Neural Architecture Search

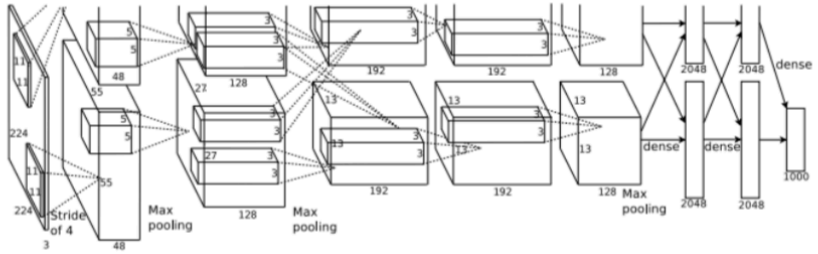
Sun, Yue

Contents

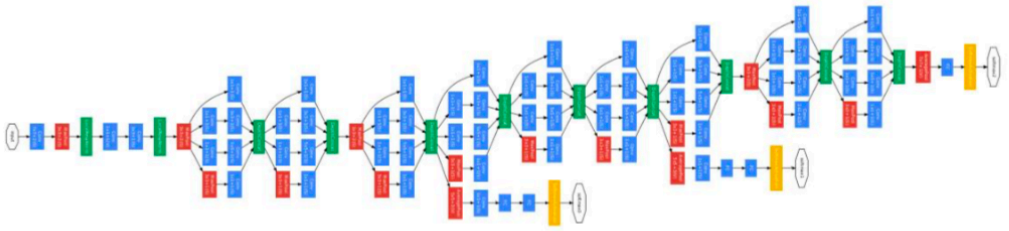
- NAS
- HPO
- Autogluon

The Architecture of a Neural Network is Crucial to its Performance

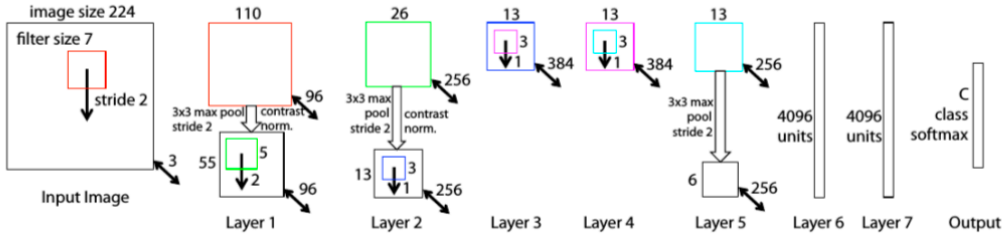
- ImageNet Winning Neural Architectures



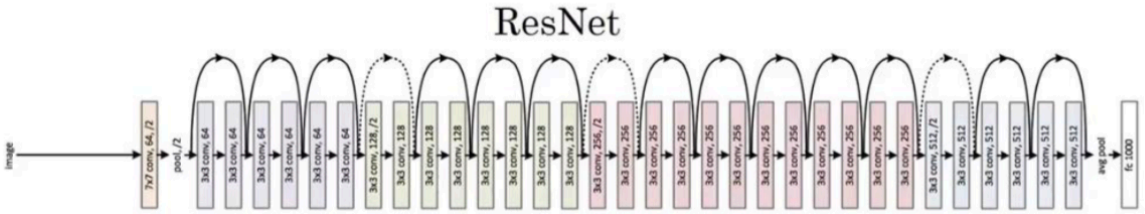
AlexNet 2012



Inception 2014



ZFNet 2013

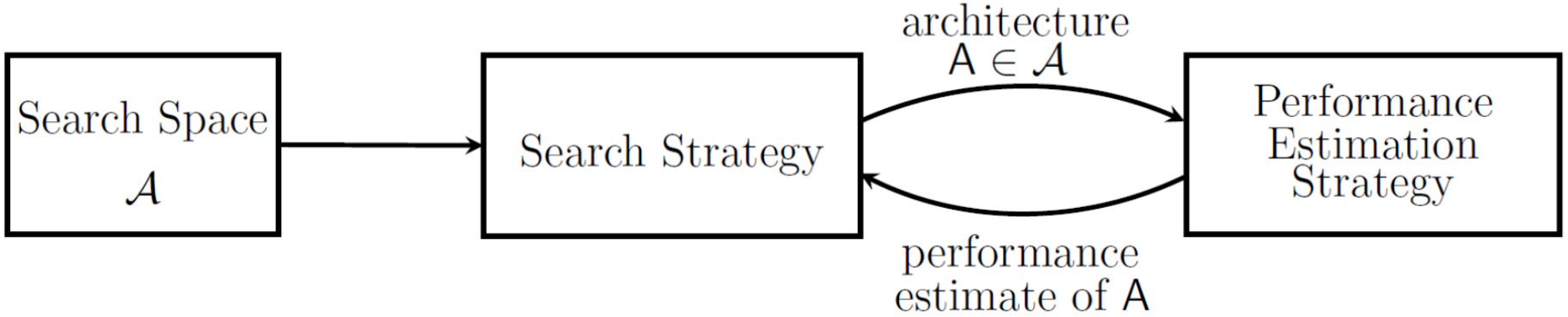


ResNet 2015

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.

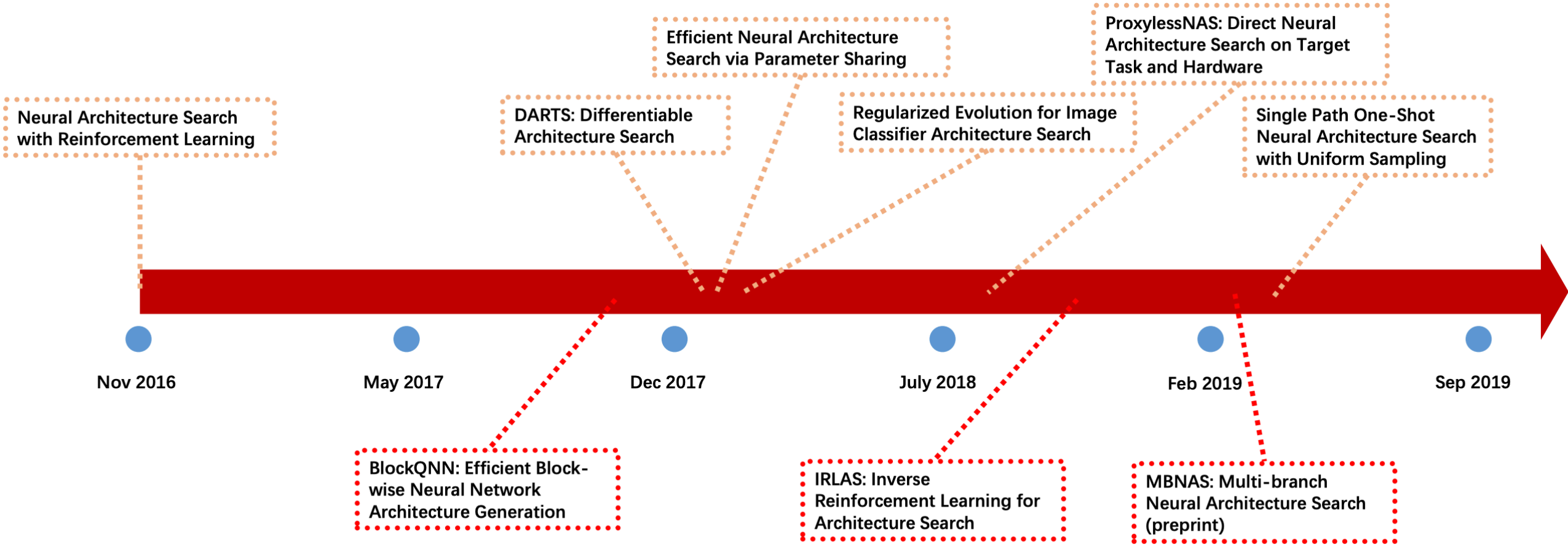
Search Strategy: the search strategy details how to explore the search space.

- *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)

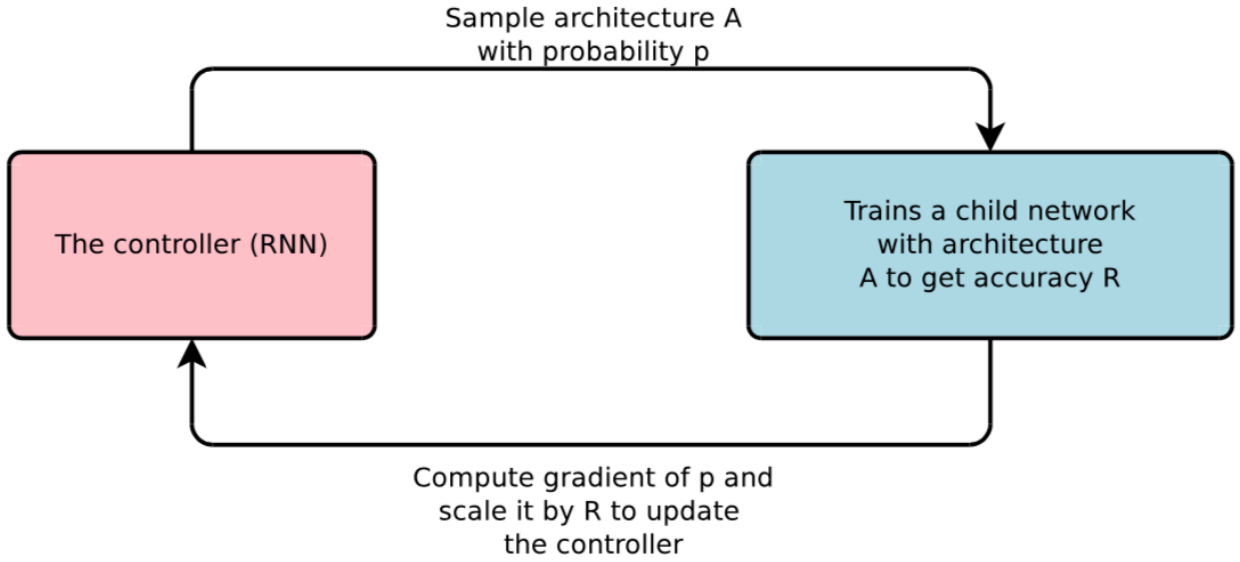


Figure 1: An overview of Neural Architecture Search.

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.

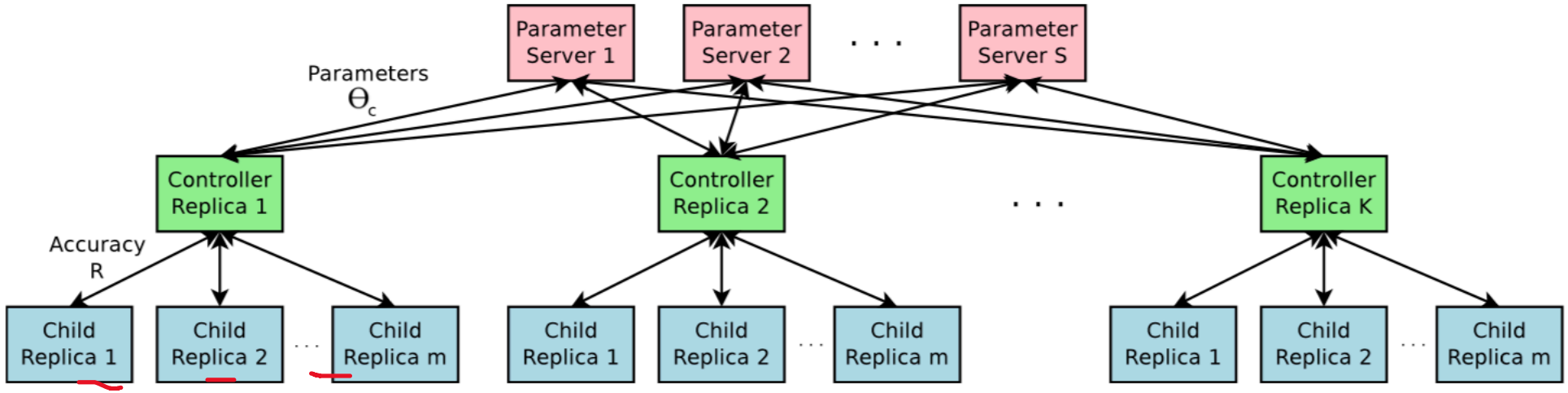


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to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

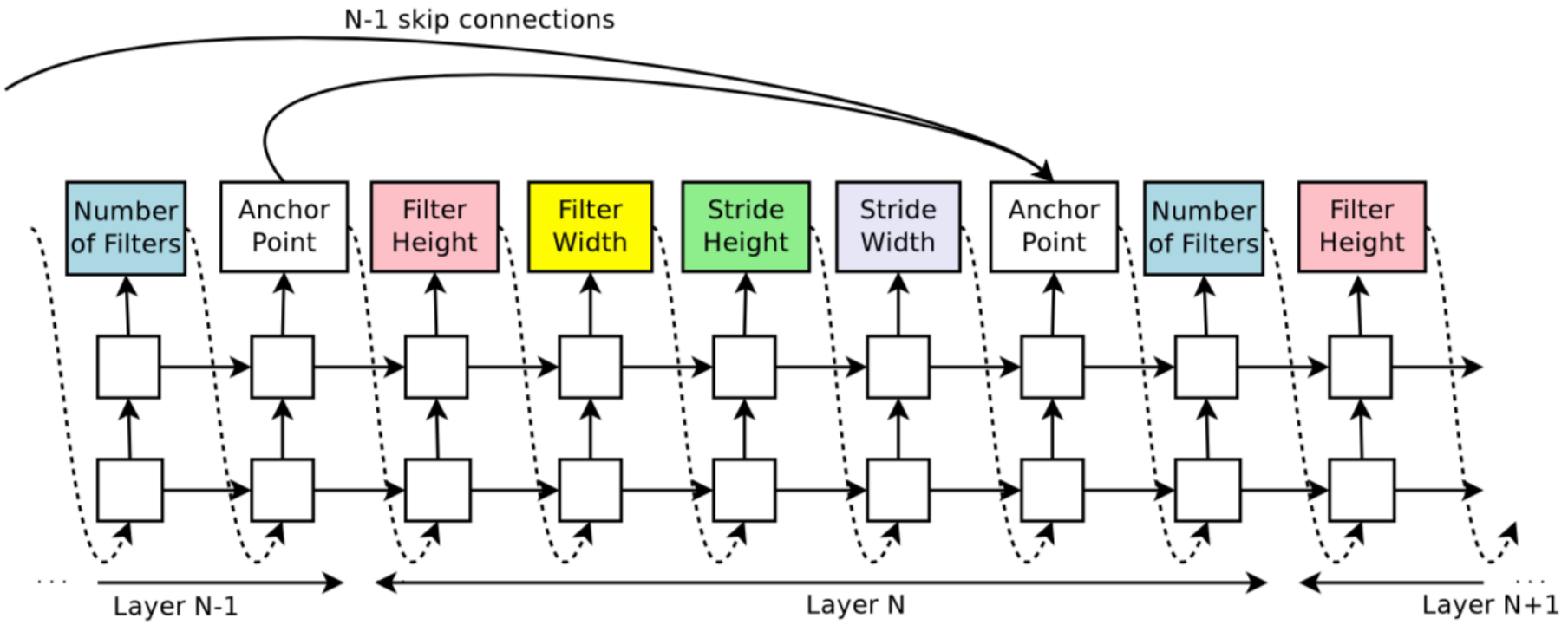


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

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

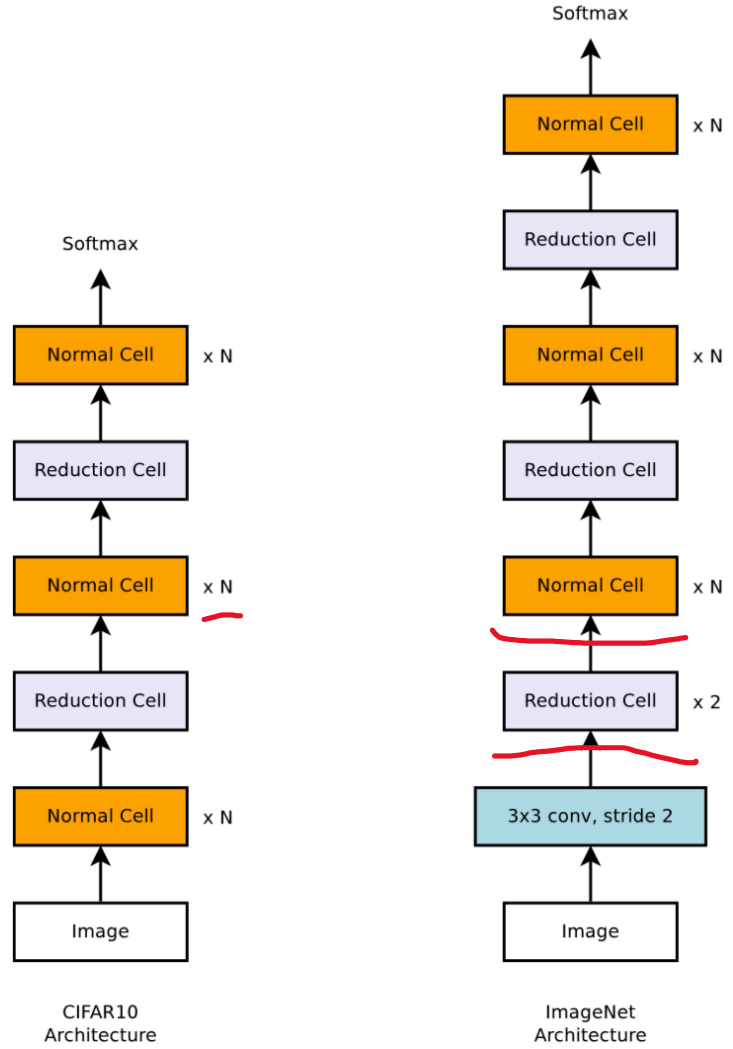
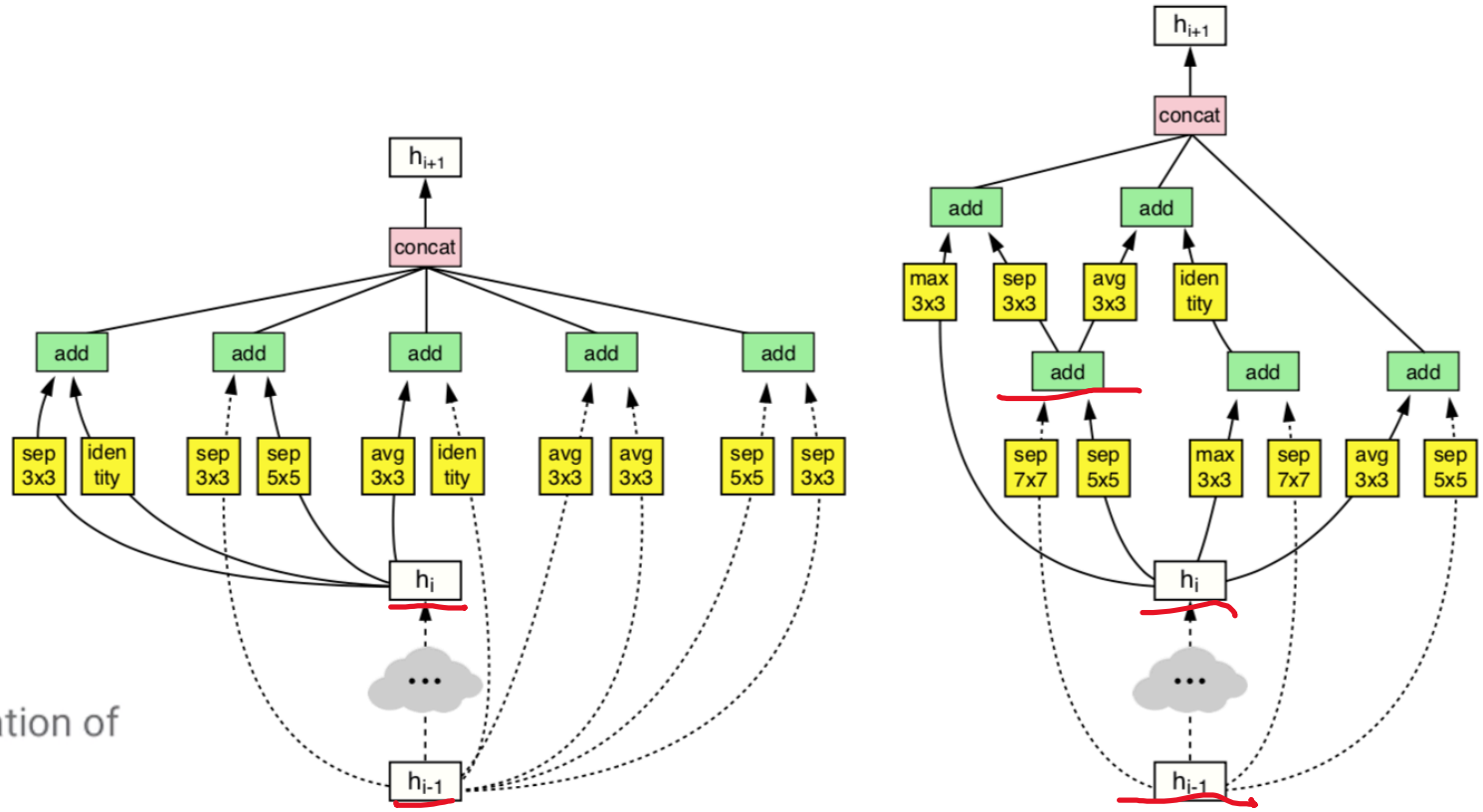


Figure 2. Scalable architectures for image classification consist of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells,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

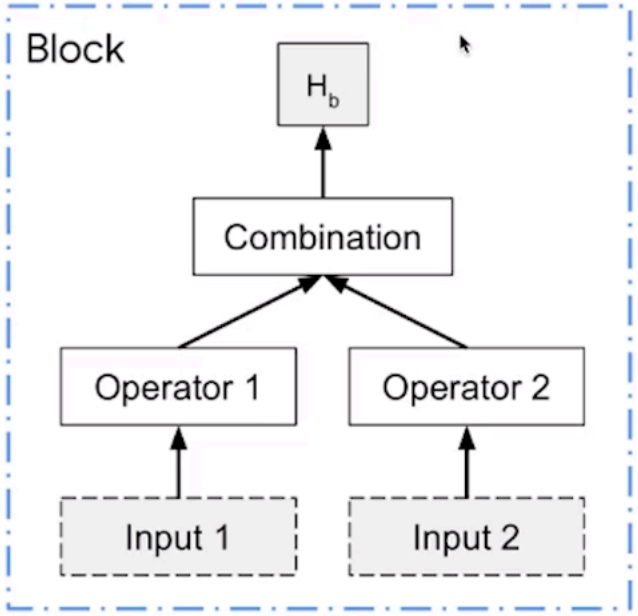


Normal Cell *Reduction Cell*

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



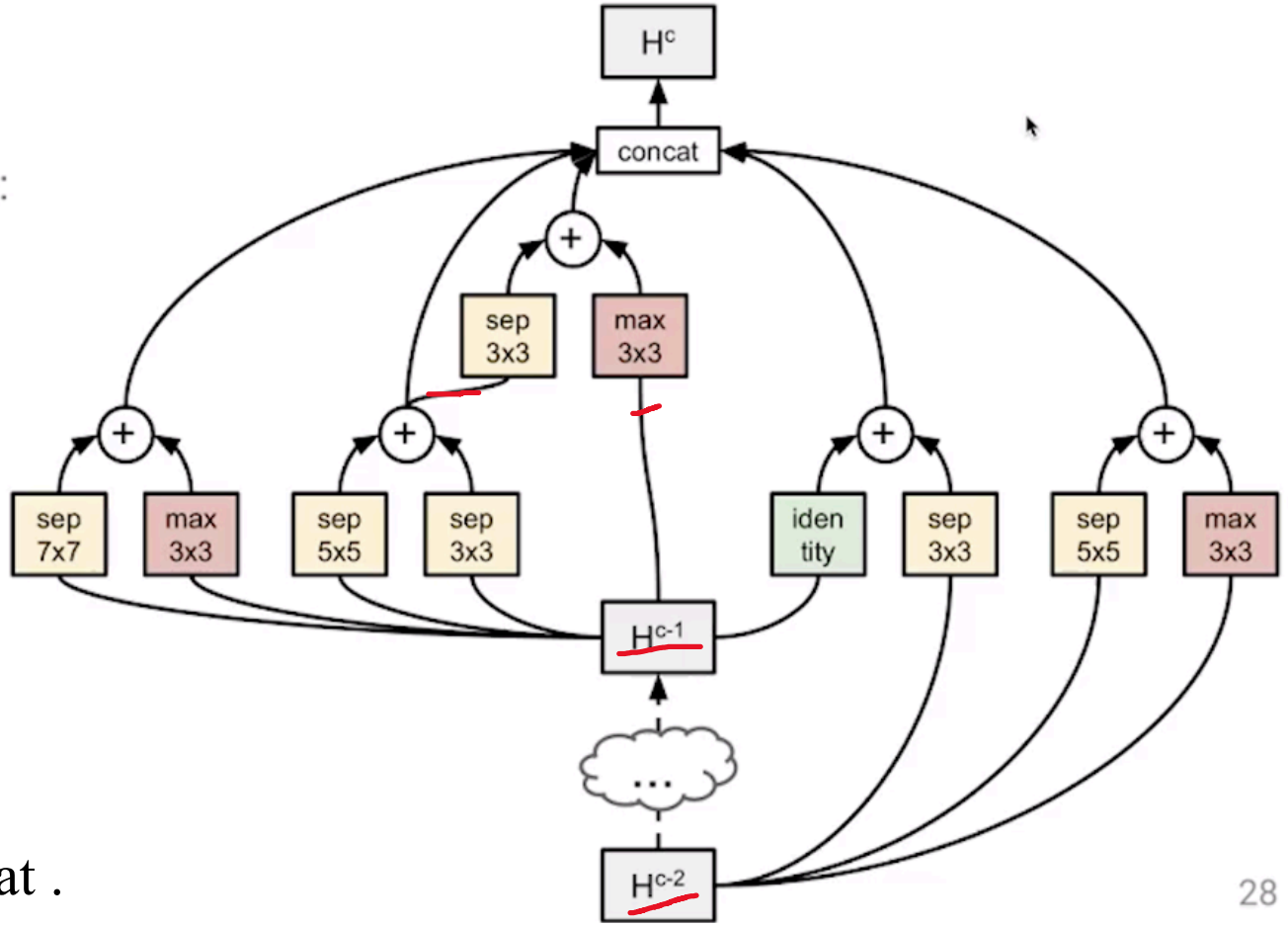
- Input 1 is transformed by Operator 1
- Input 2 is transformed by Operator 2
- Combine to give block's output
- **Input 1** and **Input 2** may select from:
  - Previous cell's output
  - Previous-previous cell's output
  - Previous blocks' output in current cell

- **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

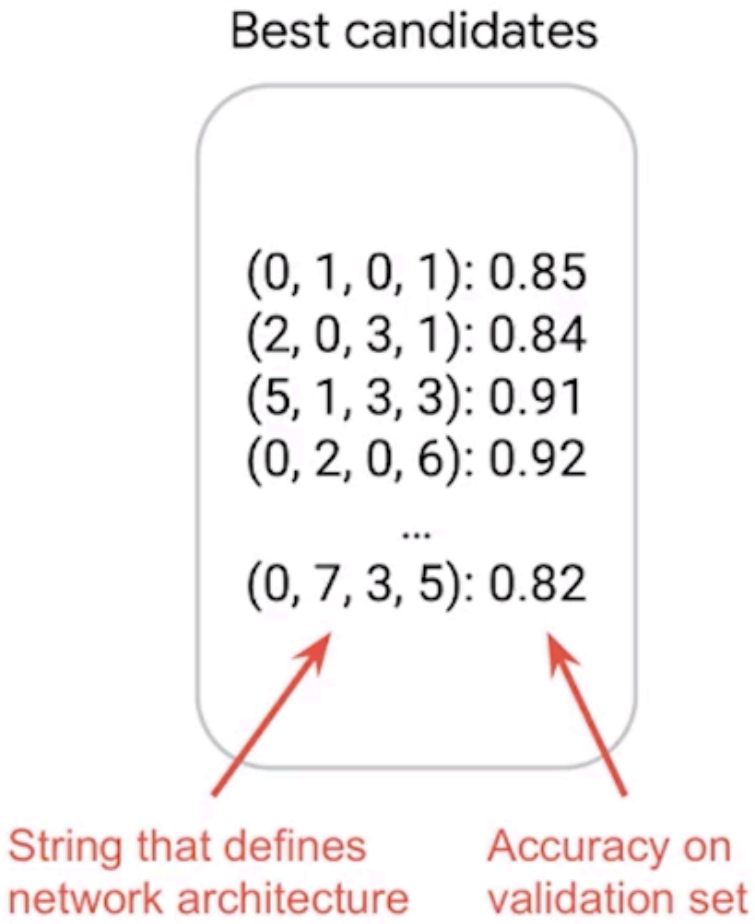
- One cell may look like:
- $2^2 * 8^2 * 1 *$
  $3^2 * 8^2 * 1 *$
  $4^2 * 8^2 * 1 *$
  $5^2 * 8^2 * 1 *$
  $6^2 * 8^2 * 1 =$
  10^{14} possible combinations!



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

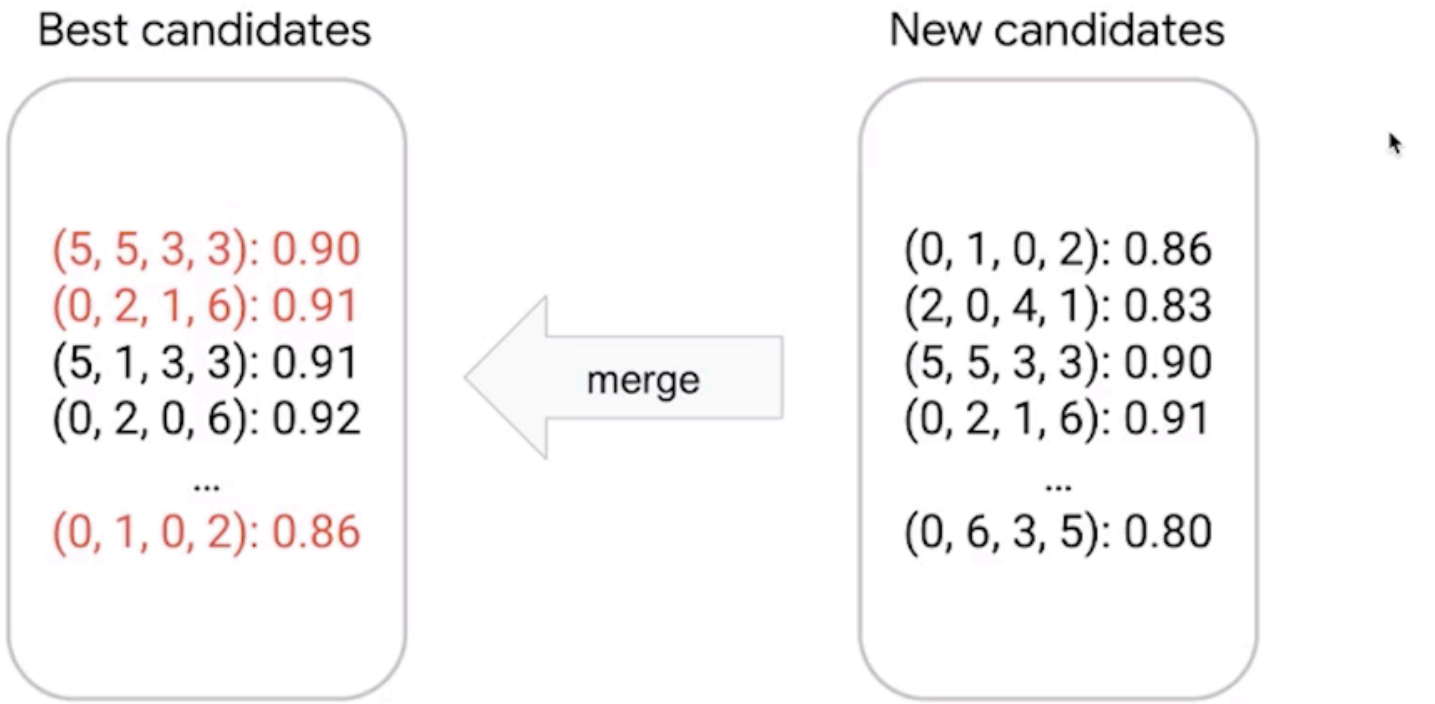
Best candidates

(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

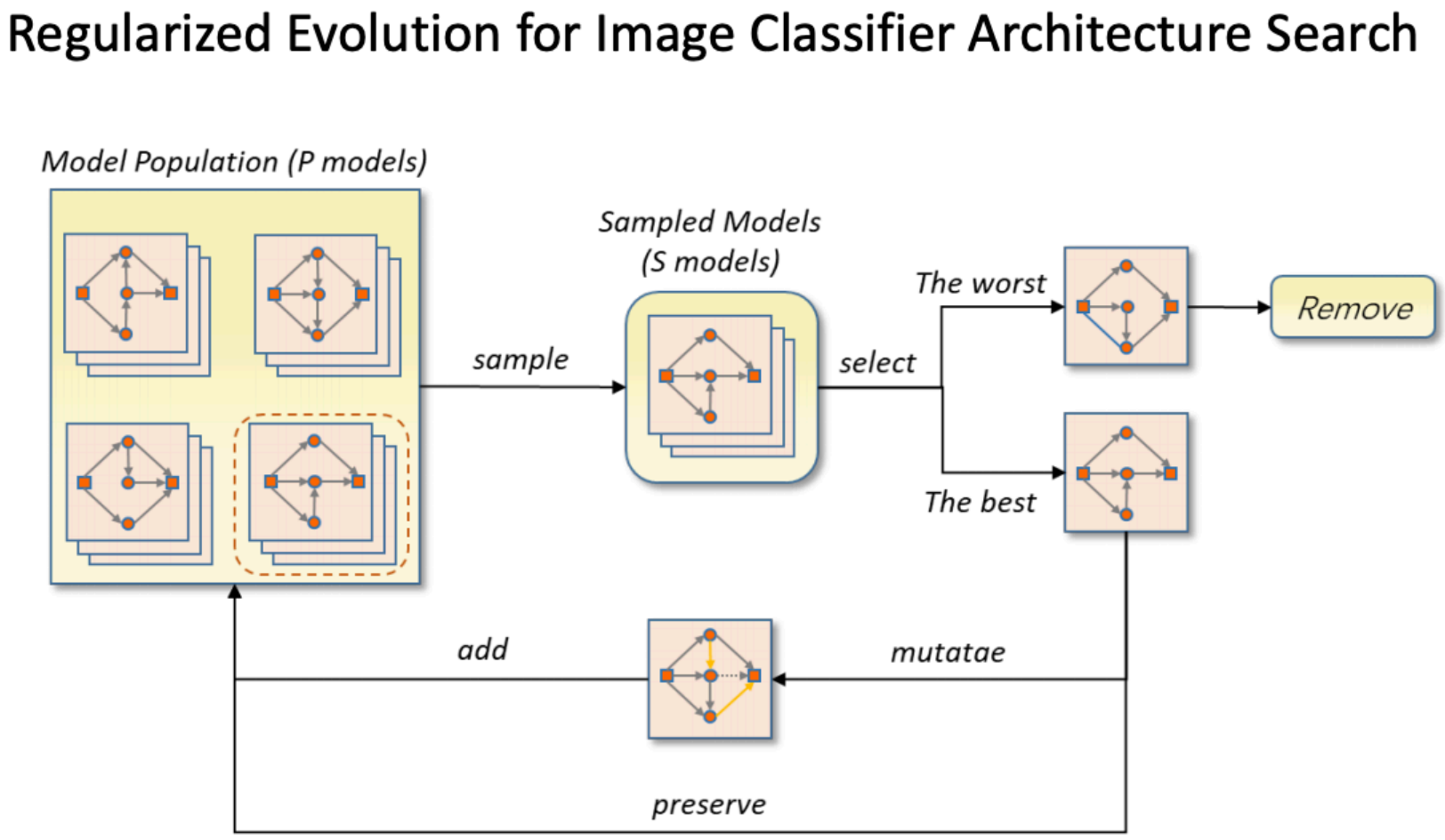
New candidates

(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

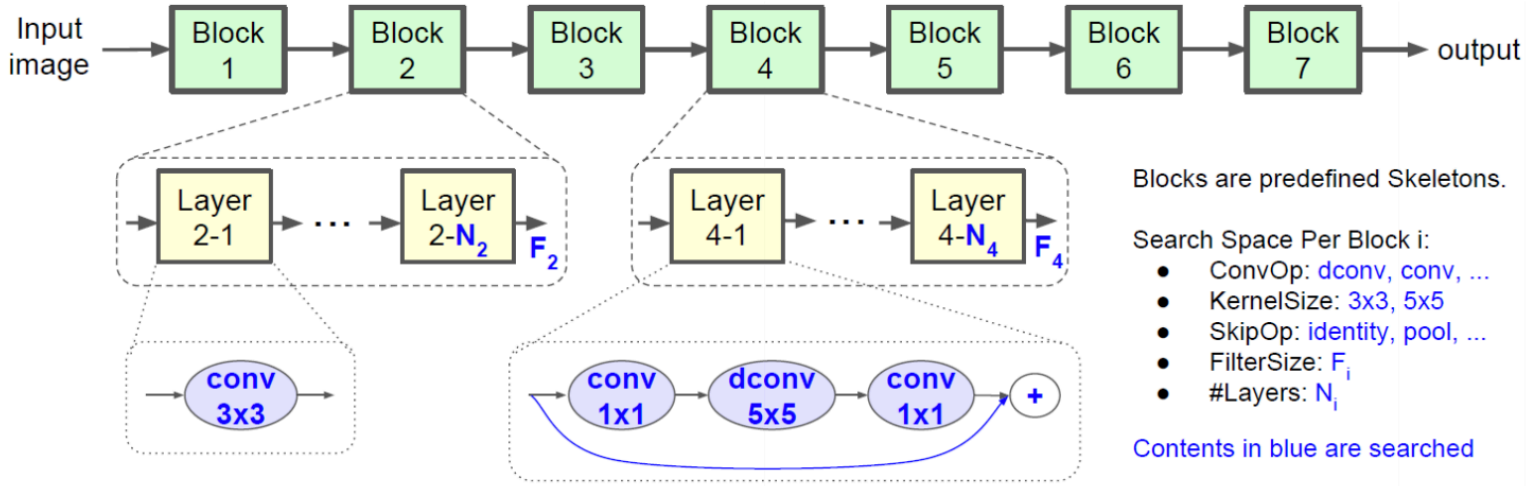


450 K40 * 7 days (cifar10)

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.

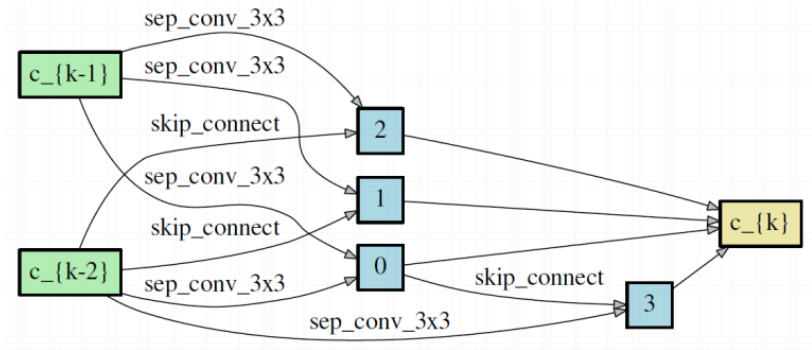


Blocks are predefined Skeletons.

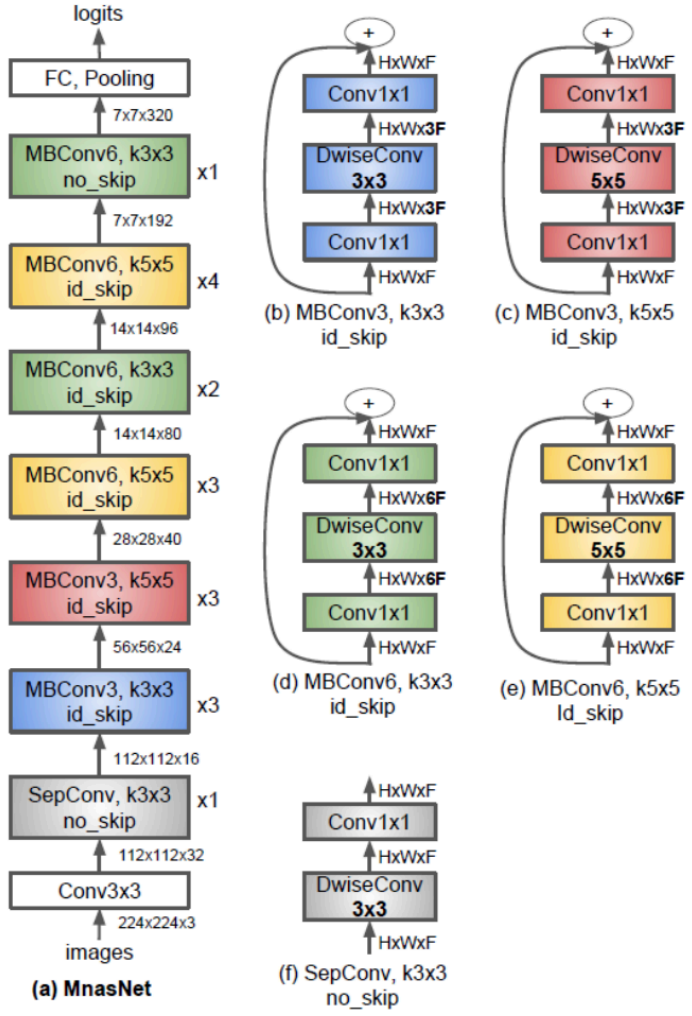
Search Space Per Block i:

- 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***.

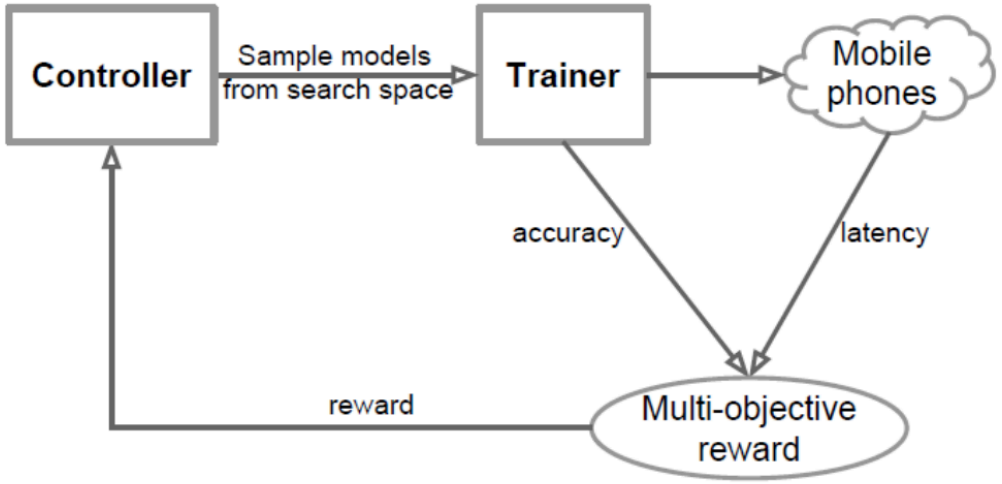


Figure 1: **An Overview of Platform-Aware Neural Architecture Search for Mobile.**

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

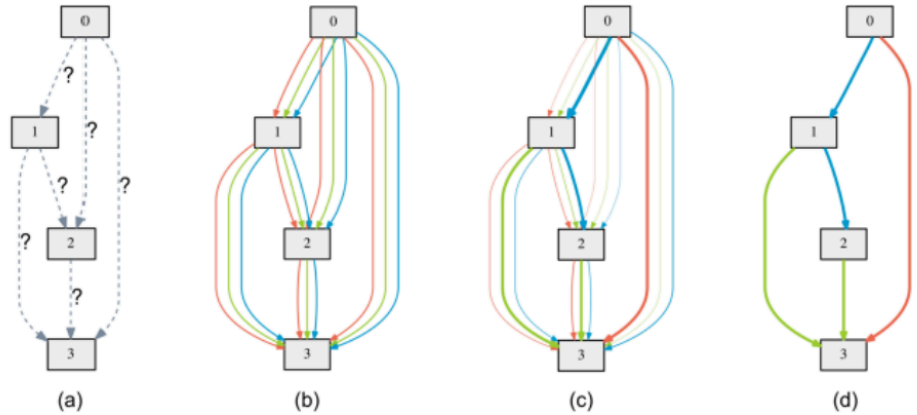


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem.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

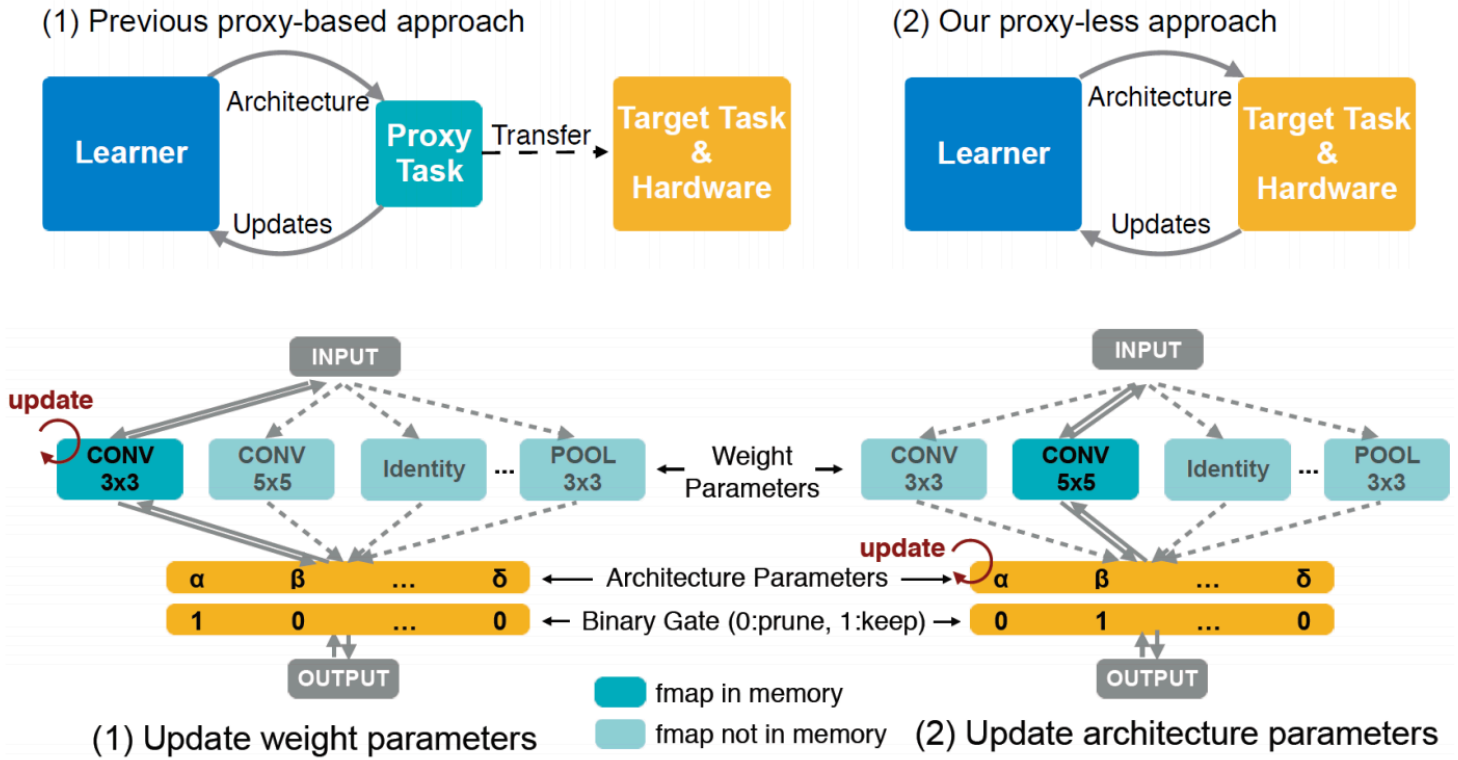


Figure 2: Learning both weight parameters and binarized architecture parameters.

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.

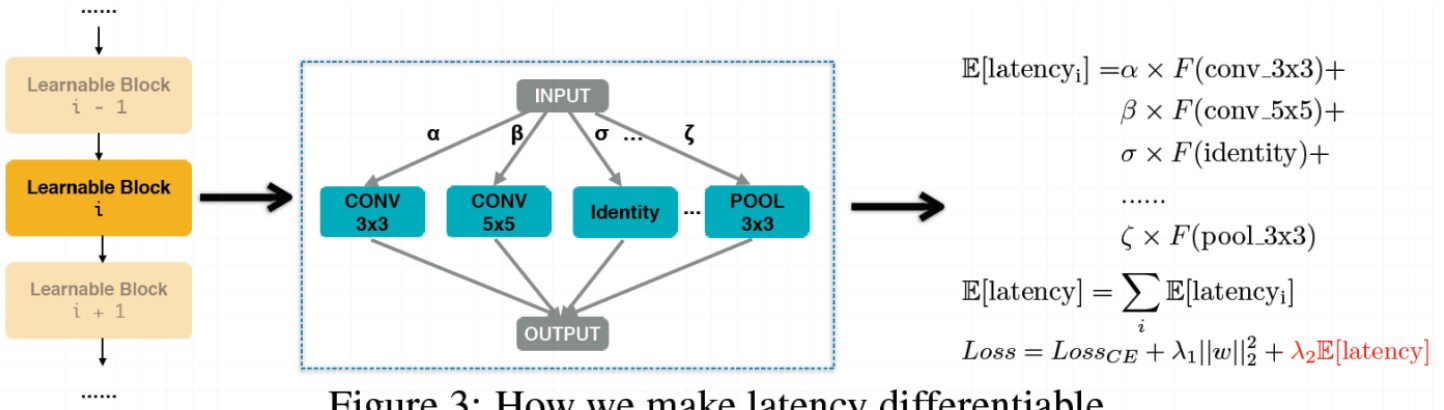


Figure 3: How we make latency differentiable.

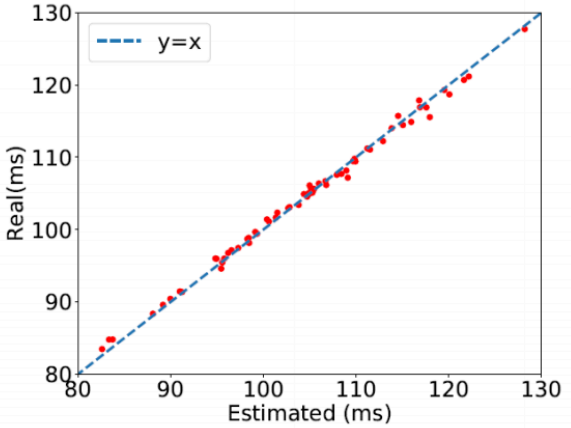


Figure 6: Our mobile latency model is close to $y = x$.

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.

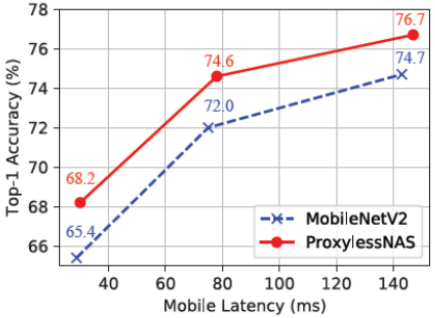


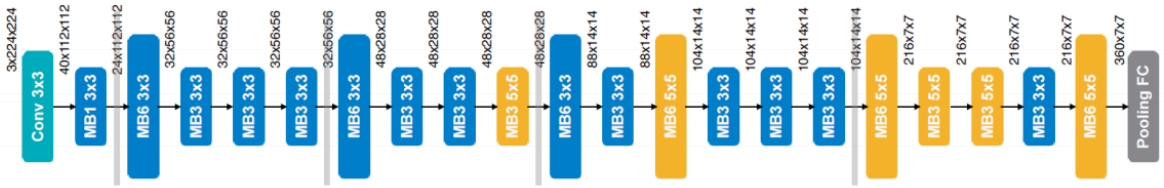
Figure 4: ProxylessNAS consistently outperforms MobileNetV2 under various latency settings.

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

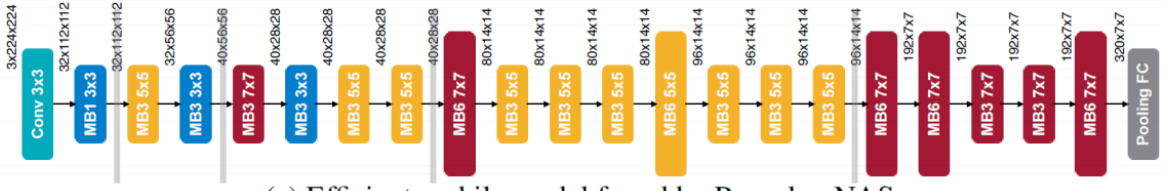
Table 4: ProxylessNAS saves GPU hours and memory (GB) by 1-2 orders of magnitude on ImageNet.[3]

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.

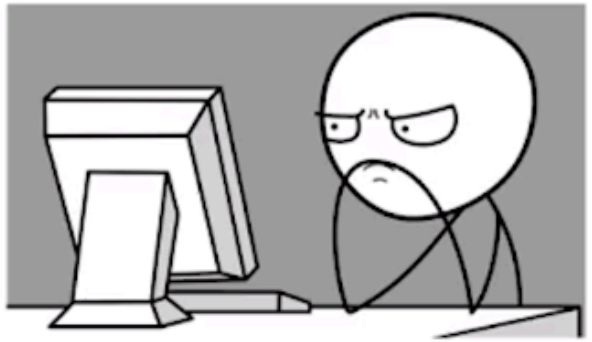


(c) Efficient mobile model found by ProxylessNAS.

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)

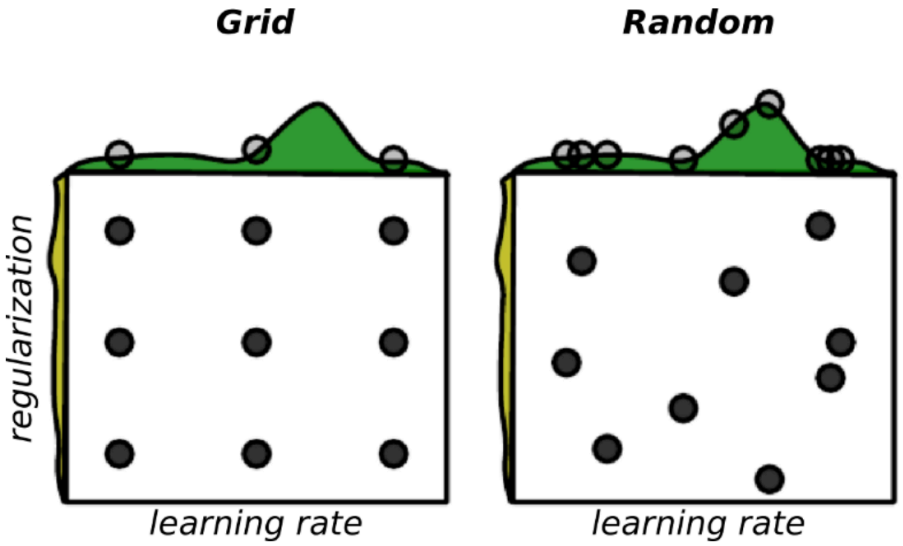
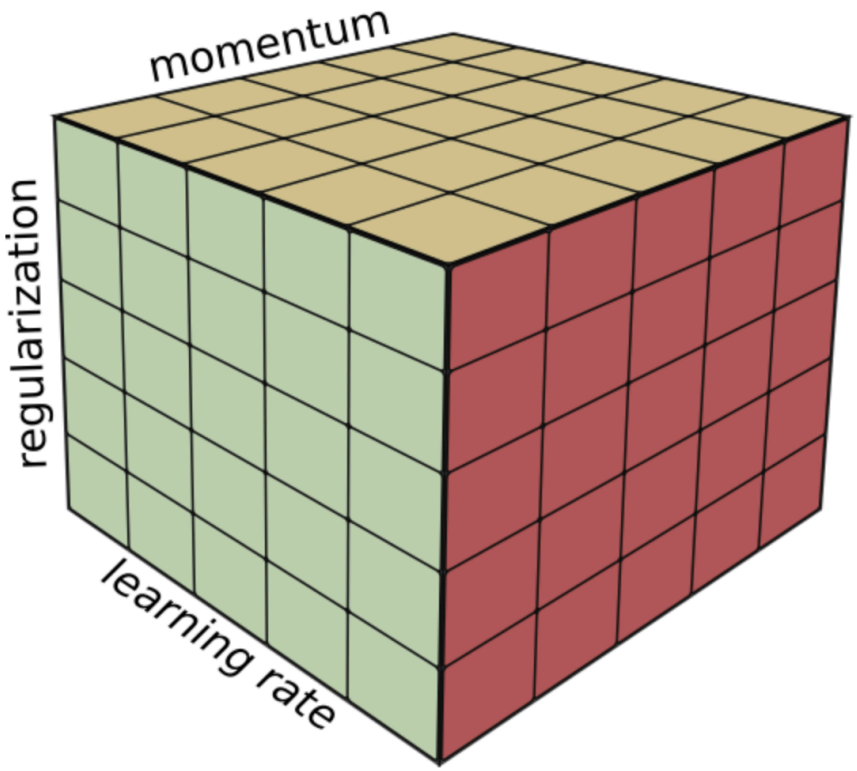


Image Credit: Random Search for Hyper-Parameter Optimization, James Bergstra, Yoshua Bengio.



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