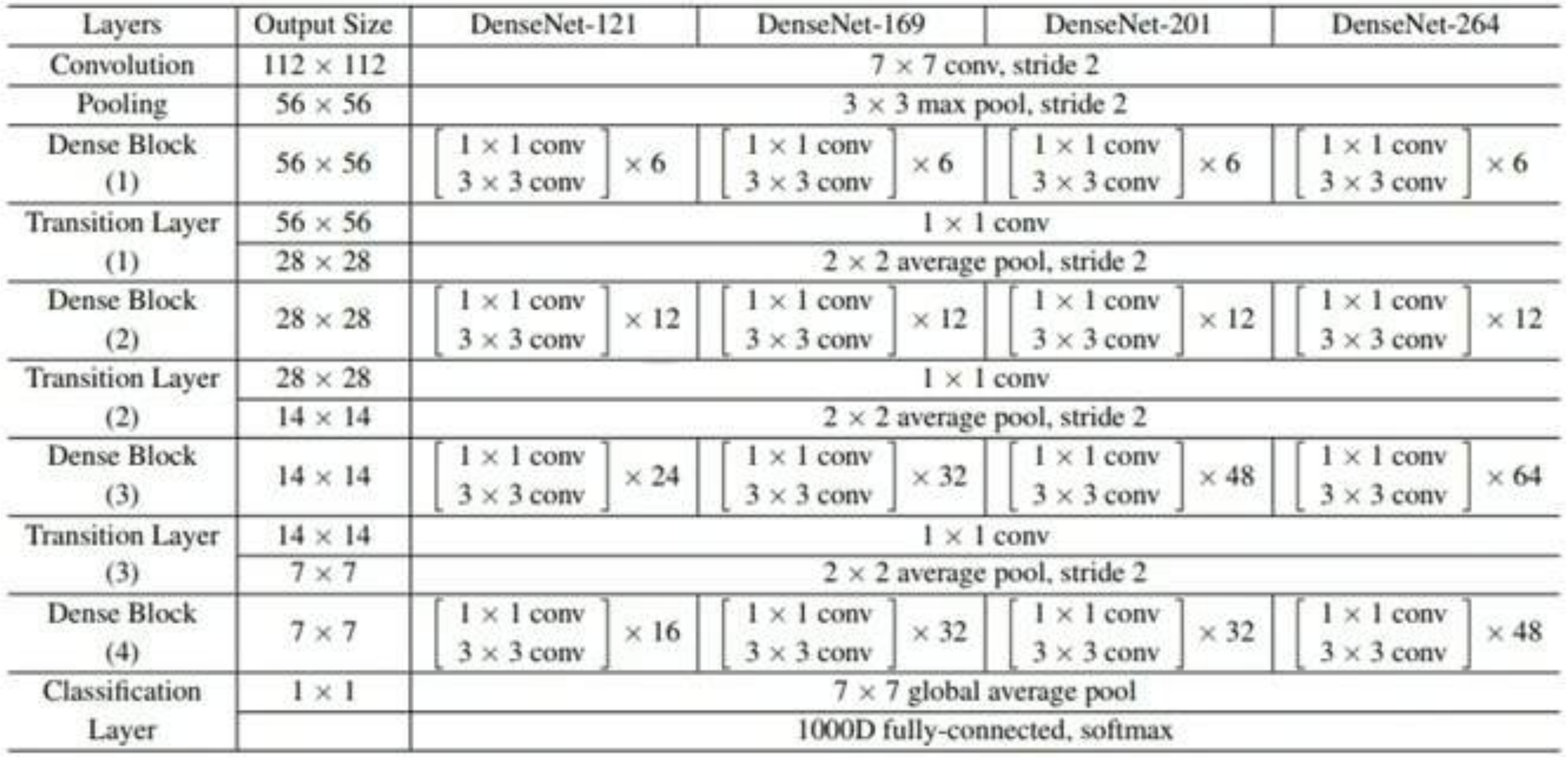
Neural Architecture Search: State-of-the-art Overview

Debadeepta Dey

Microsoft Research AI

How do you come up with this?



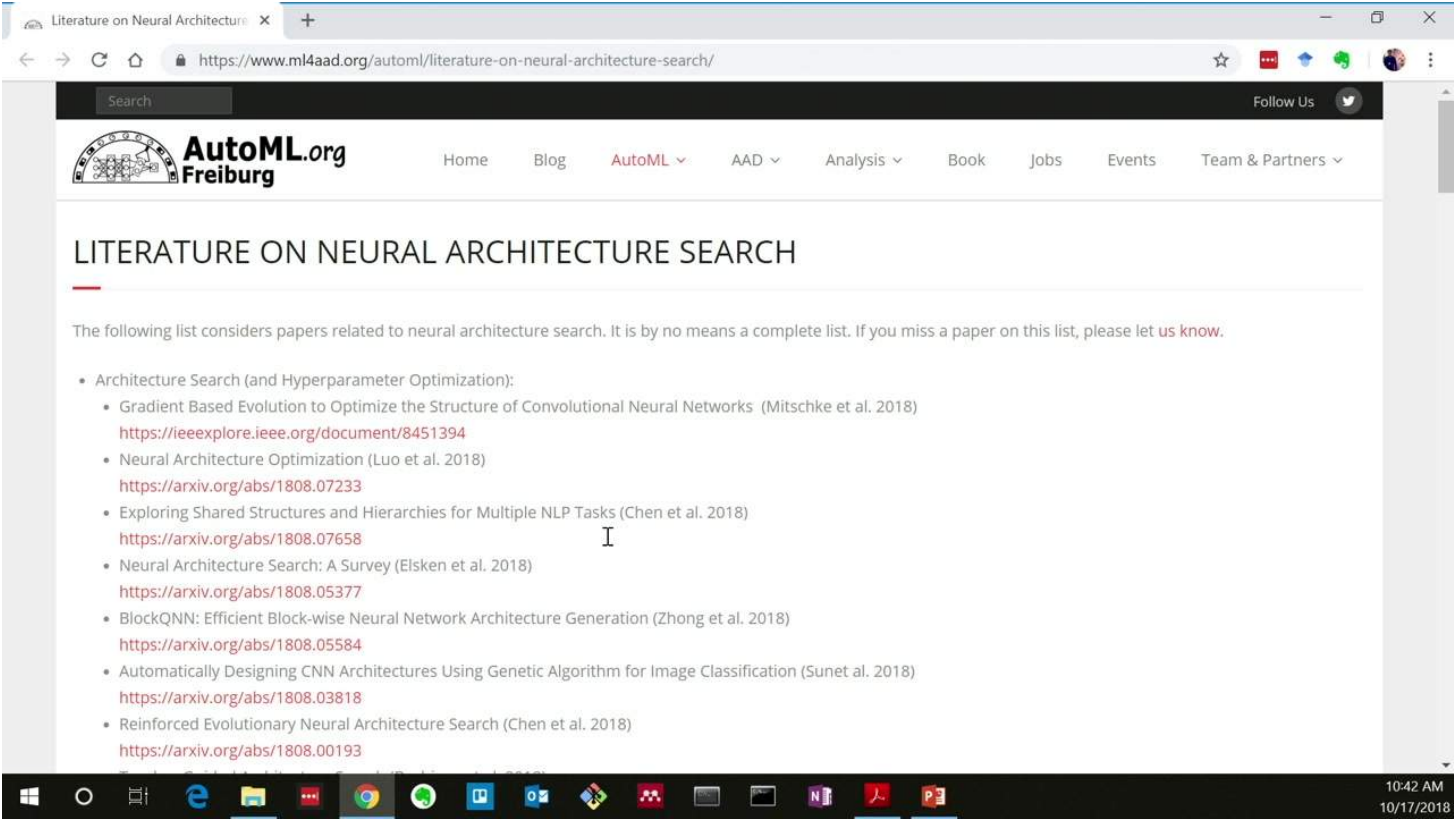
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

"I have no idea how to come up with this!" – John Langford, April 2018

Automatic Architecture Hunt (AutoML)

- Continuously updated list of NAS papers:
  - https://www.ml4aad.org/automl/literature-on-neural-architecture-search/
- Excellent survey article:
  - Neural Architecture Search: A Survey, Elsken, Metzen and Hutter, 2018



LITERATURE ON NEURAL ARCHITECTURE SEARCH

The following list considers papers related to neural architecture search. It is by no means a complete list. If you miss a paper on this list, please let us know.

- Architecture Search (and Hyperparameter Optimization):
  - Gradient Based Evolution to Optimize the Structure of Convolutional Neural Networks (Mitschke et al. 2018)
    https://ieeexplore.ieee.org/document/8451394
  - Neural Architecture Optimization (Luo et al. 2018)
    https://arxiv.org/abs/1808.07233
  - Exploring Shared Structures and Hierarchies for Multiple NLP Tasks (Chen et al. 2018)
    https://arxiv.org/abs/1808.07658
  - Neural Architecture Search: A Survey (Elsken et al. 2018)
    https://arxiv.org/abs/1808.05377
  - BlockQNN: Efficient Block-wise Neural Network Architecture Generation (Zhong et al. 2018)
    https://arxiv.org/abs/1808.05584
  - Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification (Sunet al. 2018)
    https://arxiv.org/abs/1808.03818
  - Reinforced Evolutionary Neural Architecture Search (Chen et al. 2018)
    https://arxiv.org/abs/1808.00193

Automatic Architecture Hunt (AutoML)

- Continuously updated list of NAS papers:
  - https://www.ml4aad.org/automl/literature-on-neural-architecture-search/
- Excellent survey article:
  - Neural Architecture Search: A Survey, Elsken, Metzen and Hutter, 2018

Before we dive in

- Extremely quick primer on RL
  - MDP, POMDP, HMM, Markov Chain
  - What makes a problem a RL problem?
  - Model-based vs. Model-Free RL methods.
  - Off-policy vs. On-policy.
  - Contextual Bandits (special case of RL).
  - Policy Gradients (a deeper dive).

Partially Observable Markov Decision Processes (POMDPs)

Geoff Hollinger

Graduate Artificial Intelligence

Fall, 2007

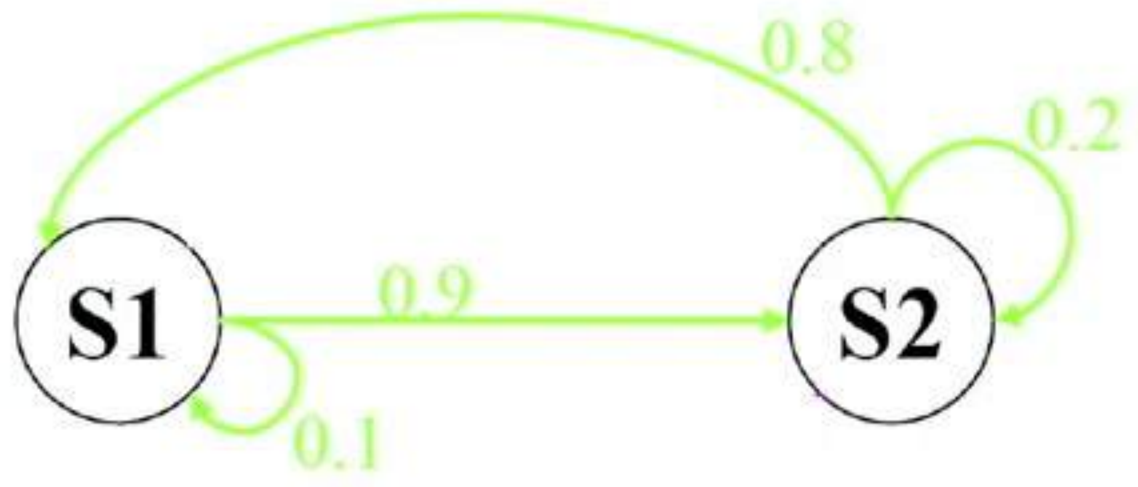
*Some media from Reid Simmons, Trey Smith, Tony Cassandra, Michael Littman, and Leslie Kaelbling

So who is this Markov guy?

- Andrey Andreyevich Markov (1856-1922)
- Russian mathematician
- Known for his work in stochastic processes
  - Later known as Markov Chains

What is a Markov Chain?

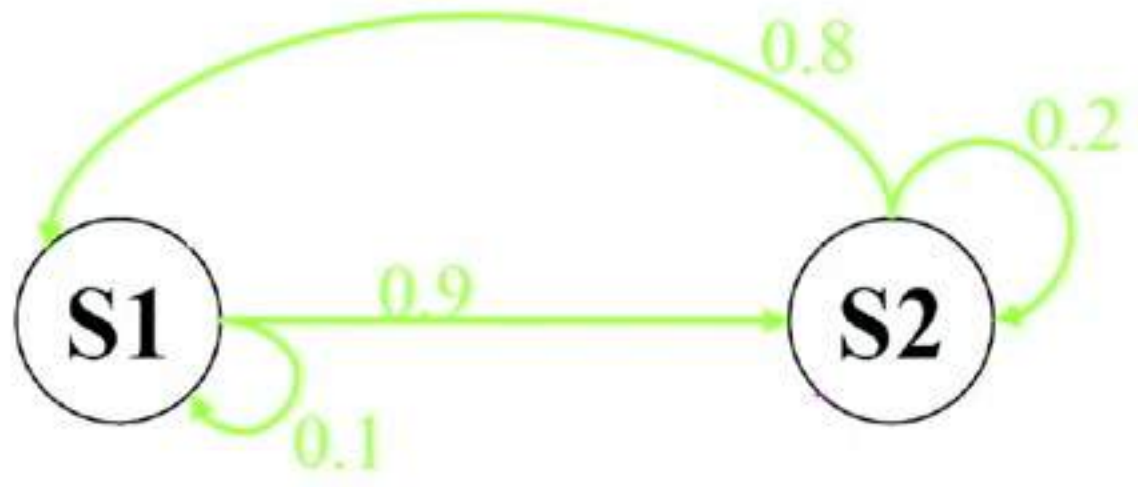
- Finite number of discrete states
- Probabilistic transitions between states
- Next state determined only by the current state
  - This is the Markov property



Rewards: S1 = 10, S2 = 0

What is a Hidden Markov Model?

- Finite number of discrete states
- Probabilistic transitions between states
- Next state determined only by the current state
- We're unsure which state we're in
  - The current states emits an observation

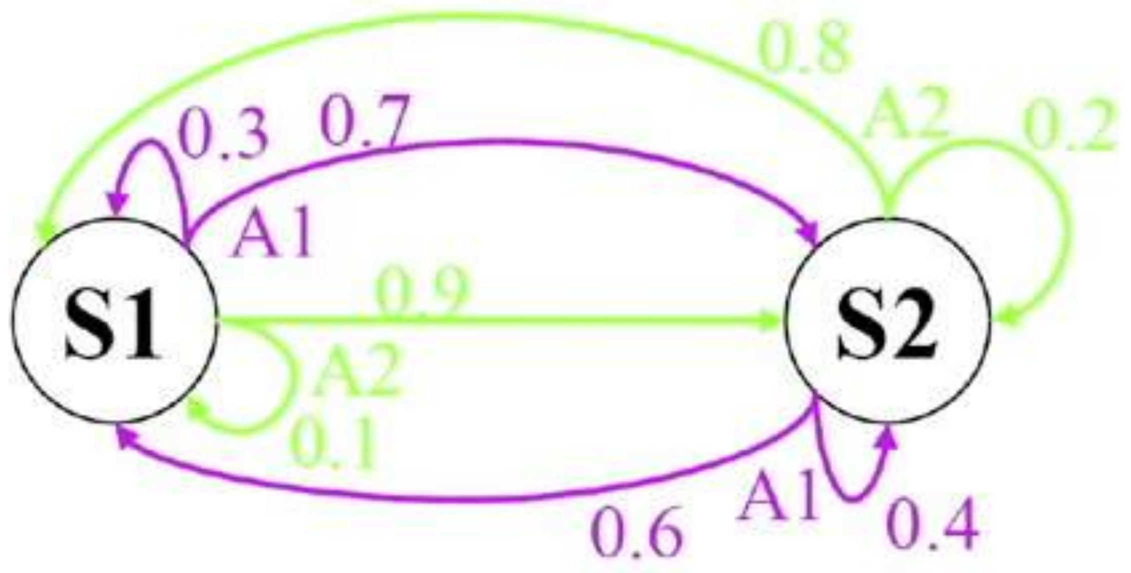


Rewards: S1 = 10, S2 = 0

Do not know state:
S1 emits O1 with prob 0.75
S2 emits O2 with prob 0.75

What is a Markov Decision Process?

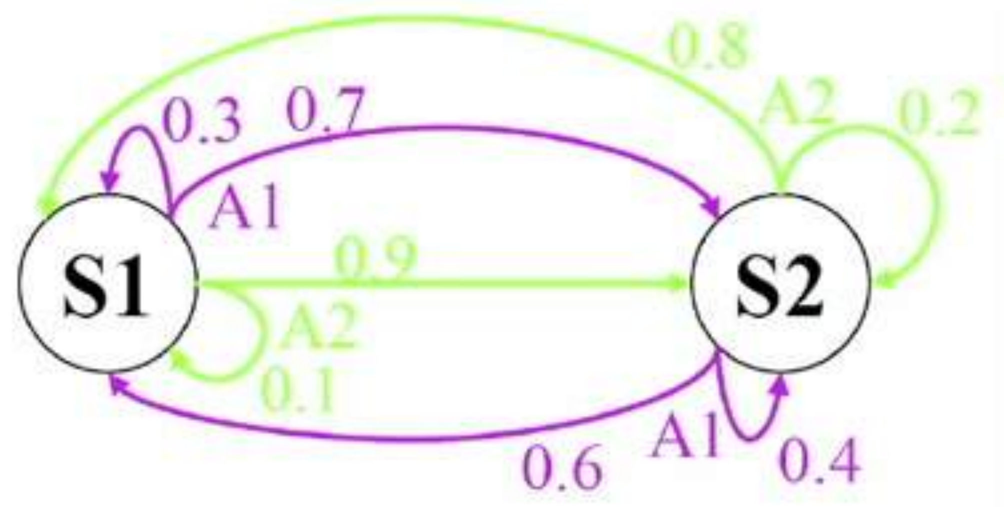
- Finite number of discrete states
- Probabilistic transitions between states ***and*** controllable actions in each state
- Next state determined only by the current state ***and*** current action
  - This is still the Markov property



Rewards: S1 = 10, S2 = 0

What is a Partially Observable Markov Decision Process?

- Finite number of discrete states
- Probabilistic transitions between states and controllable actions
- Next state determined only by the current state and current action
- We're unsure which state we're in
  - The current state emits observations



Rewards: S1 = 10, S2 = 0

Do not know state:
S1 emits O1 with prob 0.75
S2 emits O2 with prob 0.75

A Very Helpful Chart

Markov Models		Do we have control over the state transitons?	
		NO	YES
Are the states completely observable?	YES	**Markov Chain**	**MDP** Markov Decision Process
	NO	**HMM** Hidden Markov Model	**POMDP** Partially Observable Markov Decision Process

Before we dive in

- Extremely quick primer on RL
    - MDP, POMDP, HMM, Markov Chain
    - What makes a problem a RL problem?
    - Model-based vs. Model-Free RL methods.
    - Off-policy vs. On-policy.
    - Contextual Bandits (special case of RL).
    - Policy Gradients (a deeper dive).

Study material

- Introduction to Reinforcement Learning by Sutton and Barto, 2nd edition (legal free pdf available online).
- David Silver’s RL class (video and lectures available online)
- Contextual Bandits Tutorial by Alekh Agarwal and John Langford (online)
- Prediction, Learning and Games by Nicolo Cesa Bianchi et al (legal free pdf available online)

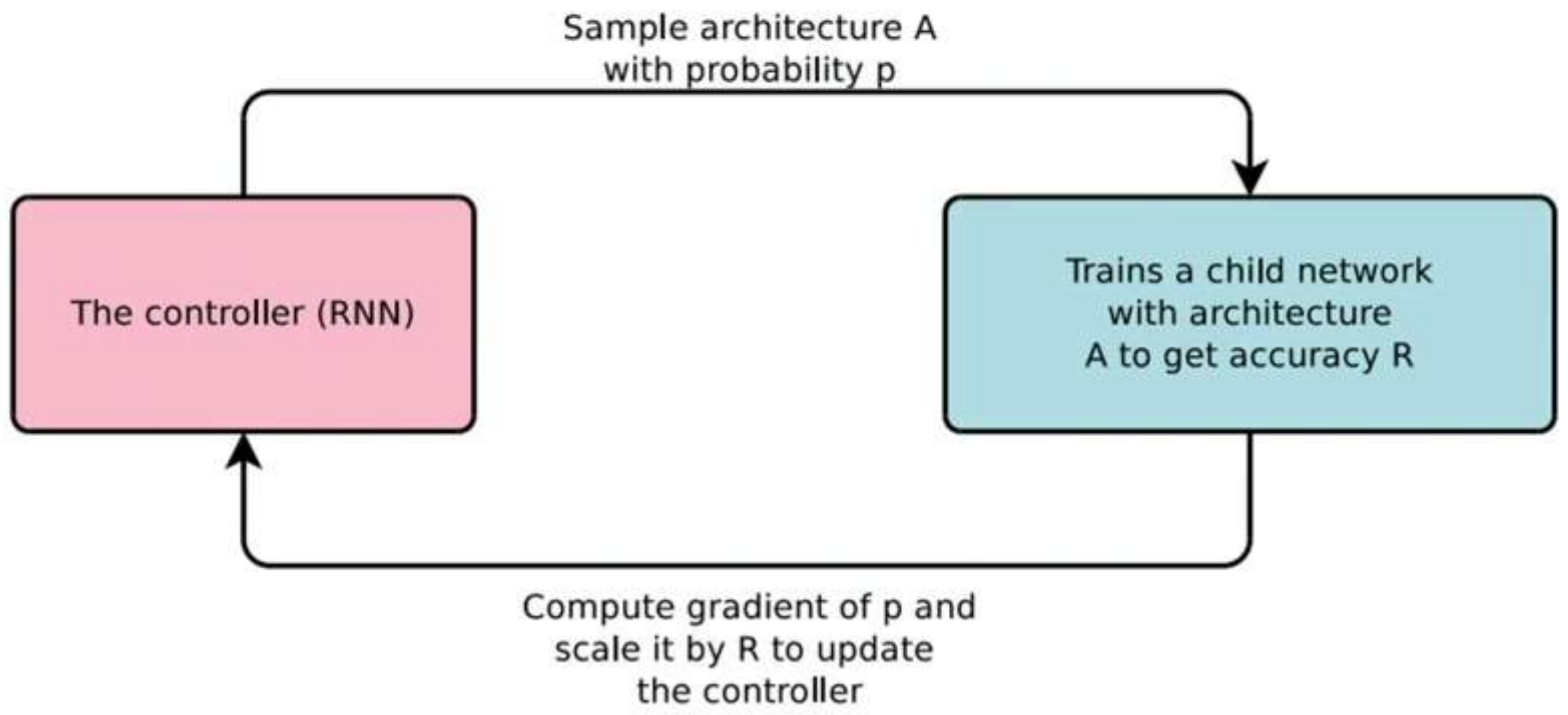


Figure 1: An overview of Neural Architecture Search.

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 $J(\theta_c)$:

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

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 Williams (1992):

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

An empirical approximation of the above quantity is:

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

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.

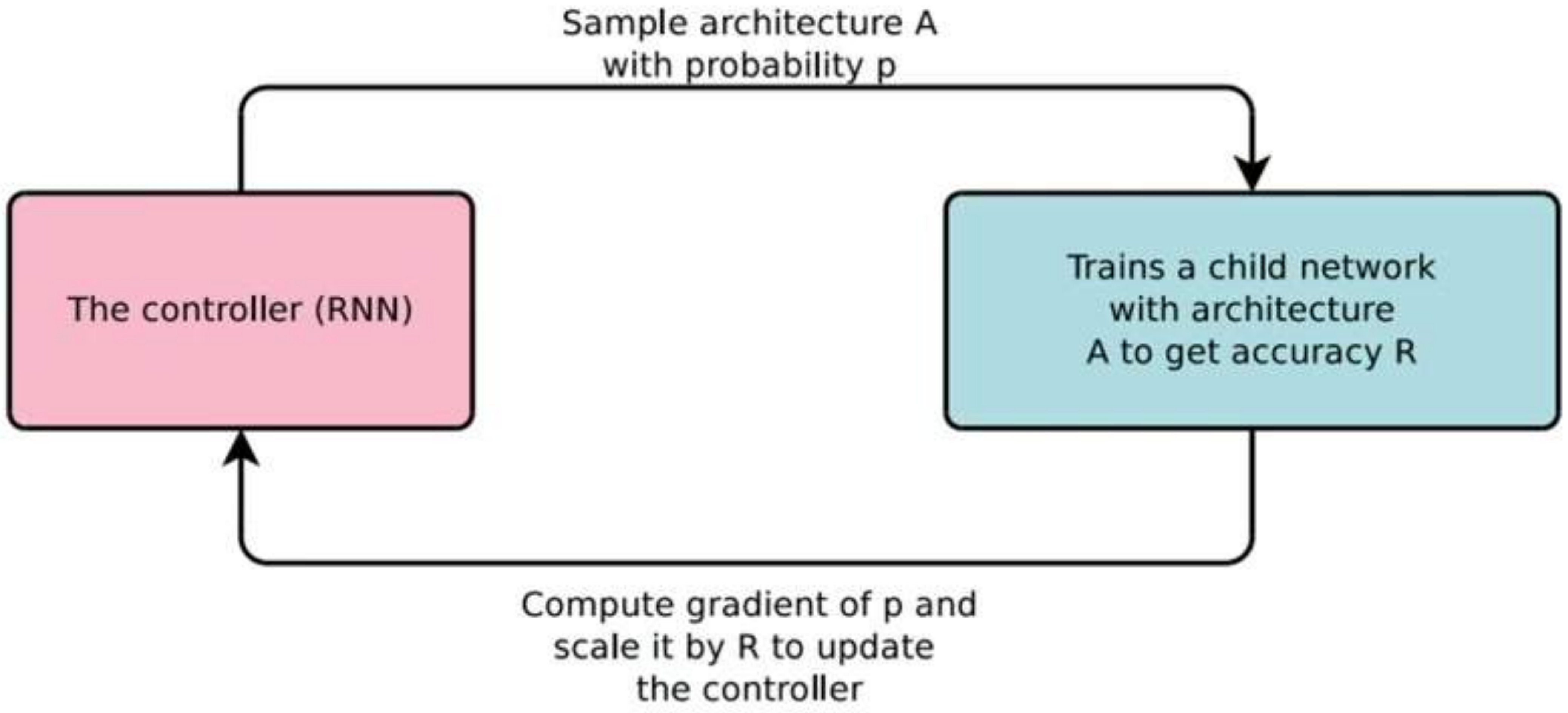


Figure 1: An overview of Neural Architecture Search.

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 $J(\theta_c)$:

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

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 Williams (1992):

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

An empirical approximation of the above quantity is:

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

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.

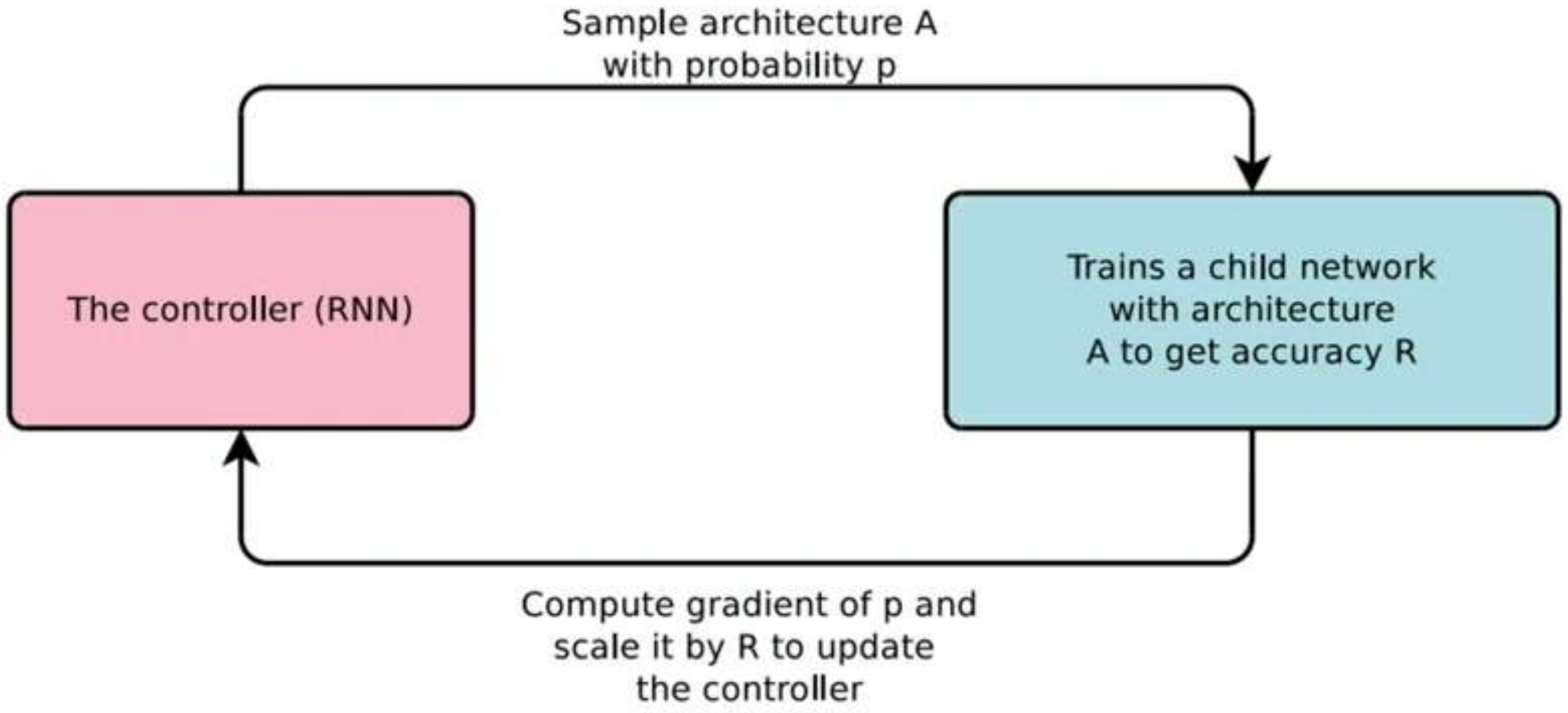


Figure 1: An overview of Neural Architecture Search.

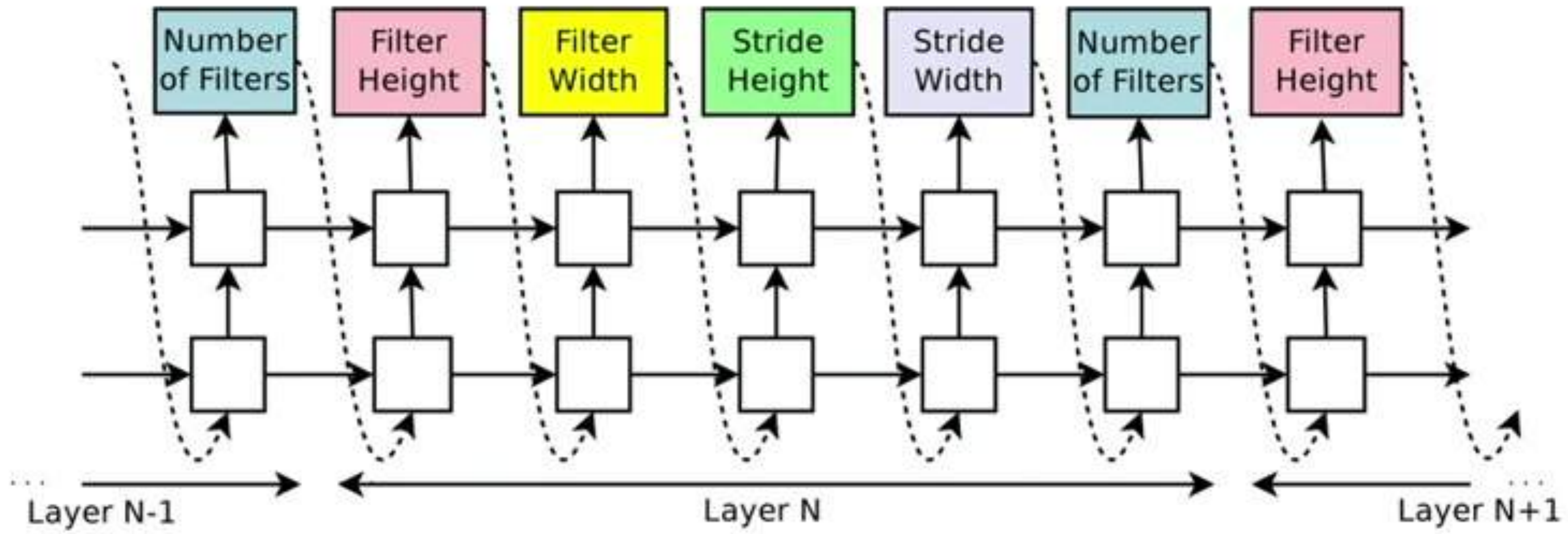


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

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 $J(\theta_c)$:

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

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 Williams (1992):

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

An empirical approximation of the above quantity is:

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

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.

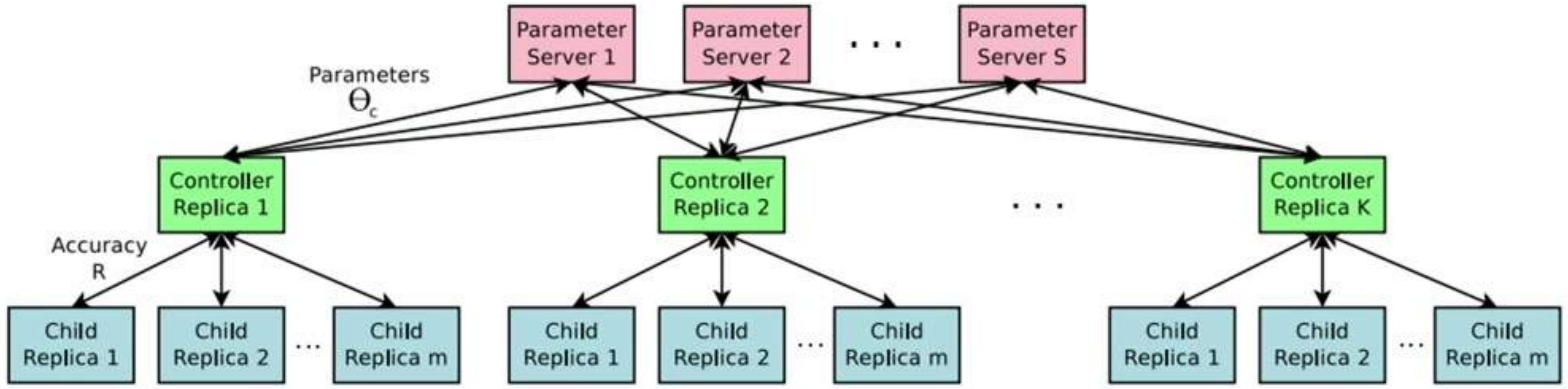


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to compute the gradients with respect to θ_c, which are then sent back to the parameter servers.

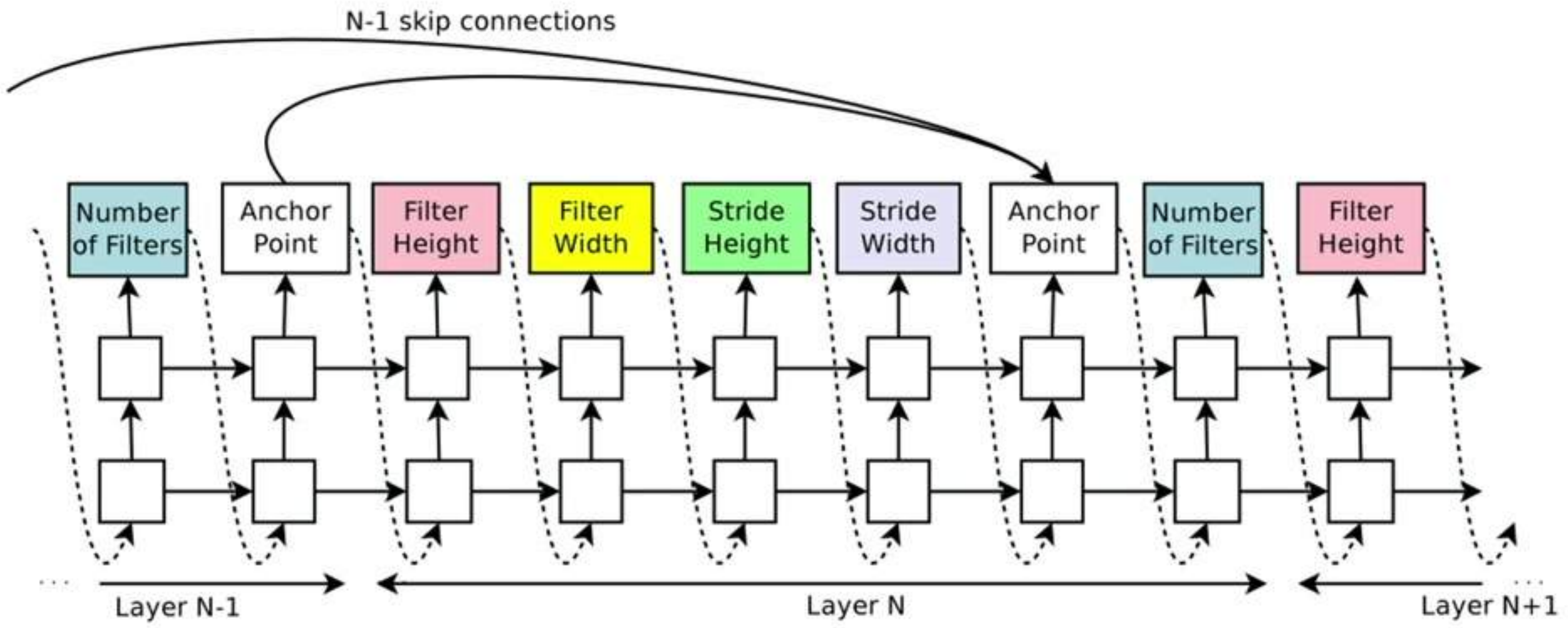


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

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

Dealing with compilation failures

- Input skip connections are concatenated along the depth dimension.
- If a layer is not connected to any layer then directly use the image as input.
- At the final layer take all layer outputs which have no connections and concatenate before sending out to the classifier.
- If input layers have smaller size then pad with zeros to make concatenation go through.

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

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

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

Table 2: Single model perplexity on the test set of the Penn Treebank language modeling task. Parameter numbers with ‡ are estimates with reference to Merity et al. (2016).

Efficient Neural Architecture Search via Parameter Sharing (Pham et al, 2018)

ENAS

On CIFAR-10 achieves a test error of 2.89%

On Penn Treebank achieves perplexity of 55.8 (NAS had 62.4)

In all experiments used a single Nvidia 1080Ti GPU

Search takes less than 16 hours

Compared to NAS >1000x reduction in search time

The main contribution of this work is to improve the efficiency of NAS by forcing all child models to share weights to eschew training each child model from scratch to convergence.

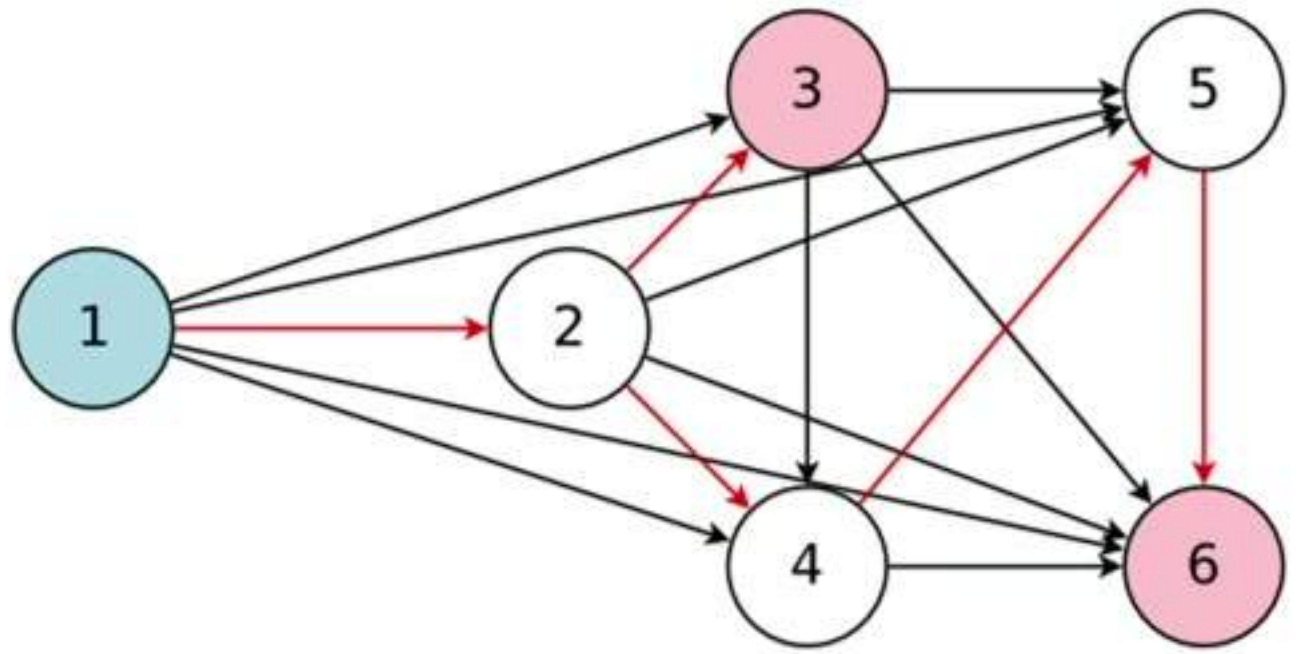


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

Recurrent cell sampling

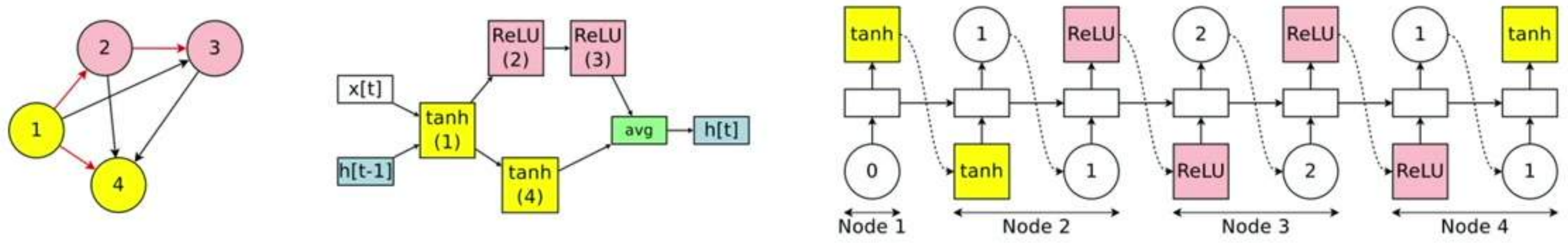


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell's output.

Convolutional cell sampling

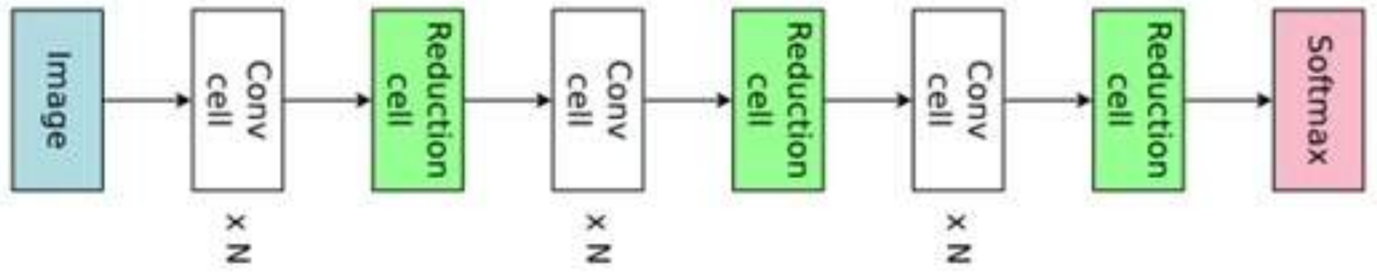


Figure 4. Connecting 3 blocks, each with N convolution cells and 1 reduction cell, to make the final network.

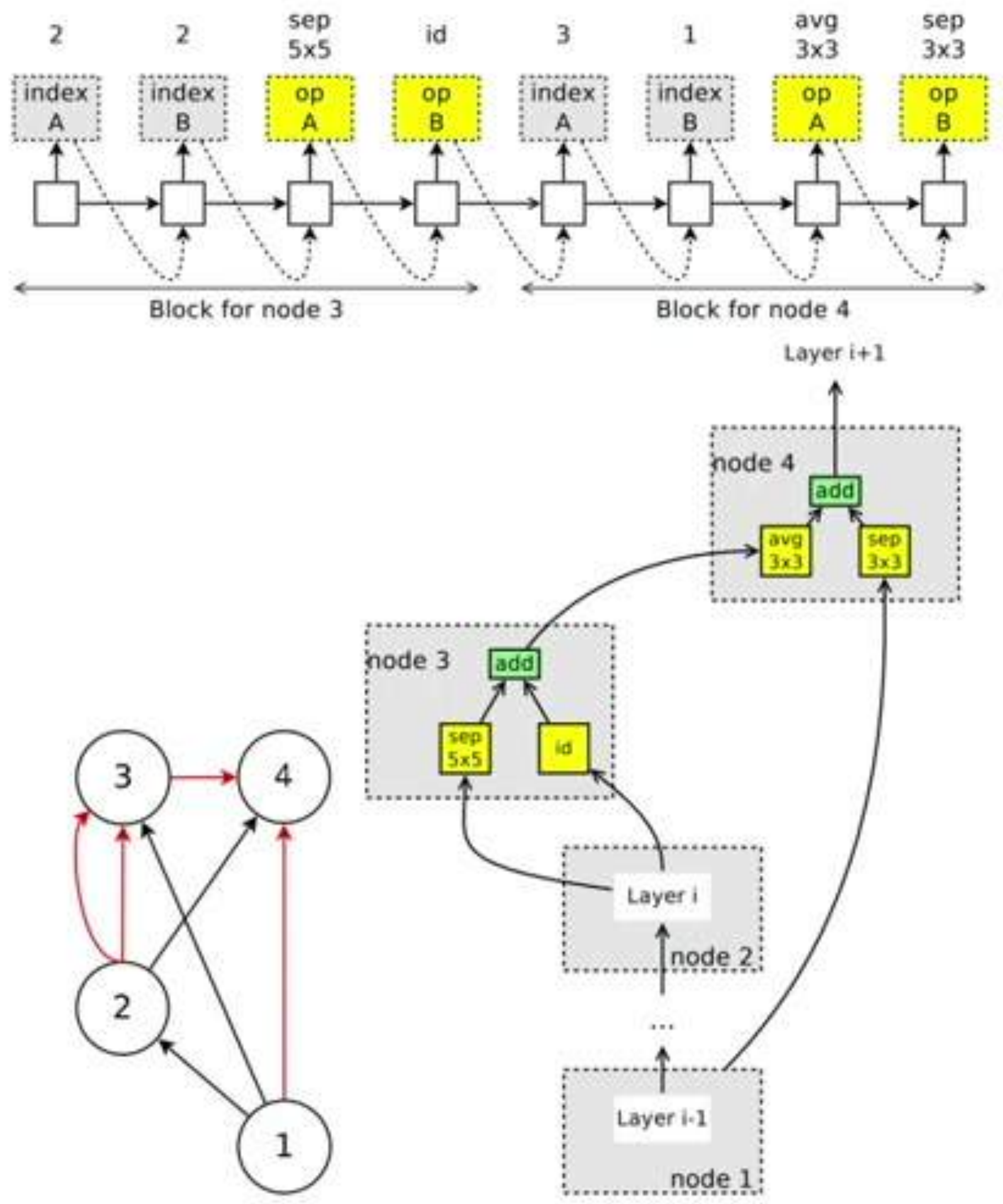


Figure 5. An example run of the controller for our search space over convolutional cells. *Top:* the controller's outputs. In our search space for convolutional cells, node 1 and node 2 are the cell's inputs, so the controller only has to design node 3 and node 4. *Bottom Left:* The corresponding DAG, where red edges represent the activated connections. *Bottom Right:* the convolutional cell according to the controller's sample.

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**

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.

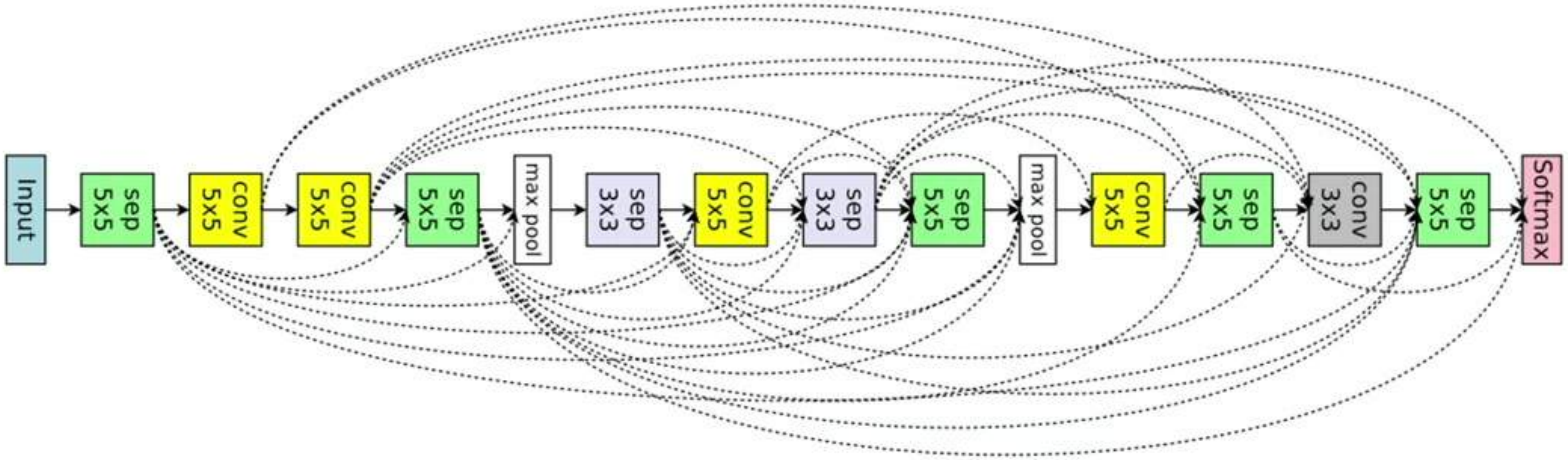
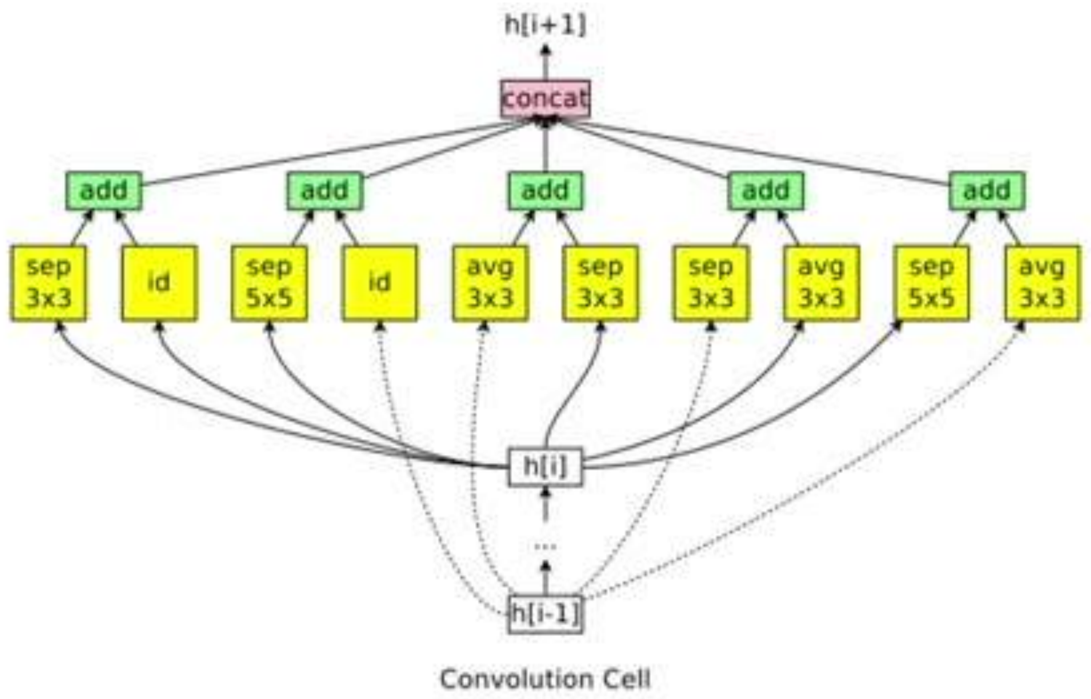


Figure 7. ENAS's discovered network from the macro search space for image classification.



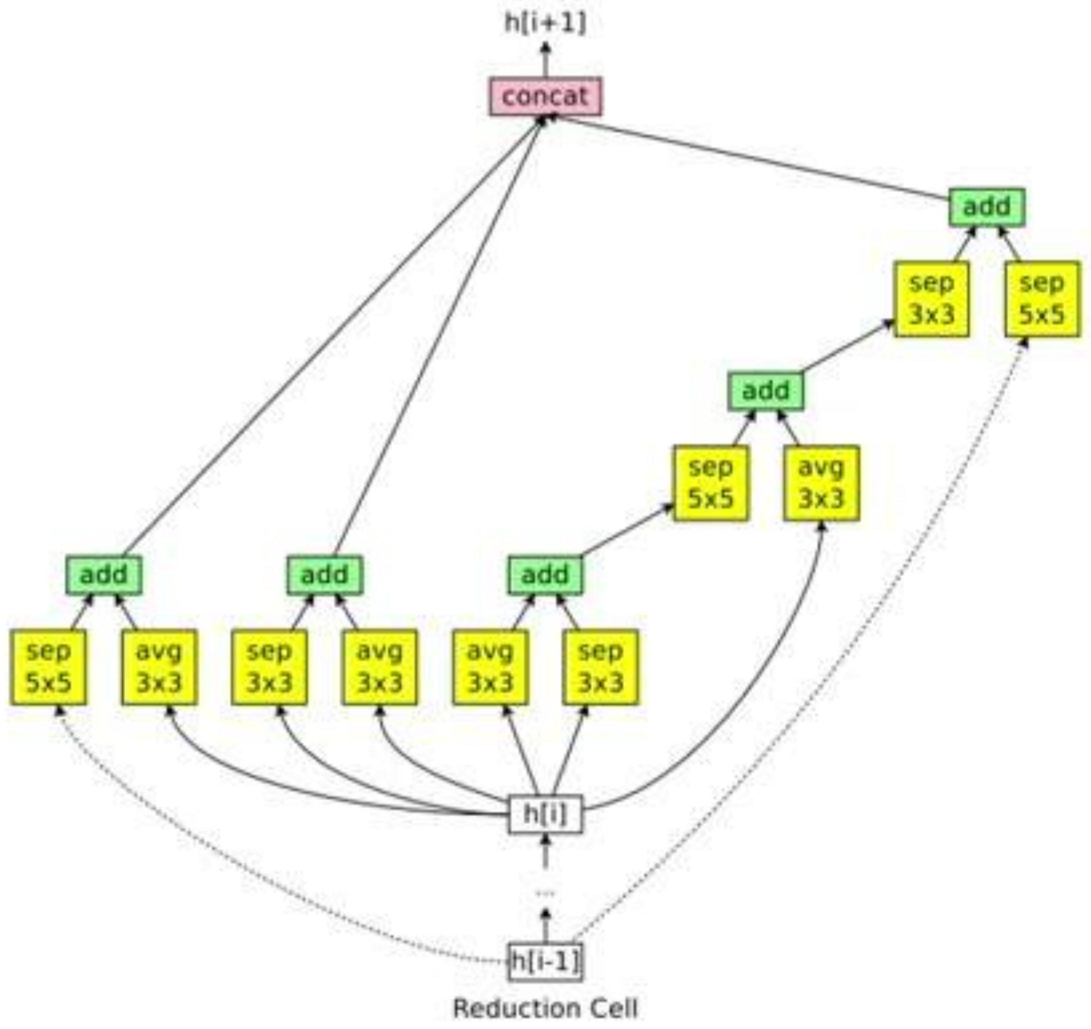


Figure 8. ENAS cells discovered in the micro search space.

Progressive Neural Architecture Search (Liu et al. 2018)

Progressive Neural Architecture Search (Liu et al. 2018)

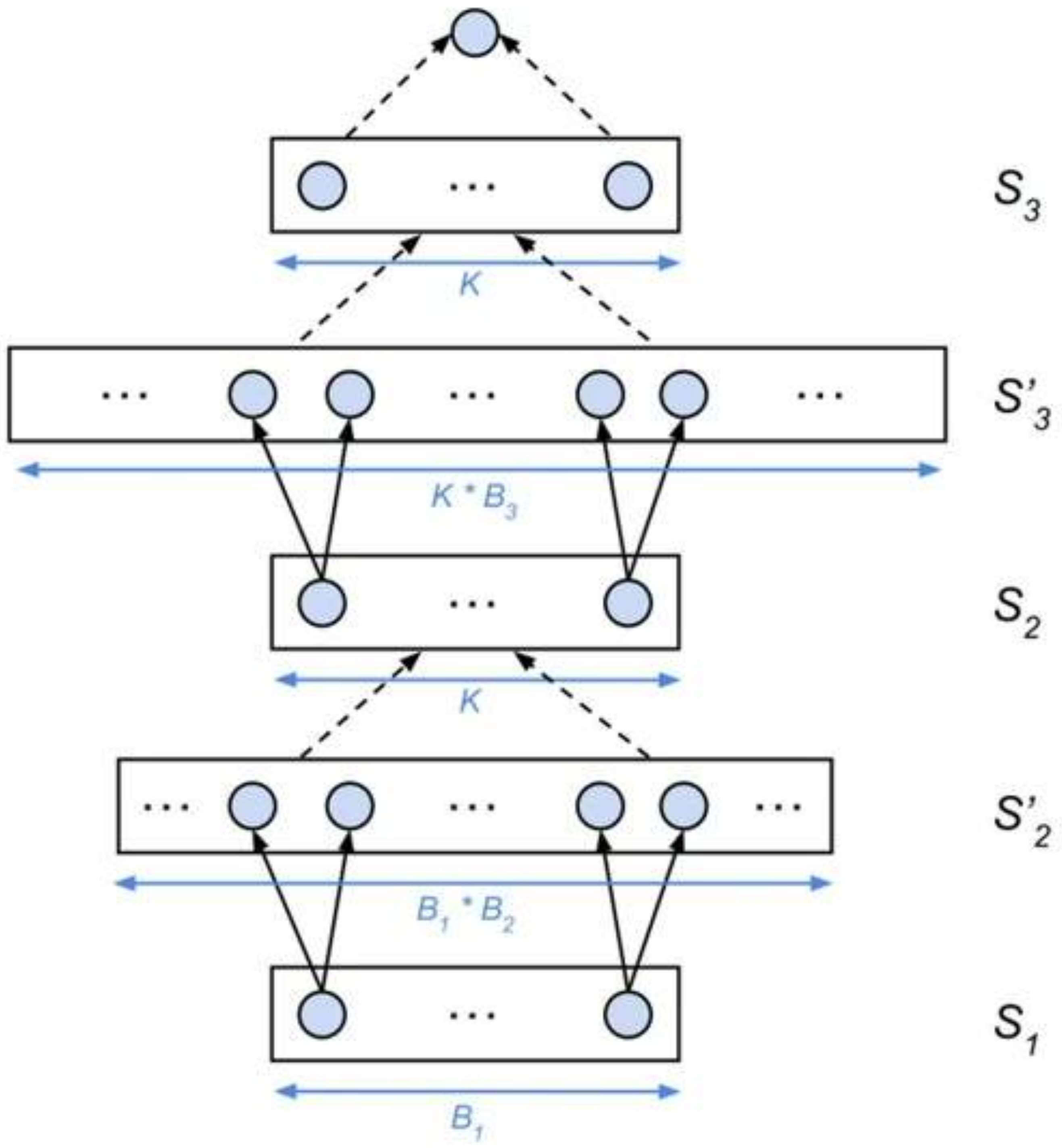


Fig. 2. Illustration of the PNAS search procedure when the maximum number of blocks is $B = 3$. Here $\mathcal{S}_b$ represents the set of candidate cells with b blocks. We start by considering all cells with 1 block, $\mathcal{S}_1 = \mathcal{B}_1$; we train and evaluate all of these cells, and update the predictor. At iteration 2, we expand each of the cells in $\mathcal{S}_1$ to get all cells with 2 blocks, $\mathcal{S}'_2 = \mathcal{B}_{1:2}$; we predict their scores, pick the top K to get $\mathcal{S}_2$, train and evaluate them, and update the predictor. At iteration 3, we expand each of the cells in $\mathcal{S}_2$, to get a subset of cells with 3 blocks, $\mathcal{S}'_3 \subseteq \mathcal{B}_{1:3}$; we predict their scores, pick the top K to get $\mathcal{S}_3$, train and evaluate them, and return the winner. $B_b = |\mathcal{B}_b|$ is the number of possible blocks at level b and K is the beam size (number of models we train and evaluate per level of the search tree).

Model	B	N	F	Error	Params	M_1	E_1	M_2	E_2	Cost
NASNet-A [1]	5	6	32	3.41	3.3M	20000	0.9M	250	13.5M	21.4-29.3B
NASNet-B [1]	5	4	N/A	3.73	2.6M	20000	0.9M	250	13.5M	21.4-29.3B
NASNet-C [1]	5	4	N/A	3.59	3.1M	20000	0.9M	250	13.5M	21.4-29.3B
Hier-EA [18]	5	2	64	3.75 ± 0.12	15.7M	7000	5.12M	0	0	35.8B[5]
AmoebaNet-B [19]	5	6	36	3.37 ± 0.04	2.8M	27000	2.25M	100	27M	63.5B[6]
AmoebaNet-A [19]	5	6	36	3.34 ± 0.06	3.2M	20000	1.13M	100	27M	25.2B[7]
PNASNet-5	5	3	48	3.41 ± 0.09	3.2M	1160	0.9M	0	0	1.0B

Table 3. Performance of different CNNs on CIFAR test set. All model comparisons employ a comparable number of parameters and exclude cutout data augmentation [32]. "Error" is the top-1 misclassification rate on the CIFAR-10 test set. (Error rates have the form $\mu \pm \sigma$, where μ is the average over multiple trials and σ is the standard deviation. In PNAS we use 15 trials.) "Params" is the number of model parameters. "Cost" is the total number of examples processed through SGD ($M_1 E_1 + M_2 E_2$) before the architecture search terminates. The number of filters F for NASNet-{B, C} cannot be determined (hence N/A), and the actual E_1, E_2 may be larger than the values in this table (hence the range in cost), according to the original authors.

Model	Params	Mult-Adds	Top-1	Top-5
MobileNet-224 [33]	4.2M	569M	70.6	89.5
ShuffleNet (2x) [34]	5M	524M	70.9	89.8
NASNet-A ($N = 4$, $F = 44$) [1]	5.3M	564M	74.0	91.6
AmoebaNet-B ($N = 3$, $F = 62$) [19]	5.3M	555M	74.0	91.5
AmoebaNet-A ($N = 4$, $F = 50$) [19]	5.1M	555M	74.5	92.0
AmoebaNet-C ($N = 4$, $F = 50$) [19]	6.4M	570M	75.7	92.4
PNASNet-5 ($N = 3$, $F = 54$)	5.1M	588M	74.2	91.9

Table 4. ImageNet classification results in the *Mobile* setting.

Model	Image Size	Params	Mult-Adds	Top-1	Top-5
ResNeXt-101 (64x4d) [36]	320×320	83.6M	31.5B	80.9	95.6
PolyNet [37]	331×331	92M	34.7B	81.3	95.8
Dual-Path-Net-131 [38]	320×320	79.5M	32.0B	81.5	95.8
Squeeze-Excite-Net [35]	320×320	145.8M	42.3B	82.7	96.2
NASNet-A ($N = 6$, $F = 168$) [1]	331×331	88.9M	23.8B	82.7	96.2
AmoebaNet-B ($N = 6$, $F = 190$) [19]	331×331	84.0M	22.3B	82.3	96.1
AmoebaNet-A ($N = 6$, $F = 190$) [19]	331×331	86.7M	23.1B	82.8	96.1
AmoebaNet-C ($N = 6$, $F = 228$) [19]	331×331	155.3M	41.1B	83.1	96.3
PNASNet-5 ($N = 4$, $F = 216$)	331×331	86.1M	25.0B	82.9	96.2

Table 5. ImageNet classification results in the *Large* setting.

DARTS: Differentiable Architecture Search (Liu et al. 2018)

DARTS

- No controllers!
- No performance prediction!
- Outperforms ENAS, PNAS.
- Cell-based (micro).
- Achieves 2.83% error on CIFAR-10.
- Uses 1000x less computation than Regularized Evolution.
- 56.1 perplexity on PennTree bank in a single GPU day.

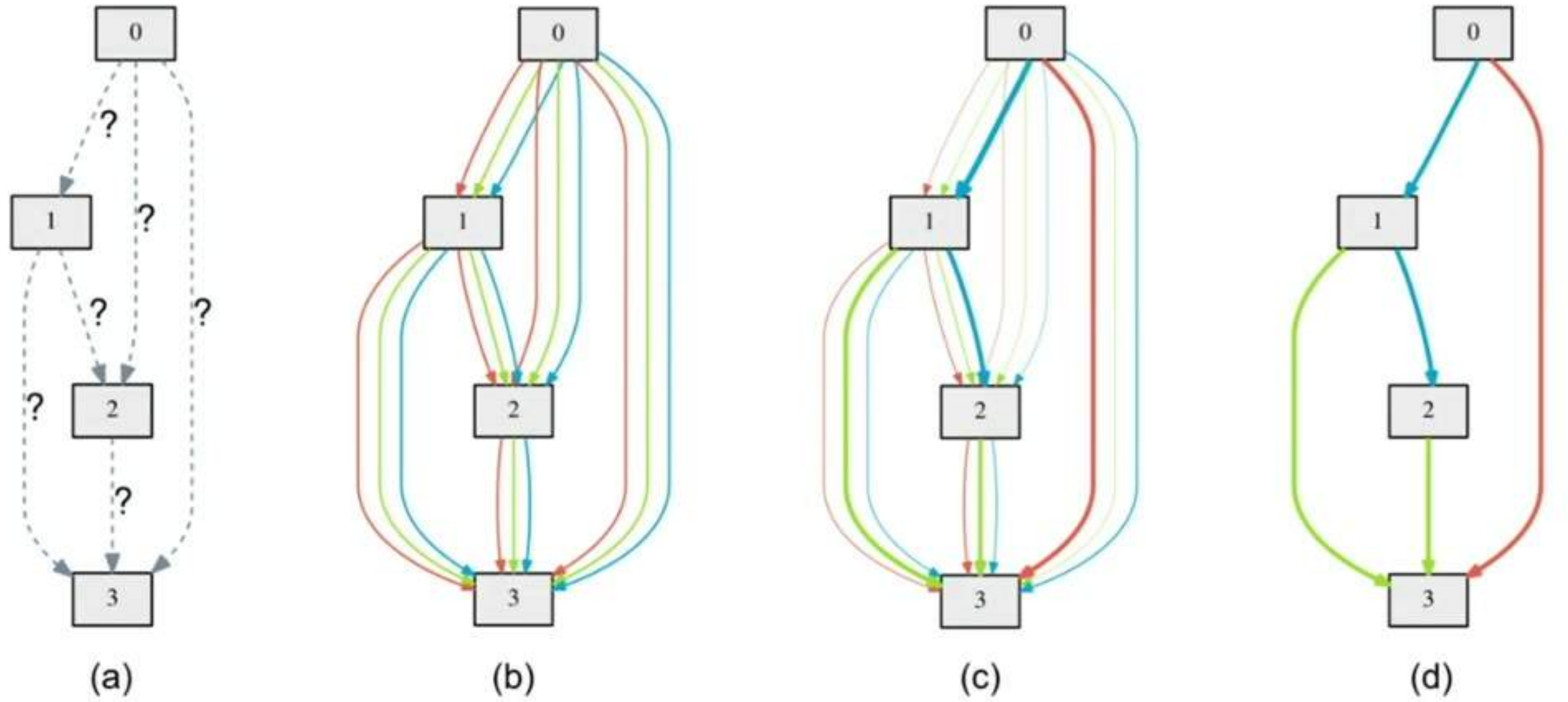


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem.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

This implies a bilevel optimization problem (Anandalingam and Friesz, 1992; Colson et al., 2007) with α as the upper-level variable and w as the lower-level variable:

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha) \tag{3}$$

$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha) \tag{4}$$

The nested formulation also arises in gradient-based hyperparameter optimization (Maclaurin et al., 2015; Pedregosa, 2016), which is related in a sense that the continuous architecture α could be viewed as a special type of hyperparameter, although its dimension is substantially higher than scalar-valued hyperparameters (such as the learning rate), and it is harder to optimize.

Algorithm 1: DARTS – Differentiable Architecture Search

```
Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i,j)$
while not converged do
    1. Update weights $w$ by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
    2. Update architecture $\alpha$ by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ for each edge $(i,j)$
```

2.4 Deriving Discrete Architectures

After obtaining the continuous architecture encoding α, the discrete architecture is derived by

1. Retaining k strongest predecessors for each intermediate node, where the strength of an edge is defined as $\max_{o \in \mathcal{O}, o \neq zero} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})}$. To make our derived architecture comparable with those in the existing works, we use $k = 2$ for convolutional cells (Zoph et al., 2017; Real et al., 2018) and $k = 1$ for recurrent cells (Pham et al., 2018b).
2. Replacing every mixed operation as the most likely operation by taking the argmax.

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10. Results marked with † were obtained by training the corresponding architectures using our setup.

Architecture	**Test Error (%)**	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	manual
NASNet-A + cutout (Zoph et al., 2017)	2.65	3.3	1800	RL
NASNet-A + cutout (Zoph et al., 2017)†	2.83	3.1	3150	RL
AmoebaNet-A + cutout (Real et al., 2018)	3.34 ± 0.06	3.2	3150	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	evolution
Hierarchical Evo (Liu et al., 2017b)	3.75 ± 0.12	15.7	300	evolution
PNAS (Liu et al., 2017a)	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	RL
Random + cutout	3.49	3.1	–	–
DARTS (first order) + cutout	2.94	2.9	1.5	gradient-based
DARTS (second order) + cutout	2.83 ± 0.06	3.4	4	gradient-based

Table 3: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	Test Error (%)		Params	$+\times$	Search Cost	Search
	top-1	top-5	(M)	(M)	(GPU days)	Method
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet 2× (v1) (Zhang et al., 2017)	29.1	10.2	∼5	524	–	manual
ShuffleNet 2× (v2) (Zhang et al., 2017)	26.3	–	∼5	524	–	manual
NASNet-A (Zoph et al., 2017)	26.0	8.4	5.3	564	1800	RL
NASNet-B (Zoph et al., 2017)	27.2	8.7	5.3	488	1800	RL
NASNet-C (Zoph et al., 2017)	27.5	9.0	4.9	558	1800	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2017a)	25.8	8.1	5.1	588	∼225	SMBO
DARTS (searched on CIFAR-10)	26.9	9.0	4.9	595	4	gradient-based

Open Problems

- How to have a truly general solution?
  - Current papers show results on:
    - CIFAR10/100 and/or transfer to ImageNet. (Vision)
    - PennTree Bank (NLP)
  - What about other datasets?
    - Are we overfitting to a few datasets?
- How to not inject domain knowledge into the search space?
  - E.g. cell-search (micro) assumes knowledge of a good outer skeleton.
  - What about when good outer skeleton is not known apriori?

- No controller.
- No predictive model.
- Simple search procedure.
    - Directly greedily search the pareto front.
    - Nice application of explore-exploit.
- Macro does better than micro!!
    - Can admit more general architectures.

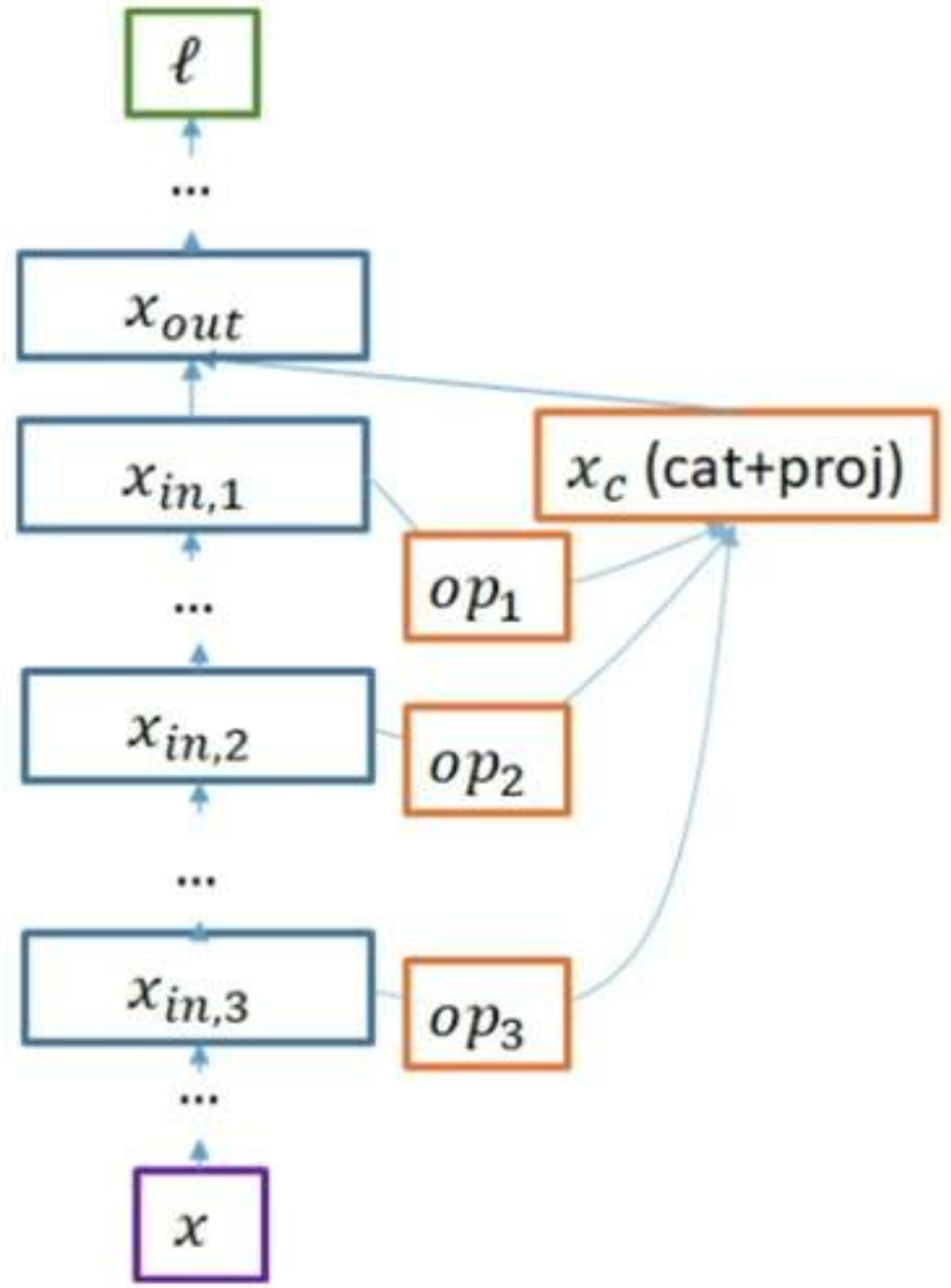


Figure 1: Incremental model change. Blue are existing layers. Orange are increments. op_i are considered part of x_c.

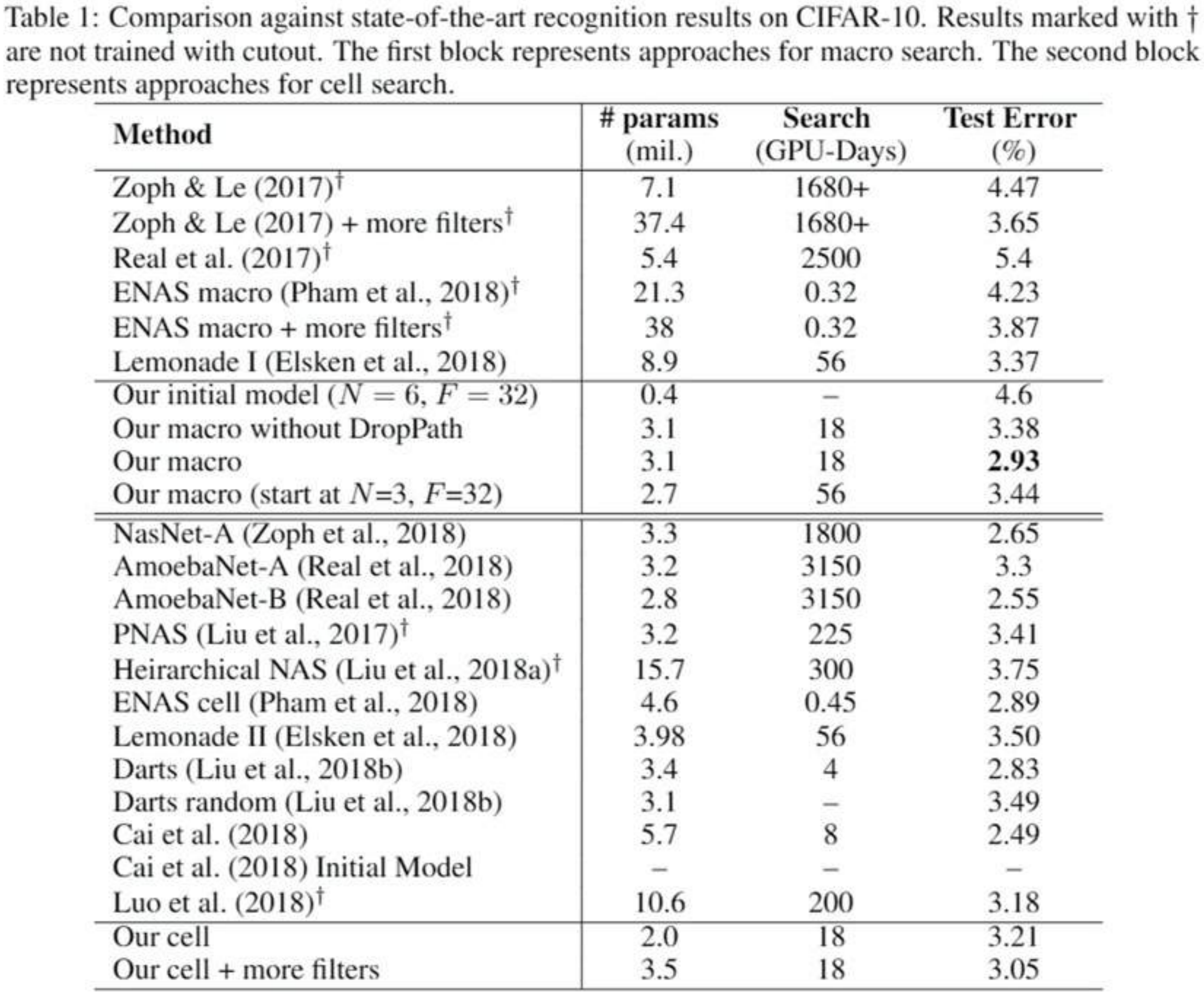


Table 1: Comparison against state-of-the-art recognition results on CIFAR-10. Results marked with † are not trained with cutout. The first block represents approaches for macro search. The second block represents approaches for cell search.

Method	**# params** (mil.)	**Search** (GPU-Days)	**Test Error** (%)
Zoph & Le (2017)†	7.1	1680+	4.47
Zoph & Le (2017) + more filters†	37.4	1680+	3.65
Real et al. (2017)†	5.4	2500	5.4
ENAS macro (Pham et al., 2018)†	21.3	0.32	4.23
ENAS macro + more filters†	38	0.32	3.87
Lemonade I (Elsken et al., 2018)	8.9	56	3.37
Our initial model ($N = 6$, $F = 32$)	0.4	–	4.6
Our macro without DropPath	3.1	18	3.38
Our macro	3.1	18	**2.93**
Our macro (start at N=3, F=32)	2.7	56	3.44
NasNet-A (Zoph et al., 2018)	3.3	1800	2.65
AmoebaNet-A (Real et al., 2018)	3.2	3150	3.3
AmoebaNet-B (Real et al., 2018)	2.8	3150	2.55
PNAS (Liu et al., 2017)†	3.2	225	3.41
Heirarchical NAS (Liu et al., 2018a)†	15.7	300	3.75
ENAS cell (Pham et al., 2018)	4.6	0.45	2.89
Lemonade II (Elsken et al., 2018)	3.98	56	3.50
Darts (Liu et al., 2018b)	3.4	4	2.83
Darts random (Liu et al., 2018b)	3.1	–	3.49
Cai et al. (2018)	5.7	8	2.49
Cai et al. (2018) Initial Model	–	–	–
Luo et al. (2018)†	10.6	200	3.18
Our cell	2.0	18	3.21
Our cell + more filters	3.5	18	3.05

Table 2: Comparison against state-of-the-art recognition results on CIFAR-100. Our models are the ones found in CIFAR-10 search. The first block represents the best human designed models. The second represents models found through automated search on CIFAR100.

Method	**# params** (mil.)	**Test Error** (%)
DenseNet-BC (k=40) (Gao Huang, 2017)	25.6	17.18
Shake-Shake (Gastaldi, 2017)	26.2	15.85
SMASHv2 (Brock et al., 2017)	16	20.6
Real et al. (2017)	40.4	23.7
Elsken et al. (2017)	22.3	23.4
Block-QNN-S (Zhong et al., 2018)	6.1	20.65
Our macro	3.1	18.74

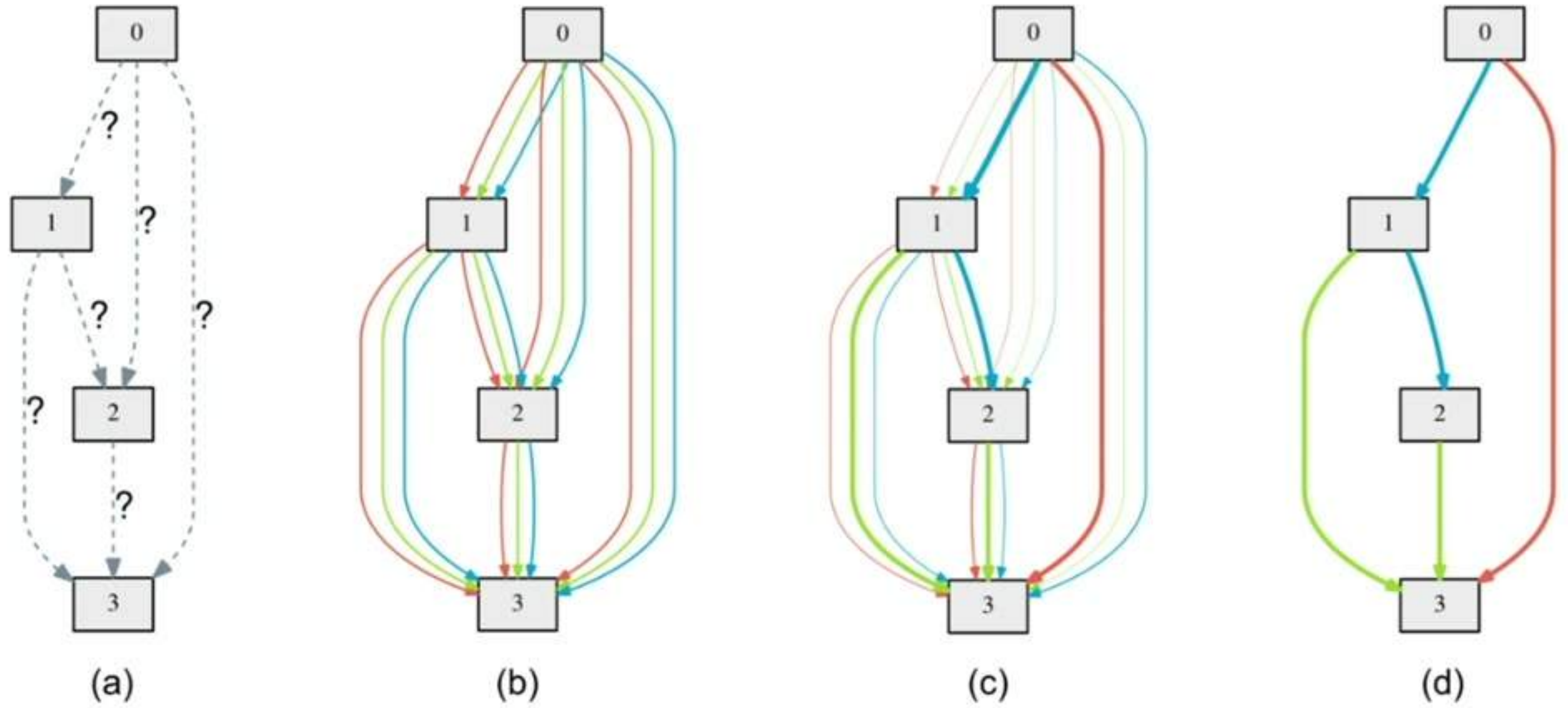


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem.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