Efficient Neural Architecture Search via Parameter Sharing

Hieu Pham [*1,2] **Melody Y. Guan** [*3] **Barret Zoph** [1] **Quoc V. Le** [1] **Jeff Dean** [1]

https://github.com/carpedm20/ENAS-pytorch

Speaker: Tao Hu
taohu620@gmail.com

[*]Equal contribution [1]Google Brain [2]Language Technology Institute, Carnegie Mellon University [3]Department of Computer Science, Stanford University. Correspondence to: Hieu Pham <hyhieu@cmu.edu>, Melody Y. Guan <mguan@stanford.edu>.

Overview

- Background
- Designing Recurrent Cells
- Designing Convolutional Networks
- Designing Convolutional Cells
- Experiments

Background

- VGG, ResNet, DenseNet, ...
- Single GPU can finish your own NAS now.

Designing Recurrent Cells

Formulation Target f: h(t) = f(h(t-1), x(t-1))

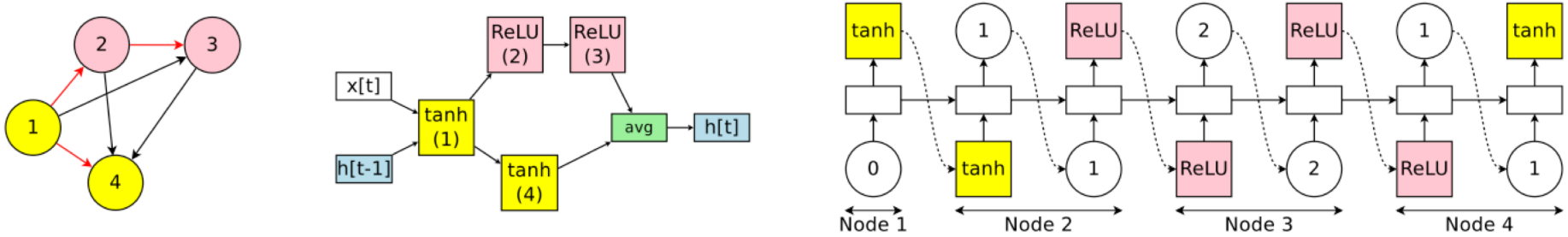
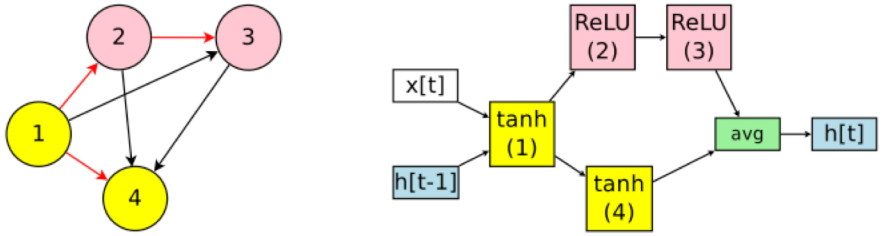


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result 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.

1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.
2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$.
3. At node 3: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 2 and the activation function ReLU. Therefore $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(\mathbf{h})})$.
4. At node 4: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function tanh, leading to $h_4 = \tanh\left(h_1 \cdot \mathbf{W}_{4,1}^{(\mathbf{h})}\right)$.
5. For the output, we simply average all the loose ends, *i.e.* the nodes that are not selected as inputs to any other nodes. In our example, since the indices 3 and 4 were never sampled to be the input for any node, the recurrent cell uses their average $(h_3 + h_4)/2$ as its output. In other words, $\mathbf{h}_t = (h_3 + h_4)/2$.

Designing Recurrent Cells



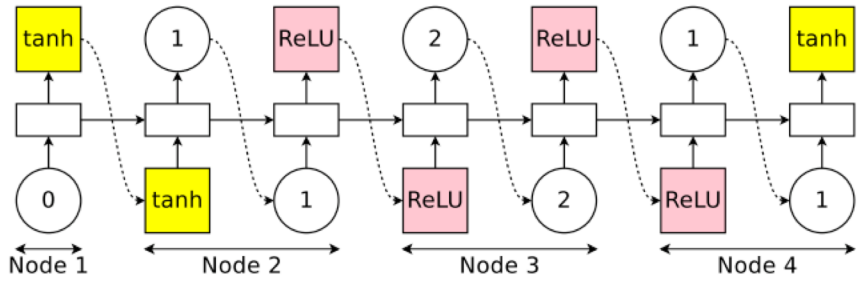


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result 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.

1->2,3,4
2->3,4
3->4

- DAG is given in advance. Actually a DAG can be decided by a node number N.
- ENAS's controller is a LSTM that decides: 1) which edges are activated and 2) which activation function are performed at each node in the DAG.
- For LSTM, h and x both act as the initial input.
- Each edge in the DAG is equipped with a sharing parameter ω.(six edges means six "ω".)
- For each iteration, the controller samples **a previous index** and an activation function.
- For the output, we simply average all the loose ends, i.e. the nodes that are not selected as inputs to any other nodes.

Designing Recurrent Cells

- **Parameter Sharing:** The local computations at each node have their own parameters, which are used only when the particular computation is activated. Therefore, ENAS's design allows parameters to be shared among all child models, i.e. architectures, in the search space.
- **Search Space:** if the recurrent cell has N nodes and we allow 4 activation **functions (namely tanh, ReLU, identity, and sigmoid)**, then the search space has **4^{N} ×N!** configurations. In our experiments, N = 12, which means there are approximately **10^{15}** models in our search space.

Designing Recurrent Cells

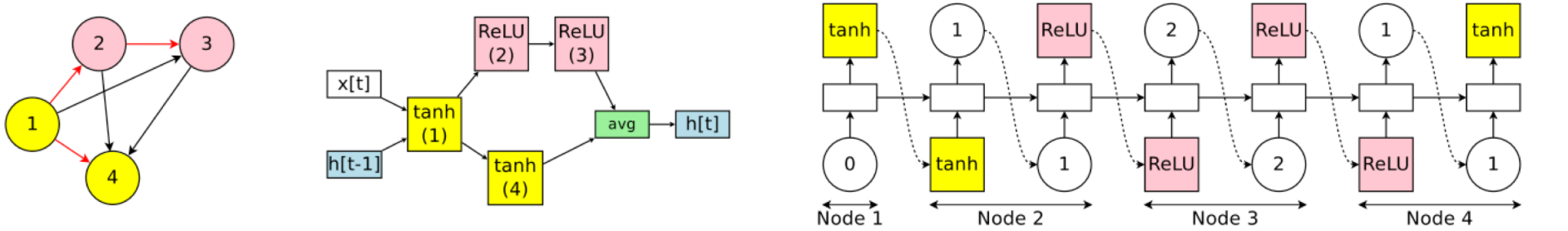


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result 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.

- Input:
  - Initial x and h.
  - a DAG decided by node number N;
  - N initial "ω";
  - activation function set: (tanh, ReLU, identity, sigmoid)
  - initial weight of the controller(actually a LSTM).
- To Learn:
  - the parameters of the controller LSTM, denoted by θ.
  - the shared parameters of the child models, denoted by ω.

Training ENAS and Deriving Architectures

- While not convergent:
  - First stage: (fix controller parameter θ)training sharing parameter ω. via mini-batch SGD during a entire pass through the training data.(typical classification problem, therefore cross-entropy loss applied.)
  - Second stage: (fix sharing parameter ω), maximize the expected reward, which is computed by the classification accuracy on the validation set. RL method is applied here.
- How to derive:
  - Sample several models from a trained ENAS model.
  - Compute its reward on a single minibatch sampled from the validation set.
  - Choose the model with the highest reward to re-train from scratch.

Designing Convolutional Networks

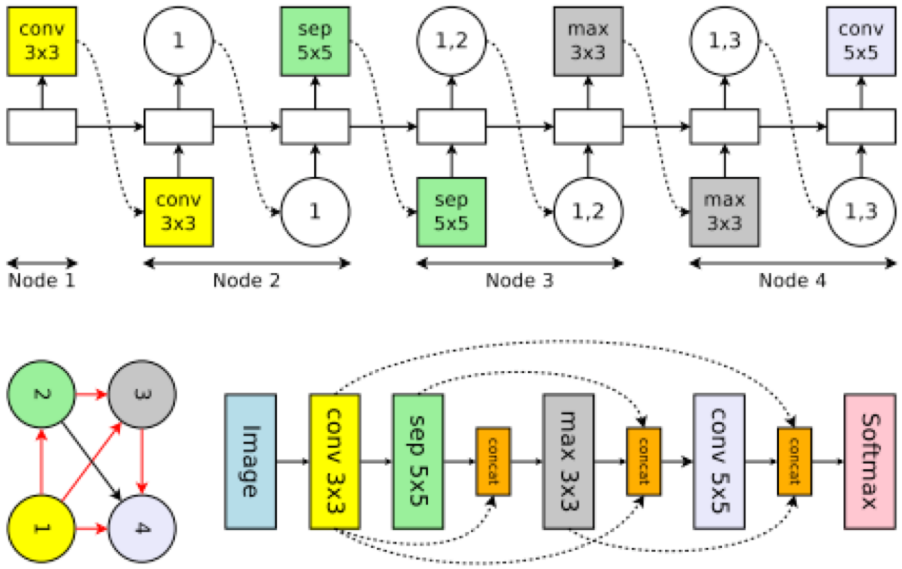


Figure 3. An example run of a recurrent cell in our search space with 4 computational nodes, which represent 4 layers in a convolutional network. *Top:* The output of the controller RNN. *Bottom Left:* The computational DAG corresponding to the network's architecture. Red arrows denote the active computational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

- One vertex may have one more input edge.(In Recurrent Cell one vertex only have one input edge) .
- The 6 operations available for the controller are: convolutions with filter sizes 3 × 3 and 5 × 5, depthwise-separable convolutions with filter sizes 3×3 and 5×5 (Chollet, 2017), and max pooling and average pooling of kernel size 3 × 3.

Designing Convolutional Cells

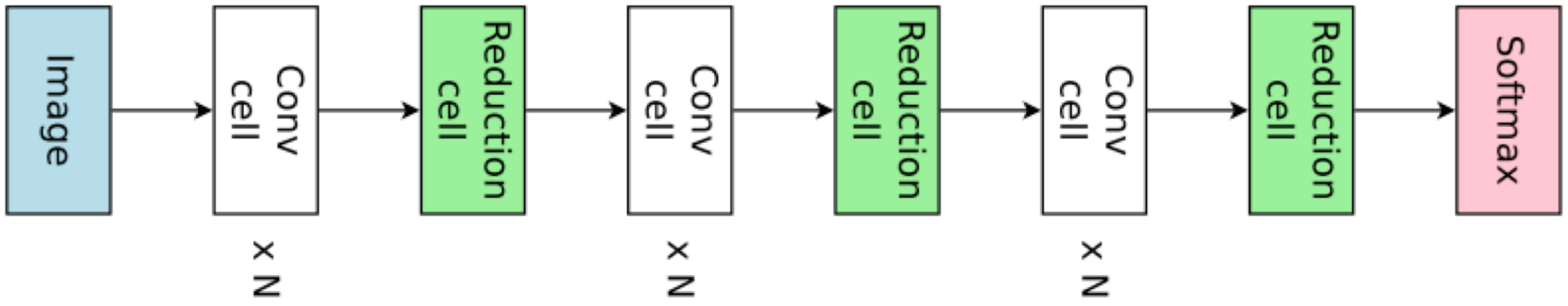


Figure 4. Connecting 3 blocks, each with N convolution cells and 1 reduction cell, to make the final network.

- Two previous nodes to be used as inputs to the current node, two according operations to apply to the two sampled nodes.
- The 5 available operations are: identity, separable convolution with kernel size 3×3 and 5×5, and average pooling and max pooling with kernel size 3×3.

Experiments- Penn Treebank

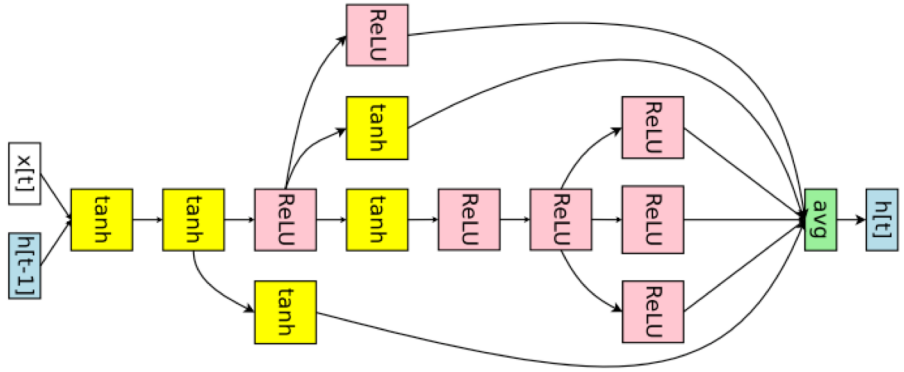


Figure 6. The RNN cell ENAS discovered for Penn Treebank.

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

Table 1. Test perplexity on Penn Treebank of ENAS and other baselines. Abbreviations: RHN is *Recurrent Highway Network*, VD is *Variational Dropout*; WT is *Weight Tying*; ℓ_2 is *Weight Penalty*; AWD is *Averaged Weight Drop*; MoC is *Mixture of Contexts*; MoS is *Mixture of Softmaxes*.

Experiments- CIFAR10

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.

Experiments- CIFAR10

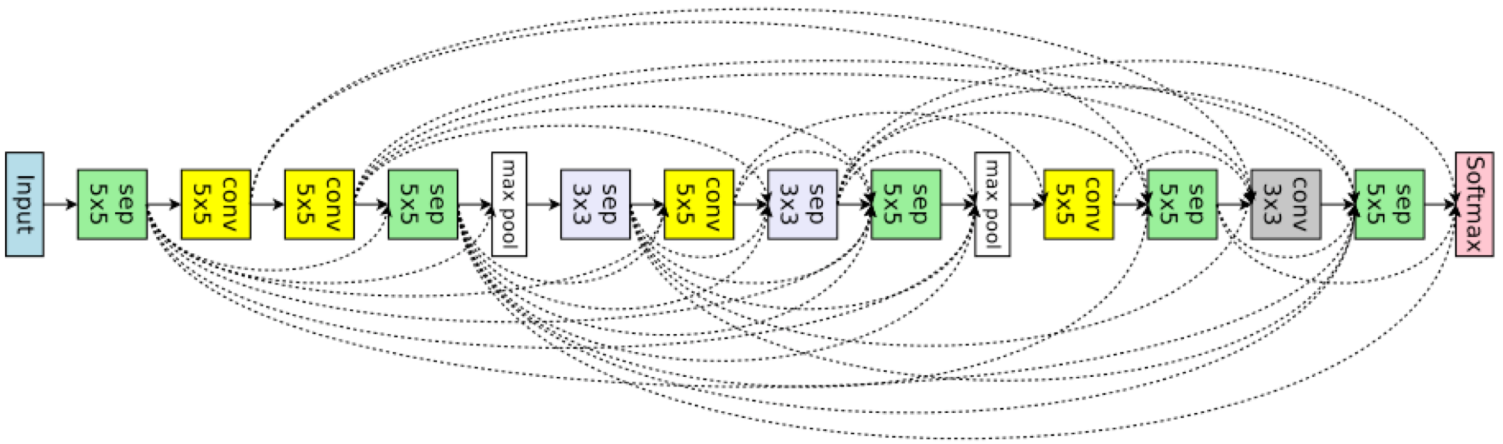


Figure 7. ENAS's discovered network from the macro search space for image classification.

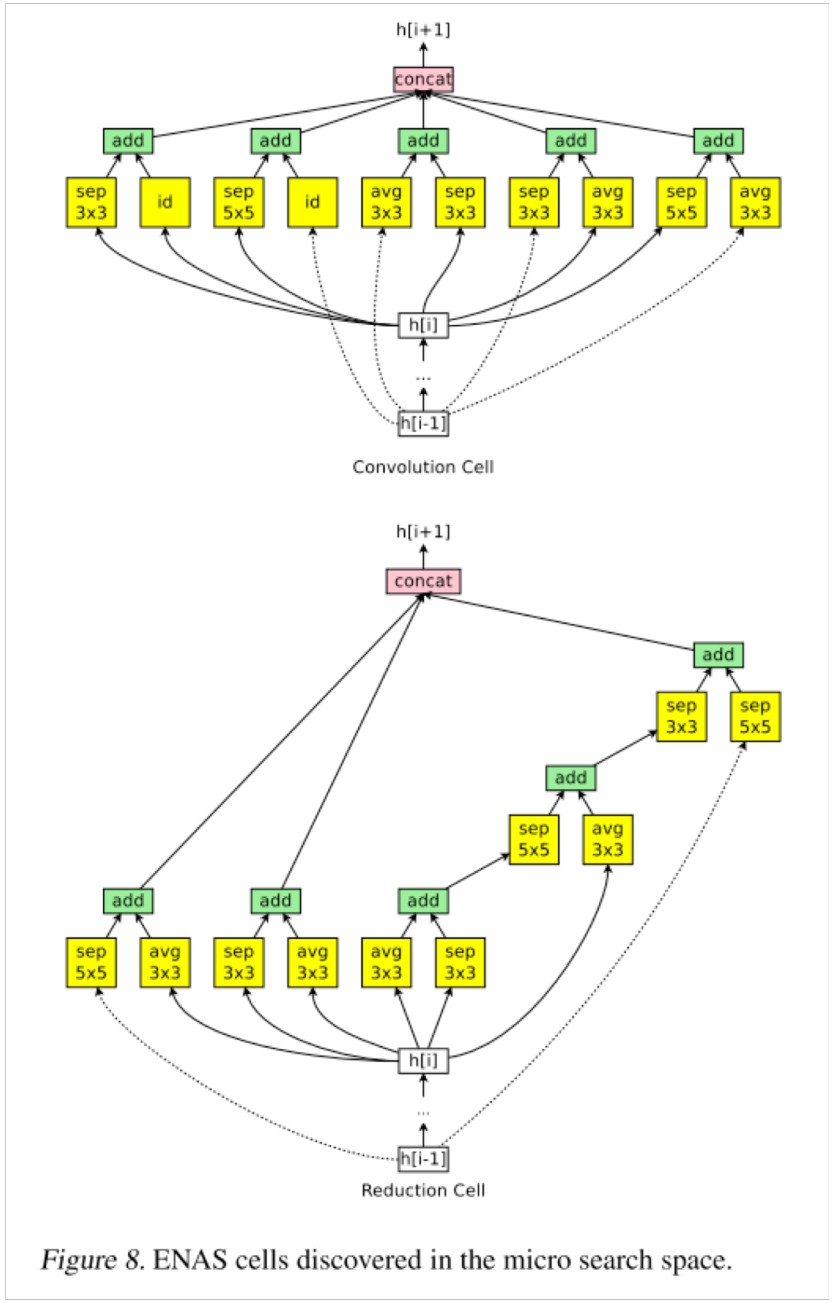


Figure 8. ENAS cells discovered in the micro search space.

Discussion

- Disabling ENAS Search 
- CIFAR to ImageNet
- High-level computer vision task(segmentation, detection)?
  - Auto-Deeplab