

本文是一篇关于 AutoML 中网络架构搜索 (neural architecture search) 的 memo。由于大部分这方面的工作针对的是卷积网络，第一部分先介绍人工设计卷积网络的一些成功经验，第二部分会对 NAS 问题给出详细的定义和描述，第三、四部分介绍目前主流的两个方法：基于进化算法和基于强化学习。

一. 人工探索卷积网络的经验

常见的卷积网络结构一般是一堆叠一些卷积层和池化层，最后用全连接层产生与具体任务相关的输出，而经过研究人员多年的探索，卷积网络在结构上有了很多有效的变体。这些经验作为 NAS 的先验知识，可以帮助我们设计不同的网络结构搜索空间，直觉上，如果一个搜索空间中包含 state-of-art 的人工网络结构，那么利用有效的搜索策略起码可以找到达到人工设计的水平的网络结构。

1. 一个 filter 是否需要同时“看到”所有的 input channel

为了把卷积操作放在多个 GPU 上训练，Alexnet 最早提出 group convolution 的概念。结构如下，可以看到第一个卷积层得到了两组 feature map，作为各自的输入在后续卷积层训练，那么对于第二个卷积层和之后的层来说，只能利用一半的 input channel（除第三层会拿到第二层产生的全部的 feature map）。这样做除了能够放在两个 GPU 上并行训练之外，还有一个优势是大大降低了参数量，具体来说，对于第二个卷积层原本 input channel 为 96，kernel size 为 5×5 ，output kernel 为 256，参数量为 $96 \times 5 \times 5 \times 256$ （得到同样数量和大小的 feature map），而现在只需要 $2 \times 48 \times 5 \times 5 \times 128$ ，参数量减少了一半，如果分成更多的 group 会继续减少。

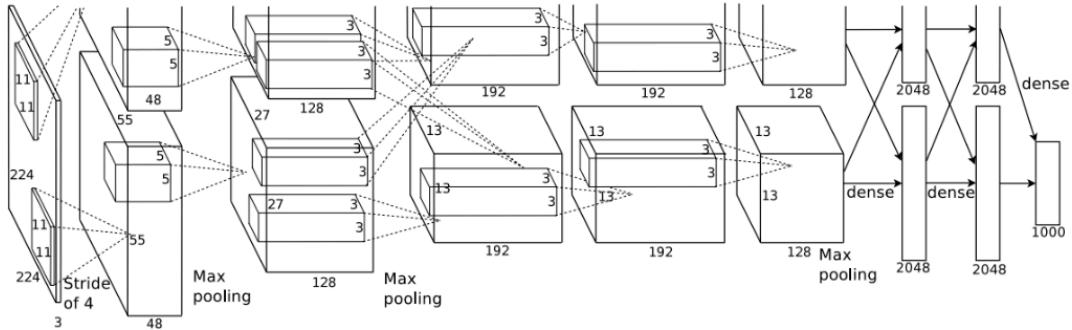
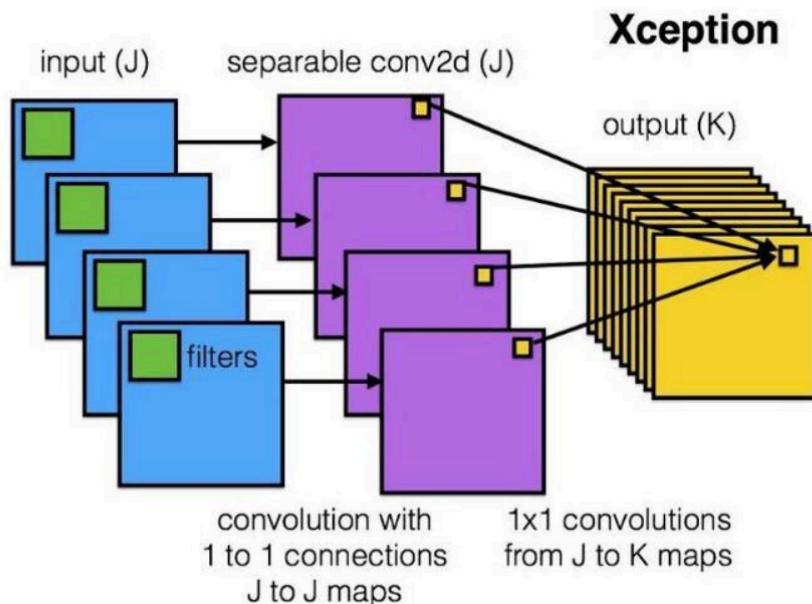


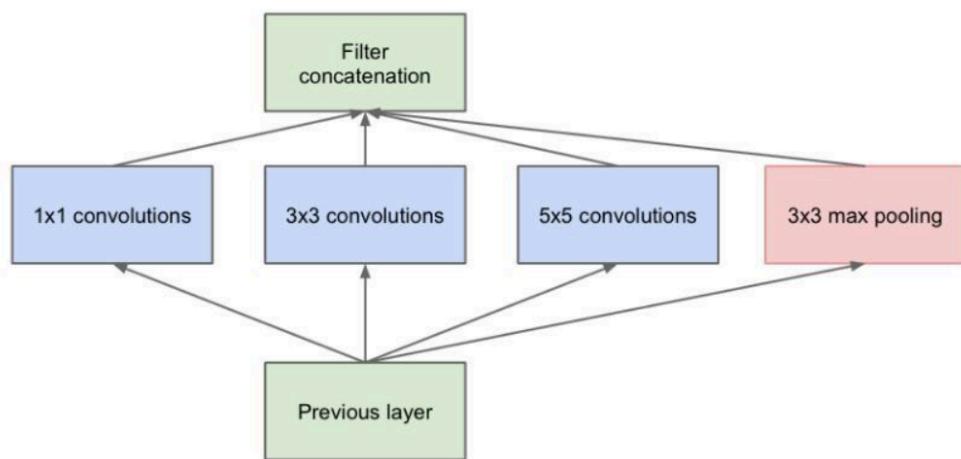
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

这种思想发展到极致就是 depthwise convolution，先对每一个 channel 进行各自的（二维）卷积操作，也就是说一个 filter 只能看到一个 channel，得到和 input channel 同样数量的一组 feature map 之后，为了利用不同 channel 在同一位置的 feature，用常规的 1×1 的 filter 进行 pointwise convolution。对于 input channel 为 3，output channel 为 256 的 3×3 卷积层，参数数量从 $3 \times 3 \times 3 \times 256$ 降到了 $3 \times 3 \times 3 + 3 \times 1 \times 1 \times 256$ 。文章在降低参数量的同时验证了效果。

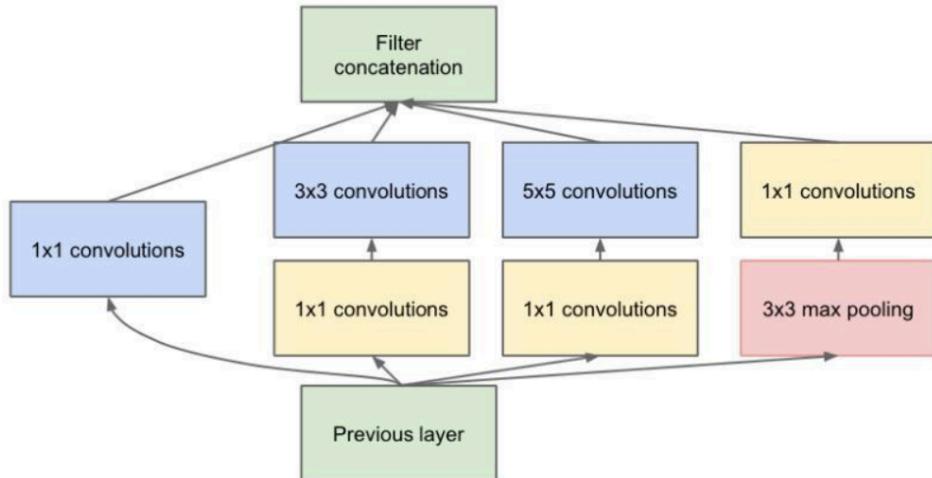


2. 卷积层中不同 filter 的 kernel size 是否需要相同

卷积层的每个 filter 对应一个 output kernel，传统的卷积网络同一个卷积层使用同样大小的 filter，但是如果可以对同一组 feature map 提取不同尺度的特征，再把这些特征结合起来，效果会不会比单一的 kernel size 的卷积核要好呢？Inception 网络做了这样的尝试，inception 结构如下图，一组 feature 经过 1×1 、 3×3 、 5×5 的卷积核处理，通过设置 padding 等方式（tf 中设置 padding 为 SAME）得到相同尺寸的 feature map，再将其在 channel 这个维度上连接起来（称为 concatenation 操作）。



为了进一步减小参数量，第二个版本的 Inception 结构中加入了一些 1×1 的卷积核。 1×1 卷积核除了可以做 pointwise convolution 来融合不同 channel 的特征，另一个重要作用可以用来升维降维。如果 input channel 为 192，output channel 为 128，kernel size 为 3×3 ，则需要的参数量为 $192 \times 3 \times 3 \times 128$ ，如果前面加上一个 1×1 的 96 个 channel 的卷积操作，则参数量是 $192 \times 1 \times 1 \times 96 + 96 \times 3 \times 3 \times 128$ 。



存在一种更充分的利用 1×1 filter 升维降维功能的结构称为 bottleneck layer，先对 channel 降维，在进行 3×3 的卷积操作后再进行升维。

3. kernel size 应该为多大

早期的卷积网络倾向于使用大的卷积核，从而获得更大的感受野，为了可以得到大小不变的 feature map，一般使用奇数 size 的卷积核。不考虑 1×1 的卷积核（感受野太小），人们发现通过 3×3 的 filter 的堆叠可以得到和更大的卷积核一样的感受野。如果 stride 为 1，两个 3×3 的 filter 的感受野为 5×5 ，三个 3×3 的 filter 感受野为 7×7 ，而参数量更小（比较 $3 \times 3 \times 2$, 5×5 ），同时引入了更多的非线性函数，所以我们一般更喜欢小而深的网络。

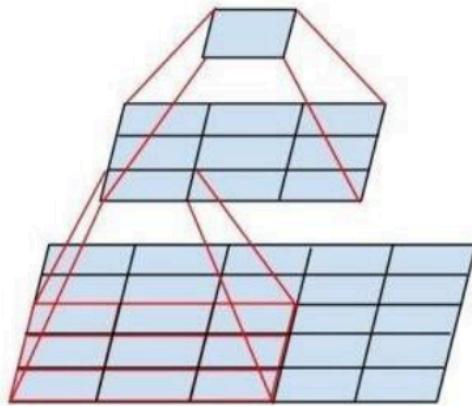


Figure 1. Mini-network replacing the 5×5 convolutions.

4. 网络加深带来的梯度消失问题

skip connection 的提出是为了解决深度网络中的梯度消失问题，Resnet 提出了一种 residual block 的结构，可以看到输入 X 通过两条不同的前馈路径得到输出，实际上卷积网络拟合的是目标输出对输入的残差。这样做的好处是，从卷积网络传回来的梯度信号可以已经发生了梯度消失，但是通过 skip connection 会得到一个没有被放大或者缩小的梯度。另外，residual block 具体的结构利用了我们上面所说的 bottleneck（见下图）。

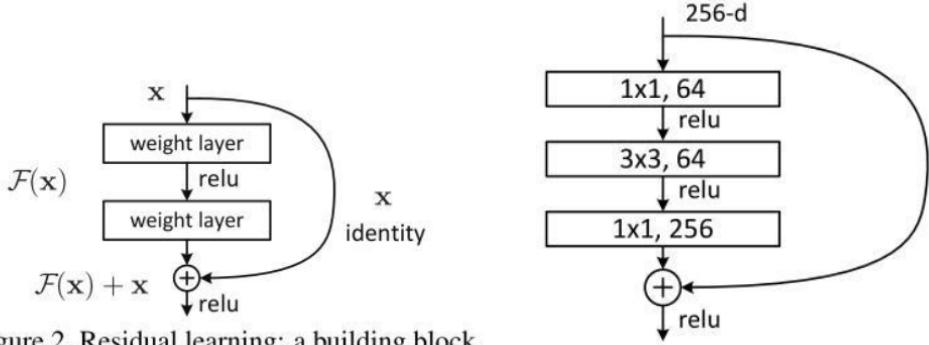


Figure 2. Residual learning: a building block.

将这种技巧应用到极致的一个结构式 DenseNet，这种结构每一层都接收前面所有层的输出，通过 concatenation 作为层的输入（不同于 ResNet 做加和），由于充分利用了各层网络特征，文章适当的减小了网络的宽度。这种有效的设计使得 DenseNet 当选了 CVPR2017 的 best paper。

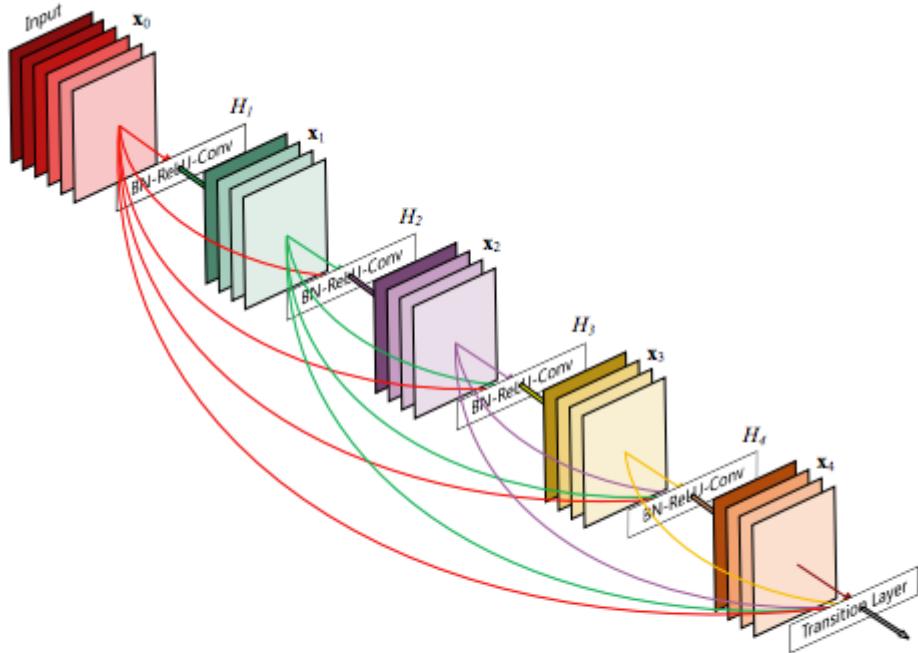


Figure 1: A 5-layer dense block with a growth rate of $k = 4$.
Each layer takes all preceding feature-maps as input.

二. NAS overview

NAS 是 automating machine learning (AutoML) 的一个子领域，可以从如下三个方面进行描述：

search space 定义了搜索空间中可能包含有什么样的结构，合理的定义搜索空间决定了生成网络架构的上限。一般有这样两种讲故事的角度，搜索空间越大，组成成分越基础，就越能避免人为偏置，发现新的结构；而引入越多的先验知识，则能有效的减小搜索空间，可以更快的产生和已有网络结构相似，但是更优化的结构。巧妙的设计一个包含更多的已有网络结构，同时很大的搜索空间是这个领域可以做的一个方向。

Search strategy 是具体的搜索方法，涉及到经典的 exploration-exploitation trade-off 问题。

performance estimation strategy 定义了 NAS 的目标，一般是验证发现的网络结构在训练集训练过之后在验证集上的表现。在搜索的过程中用来引导搜索的方向，在搜索结束后验证集上最好的网络架构在测试集上表现作为我们 NAS 算法的表现。

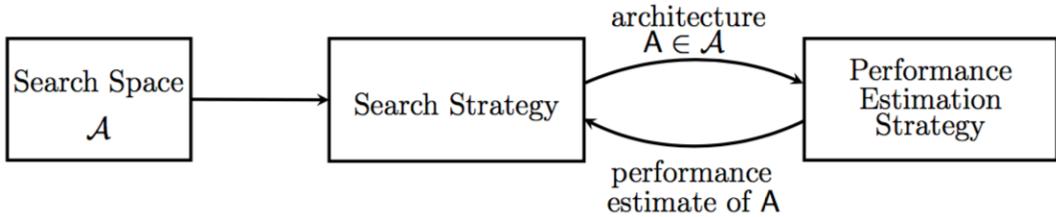
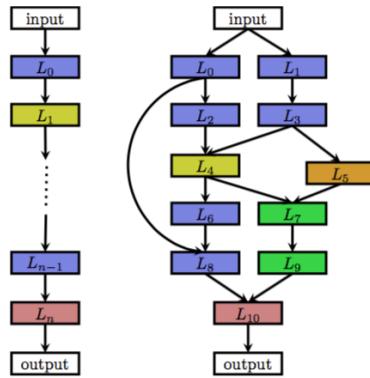


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space \mathcal{A} . The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

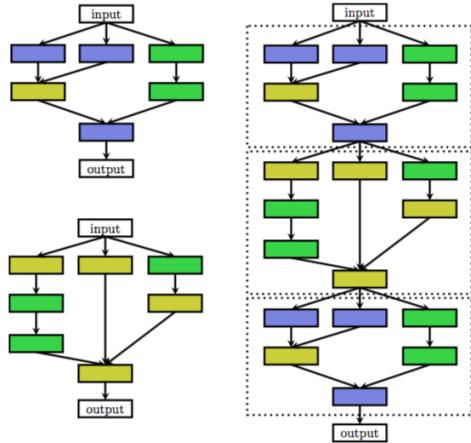
1. Search space

目前主要存在以下几种搜索空间:链式的神经网络(见下图左)，一般会被参数化为 1.最大层数，2.每层的操作（比如卷积、池化或者是上一节描述的更复杂的操作），3. 每个操作的超参数（比如卷积的 kernel size, stride 等）；更复杂的空间是加入了 skip connection 的多支网络，除了上述的参数，还要搜索 1.每层与哪些层连接，2.多层的输出作为输入如何融合等。



受启发于人工设计网络中常见的 block/cell 结构，也有一些工作是在基于 cell 的搜索空间中展开，优势在于搜索空间小，可迁移。其他工作还存在一种层次化的搜索空间，将在后续章节具体介绍。

NAS 问题搜索空间的特征是高维且不连续，这种性质使得一些超参数优化的方法---比如贝叶斯优化---不太适用于 NAS 问题，现在主要两个在发文章的方向是进化算法和强化学习。



2. 传统方法

存在在超参数空间进行搜索的传统方法：

grid search: 实际上就是穷举所有的超参数组合，逐一进行验证。往往将连续的参数离散化，然后循环遍历所有的组合，像是在一个网格中进行搜索而得名。

Random search: 对每个超参数随机采样，选择最好的组合。效果往往比 grid search 要好，因为在重要的维度上会进行更多的探索（比起 grid search 只有固定的几个值）。这种方法虽然没有什么技巧性，但是也能产生不错的网络结构，在后续介绍的方法中经常被用作 baseline 来对比。

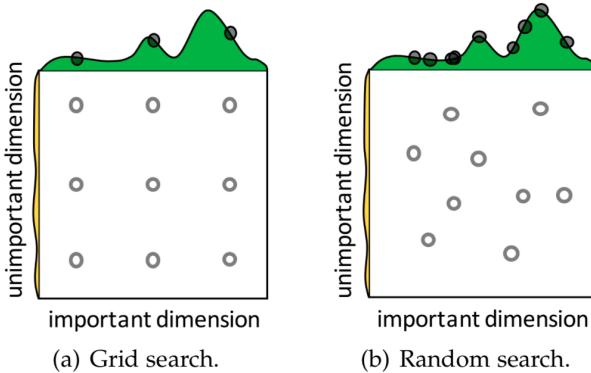


Fig. 12. Illustration of grid and random search with 9 trials in 2-D search problem. This figure also illustrates that random search does more exploration than grid search when the number of trials is same (the image is from [47]).

三. 基于强化学习的 NAS

1. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

来自 google brain，发表于 ICLR2017，最早用强化学习来做 NAS 的文章，思路很简单，将一个网络结构用一个变长的串表示，用 rnn 来学习这样的串。作为比较早的探索网络生成的方法，效率是比较低的，用了 800 个 GPU 并行训练。

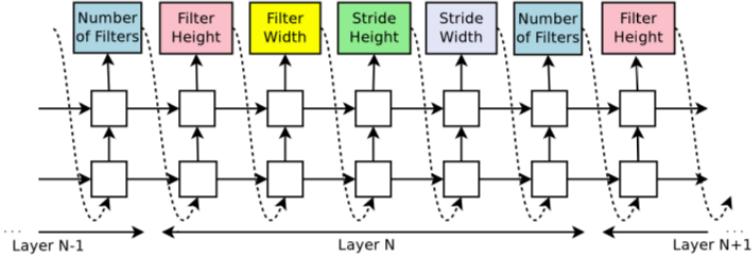


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time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

如果一个网络全部由卷积组成，那么用一个 rnn 来产生每个卷积的各个参数（如上图），其中 rnn 每步的输出都是一个 softmax 的分类结果，从预先设定的可能的值中选取参数。最后产生的结构在训练后在验证集上的准确率作为 reward，然后用 REINFORCE 算法训练，实际上是将 action 的序列作为了 state：

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

用采样代替期望，并加入 bias 来降低方差，bias 只要不和当前的 action 有关就不会引入偏差，所以直接用了之前生成的网络结构 reward 的指数移动平均。

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

在最基础的模型中，我们会得到一个链式的神将网络，为了加入 skip connection，文章用了一种类似于 attention 的方式：

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

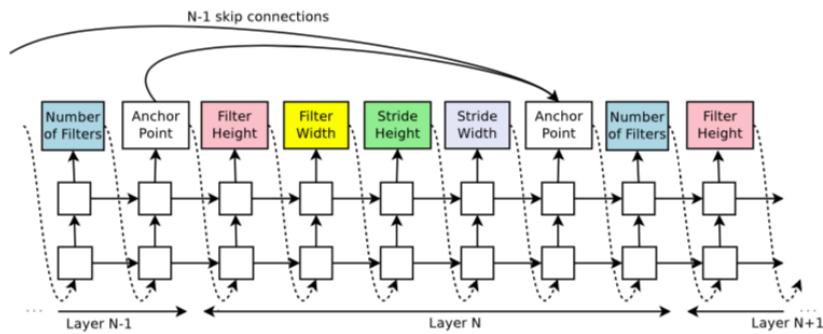
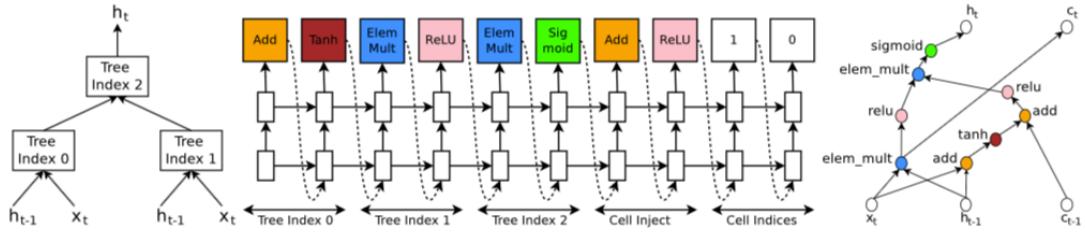


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

除了产生 CNN 之外，文章还给出了一种产生 RNN 的方法，将 RNN 看做一个树结构，如下图，用 RNN 预测 hidden state 和 input 的结合方式，以及激活函数：



作为最早探索自动生成的文章，搜索空间非常有限，像 learning rate 等都作为超参通过网格搜索再确定。最后来看下结果：

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) with Dropout/Drop-path	21 21	38.6M 38.6M	5.22 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 1202	1.7M 10.2M	5.23 4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16 28	11.0M 36.5M	4.81 4.17
ResNet (pre-activation) (He et al., 2016b)	164 1001	1.7M 10.2M	5.46 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.

第一个实验不对 stride 进行预测，不加 pooling 层，最后误差率 5.5，但注意这个网络比起同样误差率认为设计的网络来说非常浅。第三个实验中选择了两个层进行 pooling 操作，最后一个实验在最好的网络结构上每层都加了一些 filter，也就是网络变得更宽，得到了接近认为设计 state-of-art 的结果。

值得一提的是，作者之后又提出了一种基于 cell 的 search space，使用 ppo 进行搜索的模型。这种搜索空间会在下一章中介绍。

2. Designing Neural Network Architectures using Reinforcement Learning

同样发表在 ICLR2017，本文提出了一种基于 q-learning 学习 CNN 的方法。利用了 Q-learning 如下的优化目标：

$$Q_{t+1}(s_i, u) = (1 - \alpha)Q_t(s_i, u) + \alpha [r_t + \gamma \max_{u' \in \mathcal{U}(s_j)} Q_t(s_j, u')] .$$

可以看到，本文并没有建立一个 Q-function，而是使用了所谓列表法---将所有的 state-action（或者说探索到的）列出来，这是因为定义的 state 和 action 空间都是离散的。

State 的空间如下：

Layer Type	Layer Parameters	Parameter Values
Convolution (C)	$i \sim$ Layer depth $f \sim$ Receptive field size $\ell \sim$ Stride $d \sim$ # receptive fields $n \sim$ Representation size	< 12 Square. $\in \{1, 3, 5\}$ Square. Always equal to 1 $\in \{64, 128, 256, 512\}$ $\in \{(\infty, 8], (8, 4], (4, 1]\}$
Pooling (P)	$i \sim$ Layer depth $(f, \ell) \sim$ (Receptive field size, Strides) $n \sim$ Representation size	< 12 Square. $\in \{(5, 3), (3, 2), (2, 2)\}$ $\in \{(\infty, 8], (8, 4] \text{ and } (4, 1]\}$
Fully Connected (FC)	$i \sim$ Layer depth $n \sim$ # consecutive FC layers $d \sim$ # neurons	< 12 < 3 $\in \{512, 256, 128\}$
Termination State	$s \sim$ Previous State $t \sim$ Type	Global Avg. Pooling/Softmax

Table 1: **Experimental State Space.** For each layer type, we list the relevant parameters and the values each parameter is allowed to take.

state 就是在从浅到深生成过程中当前的操作（比如卷积），而不同于上一篇是当前的网络结构。注意到每种操作都包含一个参数 layer depth，那么实际上可以去掉 state 这个概念，只在 action 空间中进行探索（multi-armed bandit）。搜索过程如下：

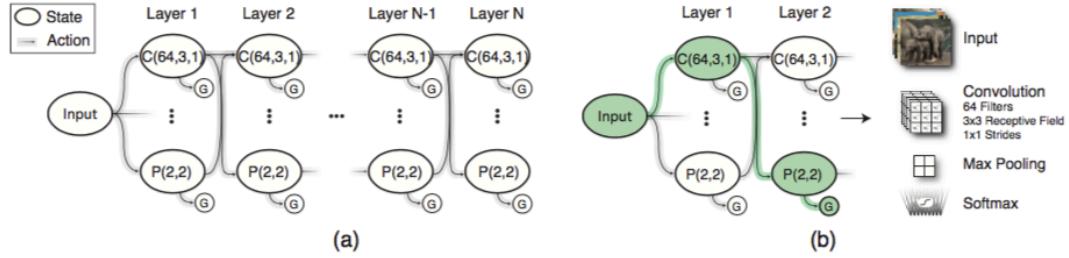


Figure 2: **Markov Decision Process for CNN Architecture Generation:** Figure 2(a) shows the full state and action space. In this illustration, actions are shown to be deterministic for clarity, but they are stochastic in experiments. $C(n, f, l)$ denotes a convolutional layer with n filters, receptive field size f , and stride l . $P(f, l)$ denotes a pooling layer with receptive field size f and stride l . G denotes a termination state (Softmax/Global Average Pooling). Figure 2(b) shows a path the agent may choose, highlighted in green, and the corresponding CNN topology.

3. Efficient Architecture Search by Network Transformation

在上一篇文章之后，有一些工作为了解决 NAS+RL 的效率问题展开，这篇发在 AAAI18 的文章就是其中的一篇。不同于前两篇将 action 的序列作为 state，本文是直接讲一个网络结构作为 state，action 为将该网络变得更深/更宽。不单单是继承网络参数，基于 net2net，本文可以在改变网络结构的同时不改变网络的输出，在此基础上训练准确率基本不会比原来更差。

来看一下整体的结构：

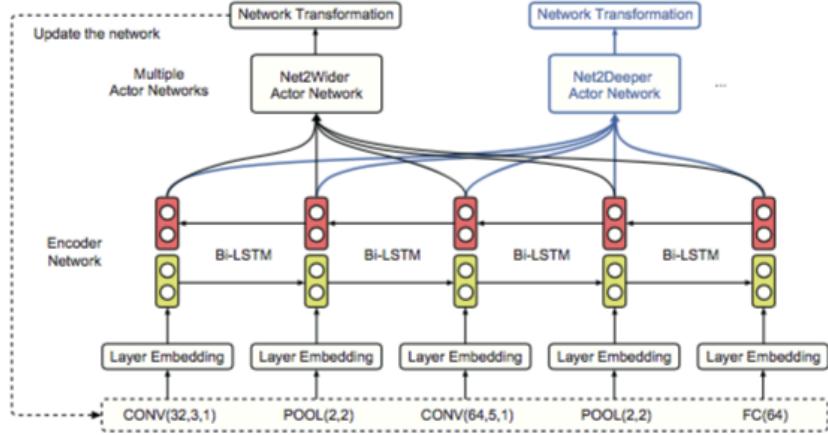


Figure 1: Overview of the RL based meta-controller in EAS, which consists of an encoder network for encoding the architecture and multiple separate actor networks for taking network transformation actions.

将每一层嵌入到一个低维表示，然后通过一个 bi-lstm 产生对宽度和深度的 action，是一个 seq2seq 结构。

Net2Wider actor:

结构如下图，对于每一个层经过 sigmoid 产生是否应该加宽的概率值：

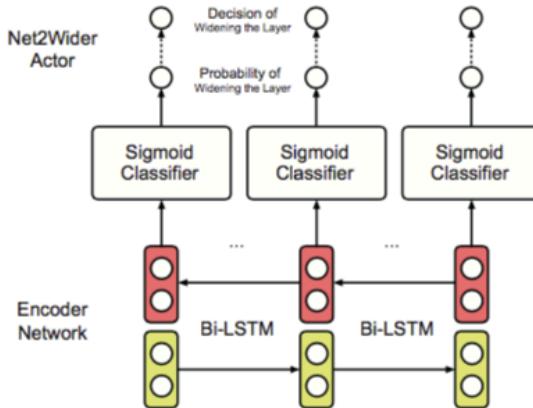


Figure 2: Net2Wider actor, which uses a shared sigmoid classifier to simultaneously determine whether to widen each layer based on its hidden state given by the encoder network.

KI 是一个卷积层，用一个四维数组表示： $(k_w^l, k_h^l, f_i^l, f_o^l)$ ，如果想令网络变宽，利用如下的 map function：

$$G_l(j) = \begin{cases} j & 1 \leq j \leq f_o^l \\ \text{random sample from } \{1, \dots, f_o^l\} & f_o^l < j \leq \hat{f}_o^l \end{cases}.$$

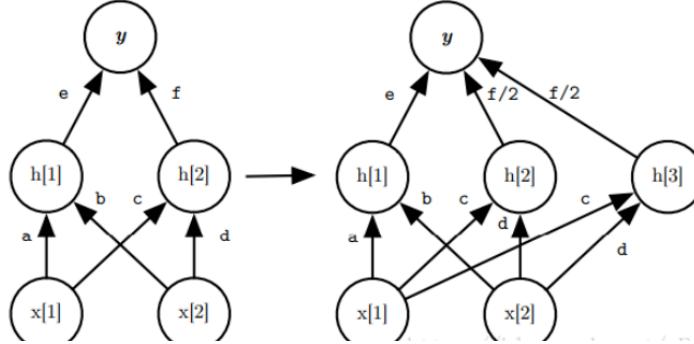
将当前的卷积层变为如下形式：

$$\hat{\mathbf{K}}_l[x, y, i, j] = \mathbf{K}_l[x, y, i, G_l(j)].$$

将下一卷积层变为:

$$\hat{\mathbf{K}}_{l+1}[x, y, j, k] = \frac{\mathbf{K}_{l+1}[x, y, G_l(j), k]}{|\{z|G_l(z) = G_l(j)\}|}.$$

对全连接层操作类似;



Net2Deeper actor:

结构如图，对于新的卷积层，只需要使用 identity filters。对于新的全连接层，权重矩阵使用单位矩阵即可。和下层保持同样的宽度，之后可以通过 Net2Wider actor 变得更宽。结构如下图：

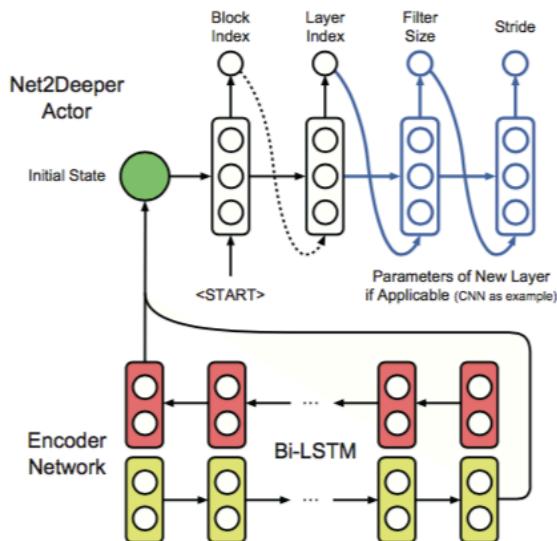


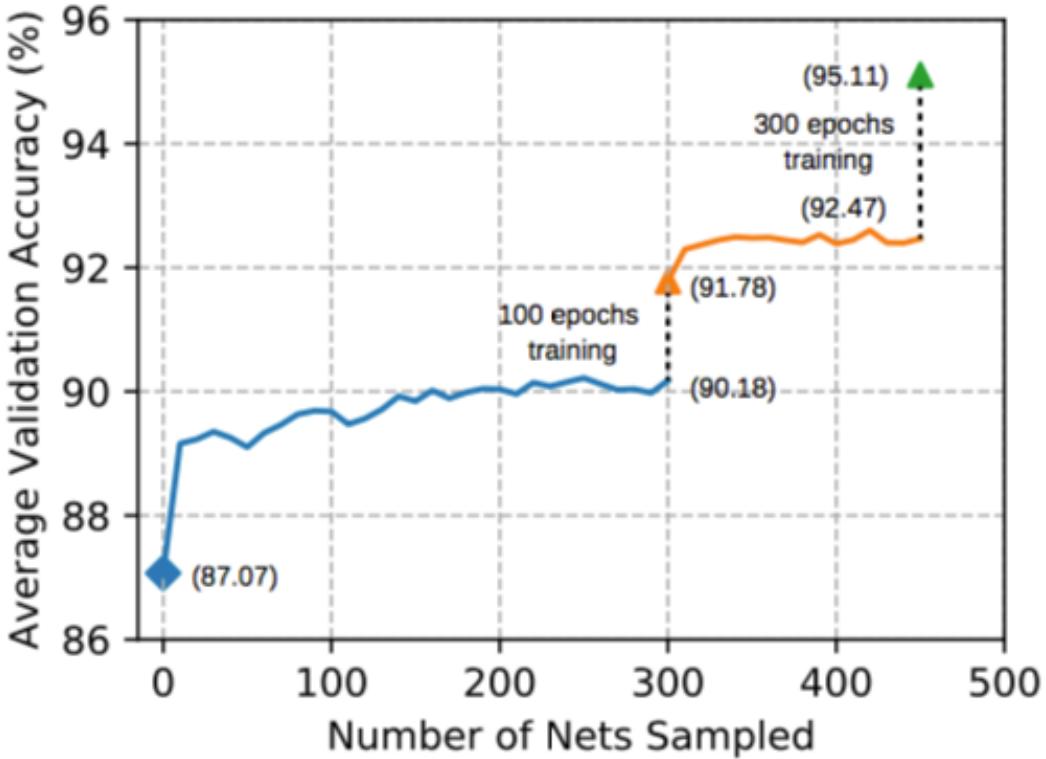
Figure 3: Net2Deeper actor, which uses a recurrent network to sequentially determine where to insert the new layer and corresponding parameters for the new layer based on the final hidden state of the encoder network given the input architecture.

整个学习的流程是这样的，以下表这样的网络作为起始点开始训练，每次随机采样 10 个现有的网络 take transformation action，具体来说，进行 5 次加深 4 次加宽，得到新的网络结构之后，训练 20 个 epoch（不一定收敛），用现在

的准确率作为 reward 来训练 meta-controller。采样 300 个网络之后，全部训练至收敛，将其中较好的网络结构作为新的起始点，进行第二阶段的训练。

Table 1: Simple start point network. $C(n, f, l)$ denotes a convolutional layer with n filters, filter size f and stride l ; $P(f, l, \text{MAX})$ and $P(f, l, \text{AVG})$ denote a max and an average pooling layer with filter size f and stride l respectively; $FC(n)$ denotes a fully-connected layer with n units; $SM(n)$ denotes a softmax layer with n output units.

Model Architecture	Validation Accuracy (%)
$C(16, 3, 1), P(2, 2, \text{MAX}), C(32, 3, 1), P(2, 2, \text{MAX}), C(64, 3, 1), P(2, 2, \text{MAX}), C(128, 3, 1), P(4, 4, \text{AVG}), FC(256), SM(10)$	87.07



四. 基于进化算法的 NAS

1. 遗传算法与进化策略

遗传算法 (GA) 是一种模拟自然界种群的进化的学习方法，传统的遗传算法是在二进制串的空间的进行探索的，我们会令一个二进制串对应真实的搜索空间的一个点，这个相互对应的过程称为编码和解码。每个串称为一个个体，一组串的集合称为种群，定义 fitness function 来评估个体的适应度，GA 以种群为单位进行进化，进化的目标即为生成 fitness 高的个体：

开始循环直至最优个体的 fitness 小于某阈值：

1. 选择：对现有种群，遵照适应度越高，选择概率越大的原则选择一部分个体放入新种群。
2. 交叉：从新种群中选择一些成对的个体作为父方和母方，进行交叉，产生两个子代，放入新种群。
3. 变异：随机选择新种群中的个体进行变异。
4. 评估新种群中每个个体的适应度

其中选择的方法有所谓轮盘赌选择（roulette wheel selection），即按照各个 fitness 的比例采样；锦标赛选择（tournament selection），即成对的选择个体，以 p 的概率淘汰适应度低的个体；排序选择（rank selection），将个体按适应度排序，个体被选择的概率与所处的位置成比例。不同选择会影响算法收敛的速度，比如使用轮盘赌选择，若出现某个个体 fitness 特别大，则很有可能在几代之内占据整个种群，使得基因池丧失了多样性。

GA 的特点是，比起我们常见的基于反向传播的神经网络，GA 不是通过梯度更新的（新的点在上一个点的邻域内），而可能产生和父母完全不同的子代，这就使得 GA 有可能找到全局最优点。其实按照这样的说法，random search 也有可能找到全局最优啊，GA 也只不过是比 random 多了保留上一代部分特征的能力，仍然是很低效的优化方法，因此也在被学界慢慢遗忘。但 GA 有个优势在于容易并行实现，在 GPU 能力越来越强的今天，GA 又被一些大公司拾起来做 AutoML。不过 GA 只被用在搜索网络架构这一层，在网络架构验证时，仍然是基于梯度下降。

2. Large-Scale Evolution of Image Classifiers

首先是来自 google brain 的一篇文章，发表在 ICML2017，最早将进化算法用在了 NAS 问题中。这篇文章的就是我们上面说的第一种讲故事的角度，文章从一个非常基础的初始点开始，使用了非常大的搜索空间和细粒度的变异算子，完全不会引入人为的偏置，从而验证了进化算法在 NAS 问题的有效性

进化算法：将每个产生的模型作为一个个体，将模型在验证集上的准确度作为 fitness，使用了锦标赛选择，即一个 worker 随机从种群中选两个模型，淘汰 fitness 小的那个，不进行交叉操作，直接将另一个模型作为父母进行变异，得到子代后计算其适应度然后放回种群。

并行：算法是并行进行的，有多个 worker 同时在这个种群上操作（population 1/4 的数量），若两个 worker 冲突，则后一个 worker 停止并重新选择。

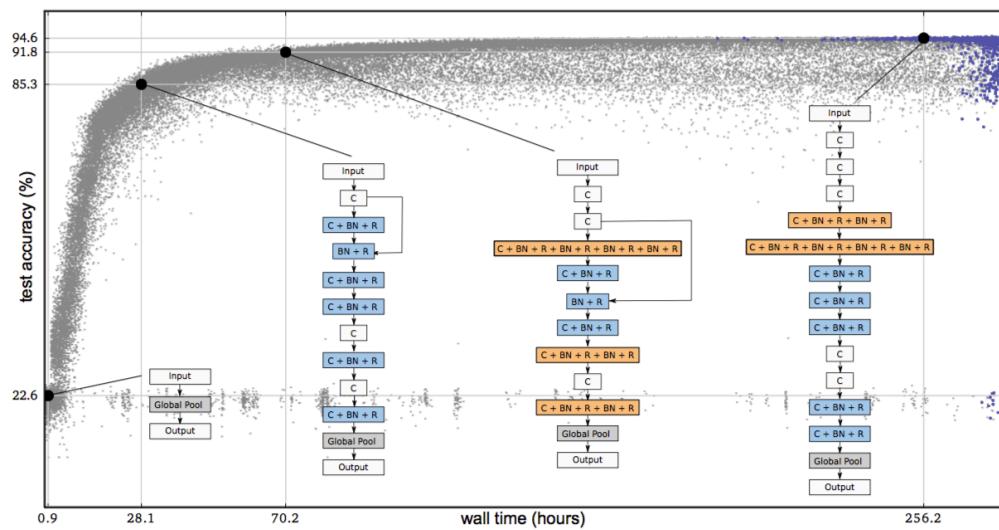
下面来看一下变异算子都有什么：

- ALTER-LEARNING-RATE (sampling details below).
- IDENTITY (effectively means “keep training”).
- RESET-WEIGHTS (sampled as in He et al. (2015), for example).
- INSERT-CONVOLUTION (inserts a convolution at a random location in the “convolutional backbone”, as in Figure 1. The inserted convolution has 3×3 filters, strides of 1 or 2 at random, number of channels same as input. May apply batch-normalization and ReLU activation or none at random).
- REMOVE-CONVOLUTION.
- ALTER-STRIDE (only powers of 2 are allowed).
- ALTER-NUMBER-OF-CHANNELS (of random conv.).
- FILTER-SIZE (horizontal or vertical at random, on random convolution, odd values only).
- INSERT-ONE-TO-ONE (inserts a one-to-one/identity connection, analogous to insert-convolution mutation).
- ADD-SKIP (identity between random layers).
- REMOVE-SKIP (removes random skip).

可以看到搜索空间很大，learning rate 都在搜索空间中，事实上搜索空间还包括模型的参数（比如某个卷积核具体的值），也就是说进化算法完成后，会得到一个 learning rate 合适的已经训练好的模型；identity 也就是保持不变，文章设置在子代产生时会继承父母的参数，而每个子代产生后会被 worker 训练固定轮数（但不保证收敛），所以还设置了 reset weights 的变异算子避免陷入局部最优；后面对 kernel size、stride、channels 的变异，也不会限制到固定的某几个值，而是从一个均匀分布采样；加入 skip 的融合方式是 add，因此 resnet 在搜索空间中，而 DenseNet 不在。

下面是一些结果，注意到网络自己学习到的模型和人为设计的并不相同，有些卷积层后有多个非线性层的叠加。

STUDY	STARTING POINT	CONSTRAINTS	POST-PROCESSING	PARAMS.	C10+	C100+
BAYESIAN (SNOEK ET AL., 2012)	3 LAYERS	FIXED ARCHITECTURE, NO SKIPS	NONE	—	90.5%	—
Q-LEARNING (BAKER ET AL., 2016)	—	DISCRETE PARAMS., MAX. NUM. LAYERS, NO SKIPS	TUNE, RETRAIN	11.2 M	93.1%	72.9%
RL (ZOPH & LE, 2016)	20 LAYERS, 50% SKIPS	DISCRETE PARAMS., EXACTLY 20 LAYERS	SMALL GRID SEARCH, RETRAIN	2.5 M	94.0%	—
RL (ZOPH & LE, 2016)	39 LAYERS, 2 POOL LAYERS AT 13 AND 26, 50% SKIPS	DISCRETE PARAMS., EXACTLY 39 LAYERS, 2 POOL LAYERS AT 13 AND 26	ADD MORE FILTERS, SMALL GRID SEARCH, RETRAIN	37.0 M	96.4%	—
EVOLUTION (OURS)	SINGLE LAYER, ZERO CONVS.	POWER-OF-2 STRIDES	NONE	5.4 M 40.4 M ENSEMB.	94.6% — 95.6%	77.0%



3. Regularized Evolution for Image Classifier Architecture Search

随后，google brain 又发表了基于进化算法的第二篇文章，发表于 AAAI 2019。在验证了进化算法的能力之后，google 尝试加入一些先验知识，使用了基于 cell 的结构，目的是要产生类似 Inception 的 block，搜索空间很小。在

google AI blog 中是这样描述的： If the evolutionary algorithm is contributing meaningfully, the final networks should be significantly better than the networks we already know can be constructed within this search space.

搜索空间·

如下图，一个前馈的类似于 Inception 的 block 称为 cell，每个 cell 接收前两个 cell 的输出作为输入。cell 有两种：normal cell 中所有的结构 stride 都是 1，不会改变 feature map 的大小，reduce cell 后面会进行一个 stride 为 2 的 reduce 操作来讲 feature map 缩小一半。一个 reduce cell 和 N 个 normal cell 组成一个 stake，网络结构是多个 stake 的重复。而我们要进行搜索的是两种 cell 的结构。

具体来说，cell 接收两组 feature map 作为输入，然后先通过一个 pairwise combination 得到第三组 feature map，之后通过不断从已有的 feature map 中选两组进行结合产生新的 feature map，最后没有被使用的输出结合起来作为 cell 的输出。其中使用的操作是从一个固定的集合中选择，包含常见的卷积、池化操作，也可以不进行操作（None）。

Cell 训练好以后，还需手动调整两个参数，分别是每个卷积层的 output channel F 和每个 stack 中 normal cell 的数量。

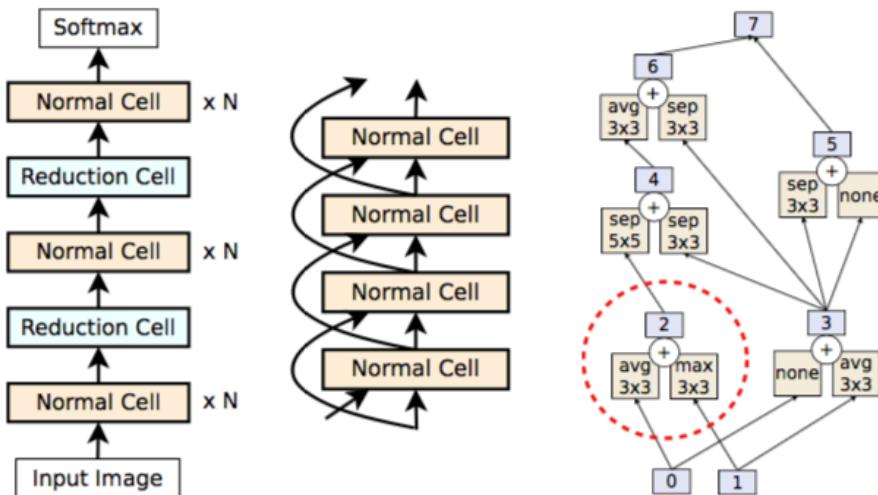


Figure 1: NASNet Search Space [54]. LEFT: the full outer structure (omitting skip inputs for clarity). MIDDLE: detailed view with the skip inputs. RIGHT: cell example. Dot-dashed line demarcates a pairwise combination.

进化算法：

文章使用了锦标赛选择，即随机选择 S 个个体，其中 fitness 高的可以产生子代，但是稍微不同的一点在于，fitness 低的并不会被淘汰。文章通过使用一个队列存放所有存活的模型，总是淘汰最老的。这种进化算法称为 aging evolution，这种算法的优势在于不会过早收敛，具体来说，锦标赛选择使得 fitness 相对低的个体也有机会产生子代，aging evolution 使得优秀的个体也会被淘汰，使种群的多样性更好。

Algorithm 1 Aging Evolution

```
population ← empty queue           ▷ The population.  
history ← Ø                         ▷ Will contain all models.  
while |population| < P do          ▷ Initialize population.  
    model.arch ← RANDOMARCHITECTURE()  
    model.accuracy ← TRAINANDEVAL(model.arch)  
    add model to right of population  
    add model to history  
end while  
while |history| < C do            ▷ Evolve for C cycles.  
    sample ← Ø                      ▷ Parent candidates.  
    while |sample| < S do  
        candidate ← random element from population  
        ▷ The element stays in the population.  
        add candidate to sample  
    end while  
    parent ← highest-accuracy model in sample  
    child.arch ← MUTATE(parent.arch)  
    child.accuracy ← TRAINANDEVAL(child.arch)  
    add child to right of population  
    add child to history  
    remove dead from left of population      ▷ Oldest.  
    discard dead  
end while  
return highest-accuracy model in history
```

变异算子也非常的简单，有两种变异的方式供选择：hidden state 变异，即更改 cell 中某一个操作的输入；Op 变异，即对某一个操作进行替换。

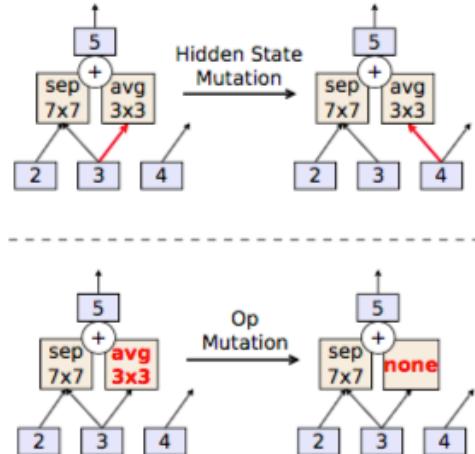


Figure 2: Illustration of the two mutation types.

实验结果表明，这样设计的网络在 Imagenet 数据集上取得了 state-of-art 的结果。

4. HIERARCHICAL REPRESENTATIONS FOR EFFICIENT ARCHITECTURE SEARCH

发表于 ICLR2018，CMU 和 google 联合的工作，另辟蹊径的提出了一种层次化的搜索空间。前两篇文章中其实都包含一个 encoding 的过程，也就是将具体的网络结构表示为一个 graph，这个 graph 的作用类似于传统 GA 的二进制串，但是都比较直观。而这篇文章最重要的贡献就在于一种层次化的网络结构表示方法：

先看单层，文章用一个单个输入单个输出的有向图来表示一个网络，其中节点是 feature map，也就是数据，边为各种基础操作。一个网络结构被表示为 (G, o) ，其中 o 是各个操作的索引组成的向量， G 是一个邻接矩阵，记录两个 feature map 之间使用哪个操作：

1. A set of available operations $\mathcal{O} = \{o_1, o_2, \dots\}$.
2. An adjacency matrix G specifying the neural network graph of operations, where G_{ij} means that the k -th operation o_k is to be placed between nodes i and j .

第 i 个 feature map 是以前 $i-1$ 个 feature map 作为输入，merge 是在宽度上的合并，即 concatenation：

$$x_i = \text{merge} [\{o_{G_{ij}}(x_j)\}_{j < i}], \quad i = 2, \dots, |G|$$

注意，这种表示方式 feature map 的数量是固定的，即作为一个超参需要手动调整。

到目前为止，网络看起来有点像上一篇的 cell 的定义，下面引入多层次表示，将这样形成的“单元”作为更高层的基础操作：

$$o_m^{(\ell)} = \text{assemble} \left(G_m^{(\ell)}, \mathcal{O}^{(\ell-1)} \right), \quad \forall \ell = 2, \dots, L$$

上式表示，第 1 层的第 m 个 op，由第 1-1 层的 ops 根据矩阵 G 组装。

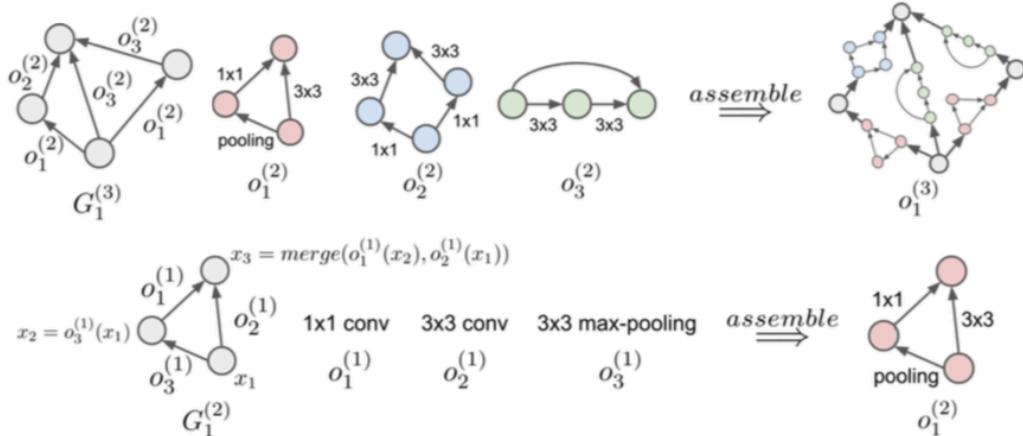


Figure 1: An example of a three-level hierarchical architecture representation. The bottom row shows how level-1 primitive operations $o_1^{(1)}, o_2^{(1)}, o_3^{(1)}$ are assembled into a level-2 motif $o_1^{(2)}$. The top row shows how level-2 motifs $o_1^{(2)}, o_2^{(2)}, o_3^{(2)}$ are then assembled into a level-3 motif $o_1^{(3)}$.

来看下第一层的的 ops 都有什么：

- 1×1 convolution of C channels
- 3×3 depthwise convolution
- 3×3 separable convolution of C channels
- 3×3 max-pooling
- 3×3 average-pooling
- identity

卷积的 `stride` 都为 1, `padding` 为 SAME, 后面都加 BN 和 relu。作者认为这些简单的操作就足够形成复杂的网络, 更大的感受野可以堆叠多个卷积层, 想要更宽的卷积网络可以通过上面描述的 `merge` 等效实现。这是一个良好的定义---有足够的好的网络结构包含在搜索空间中, 需要搜索的超参数也不是很多(排除了卷积 pooling 的宽度、size、stride, 激活层等等)。

定义好了搜索空间, 来看进化算法, 变异算子很简单就是将 graph 中某个 op 从 k 变为 k' , 其中 l, m, I, j 都是随机采样的:

$$[G_m^{(\ell)}]_{ij} = k'$$