Neural Architecture Search

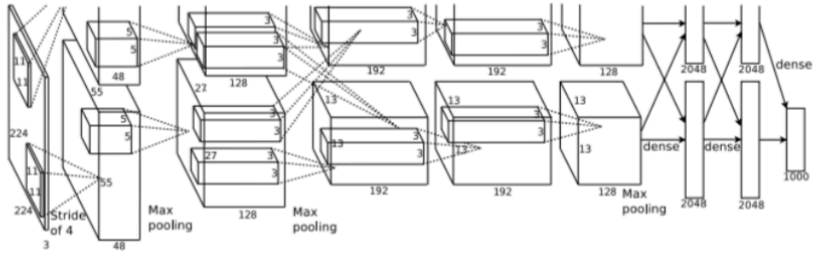
Sun, Yue

Contents

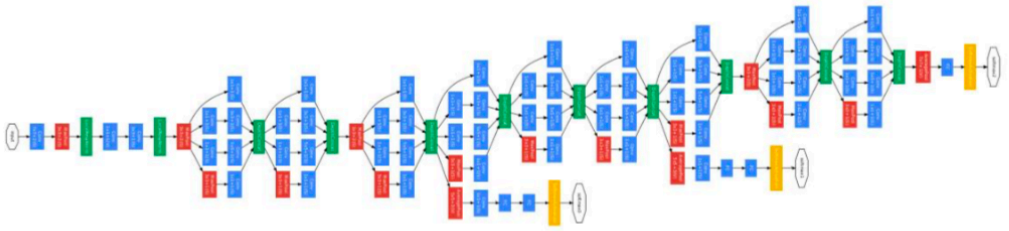
- NAS
- HPO
- Autogluon

The Architecture of a Neural Network is Crucial to its Performance

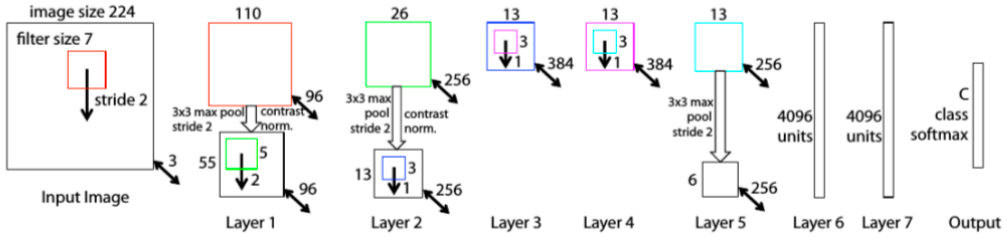
- ImageNet Winning Neural Architectures



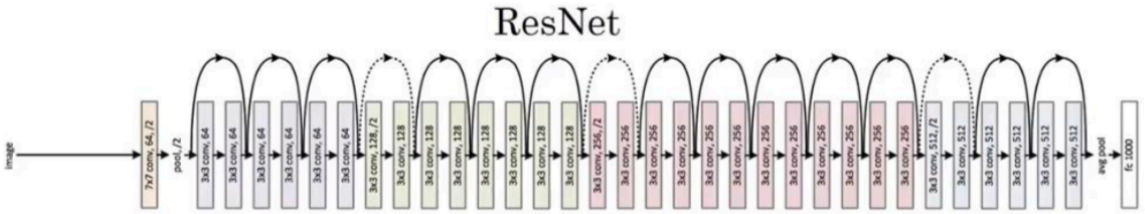
AlexNet 2012



Inception 2014



ZFNet 2013

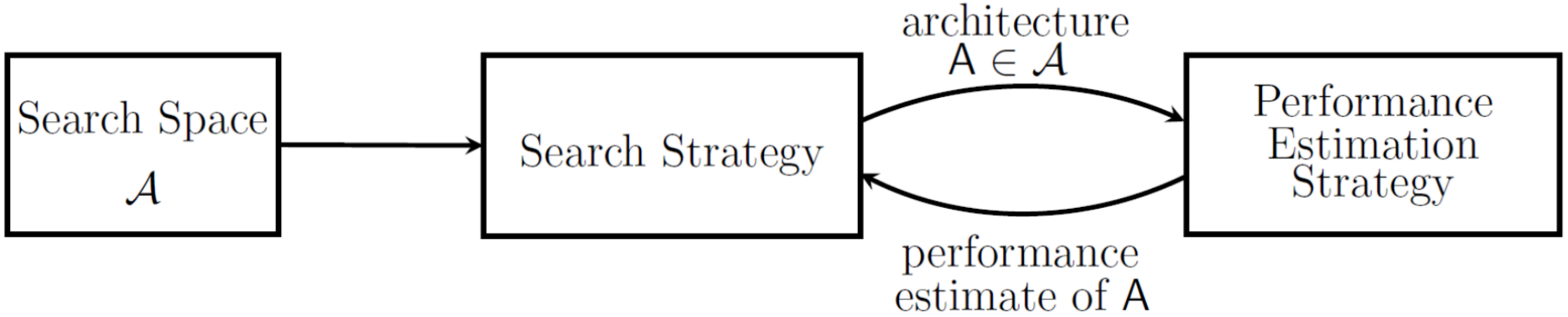


ResNet 2015

https://gluon-cv.mxnet.io/model_zoo/classification.html

Neural Architecture Search （NAS）

- Neural Architecture Search can be divided as three parts:



Search Space: the search space defines which neural architectures a NAS approach might discover in principle.

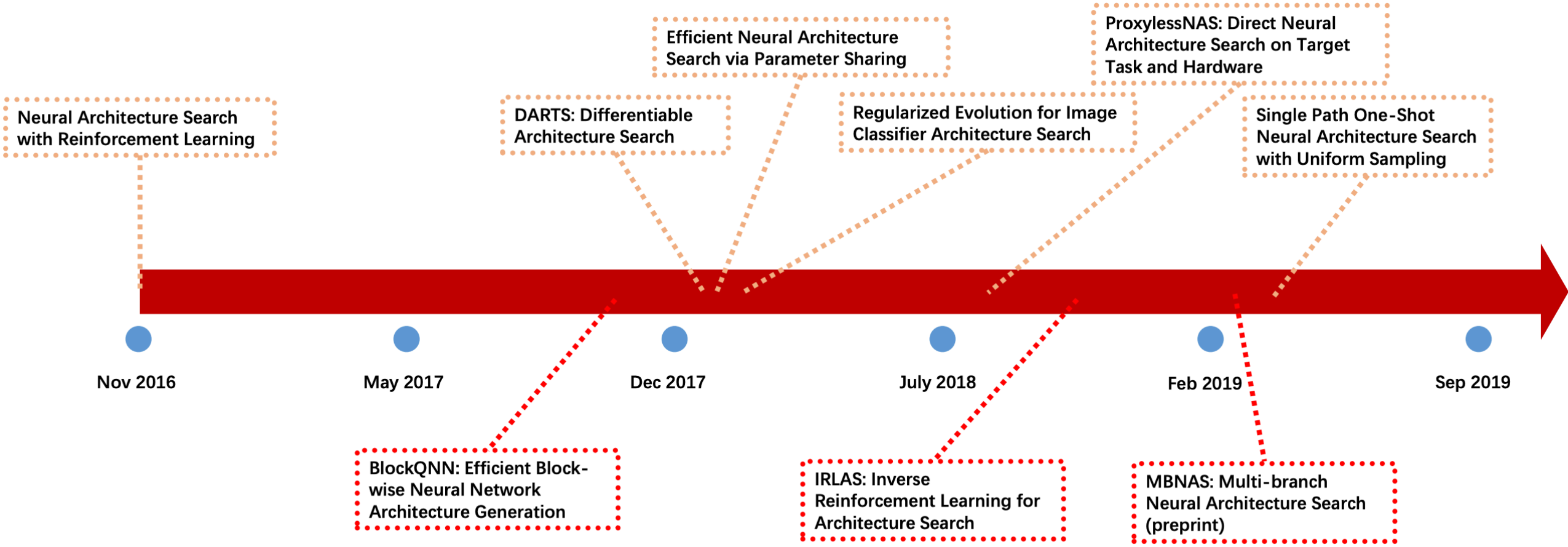
Search Strategy: the search strategy details how to explore the search space.

- *Reinforcement Learning (RL) based: NAS, NASNet, MnasNet …*
- *Evolutionary Algorithm (EA) based: AmoebaNet, NSGANet …*
- *Gradient based: DARTS, ProxylessNAS …*

Performance Estimation Strategy: the process of estimating the performance of architecture.

- *Lower fidelities*
- *Learning Curve Extrapolation*
- *Parameters Sharing*
- *One-Shot Architecture Search*

Time line of NAS



NAS

- Key point: Generate Recurrent Architectures
  - use a controller to generate architectural hyperparameters of neural networks (as a sequence of tokens)
  - filter height, filter width, stride height, stride width, and number of filters for one layer and repeats.
  - reward signal R is non-differentiable, a policy gradient method which we use to update parameters θc (θc, are then optimized in order to maximize the expected validation accuracy) REINFORCE rule from Williams.(1992)

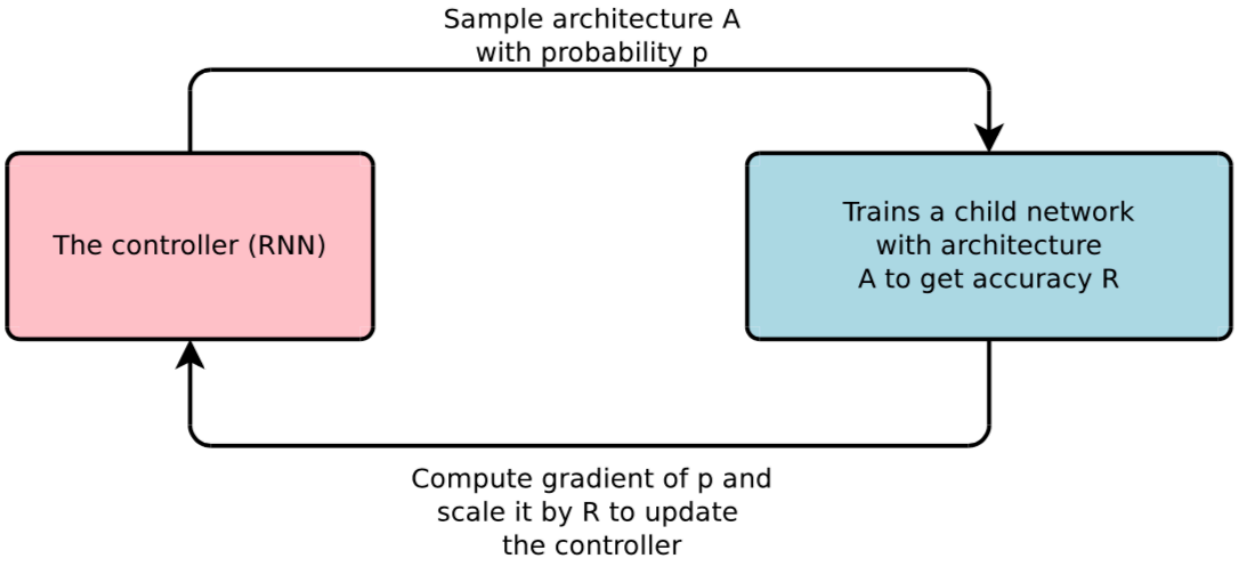


Figure 1: An overview of Neural Architecture Search.

NAS

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):

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

An empirical approximation of the above quantity is:

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

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.

NAS

S parameter servers; K controller replicas; sample m child network;

Each child model is recorded the θc to update.

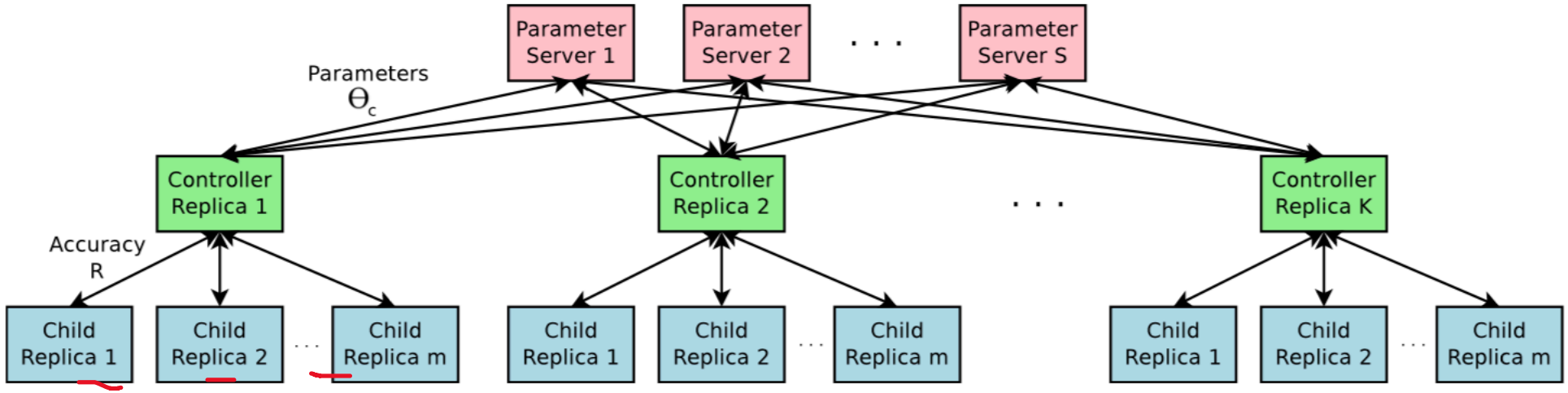


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to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

NAS

To enable the controller to predict skip connections like Resnet.

At layer N , we add an anchor point which has N − 1 content-based sigmoids to indicate the previous layers that need to be connected.

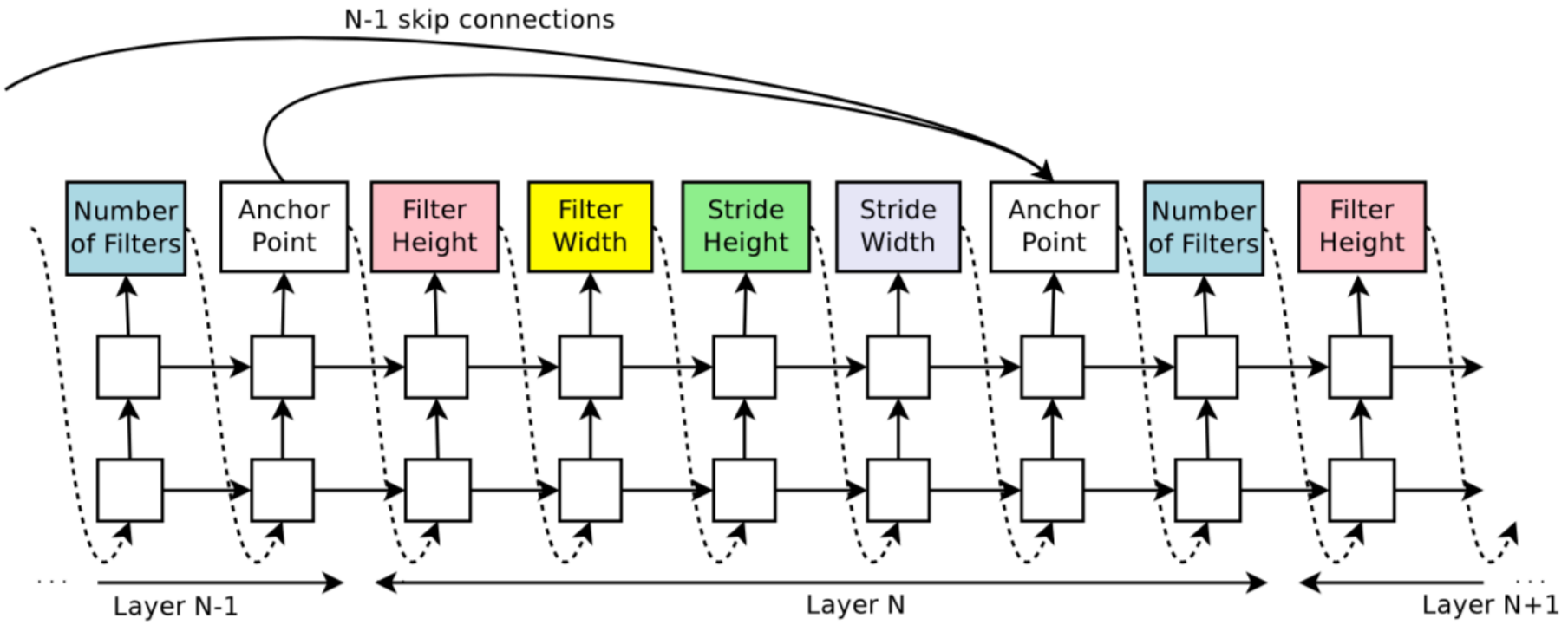


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

Experiment:

Dataset: Cifar10

Space: For every convolutional layer,

a filter height in [1, 3, 5, 7]

a filter width in [1, 3, 5, 7],

a number of filters in [24, 36, 48, 64].

For strides, we perform two sets of experiments, one where we fix the strides to be 1, and one where we allow the controller to predict the strides in [1, 2, 3].

Details: a child model is constructed and trained for 50 epochs. The reward used for updating the controller is maximum validation accuracy of the last 5 epochs

800 K40 * 28days (Cifar)

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.

Search Space

- Similar to Zoph et al.(2018)

NASNet

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

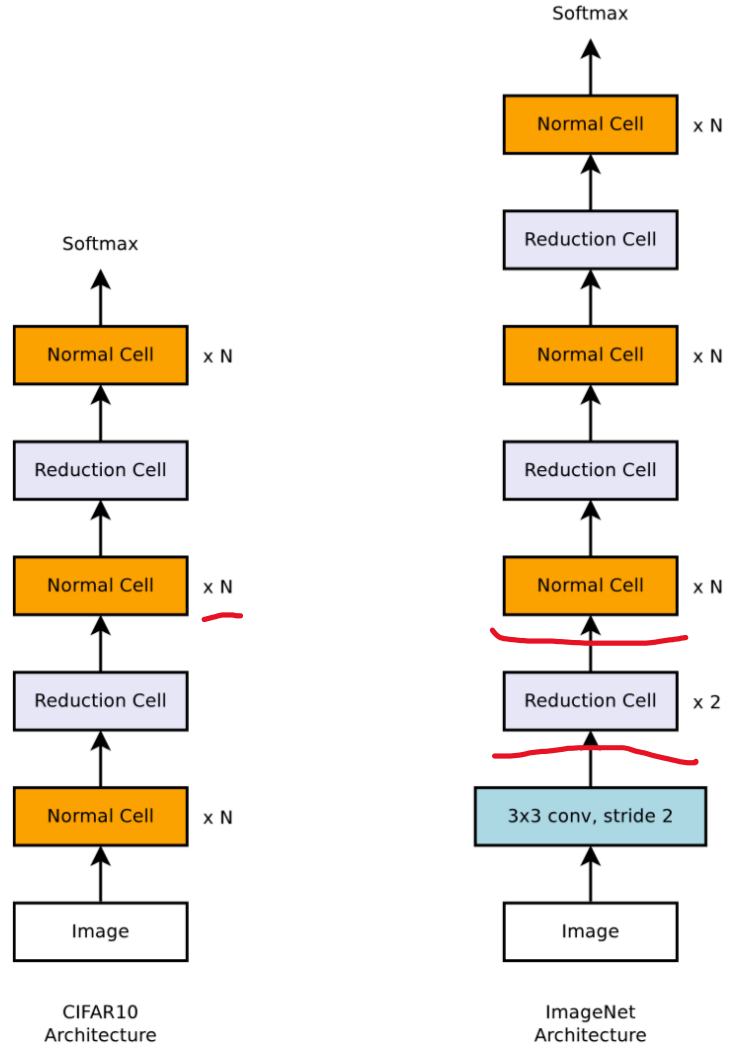
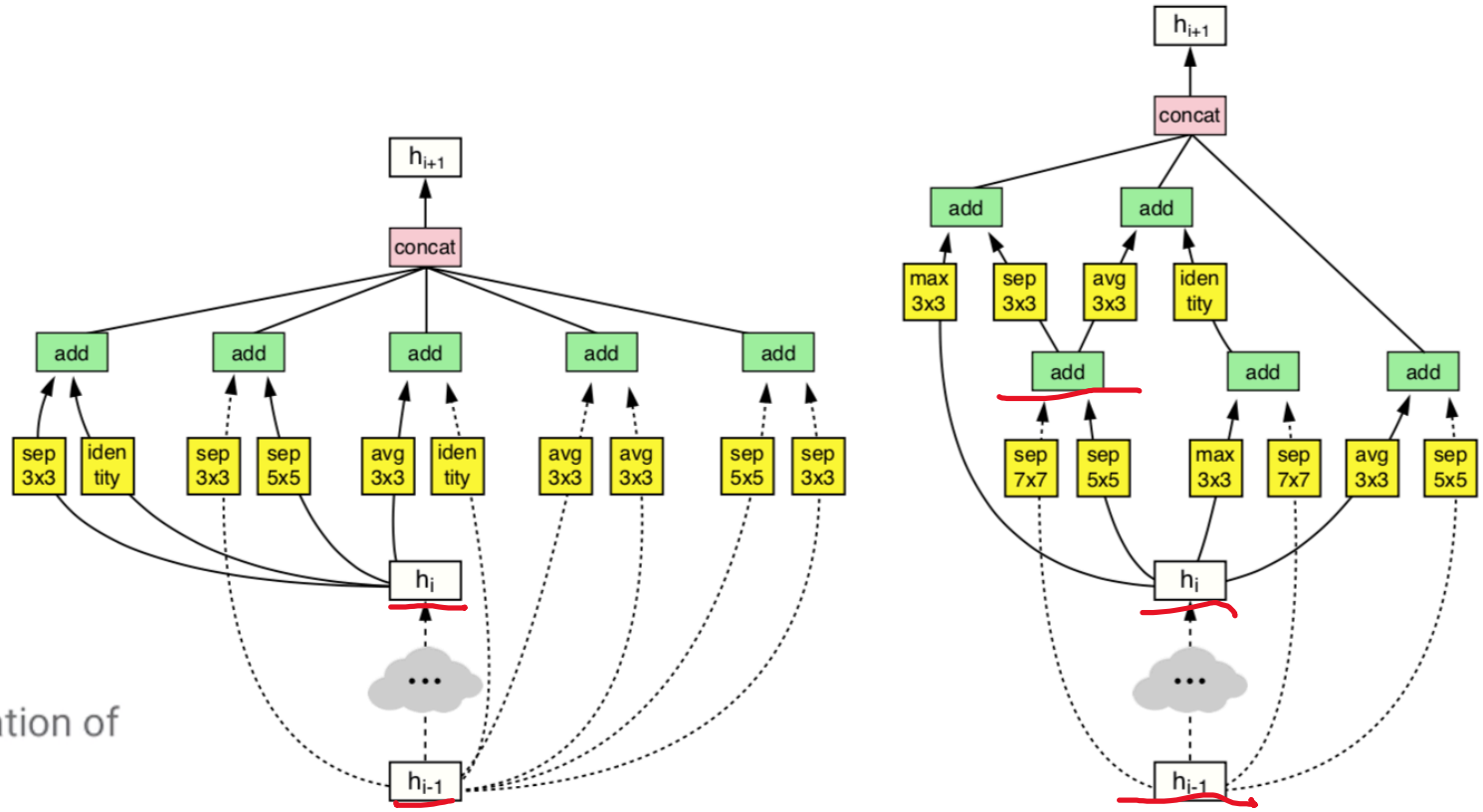


Figure 2. Scalable architectures for image classification consist of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells, N, can vary in our experiments.

Search Space

Block->Cell

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

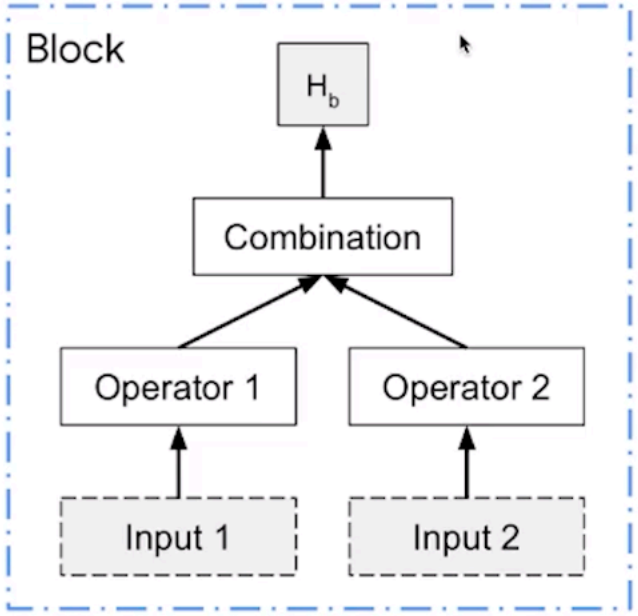


Normal Cell *Reduction Cell*

Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

Search Space

Within a block



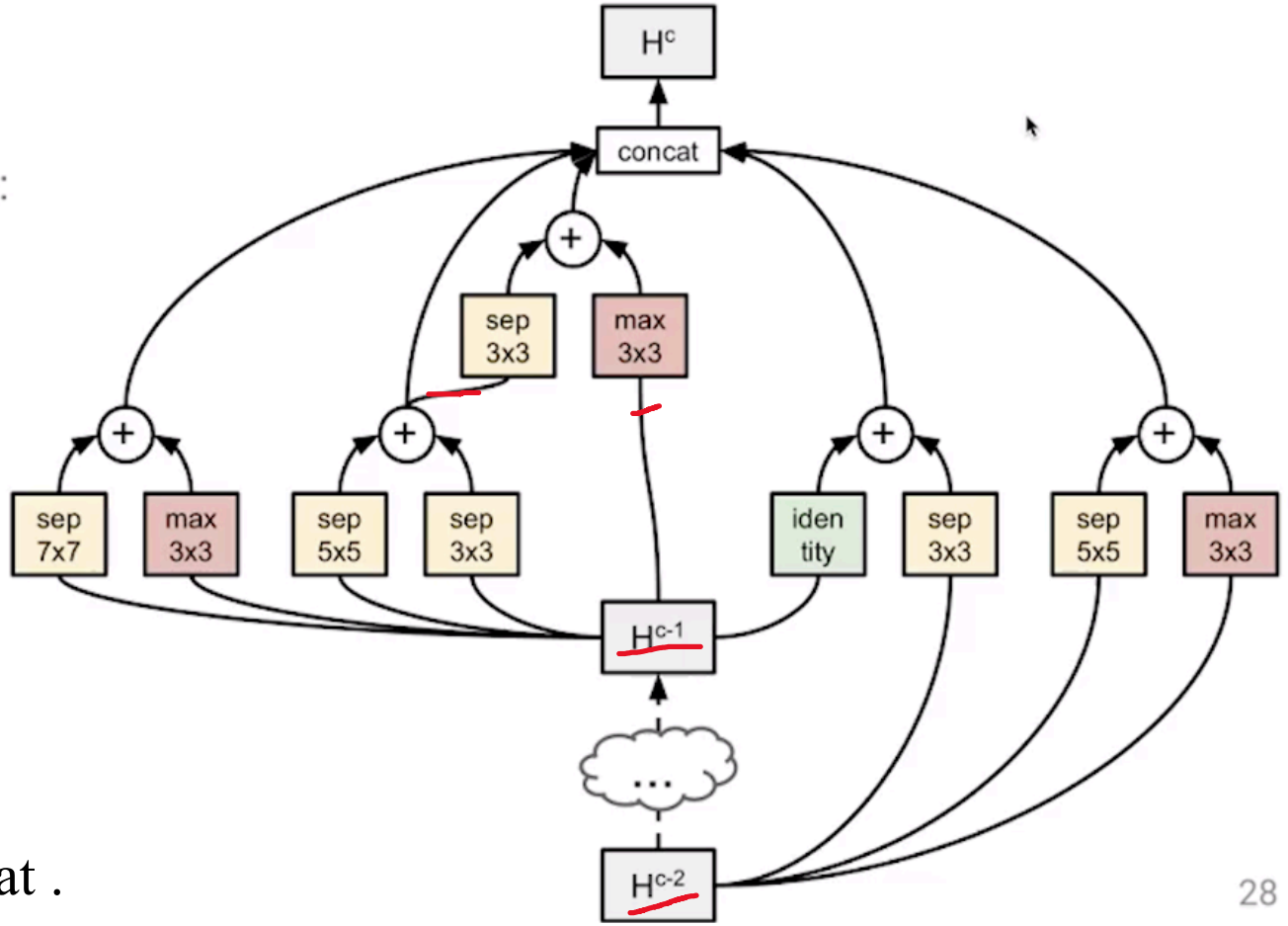
- Input 1 is transformed by Operator 1
- Input 2 is transformed by Operator 2
- Combine to give block's output
- **Input 1** and **Input 2** may select from:
  - Previous cell's output
  - Previous-previous cell's output
  - Previous blocks' output in current cell

- **Combination** is element-wise addition
- **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

Architecture Search Space Summary

每一个+ 表示一个block结束，elementwise addition
前两个cell 以及前一个cell的output

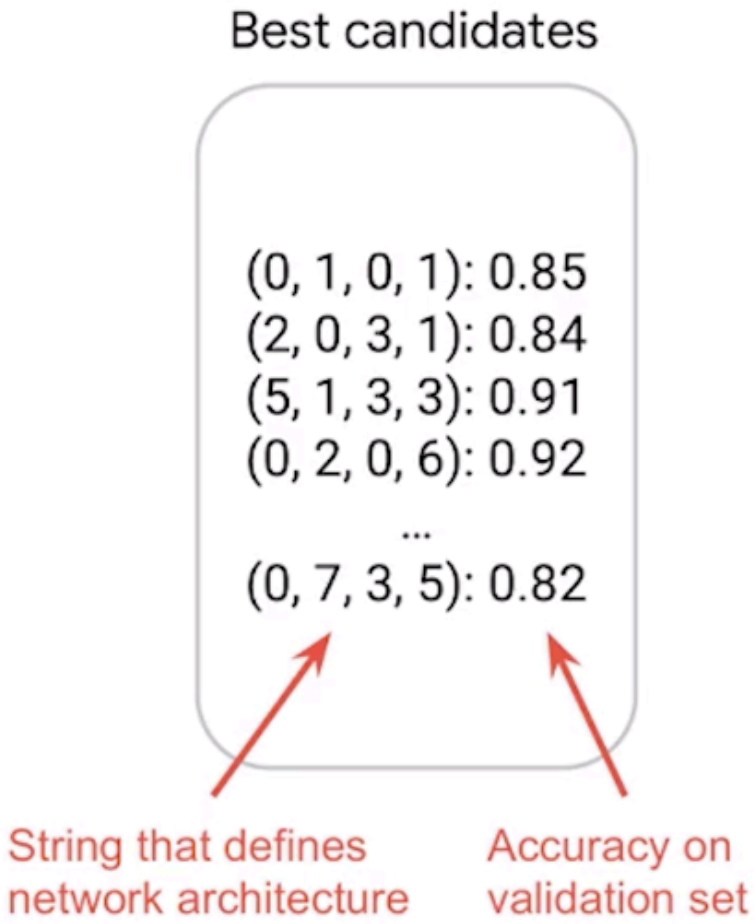
- One cell may look like:
- $2^2 * 8^2 * 1 *$
  $3^2 * 8^2 * 1 *$
  $4^2 * 8^2 * 1 *$
  $5^2 * 8^2 * 1 *$
  $6^2 * 8^2 * 1 =$
  10^{14} possible combinations!



Network的结构提前定义不搜索，
Cell的search space已经非常大了
random search is a difficult baseline to beat .
500 K40 * 4 days (cifar10)

Evolutionary Algorithms for NAS

- AmoebaNet



Evolutionary Algorithms for NAS

Evolutionary Algorithms for NAS

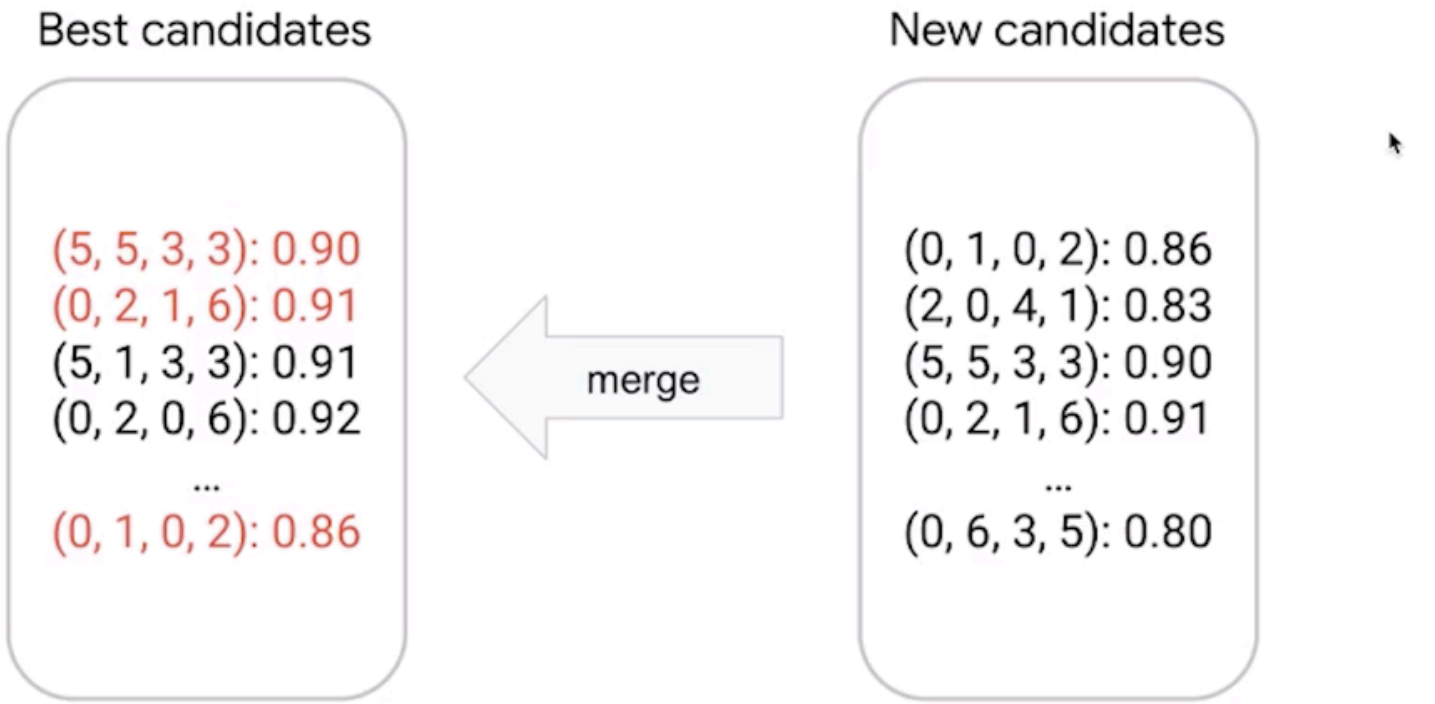
Best candidates

(0, 1, 0, 1): 0.85
(2, 0, 3, 1): 0.84
(5, 1, 3, 3): 0.91
(0, 2, 0, 6): 0.92
...
(0, 7, 3, 5): 0.82

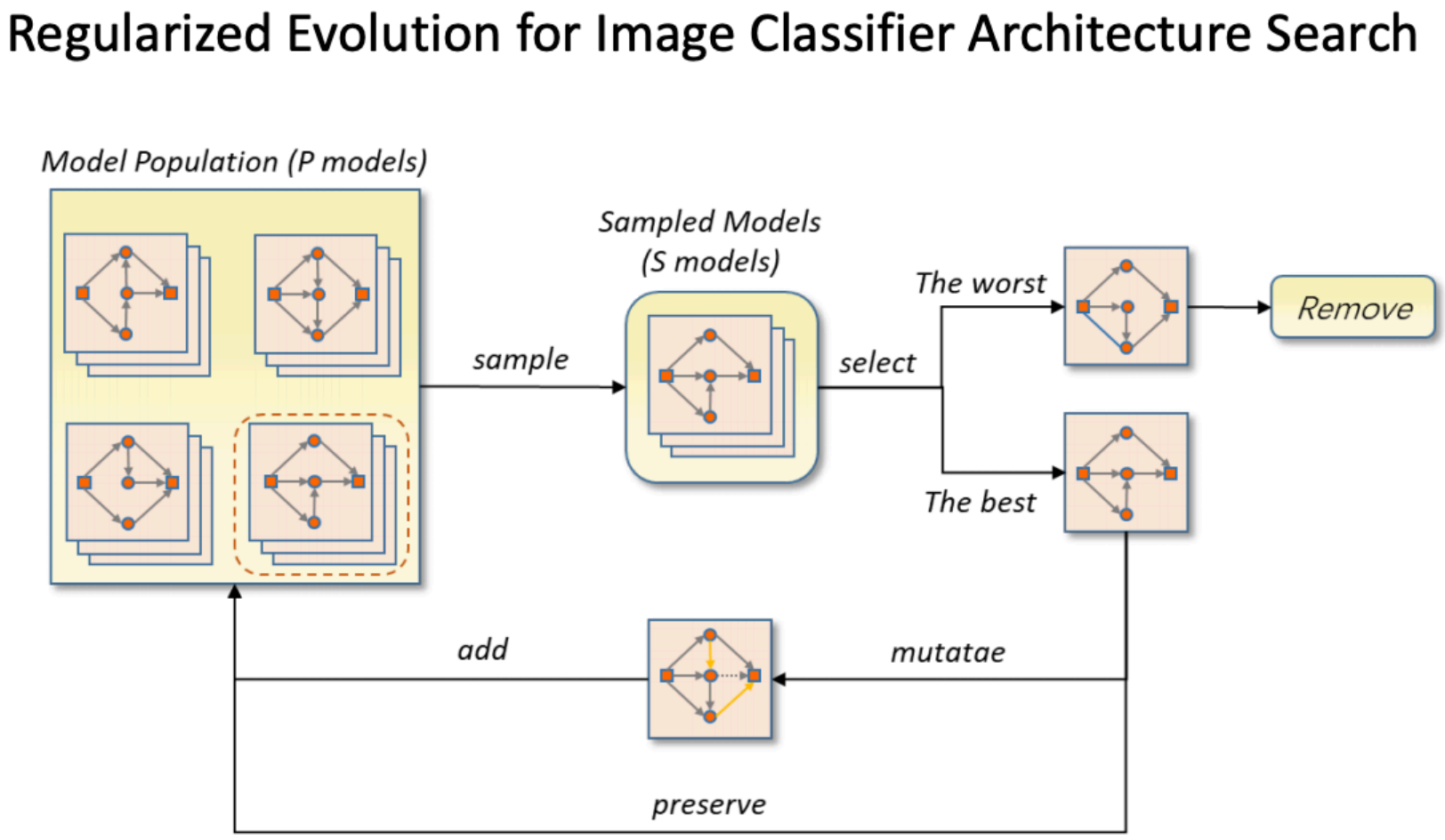
New candidates

(0, 1, 0, 2): 0.86
(2, 0, 4, 1): 0.83
(5, 5, 3, 3): 0.90
(0, 2, 1, 6): 0.91
...
(0, 6, 3, 5): 0.80

Evolutionary Algorithms for NAS



Evolutionary Algorithms for NAS

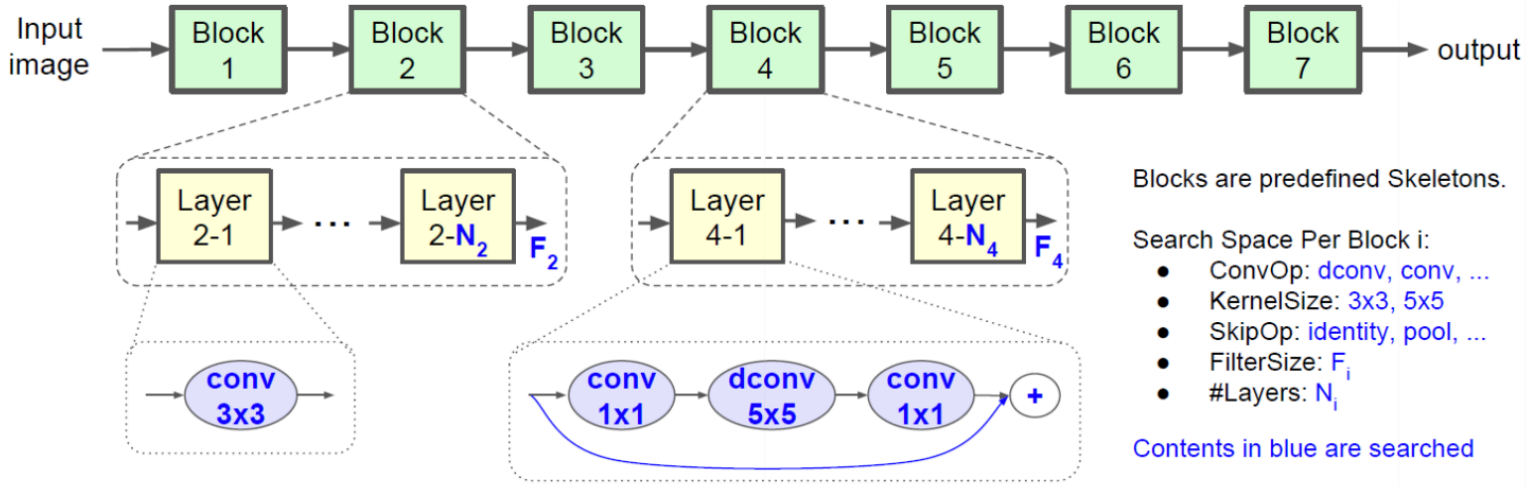


450 K40 * 7 days (cifar10)

Search Strategy

- Mnas

We propose a novel **factorized hierarchical search space** to maximize the on-device resource efficiency of mobile models, by striking the right balance between flexibility and search space size.

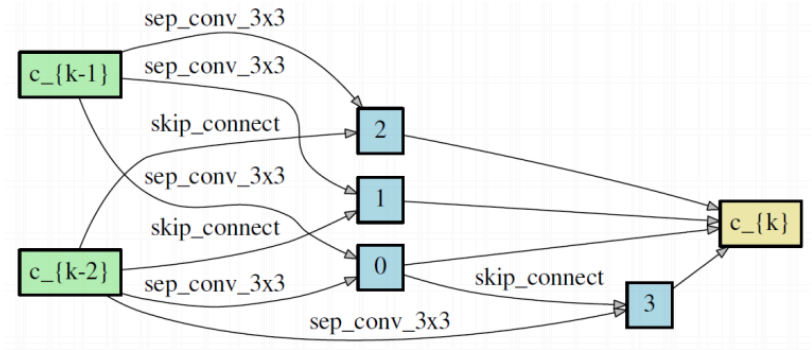


Blocks are predefined Skeletons.

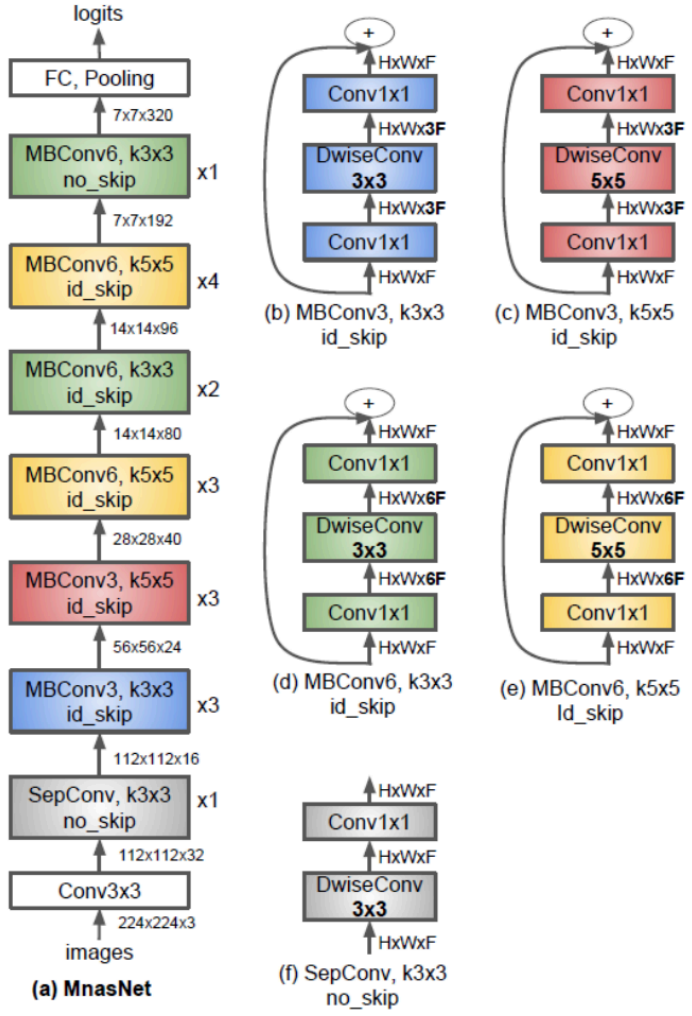
Search Space Per Block i:

- ConvOp: dconv, conv, ...
- KernelSize: 3x3, 5x5
- SkipOp: identity, pool, ...
- FilterSize: F_i
- #Layers: N_i

Contents in blue are searched



Compared with the cell-based search space, the search space of MnasNet is more latency-friendly and topology-simple.



Search Strategy

- Mnas

We introduce a multi-objective neural architecture search approach based on reinforcement learning, which is capable of finding ***high accuracy*** CNN models with ***low real-world inference latency***.

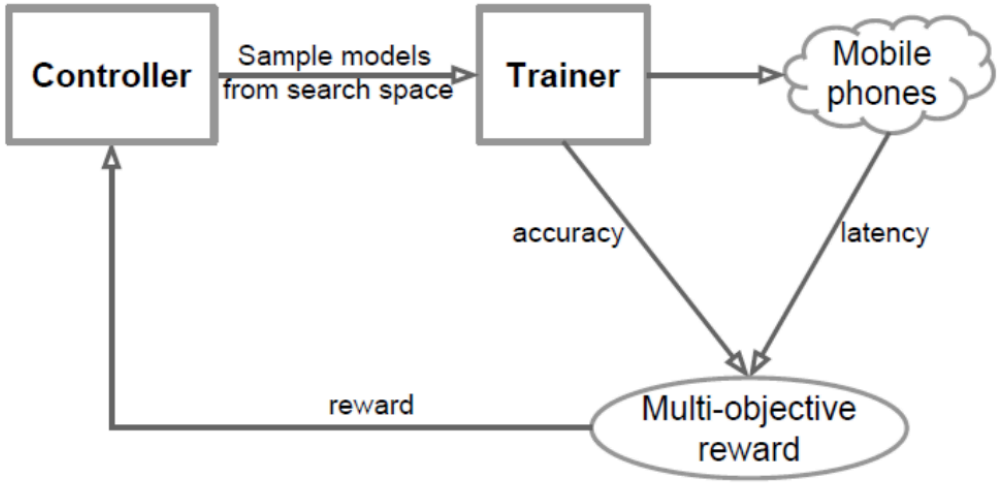


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile.**

Use a customized weighted product method to approximate Pareto optimal solutions, by setting the optimization goal as:

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{w}$$

$$w = \begin{cases} \alpha, & \text{if } LAT(m) \leq T \\ \beta, & \text{otherwise} \end{cases} \qquad \alpha = \beta = -0.07$$

Search Strategy

- Darts （Gradient- based）

we have a bunch of a powerful gradient-based optimization algorithms

First to formulate NAS task in a differentiable manner

Continuous relaxation of the architecture representation
Efficient search of the architecture using gradient descent

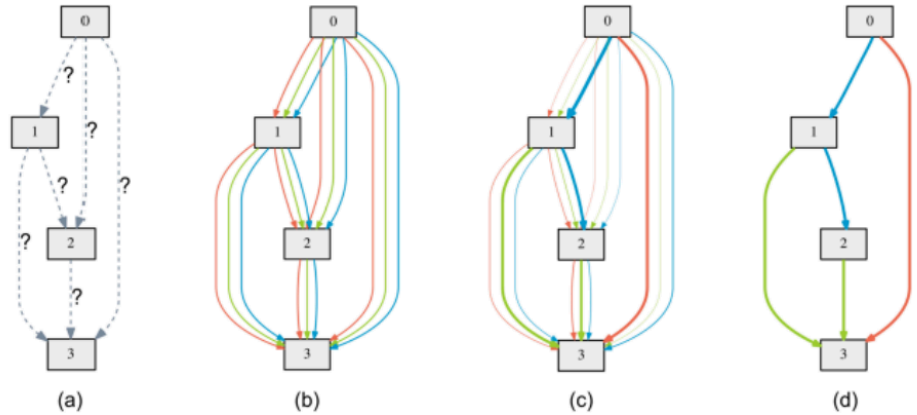


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem.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

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 α 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)

$$x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)}) \qquad \bar{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

Search Strategy

- Darts （Gradient- based）

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	+×	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

Search Strategy

ProxylessNAS

ProxylessNAS directly learns architectures on the large-scale dataset (e.g. ImageNet) **without any proxy** while still allowing a large candidate set and removing the restriction of repeating blocks. It effectively **enlarged the search space** and achieved better performance.

We provide a new path-level pruning perspective for NAS, showing a close connection between NAS and model compression. We save the **memory consumption** by one order of magnitude by using **path-level binarization**.

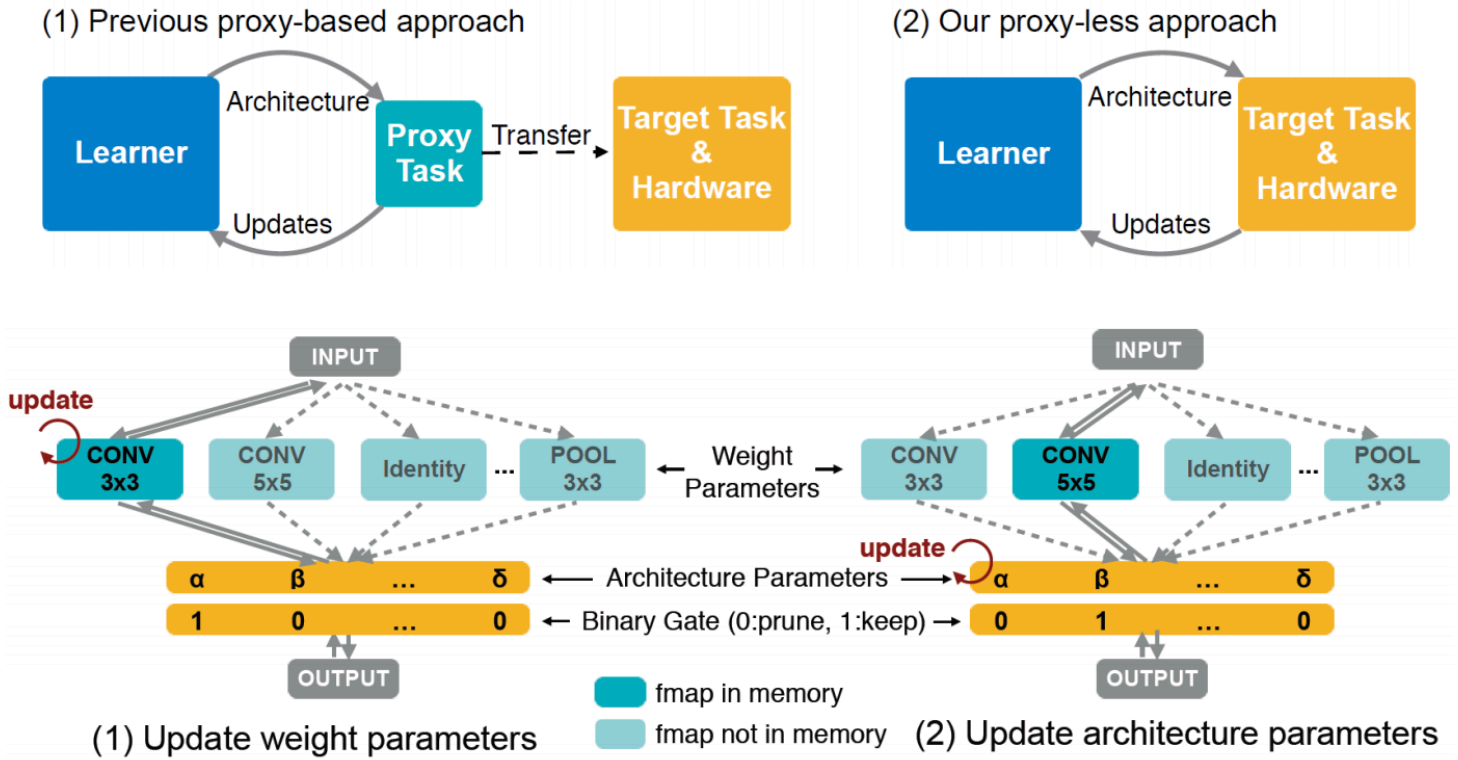


Figure 2: Learning both weight parameters and binarized architecture parameters.

Search Strategy

ProxylessNAS

*Proxyless-NAS enables hardware-aware neural network specialization that's exactly optimized for the **target hardware**.*

Optimize both accuracy and latency based on the differentiable framework.

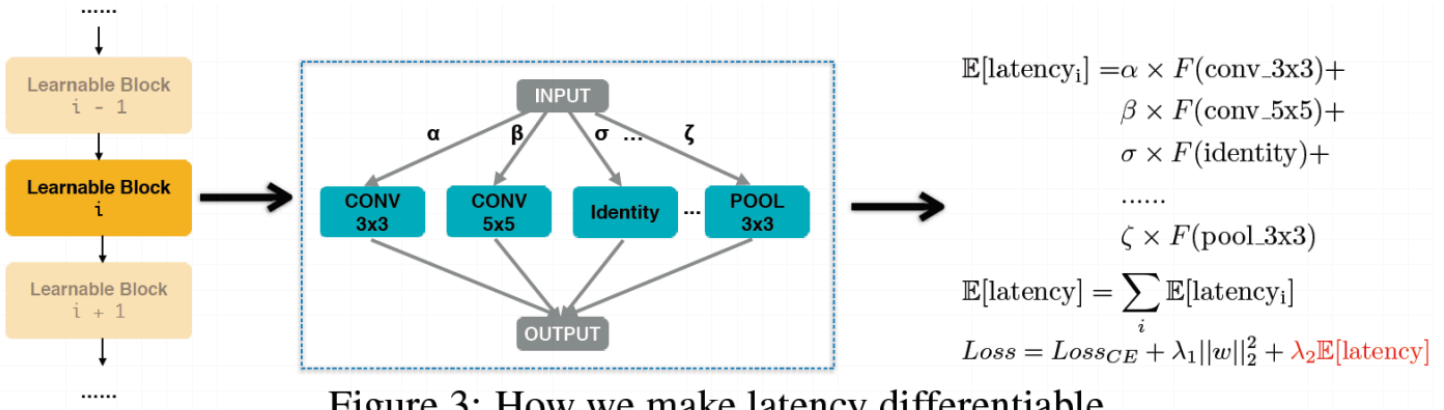


Figure 3: How we make latency differentiable.

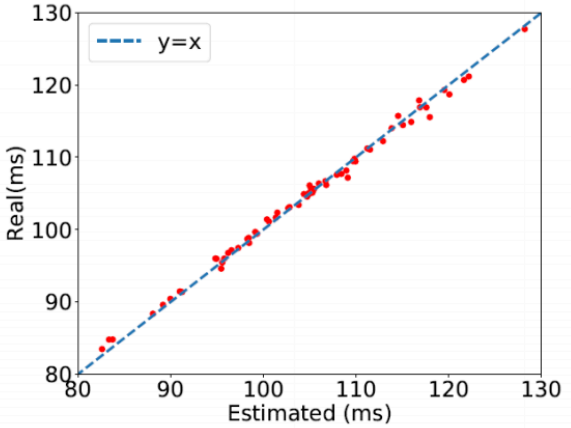


Figure 6: Our mobile latency model is close to $y = x$.

Search Strategy

ProxylessNAS

Model	Top-1	Top-5	Mobile latency
MobileNetV1 (Howard et al., 2017)	70.6	89.5	113ms
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	75ms
NASNet-A (Zoph et al., 2018)	74.0	91.3	183ms
AmoebaNet-A (Real et al., 2018)	74.5	92.0	190ms
MnasNet (Tan et al., 2018)	74.0	91.8	76ms
MnasNet (our impl.)	74.0	91.8	79ms
Proxyless-G (mobile)	71.8	90.3	83ms
Proxyless-G + Latency Loss (mobile)	74.2	91.7	79ms
Proxyless-R (mobile)	**74.6**	**92.2**	78ms

Table 2: ProxylessNAS achieves state-of-the art accuracy (%) on ImageNet (under mobile latency constraint $\leq 80ms$) with 100× less search cost in GPU hours.

Model	Top-1	Top-5	GPU latency
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	6.1ms
ShuffleNetV2 (1.5) (Ma et al., 2018)	72.6	-	7.3ms
ResNet-34 (He et al., 2016)	73.3	91.4	8.0ms
DARTS (Liu et al., 2018c)	73.1	91.0	-
NASNet-A Zoph et al. (2018)	74.0	91.3	38.3ms
MnasNet (Tan et al., 2018)	74.0	91.8	6.1ms
Proxyless (GPU)	**75.1**	**92.5**	**5.1ms**

Table 3: Accuracy (%) and GPU latency on ImageNet.

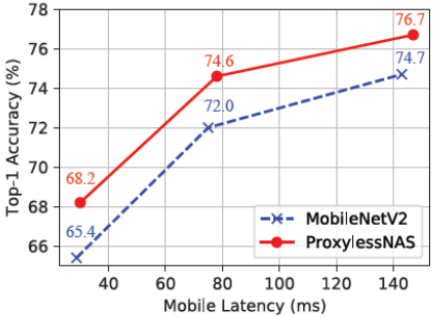


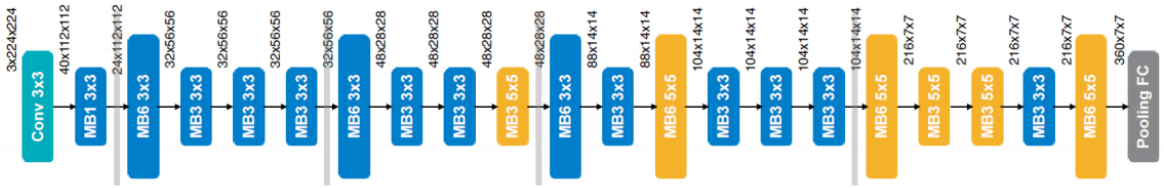
Figure 4: ProxylessNAS consistently outperforms MobileNetV2 under various latency settings.

Method	GPU hours	GPU memory
NASNet	10^4	$\mathbf{10^1}$
DARTS	$\mathbf{10^2}$	10^2
Mnas	10^4	$\mathbf{10^1}$
Ours	$\mathbf{10^2}$	$\mathbf{10^1}$

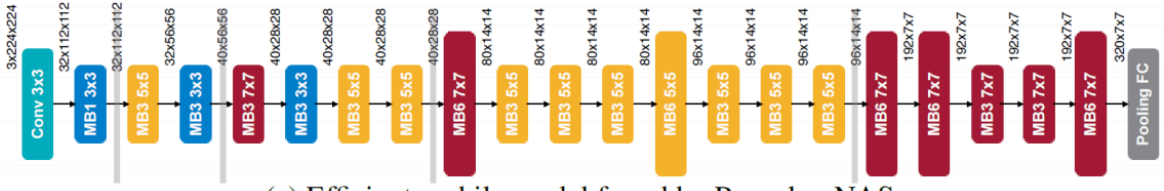
Table 4: ProxylessNAS saves GPU hours and memory (GB) by 1-2 orders of magnitude on ImageNet.[3]

Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

(a) Efficient GPU model found by ProxylessNAS.



(b) Efficient CPU model found by ProxylessNAS.

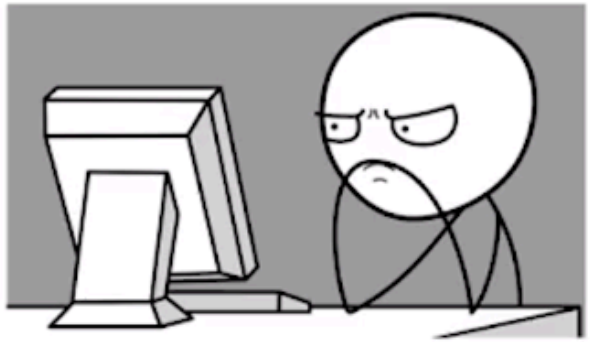


(c) Efficient mobile model found by ProxylessNAS.

Hyperparameter Optimzation (HPO)

Machine Learning solution	Parameter	Hyperparameter
Neural Network	Automated :)	Not quite automated :(

Key of AutoML



Search method

- Grid search
- Random search
- Skopt (Scikit-learn Optimzation)

http://scikit-optimize.github.io/optimizer/index.html#skopt.optimizer.Optimizer

Hyperparameter Optimzation (HPO)

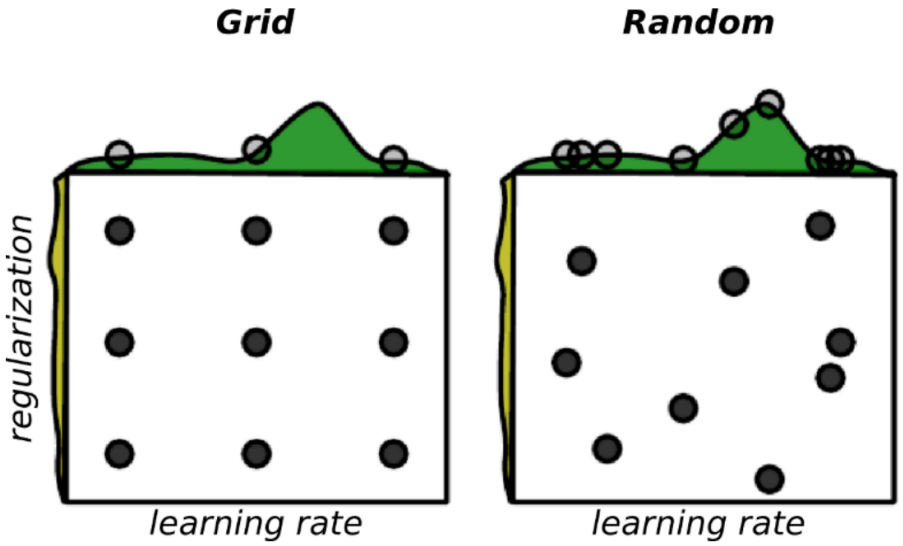
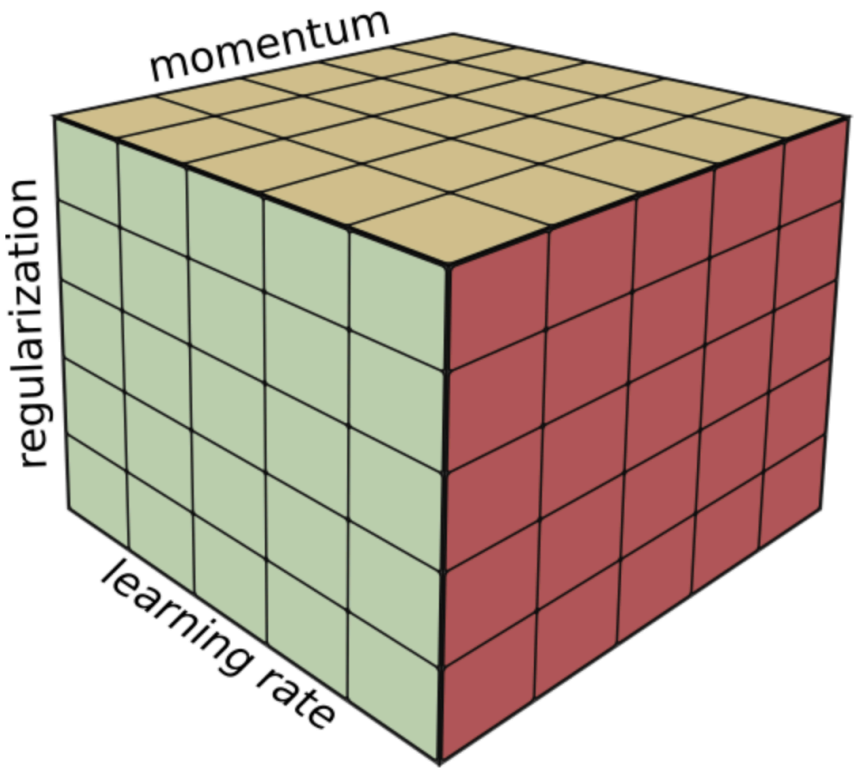


Image Credit: Random Search for Hyper-Parameter Optimization, James Bergstra, Yoshua Bengio.



Optimizing over 3 hyper-parameters using Grid Search.

For Grid Search, we would be running 125 training runs, but only exploring five different values of each parameter.

On the other hand, with Random Search, we would be exploring 125 different values of each.

Grid Search is that it is an exponential time algorithm. Its cost grows exponentially with the number of parameters.

In other words, if we need to optimize p parameters and each one takes at most v values, it runs in $O(v^p)$ time.

Hyperparameter Optimzation (HPO)

If we want to try values for the learning rate, say within the range of 0.1 to 0.0001, we do:

```
experiments = 25
for i in range(experiments):

    # sample from a Uniform distribution on a log-scale
    learning_rate = 10**np.random.uniform(-1,-4) # Sample learning rate candidates in the range (0.1
    regularization = 10**np.random.uniform(-2,-5) # Sample regularization candidates in the range (0

    # do your thing with the hyper-parameters
```

Note that we are sampling values from a uniform distribution on a log scale.

```
# DO NOT DO THIS
experiments = 25
for i in range(experiments):
  # uniform distribution on a log-scale for
  learning_rate = np.random.uniform(0.1,0.0004)
  regularization = np.random.uniform(0.01,0.00001)

  # do your thing with the hyper-parameters
```

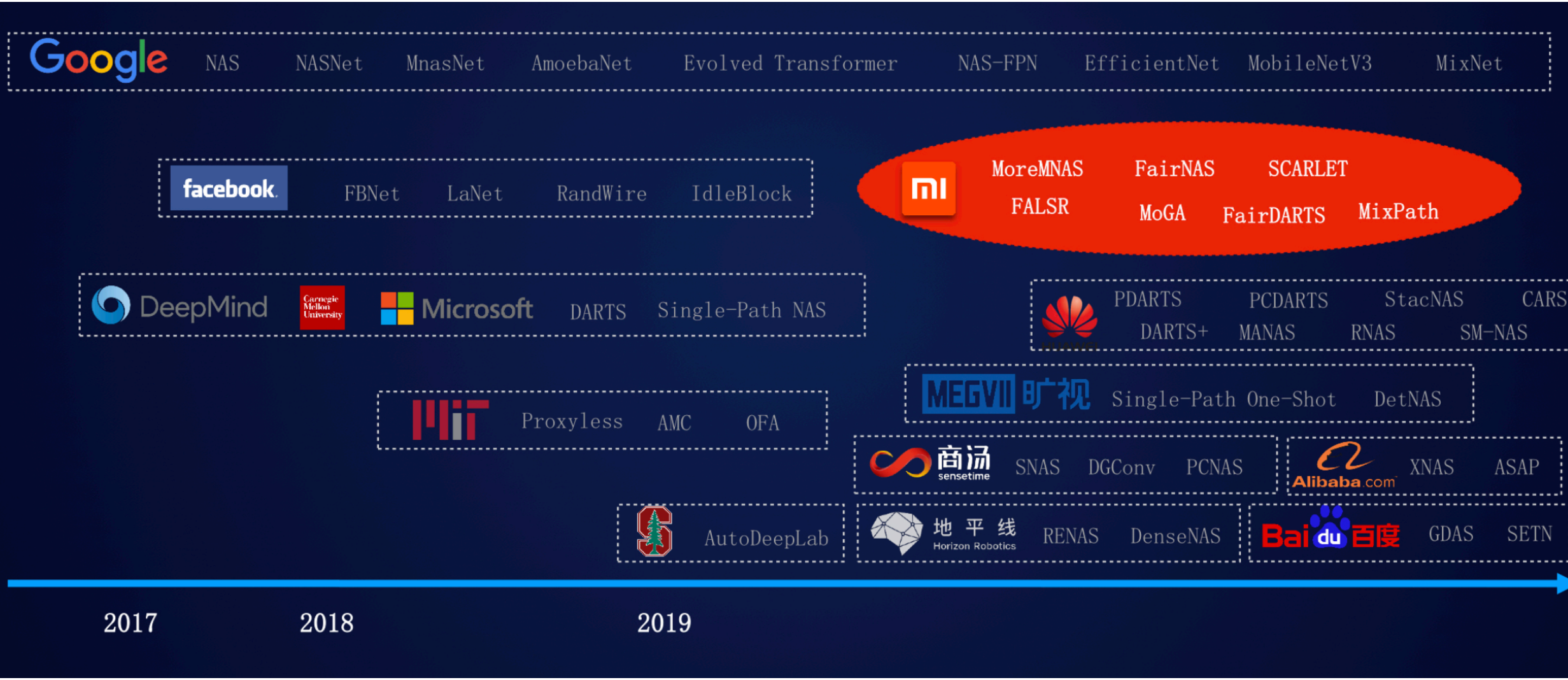
If we do not use a log-scale, the sampling will not be uniform within the given range. In other words, you should not attempt to sample values like:

Autogluon

- https://github.com/awslabs/autogluon

Panel

- Suggestions for Autogluon
- RL/ES/GB(gradient-based) NAS
- Reproducibility for NAS
- Pruning or NAS


Google
NAS
NASNet
MnasNet
AmoebaNet
Evolved Transformer
NAS-FPN
EfficientNet
MobileNetV3
MixNet
facebook
FBNet
LaNet
RandWire
IdleBlock
MoreMNAS
FairNAS
SCARLET
FALSR
MoGA
FairDARTS
MixPath
DeepMind
Carnegie Mellon University
Microsoft
DARTS
Single-Path NAS
PDARTS
PCDARTS
StacNAS
CARS
DARTS+
MANAS
RNAS
SM-NAS
MEGVII 旷视
Single-Path One-Shot
DetNAS
Proxyless
AMC
OFA
商汤 sensetime
SNAS
DGConv
PCNAS
Alibaba.com
XNAS
ASAP
AutoDeepLab
地平线 Horizon Robotics
RENAS
DenseNAS
Baidu 百度
GDAS
SETN
2017
2018
2019

Efficient Neural Architecture Search

Sun, Yue

Contents

- ENAS
- One-Shot
- DARTS
- SPOS
- FairNAS
- DARTS+
- Fair Darts
- Exploring Randomly Wired Neural Networks for Image Recognition

主流工作向可行性延伸

- One is the traditional searching approach, where each child model in search space runs as an independent trial, the performance result is sent to tuner and the tuner generates new child model.
- 实现方式RL、EA方式近似，每一个搜出来的模型都要traning form scratch 所以这种方式比较浪费时间
- We observe that the computational bottleneck of NAS is the training of each child model to convergence, only to measure its accuracy whilst throwing away all the trained weights.
- 权重被浪费掉了，好处是非常准确地估计每个模型的精度而且对每一个模型是公平的，相同setting下，缺点是浪费时间，NAS其实是一个多目标问题，时效性是限制准确性因素，
- 所以我们需要很牺牲一下搜索的精度换取时间的高效搜索，也想更加灵活地得到想要的网络结构（针对不同的目标）

- Computationally feasible 所以NAS的工作很多往计算可行性方向拓展：几类方法->对应的工作
- Lower fidelities 低保真度
- 数据集验证集的子集测试，train少量epoch估计就进行比较（但是这个不work->但是noah的一个工作证明了某些任务到后期精度是倒挂前面的模型）代理评测
- Parameters sharing 参数共享 （渐进的思想结构渐进复杂类似于不断finetune、更换连边的思想Enas）
- One-shot Architecture Search （定义搜索空间，Supernet超大网络结构，然后sample child model）The other is the so-called one-shot NAS, where a supernet is built based on search space, and using one shot training to generate good-performing child model.
- GB的方法，这个最著名的工作就是darts，darts不太稳定，没法证明双边优化到全局最优解
- 大家都是想用更快更好地方式实现每个trial都训练到底convergence的结果

- 发现一件事搜索空间设计就像之前dl的数据分布相关得问题，更好的搜索策略就像有更好的optimization的方法，更快的estimation的方法就是我们研究inference的过程怎样能更快在怎样的设备端部署。以前是利用更好的先验知识设计网络结构，现在是用会更加好的先验知识设计搜索空间
- 不公平比较，数据集/backbone/计算资源，Search space 公平比较（本身给予很多先验知识） -> NAS benchmark

ENAS

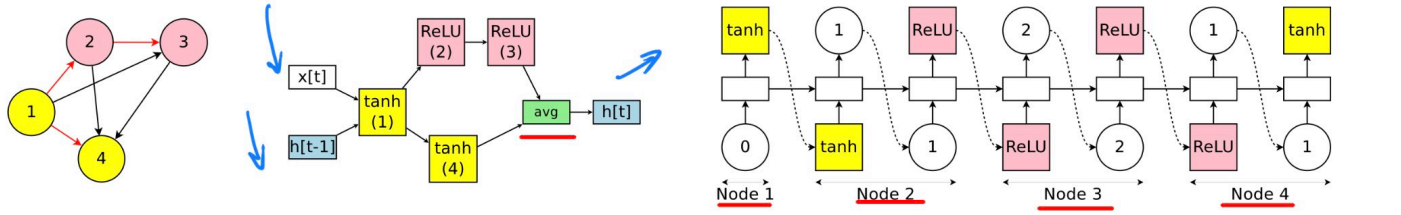


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that correspond to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of th controller RNN that result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

四个计算节点构成的有向无环图，未被采样的节点最后avg作为输出
Force all child models to share weights to eschew training each child model from scratch to convergence.
节点是运算，而边是信息流，节点固定，Enas要学习和挑选的就是节点之间的连线关系

- Enas的控制器是一个RNN，它决定哪些边被激活，在node是哪些计算，每一个节点的拓扑结构都是一个二叉树
- 训练分为两个阶段：
- 总体一个epoch：# epoch 150 每个epoch、完整实现一次结构搜索并且将acc得到
- Sample model and train # child_steps 500 在训练集上得到一个较好的weight 常规步骤
- Train sampler (mutator) 固定w 更新策略参数theta
- # mutator_step 50 在验证集上的step # mutator_steps_aggregate 20 RL控制器的一个微型批处理中将汇总的步骤数。
- 在enas有两组参数一个控制lstm的参数theta 决定控制器策略和模型权重w，model从theta的函数中采样，可以采样任意一个结构去更新梯度
- 评估模型时用的一组权重，在未被训练的验证集中测试，取AUC或者准确率最高的结构，权重会在每个epoch更新，只对一小批验证图像进行测试
- RNN对每个decision block 进行采样，连接哪些节点以及 使用什么op
- w都是随机初始化，并在每个神经网络架构中从头开始训练的。在ENAS，这些参数是所有神经网络架构共享的。如果下一次controller得到的神经网络架构如下，它参数由w 与上面神经网络架构相同的。

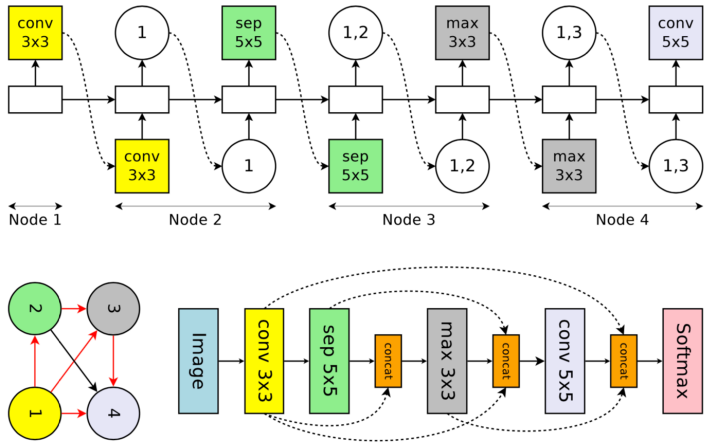
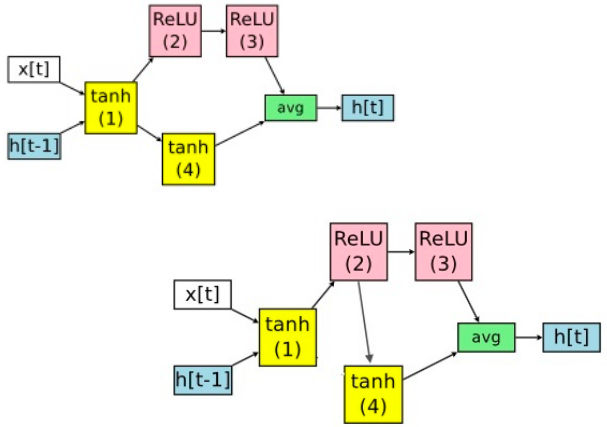


Figure 3. An example run of a recurrent cell in our search space with 4 computational nodes, which represent 4 layers in a convolutional network. *Top:* The output of the controller RNN. *Bottom Left:* The computational DAG corresponding to the network's architecture. Red arrows denote the active computational paths. *Bottom Right:* The complete network. Dotted arrows denote skip connections.



Darts

First to formulate NAS task in a differentiable manner

Continuous relaxation of the architecture representation
Efficient search of the architecture using gradient descent

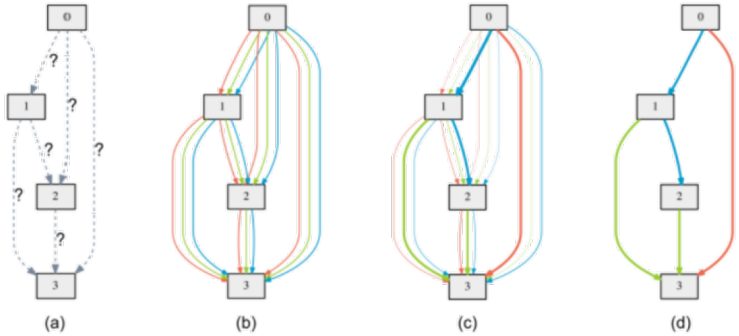


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem.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

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 α 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)

$$x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)}) \qquad \bar{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

- Darts (可微分架构搜索) 将结构参数和权重系数weight转化为 双层优化问题
- 本文使用连续松弛将离散的架构搜索空间转化为连续可微的搜索空间，从而可以使用梯度下降实现高效搜索，并可用于搜索卷积架构和循环架构（当时流行的搜索方式基本都是离散的搜索方式，如强化学习和进化算法等），但是本文依然是通过搜索Cell然后叠加来搜索架构。
- 定义搜索空间：搜索Cell，每个Cell是一个包含N个顺序节点的有向无环图，每个节点可看作是一张特征图的潜在表示，每条边即是一种实现节点转换的操作。于是将学习Cell的任务转换为学习它的每一边的操作。
- 连续松弛、优化和近似：
- （1）连续松弛：使用如下公式将离散搜索空间连续化。我理解为将每一条边看成一个混合的操作集（即本来是一条边，但是将所有的可选操作都作为边添加上去，并在边上添加概率值），训练的目标即变成了学习每一条边操作的权重和概率值，训练好了以后，保留最大概率的边即可得到最终所需的网络架构。
- （2）优化：因为现在需要同时学习每一个操作中的权重和这个操作保留下来的概率，从而变成了一个双优化的问题，如下公式所示。
- （3）近似：由于这种双优化的问题很难求得精确解（因为需要反复迭代求解两个参数），所以采用一种近似的迭代优化步骤来交替更新两个参数，算法如下。

优势速度快但是结果不稳定，这类算法的缺点在Darts+和fair darts中有详细的分析

One-Shot

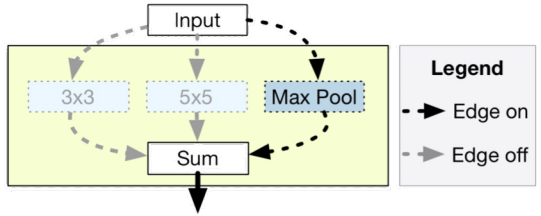


Figure 1. Example building block used in a one-shot model. The search space contains three different operations; the one-shot model adds their outputs together. Some of the operations are removed or zeroed out at evaluation time.

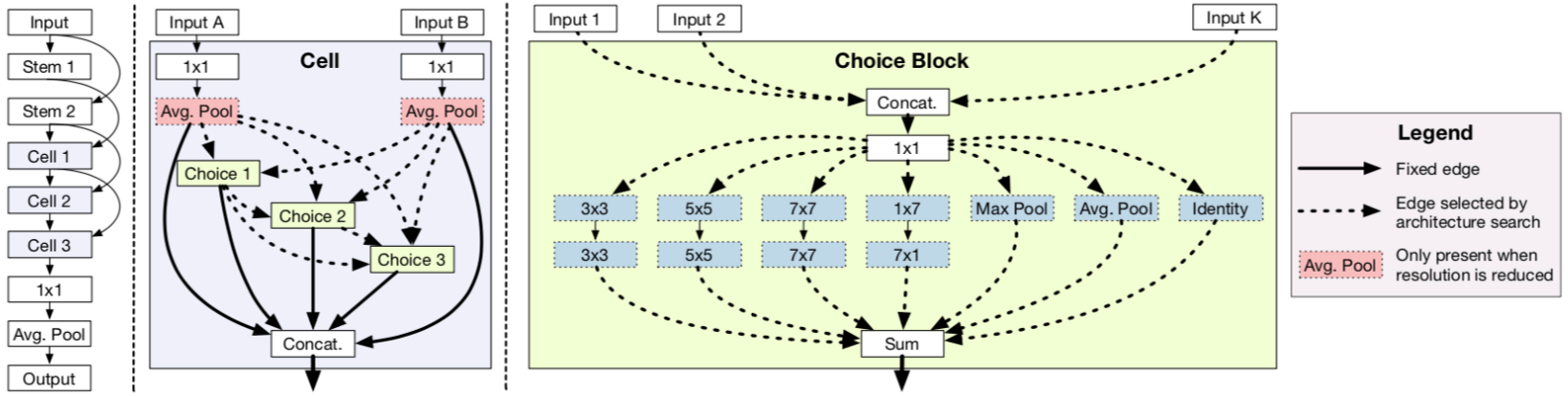


Figure 3. Diagram of the one-shot architecture used in our experiments. Solid lines indicate components that are present in every architecture, while dashed lines indicate optional components that are part of the search space.

- One-Shot: One promising direction is sharing weights between models rather than training thousands of separate models from scratch, one can train a single large network capable of emulating any architecture in the search space.
- One-Shot模型的一个例子如图所示，我们可以在网络的特定位置使用3x3卷积、5x5卷积或最大池层三种操作的一种。不同于训练三个单独的模型，one-shot方法可以训练包含三种操作的单个模型。在评估时，可以选择将其中两种操作进行置零来确定哪种操作的预测精度最高。
- 中间的三个Cell是相同的，每个Cell的构造如上图第二部分所示，但是在本文的实验中作者设定的Choice Block为4，图中少画了一个。每个Choice Block的构造如上图第三部分所示，设定每个Choice Block只能接受最多两个最近Cell 的输入，或者同一个Cell中其他Choice Block的输入，所以每个Choice Block可接受最少1个输入，最多7个输入。每个Choice Block中有7个可选操作，最多每次选2个操作，最少每次选1个操作。
- 使用One-Shot模型比较灵活，将结构参数与模型训练解耦，设计搜索空间-训练supernet评估-选出有潜力的子模型从头训练再评估。
- 考虑模型的鲁棒性：因为评估的时候会去掉大量的分支，只留下所选择的路径，所以模型需要具备去掉分支后进行评估的鲁棒性，在训练的过程中使用Dropout来解决这一问题。

SPOS

- *First*, the weights in the supernet are **deeply coupled**. It is unclear why **inherited weights for a specific architecture are still effective**.
- *Second*, joint optimization introduces further coupling between the architecture parameters and supernet weights. **The greedy nature of the gradient based methods inevitably introduces bias in both architecture distribution and supernet weights**. This easily misleads the architecture search.
- As firstly observed in [2], the key to the success of one-shot is that **the accuracy of an architecture using inherited weights should be predictive for the accuracy using optimized weights.** Thus, we propose that the supernet training should be stochastic, in that all architectures can have their weights optimized simultaneously.
- 继承超网权重的复用模型可以和train from scratch的模型精度相当
- To reduce the weight coupling in the supernet, we propose using a simple search space**, single path supernet**, that only contains single path architectures. **For training, we use a hyperparameter-free method, uniform sampling, to treat all architectures equally** . (均匀分布采样)

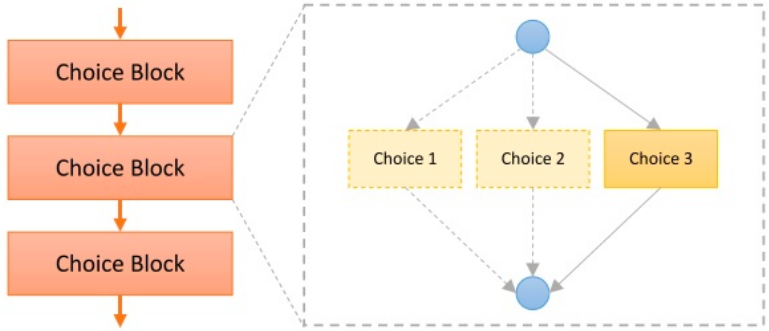
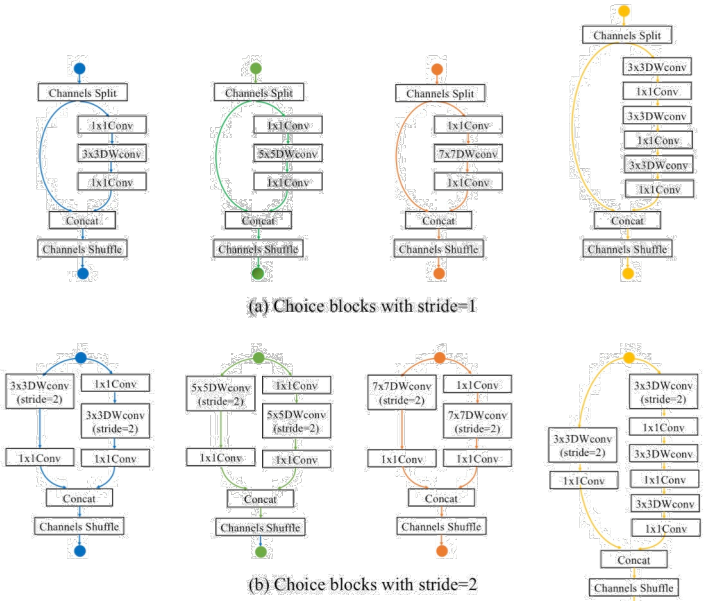


Figure 1. Architecture of a single path supernet. It consists of a series of *choice blocks*. Each has several *choices*. Only one choice is invoked at the same time.



- 搜索空间基础的搜索块参照ShuffleNet v2设计，**权重优化和架构参数搜索这一双优化问题解耦分成两个步骤来进行 ->用supernet再根据不同任务单独sample chlid model更加高效**
- The evolutionary algorithm is flexible in dealing with different constraints in Eq. (3), because the mutation and crossover processes can be directly controlled to gener- ate proper candidates to satisfy the constraints.
- 使用进化算法选择网络架构时可以精准满足不同的约束（如FLOPs和Latency等）。多目标优化，想flops、latency这类约束不需要花费太长时间

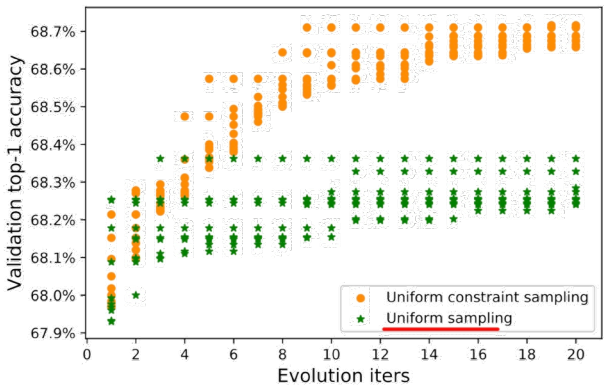


Figure 2. Evolutionary architecture search (see Sec. 3.4) on the single path supernets with different sampling strategies.

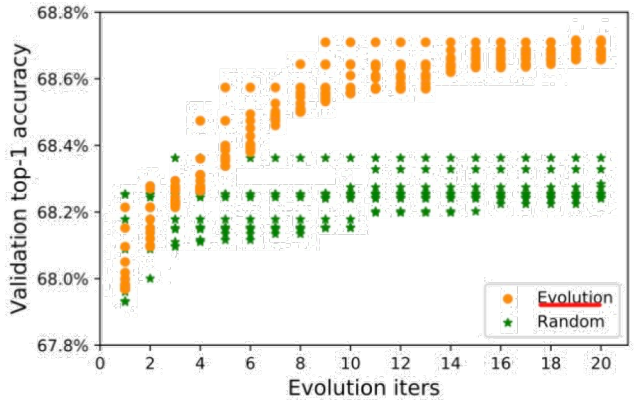


Figure 3. Evolutionary vs. random architecture search. See Sec. 3.4 and 4 for details.

FairNAS

- Firstly, we prove it is due to unfair bias that **the supernet misjudges submodels' performance.** It is inevitable in current one-shot approaches.

练的比较充分。因此当我们使用single path策略训练supernet时，一个基本的要求是，每个子模型有相同的概率被采样和训练，SPOS中的采样策略可以满足这一点，即设某一层中有m个候选模块，我们一共（迭代）采样n次，设某一层中的某个候选操作 x_{li} 被选中的次数为 Y_{li}，那么 Y_{li} 很显然服从二项分布 $B(n, \frac{1}{m})$ 。

但是仅仅满足期望公平是不够的，for example，设某一层中只有2个候选操作，在n次采样之后，这两个候选操作被选择的次数分别为 Y_{l1} 和 Y_{l2} ，那么 $P(Y_{l1} = Y_{l2}) = C_n^{\frac{n}{2}} * (\frac{1}{2})^n$ ，那么可以证明，当 $n \to +\infty$ 时 $P(Y_{l1} = Y_{l2}) \to 0$ 。即期望公平无法保证严格的采样次数相等。

此外，以前的类似方法中，每采样一次之后就立刻使用BP算法更新权值，所以即使每个block被采样的次数是完全一样的，但如果采样顺序不同，最终得到的权值也不同，如果学习率是变化的，结果会更加复杂。

- The supernet gets its weights updated after accumulating gradients from each single-path model. All operations are thus ensured to be **equally sampled and trained within every step t.**
- (1)对于每个layer有m个候选操作的supernet，每次不放回地抽取m个模型，保证每层中的每个候选操作都被采样且仅被采样一次（使得每个候选操作被采样的频率严格相等）。
- (2)每一次采样完毕之后，计算并叠加对应权重的反传梯度但是并不更新，等到m个候选操作都被采样一遍之后，再更新梯度（解决了采样顺序带来的问题）。
- 显然这种采样方式比SPOS中的采样方式要公平地多，但是在network的层面上说，不同的network被采样出的频率只能满足期望公平，作者也提到：完完全全的公平是非常困难的。

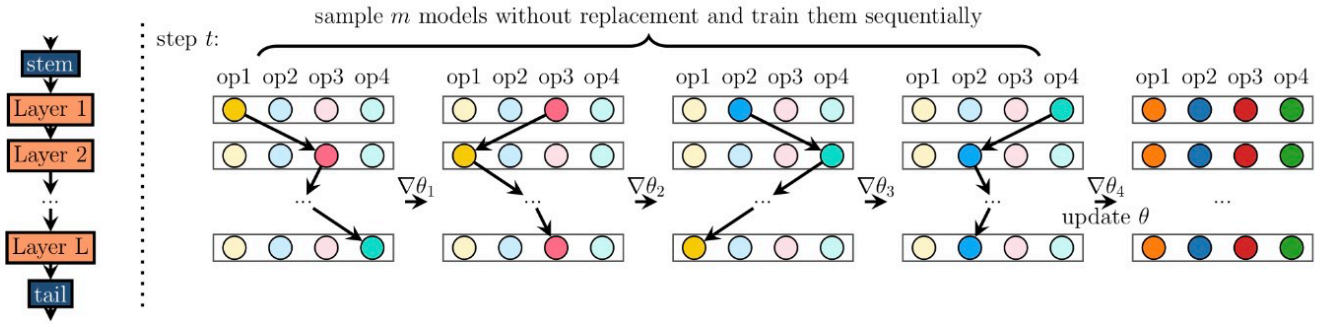
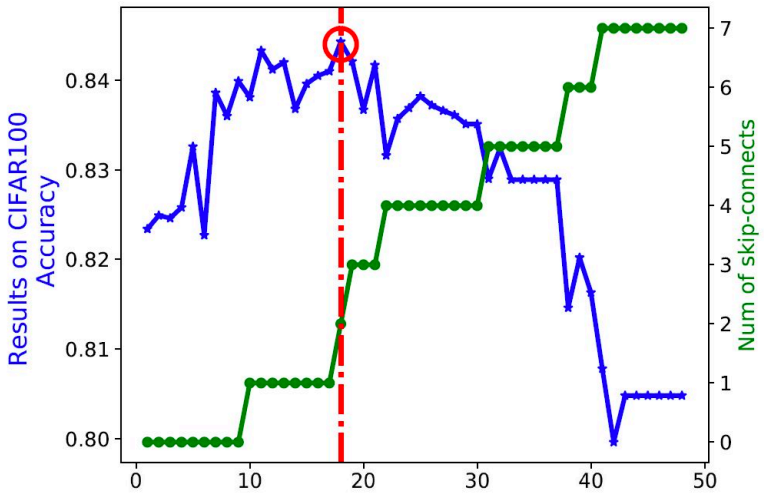
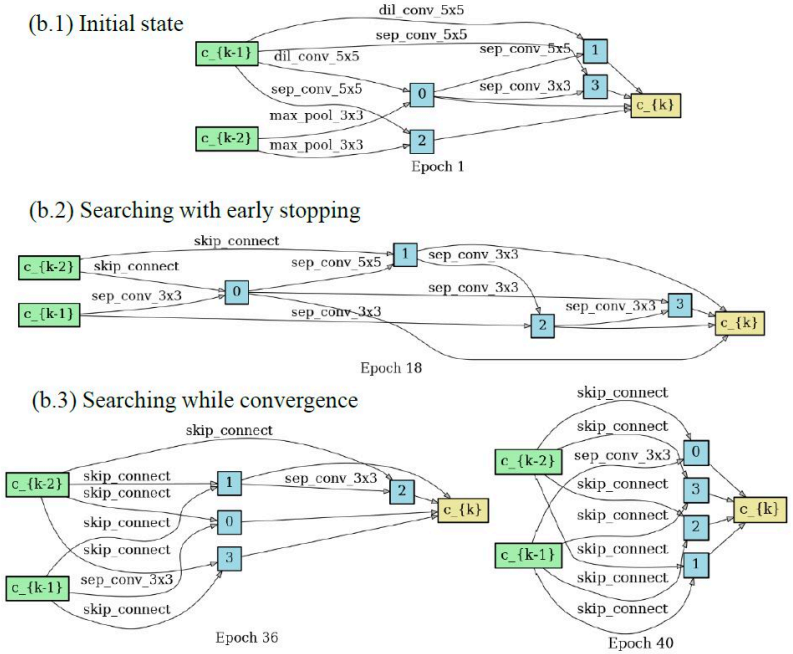


Fig. 2. Our *strict fairness sampling and training* strategy for supernet. A supernet training step t consists of training m models, each on one batch of data. The supernet gets its weights updated after accumulating gradients from each single-path model. All operations are thus ensured to be equally sampled and trained within every step t. There are $(6!)^{18}$ choices per step in our experiments[2]

DARTS+





- DARTS 使用了如下的两层优化（Bi-Level Optimization）来搜索
- Collapse of DARTS: 当搜索轮数过大时，搜索出的架构中会包含很多的 skip-connect，从而性能会变得很差。
- 举个例子，让我们来考虑在 CIFAR100 上用 DARTS 做搜索。从下图可以看出，当 search epoch（横轴）比较大的时候，skip-connect 的 alpha 值（绿线）将变得很大。在 DARTS 最后选出的网络架构中，skip-connect 的数量也会随着 search epoch 变大而越来越多。
- 在一个节点数固定的 cell 中，skip-connect 的数量越多，会导致网络变得越浅。相比于深度网络，浅度网络可学习的参数更少，具有的表达能力更弱。因此，在 DARTS 搜出的网络架构中，skip-connect 的数量太多会导致性能急剧变差。例如，在上图中，当 skip-connect 的数量超过 2 个的时候，网络的性能（蓝线）开始降低。下图直观展示了随着 search epoch 变大，网络结构由深变浅的过程。
- cooperation vs competition : DARTS 发生 Collapse 背后的原因是在两层优化中，alpha 和 w 的更新过程存在先合作（cooperation）后竞争（competition）的问题。粗略来说，在刚开始更新的时候，alpha 和 w 是一起被优化，从而 alpha 和 w 都是越变越好。渐渐地，两者开始变成竞争关系，由于 w 在竞争中比 alpha 更有优势（比如，w 的参数量大于 alpha 的参数量，One-Shot 模型在大多数 alpha 下都能收敛，等等），alpha 开始被抑制，因此网络架构出现了先变好后变差的结果，也就是上图中蓝线的情况。
- 在搜索过程的初始阶段，One-Shot 模型欠拟合到数据集，因此在搜索过程刚开始的时候，alpha 和 w（也就是 One-Shot 模型的参数）都会朝着变好的方向更新，这就是合作的阶段。由于整个 One-Shot 模型中，前面的 cell 比后面的 cell 能接触到更干净的数据，如果我们允许不同的 cell 可以拥有不同的网络结构（打破 DARTS 中 cell 共享网络结构的设定），那么前面的 cell 会比后面的 cell 更快地学到特征。一旦前面的 cell 已经学到了不错的特征表达，而后面的 cell 学到的特征表达相对较差，那么后面的 cell 接下来会倾向于选择 skip-connect，来把前面 cell 已经学好的特征表达直接传递到后面。下图是打破 DARTS 中 cell 共享网络结构的设定下，搜出来的网络结构图：可以看到，前面的 cell 大部分都是卷积算子，而靠后的 cell 大部分都是 skip-connect。
- **早停准则** *当一个 normal cell 中出现两个及两个以上的 skip-connect 的时候，搜索过程停止。*

Fair DARTS

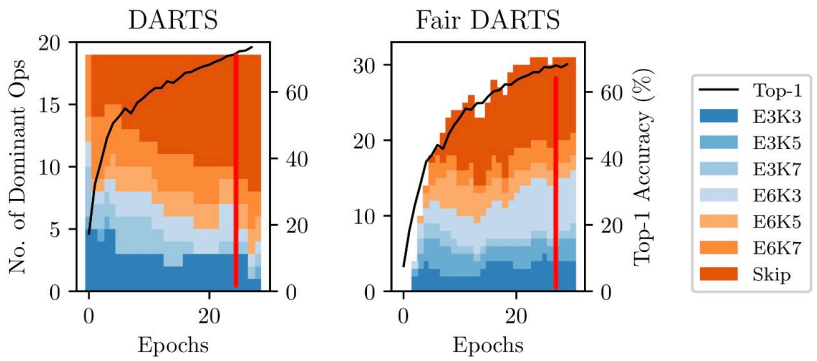


Figure 1. The number of dominant operations (in all 19 layers) of DARTS and Fair DARTS searching on ImageNet (in search space S_2). In DARTS, skip connections incrementally suppress others. In Fair DARTS, all operations develop independently.

- First, skip connections during optimization is caused by an **unfair advantage in an exclusive competition**.
- Insight 1: The root cause of excessive skip connections is the inherent unfair competition. The skip connection has an unfair advantage **by forming a residual module** which is convenient for the training of supernet, but **not equally beneficial for the performance of the outcome network.**

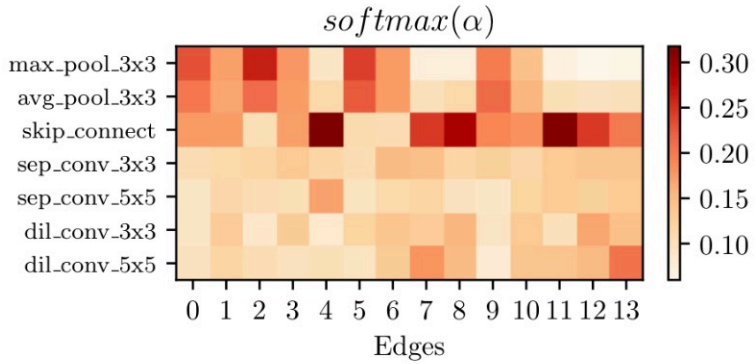


Figure 4. Heatmap of softmax values in the last searching epoch when running DARTS on CIFAR-10 (in search space S_1).

- Second, there is a **non-negligible incongruence** when discretizing continuous architectural weights to a one-hot representation. Relaxing **from discrete categorical choices to continuous** ones should make a good approximation
- We run DARTS on CIFAR-10 and display softmax(α) of the last iteration in Figure 4. **The largest value is about 0.3 while the smallest one is above 0.14.** On one hand, it is not a good approximation to meet Equation 3, i.e., **this leads to a non-negligible gap between the derived model and its supernet**, as observed in Section 3.2. In the worst cases, the gap becomes so large that it turns into pure noise for choice selection.
- On the other hand, the range is too narrow to differentiate between 'good' and 'bad' operations. For instance in Figure 4, the top-2 highest values are very close on the 1st, 3rd, 6th, and 9th edge. **It's hard to say that an edge weighted by 0.26 is better than another by 0.24.**

Fair DARTS

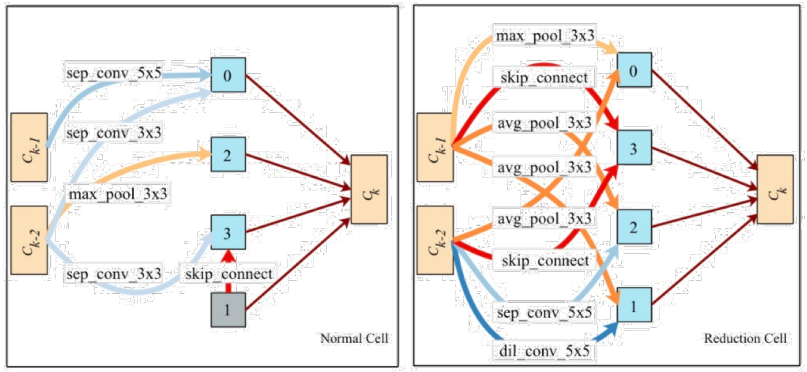


Figure 7. FairDARTS-a cells searched on CIFAR-10 in S_1.

- Stepping out the Pitfalls of Skip Connections by **Cooperation**
- Based on Insight 1, we propose a **cooperative mechanism** to eliminate the existing unfair advantage. Not only should we exploit skip connection for smoother information flow, but we also have to provide equal opportunities for other operations. In a word, they need to avoid being trapped by unfair advantage from skip connections.
- On this regard, we apply a **sigmoid activation** for each αi,j, so that each operation can be switched on or off independently without being suppressed.

- Like DARTS [18], the architectural weight can be optimized through back propagation. From Equation 8, the search objective is to find an architecture of high accuracy with **a good approximation from continuous to discrete**.

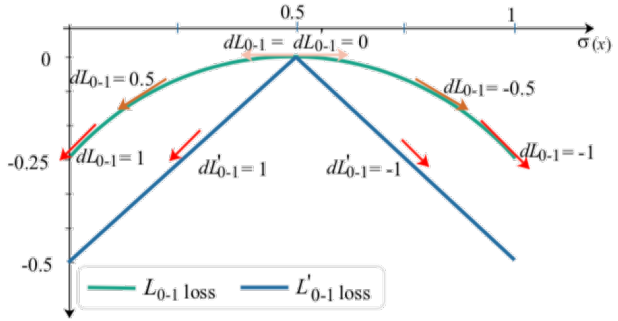


Figure 6. Illustration about the auxiliary loss design: L_{0-1} (proposed) and L'_{0-1} (control).

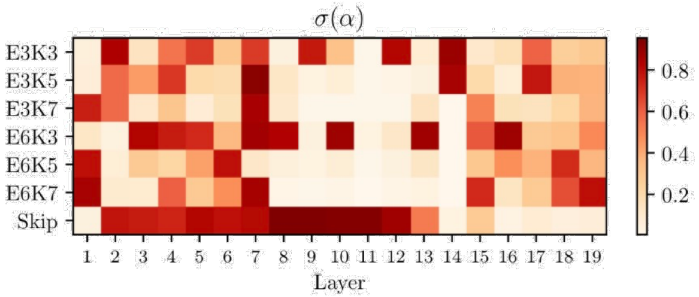
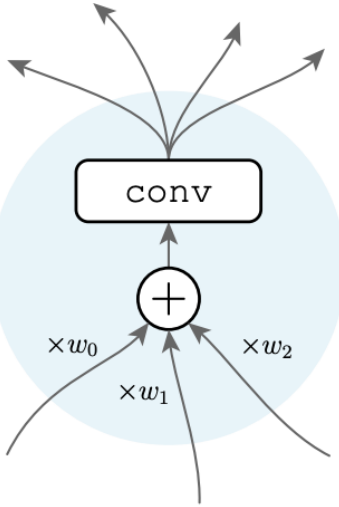


Figure 10. Heatmap of sigmoid values in the last searching epoch when running Fair DARTS in S_2. Many operations other than skip connections are also activated in parallel.

- Like resnet : short cut
- To be fair, we select **at most two choices per layer** if there are more than two **above σ>threshold(0.75)** and use the same training tricks as [27]. We exclude squeeze and excitation [11] and refrain from using AutoAugment [6] tricks though they can boost the classification accuracy further.

Exploring Randomly Wired Neural Networks for Image Recognition

- NAS (Search space) -> Better Network Generator vs Better Network
- Edge operations
- Node operations : 3 input/4 output， learnable positive (w0、w1、w2) ->Aggregation.
- Transformation : ReLU-convolution-BN has the same numble of input/output channels(useless switching stages)
- Input and output nodes : no convolution， compute (unweighted) average form all nodes.



stage	output	*small regime*	*regular regime*
conv_1	112×112	3×3 conv, $C/2$	
conv_2	56×56	3×3 conv, C	***random wiring*** $N/2, C$
conv_3	28×28	***random wiring*** N, C	***random wiring*** $N, 2C$
conv_4	14×14	***random wiring*** $N, 2C$	***random wiring*** $N, 4C$
conv_5	7×7	***random wiring*** $N, 4C$	***random wiring*** $N, 8C$
classifier	1×1	1×1 conv, 1280-d global average pool, 1000-d *fc*, softmax	

Table 1. **RandWire architectures** for small and regular computation networks. A random graph is denoted by the node count (N) and channel count for each node (C). We use conv to denote a ReLU-Conv-BN triplet (expect conv_1 is Conv-BN). The input size is 224×224 pixels. The change of the output size implies a stride of 2 (omitted in table) in the convolutions that are right after the input of each stage.

- Random Graph Models -> （DAG）
- Erdos-Renyi（ER）、Barabasi-Albert（BA）、Watts-Strogatz（WS）

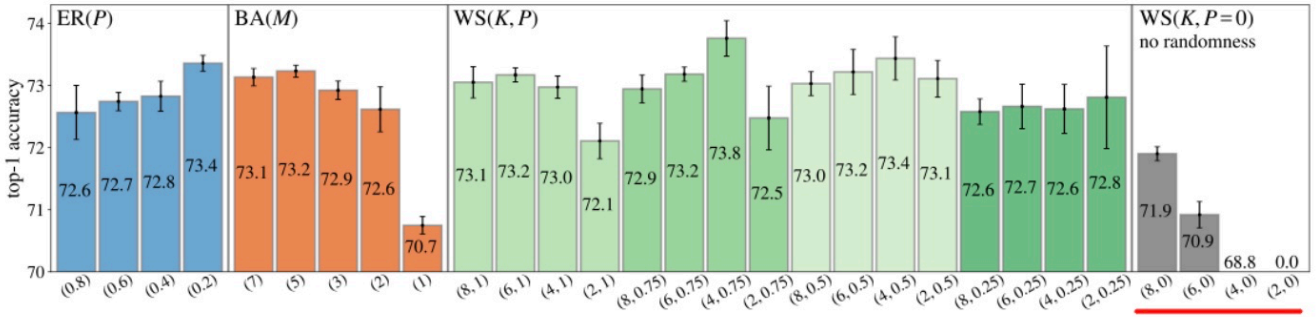


Figure 3. **Comparison on random graph generators: ER, BA, and WS** in the small computation regime. Each bar represents the results of a generator under a parameter setting for P, M, or (K, P) (tagged in x-axis). The results are ImageNet top-1 accuracy, shown as mean and standard deviation (std) over 5 random network instances sampled by a generator. At the rightmost, WS$(K, P{=}0)$ has no randomness.

Exploring Randomly Wired Neural Networks for Image Recognition

- We explored randomly wired neural networks driven by **three classical random graph models** from graph theory.
- The results were surprising: the mean accuracy of these models is competitive with hand-designed and optimized models from recent work on neural architecture search. Our exploration was enabled by the novel concept of a network generator. We hope that **future work exploring new generator designs may yield new, powerful networks designs.**

network	top-1 acc.	top-5 acc.	FLOPs (M)	params (M)
MobileNet [15]	70.6	89.5	569	4.2
MobileNet v2 [40]	74.7	-	585	6.9
ShuffleNet [54]	73.7	91.5	524	5.4
ShuffleNet v2 [30]	**74.9**	92.2	591	7.4
NASNet-A [56]	74.0	91.6	564	5.3
NASNet-B [56]	72.8	91.3	488	5.3
NASNet-C [56]	72.5	91.0	558	4.9
Amoeba-A [34]	74.5	92.0	555	5.1
Amoeba-B [34]	74.0	91.5	555	5.3
Amoeba-C [34]	**75.7**	**92.4**	570	6.4
PNAS [26]	74.2	91.9	588	5.1
DARTS [27]	73.1	91.0	595	4.9
RandWire-WS	**74.7**$_{\pm 0.25}$	**92.2**$_{\pm 0.15}$	583$_{\pm 6.2}$	5.6$_{\pm 0.1}$

Table 2. **ImageNet: small computation regime** (*i.e.*, <600M FLOPs). RandWire results are the mean accuracy (±std) of 5 random network instances, with WS$(4, 0.75)$. Here we train for 250 epochs similar to [56, 34, 26, 27], for fair comparisons.

network	top-1 acc.	top-5 acc.	FLOPs (B)	params (M)
ResNet-50 [11]	77.1	93.5	4.1	25.6
ResNeXt-50 [52]	78.4	94.0	4.2	25.0
RandWire-WS, C=109	**79.0**$_{\pm 0.17}$	**94.4**$_{\pm 0.11}$	4.0$_{\pm 0.09}$	31.9$_{\pm 0.66}$
ResNet-101 [11]	78.8	94.4	7.8	44.6
ResNeXt-101 [52]	79.5	94.6	8.0	44.2
RandWire-WS, C=154	**80.1**$_{\pm 0.19}$	**94.8**$_{\pm 0.18}$	7.9$_{\pm 0.18}$	61.5$_{\pm 1.32}$

Table 3. **ImageNet: regular computation regime** with FLOPs comparable to ResNet-50 (top) and to ResNet-101 (bottom). ResNeXt is the 32×4 version [52]. RandWire is WS$(4, 0.75)$.

network	test size	epochs	top-1 acc.	top-5 acc.	FLOPs (B)	params (M)
NASNet-A [56]	331^2	>250	82.7	96.2	23.8	88.9
Amoeba-B [34]	331^2	>250	82.3	96.1	22.3	84.0
Amoeba-A [34]	331^2	>250	82.8	96.1	23.1	86.7
PNASNet-5 [26]	331^2	>250	82.9	96.2	25.0	86.1
RandWire-WS	320^2	100	81.6$_{\pm 0.13}$	95.6$_{\pm 0.07}$	16.0$_{\pm 0.36}$	61.5$_{\pm 1.32}$

Table 4. **ImageNet: large computation regime**. Our networks are the same as in Table 3 (C=154), but we evaluate on 320×320 images instead of 224×224. Ours are only trained for 100 epochs.

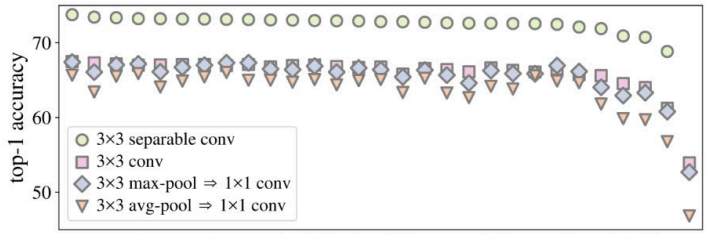


Figure 6. **Alternative node operations**. Each column is the mean accuracy of the same set of 5 random graphs equipped with different node operations, sorted by "3×3 separable conv" (from Figure 3). The generators roughly maintain their orders of accuracy.

Node operations. Thus far, all models in our experiment use a 3×3 separable convolution as the "conv" in Figure 2. Next we evaluate alternative choices. We consider: (i) 3×3 (regular) convolution, and (ii) 3×3 max-/average-pooling followed by a 1×1 convolution. We replace the transformation of *all nodes* with the specified alternative. We adjust the factor C to keep the complexity of all alternative networks.

Figure 6 shows the mean accuracy for each of the generators listed in Figure 3. Interestingly, almost all networks still converge to non-trivial results. Even "3×3 pool with 1×1 conv" performs similarly to "3×3 conv". The network generators roughly maintain their accuracy ranking despite the operation replacement; in fact, the Pearson correlation between any two series in Figure 5 is 0.91~0.98. This suggests that the network wiring plays a role somewhat orthogonal to the role of the chosen operations.

One-Shot Model Neural Architecture Search

Sun, Yue