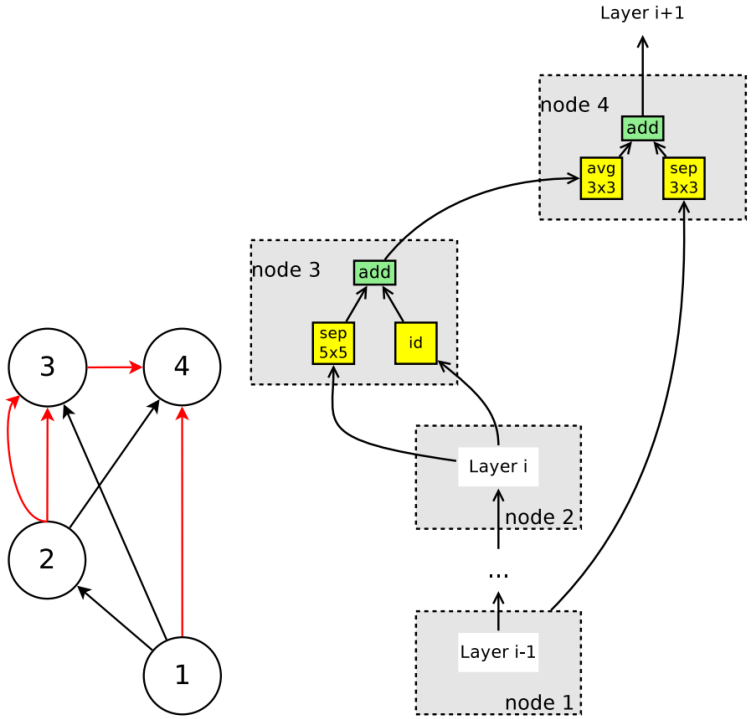
Step 3. Select an operation to apply to the hidden state selected in Step 1.

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.

CNN Space Example

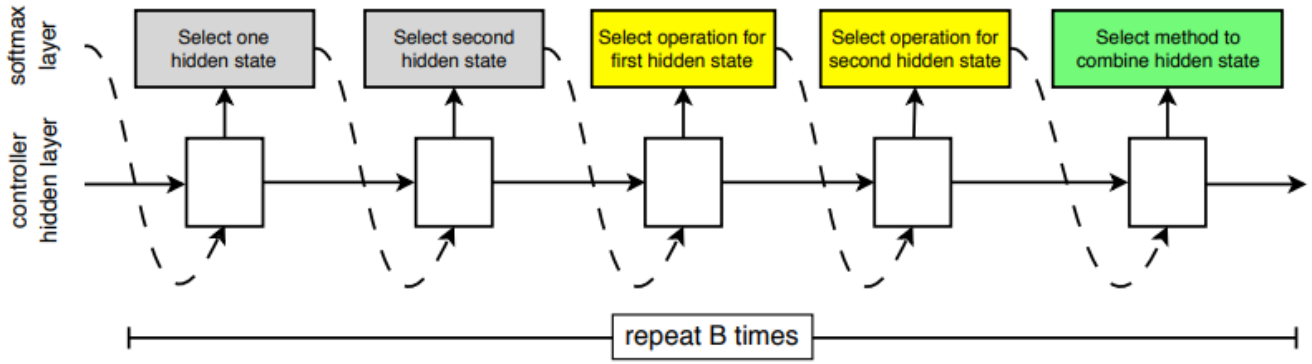


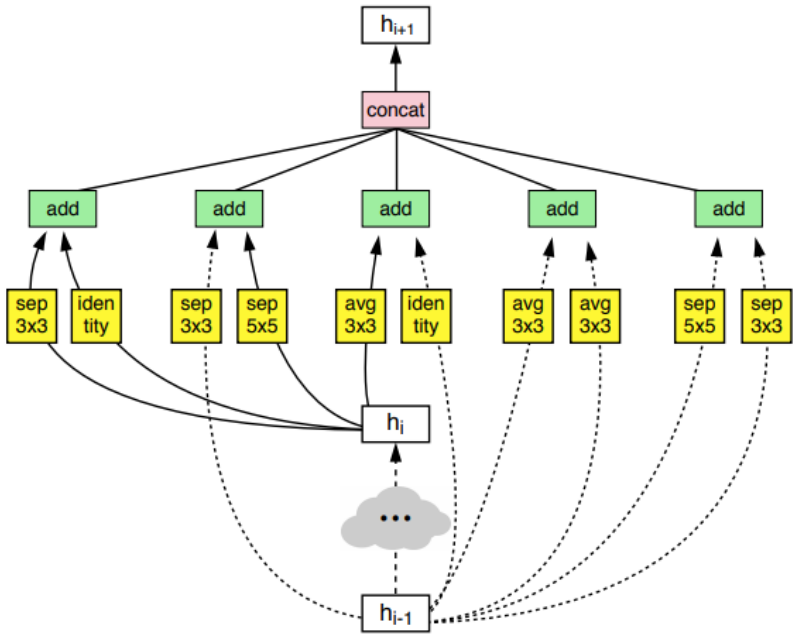


1. Nodes 1, 2 are input nodes, so no decisions are needed for them. Let h_1, h_2 be the outputs of these nodes.
2. At node 3: the controller samples two previous nodes and two operations. In this example, it samples node 2, node 2, separable conv 5x5, and identity. This means that $h_3 = \text{sep\_conv\_5x5}(\text{h}_2) + \text{identity}(\text{h}_2)$
3. At node 4: the controller samples node 3, node 1, avg pool 3x3, and sep conv 3x3. This means that $h_4 = avg\_pool\_3x3(h_3) + sep\_conv\_3x3(h_1)$
4. Since all nodes but h_4 were used as inputs to at least another node, the only loose end, h_4, is treated as the cell's output. If there are multiple loose ends, they will be concatenated along the depth dimension to form the cell's output.

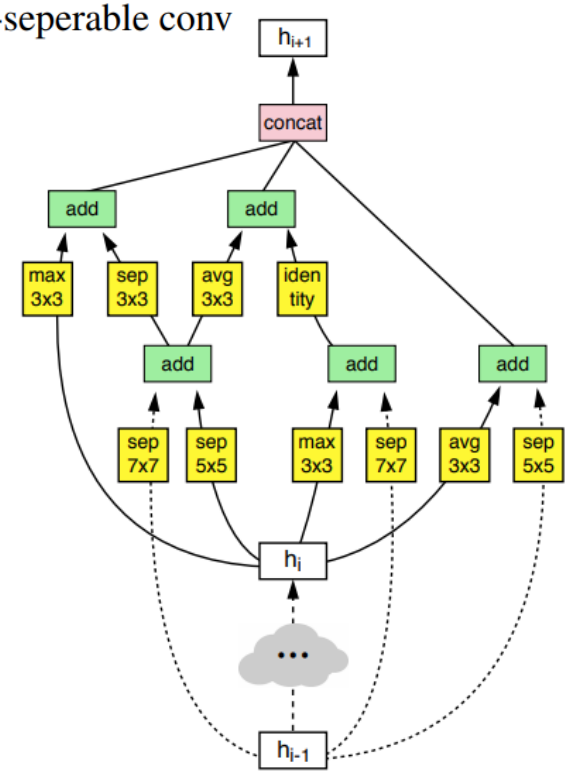
Searched Cell

- 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





Normal Cell



Reduction Cell

ScheduledDropPath, each path in the cell is dropped out with a probability that is linearly increased over the course of training

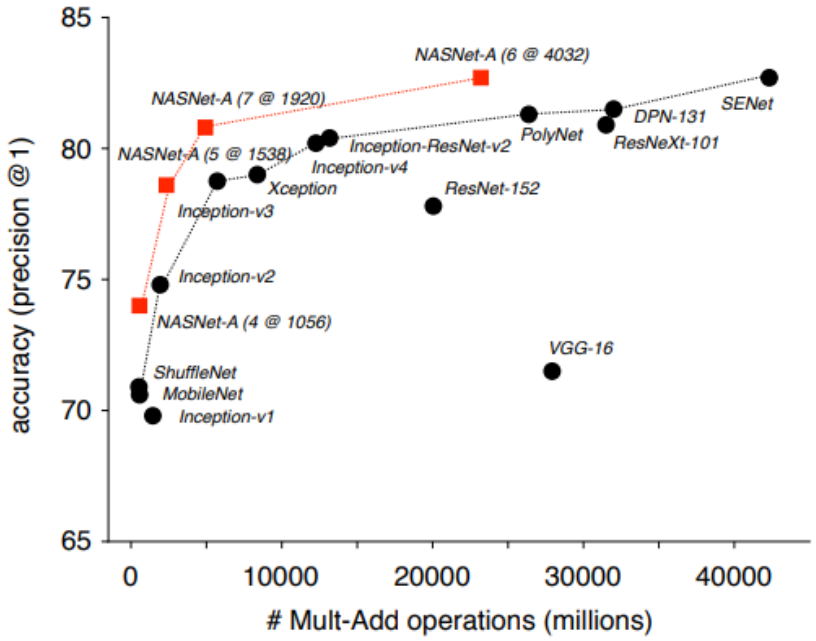
Results

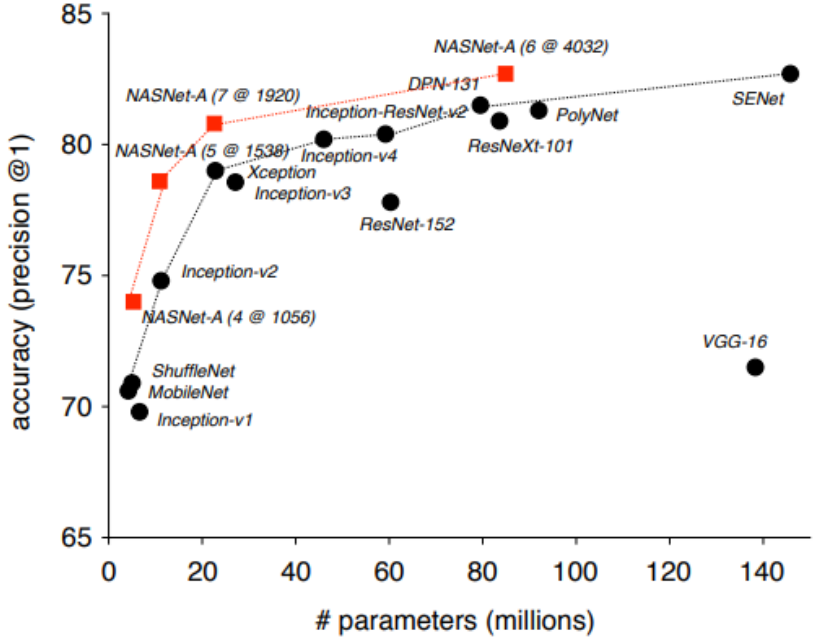
- 7x faster search than previous method

model	depth	# params	error rate (%)
DenseNet ($L = 40, k = 12$) [26]	40	1.0M	5.24
DenseNet($L = 100, k = 12$) [26]	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) [26]	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) [26]	190	25.6M	3.46
Shake-Shake 26 2x32d [18]	26	2.9M	3.55
Shake-Shake 26 2x96d [18]	26	26.2M	2.86
Shake-Shake 26 2x96d + cutout [12]	26	26.2M	2.56
NAS v3 [71]	39	7.1M	4.47
NAS v3 [71]	39	37.4M	3.65
NASNet-A (6 @ 768)	-	3.3M	3.41
NASNet-A (6 @ 768) + cutout	-	3.3M	2.65
NASNet-A (7 @ 2304)	-	27.6M	2.97
NASNet-A (7 @ 2304) + cutout	-	27.6M	2.40
NASNet-B (4 @ 1152)	-	2.6M	3.73
NASNet-C (4 @ 640)	-	3.1M	3.59

Results

- Transferred cell with conventional knowledge
  - stride 2 with channel doubling
  - A @ B means A cell repeat with B channels in the penultimate layer





Results

- Searched cell is superior to other vision tasks
  - Many of image applications share common features

Model	resolution	mAP (mini-val)	mAP (test-dev)
MobileNet-224 [24]	600 × 600	19.8%	-
ShuffleNet (2x) [70]	600 × 600	24.5%†	-
NASNet-A (4 @ 1056)	600 × 600	**29.6%**	-
ResNet-101-FPN [36]	800 (short side)	-	36.2%
Inception-ResNet-v2 (G-RMI) [28]	600 × 600	35.7%	35.6%
Inception-ResNet-v2 (TDM) [52]	600 × 1000	37.3%	36.8%
NASNet-A (6 @ 4032)	800 × 800	41.3%	40.7%
NASNet-A (6 @ 4032)	1200 × 1200	**43.2%**	**43.1%**
ResNet-101-FPN (RetinaNet) [37]	800 (short side)	-	39.1%

Progressive NAS

Simple to Complex

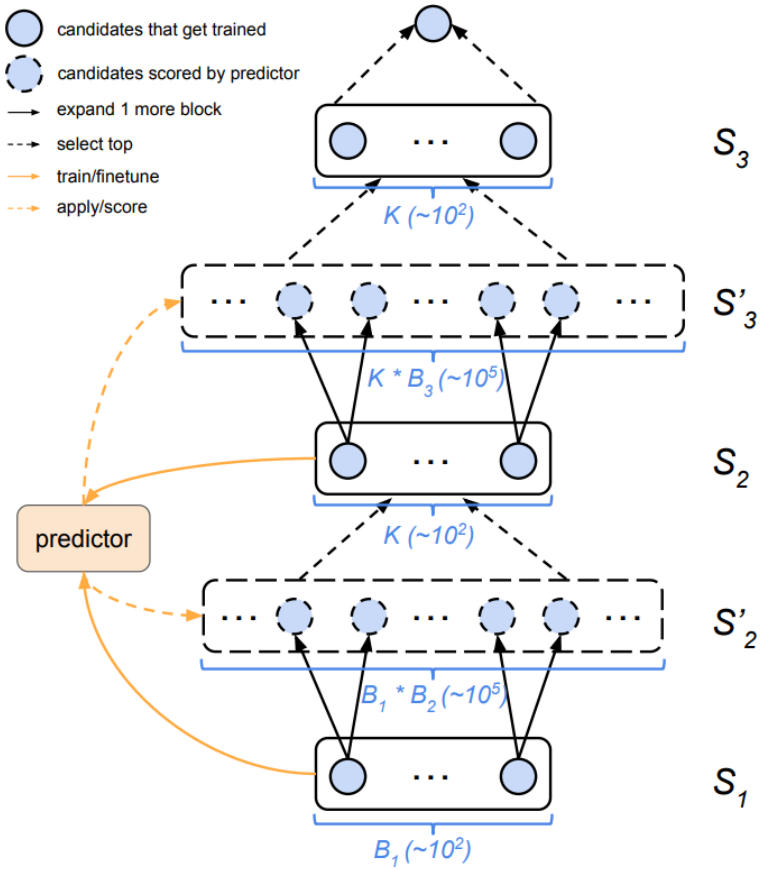
- NASNet uses a 50-step RNN as a controller
  - It is difficult to directly navigate in an exponentially large search space

- Search the space in a progressive order, simplest models first
- Try to estimate the performance based on performance prediction network
- Example
  - Start by constructing all possible cell structures and evaluate all the models in parallel
  - Expand each one by adding all of the possible block structures?
    - At every step, the candidates is increased exponentially
  - Instead of training all candidates, K most promising cells are selected based on **the performance prediction network**
- This idea will be extended to neural-predictor based method

Progressive NAS Algorithm

Algorithm 1 Progressive Neural Architecture Search (PNAS).

Inputs: B (max num blocks), E (max num epochs), F (num filters in first layer), K (beam size), N (num times to unroll cell), trainSet, valSet.
$\mathcal{S}_1 = \mathcal{B}_1$ // *Set of candidate structures with one block*
$\mathcal{M}_1 = \text{cell-to-CNN}(\mathcal{S}_1, N, F)$ // *Construct CNNs from cell specifications*
$\mathcal{C}_1 = \text{train-CNN}(\mathcal{M}_1, E, \text{trainSet})$ // *Train proxy CNNs*
$\mathcal{A}_1 = \text{eval-CNN}(\mathcal{C}_1, \text{valSet})$ // *Validation accuracies*
$\pi = \text{fit}(\mathcal{S}_1, \mathcal{A}_1)$ // *Train the reward predictor from scratch*
for $b = 2 : B$ **do**
  $\mathcal{S}'_b = \text{expand-cell}(\mathcal{S}_{b-1})$ // *Expand current candidate cells by one more block*
  $\hat{\mathcal{A}}'_b = \text{predict}(\mathcal{S}'_b, \pi)$ // *Predict accuracies using reward predictor*
  $\mathcal{S}_b = \text{top-K}(\mathcal{S}'_b, \hat{\mathcal{A}}'_b, K)$ // *Most promising cells according to prediction*
  $\mathcal{M}_b = \text{cell-to-CNN}(\mathcal{S}_b, N, F)$
  $\mathcal{C}_b = \text{train-CNN}(\mathcal{M}_b, E, \text{trainSet})$
  $\mathcal{A}_b = \text{eval-CNN}(\mathcal{C}_b, \text{valSet})$
  $\pi = \text{update-predictor}(\mathcal{S}_b, \mathcal{A}_b, \pi)$ // *Finetune reward predictor with new data*
end for
Return $\text{top-K}(\mathcal{S}_B, \mathcal{A}_B, 1)$



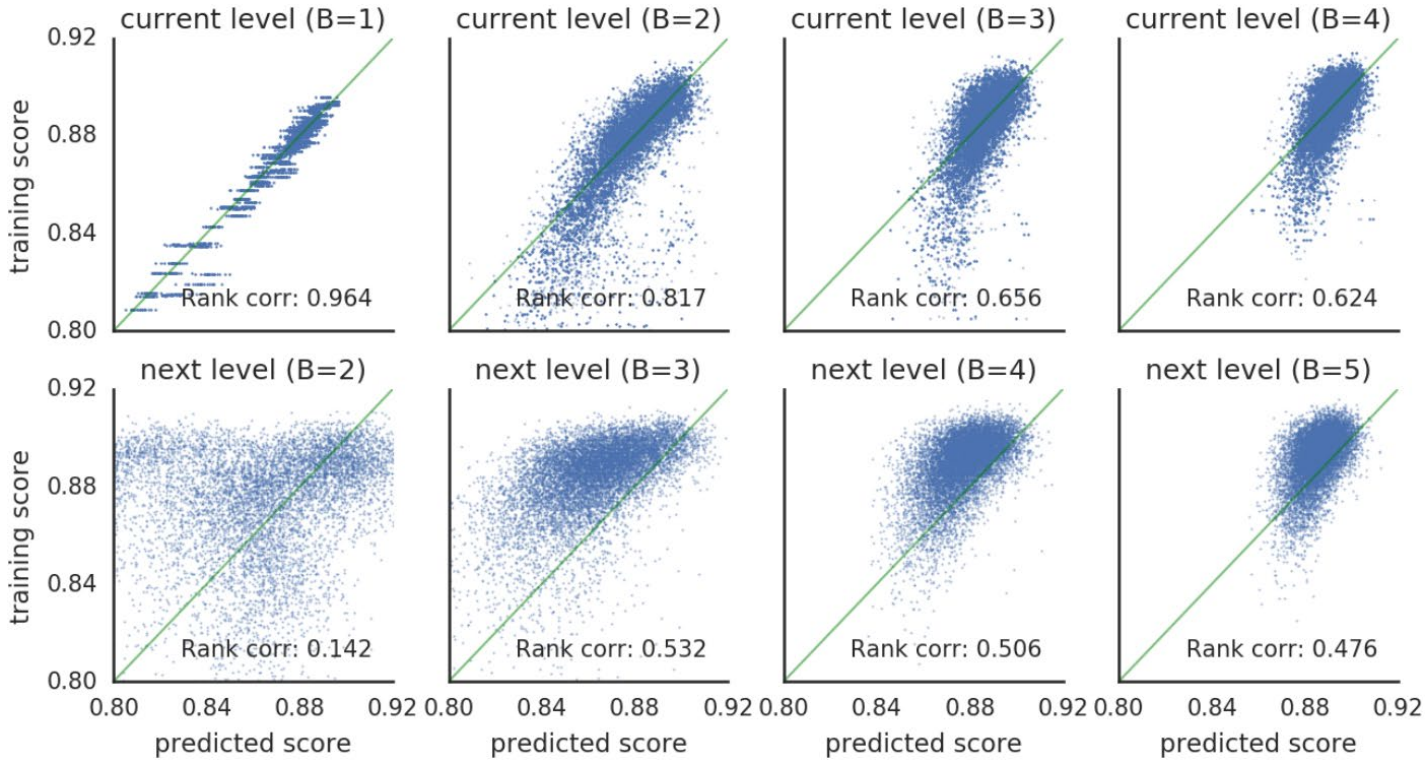
Performance Prediction with Surrogate Model

- Required properties
  - Handle variable-size inputs
    - predictor need to work for variable-length input strings
  - Correlated with true performance
    - The predictor need to rank models in roughly the same order as their true performance
  - Sample efficiency
    - We want to train and evaluate as few cells as possible

- Use LSTM and use sigmoid output to regress the validation accuracy
- 5-fold prediction model
  - Split sample into 5 sets, and train a predictor using 4/5 datasets
  - Ensemble to get the prediction value
    - Helpful to minimize the variance of the predictions

Results

- The predictor performs well on models from the training set, but not so well when predicting larger models
  - However, performance does increase as the predictor is trained on more (and larger) cells



Results

- PNAS is about 8 times faster than NAS when taking into account the total cost.

Model	B	N	F	Error	Params	M_1	E_1	M_2	E_2	Cost
NASNet-A [41]	5	6	32	3.41	3.3M	20000	0.9M	250	13.5M	21.4-29.3B
NASNet-B [41]	5	4	N/A	3.73	2.6M	20000	0.9M	250	13.5M	21.4-29.3B
NASNet-C [41]	5	4	N/A	3.59	3.1M	20000	0.9M	250	13.5M	21.4-29.3B
Hier-EA [21]	5	2	64	3.75±0.12	15.7M	7000	5.12M	0	0	35.8B[9]
AmoebaNet-B [27]	5	6	36	3.37±0.04	2.8M	27000	2.25M	100	27M	63.5B[10]
AmoebaNet-A [27]	5	6	36	3.34±0.06	3.2M	20000	1.13M	100	27M	25.2B[11]
PNASNet-5	5	3	48	3.41±0.09	3.2M	1160	0.9M	0	0	1.0B

Efficient Neural Architecture Search via Parameter Sharing

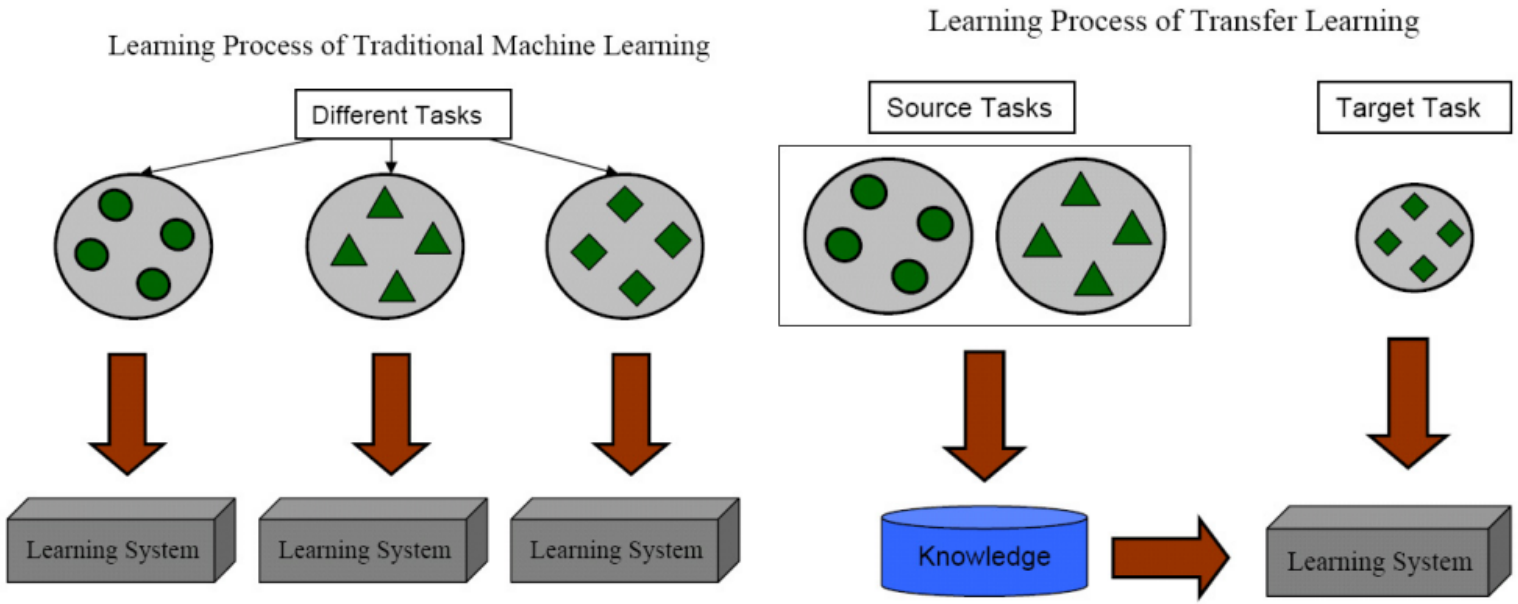
RL-based NAS is Expensive

- First NAS uses 800 GPUs with 28 days
  - ~540000 GPU hours
- With cell-based design, overall search space is reduced significantly
  - 7x speed up – still ~77000 GPU hours
- For popularization, we need to speed up the search process

Reuse Search Result?

- In transfer learning, pre-training is highly helpful to improve the quality of the network with smaller training iteration

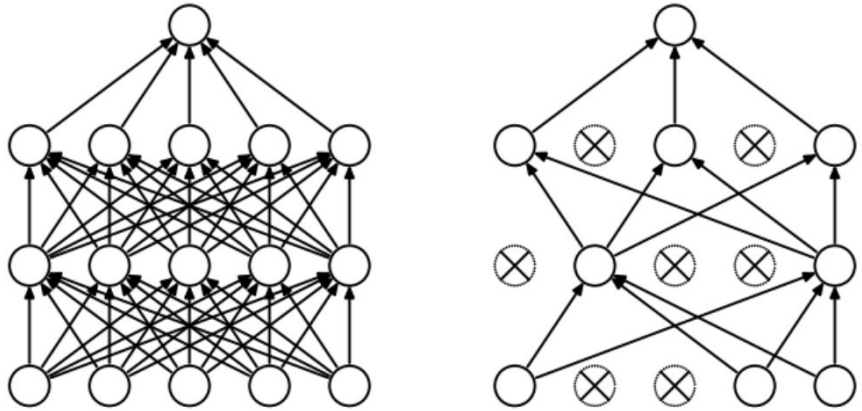
- After a single search process, we have a well-trained weights for the target task
  - Is there any way to exploit the pre-trained weight for the following search process?



Idea Sketch

- Hint 1:
  - In the view of transfer learning, if one of the evaluated network is similar to our newly sampled network, the pre-trained weight can be used as a good initialization point

- Hint 2:
  - Dropout samples subset of networks and train them iteratively
  - Not only the base network is trained, but also the subset networks are trained well



(a) Standard Neural Net (b) After applying dropout.

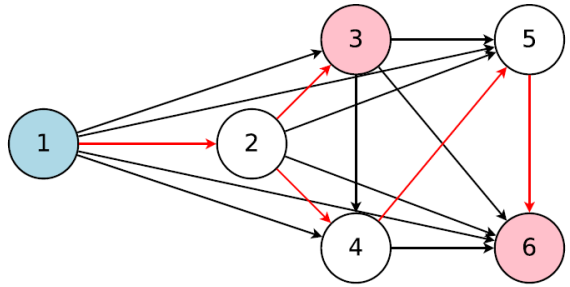
Parameter Sharing

- Prepare a gigantic network that can cover the entire sampled network
  - Every sampled networks can be a part of a gigantic network

- Instead of training each sampled network independently, jointly train the sampled networks
  - The sample networks can be seen as a coarse-grained dropout version of the gigantic network
  - All sampled networks share the same weight
  - By iteratively update the sampled network in parallel, each sampled network could be converged to a good optimal point.

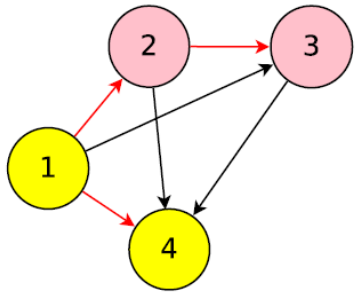
- We need large memory space to store the entire weights
  - Prepare all weights for the gigantic network
  - Tradeoff between storage overhead and computation cost

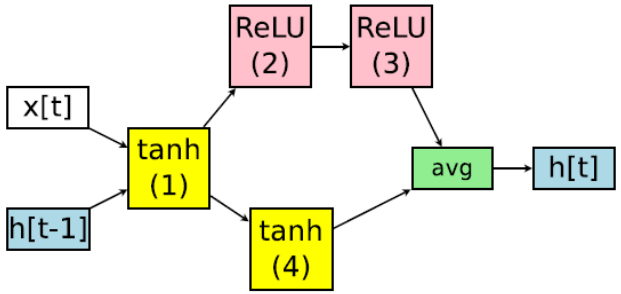
DAG Representation

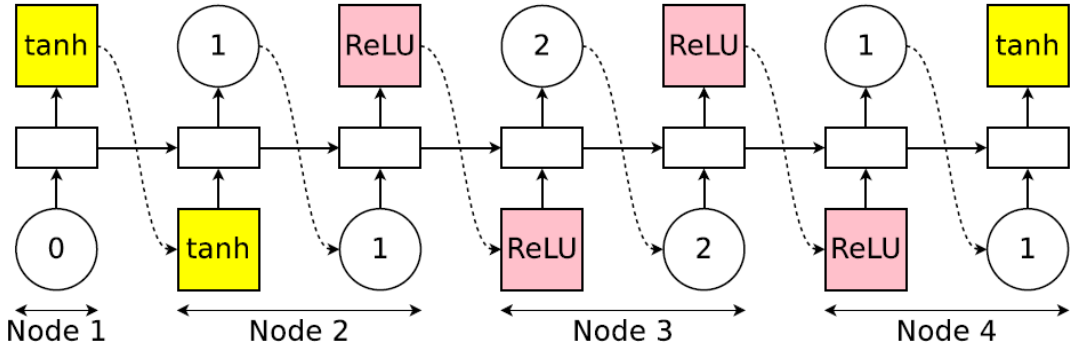


- Represent NAS's search space using a single directed acyclic graph (DAG).
  - Node i: i^{th} feature map
  - Edge i to j: set of operations from node i to node j.
    - computation operation, i.e. convolution, pooling, identity, …
  - DAG is the superposition of all possible child models in a search space of NAS
    - When we sample a network from the search space, it can be a subgraph in DAG

DAG Example







1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means $h_1 = \tanh\left(x_t \cdot W^x + h_{t-1} \cdot W_1^h\right)$
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, $h_2 = ReLU\left(h_1 \cdot W_{2,1}^h\right)$
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, $h_3 = ReLU\left(h_2 \cdot W_{3,2}^h\right)$
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 $h_4 = \tanh\left(h_1 \cdot W_{4,1}^h\right)$
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 other words, $h_t = \frac{h_3 + h_4}{2}$

Training Strategy

- Iterative update between the shared parameters w and the controller parameters θ

- 1. Update for the shared parameters w
  - Fix the controller's policy $\pi(m;\theta)$ and perform stochastic gradient descent (SGD) on w to minimize the expected loss function $E_{m\sim\pi}[L(m;\pi)]$
  - The gradient is computed using the Monte Carlo estimate

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

    - Surprisingly, $M = 1$ works just fine.
      - Sample a single model from $\pi(m;\theta)$ and update the weights

Training Strategy

- Iterative update between the shared parameters w and the controller parameters θ

- 2. Update the controller parameters θ
  - Fix w and update the policy parameter θ aiming to maximize the expected reward $E_{m\sim\pi(m;\theta)}[R(m,w)]$, the accuracy on the validation set

- Delivering Architectures
  - After the iterative update, sample several models from the trained policy $\pi(m;\theta)$
  - Compute its reward on a single minibatch sampled from the validation set.
  - Take only the model with the highest reward to re-train from scratch.

CNN Space Design

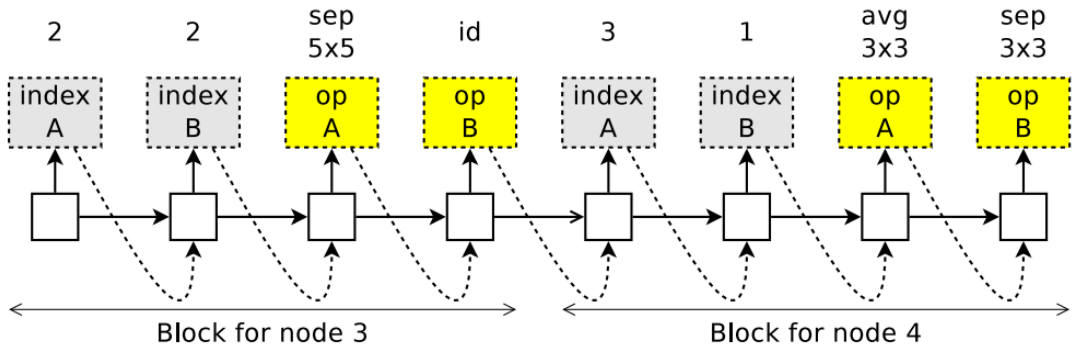
- Cell-based approach with 2 to 1 node design
  - Each cell has a strict restrictions but has fluent expression capability.

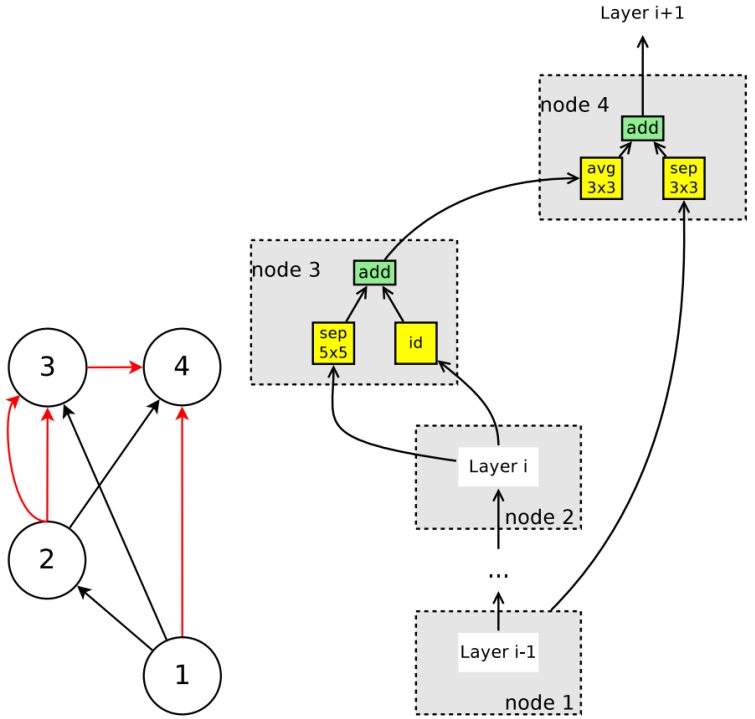
  - 1) two previous nodes to be used as inputs to the current node
  - 2) two operations to apply to the two sampled nodes.

  - Node 1 and 2 are the outputs of the two previous cells in the final network

  - The 5 available operations are: identity, separable convolution with kernel size 3 × 3 and 5 × 5, and average pooling and max pooling with kernel size 3×3.

CNN Space Example

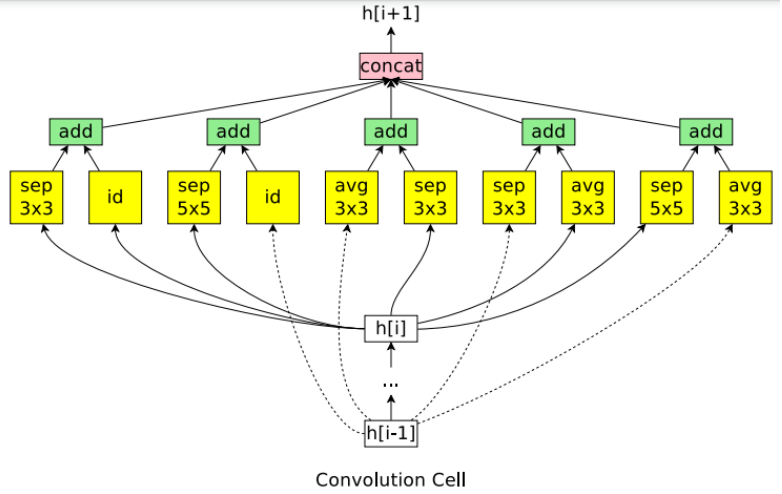




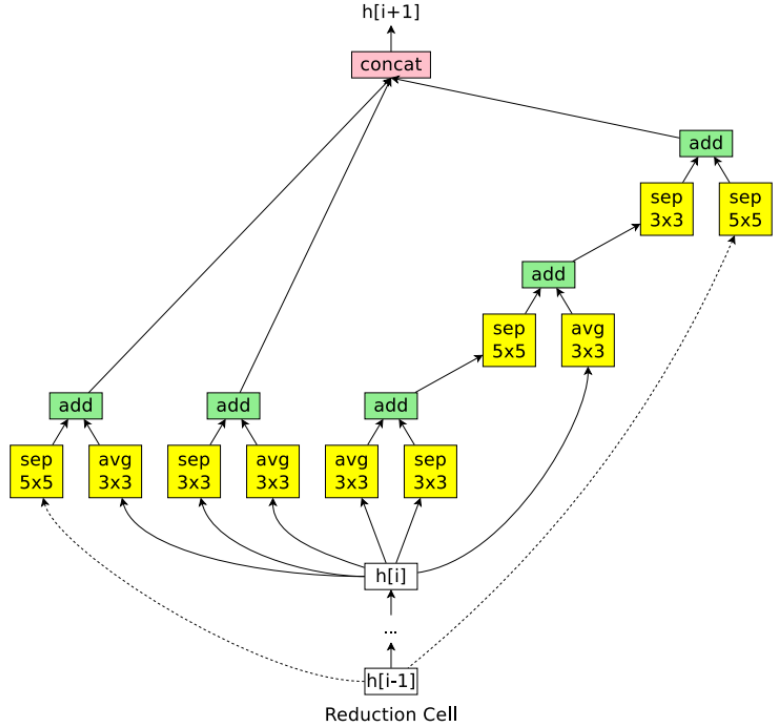
1. Nodes 1, 2 are input nodes, so no decisions are needed for them. Let h_1, h_2 be the outputs of these nodes.
2. At node 3: the controller samples two previous nodes and two operations. In this example, it samples node 2, node 2, separable conv 5x5, and identity. This means that $h_3 = \text{sep\_conv\_5x5}(\text{h}_2) + \text{identity}(\text{h}_2)$
3. At node 4: the controller samples node 3, node 1, avg pool 3x3, and sep conv 3x3. This means that $h_4 = avg\_pool\_3x3(h_3) + sep\_conv\_3x3(h_1)$
4. Since all nodes but h_4 were used as inputs to at least another node, the only loose end, h_4, is treated as the cell's output. If there are multiple loose ends, they will be concatenated along the depth dimension to form the cell's output.

Results

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**



Convolution Cell



Reduction Cell

Table 2. Classification errors of ENAS and baselines on CIFAR-10. In this table, the first block presents DenseNet, one of the state-of-the-art architectures designed by human experts. The second block presents approaches that design the entire network. The last block presents techniques that design modular cells which are combined to build the final network.