AutoGrow: Automatic Layer Growing in Deep Convolutional Networks

Wei Wen  
Duke University  
wei.wen@duke.edu

Feng Yan  
University of Nevada – Reno  
fyan@unr.edu

Hai Li  
Duke University  
hai.li@duke.edu

Abstract

We propose AutoGrow to automate depth discovery in Deep Neural Networks (DNNs): starting from a shallow seed architecture, AutoGrow grows new layers if the growth improves the accuracy; otherwise, the growth stops and the network depth is discovered. The residual and plain blocks are used as growing sub-modules to study DNNs with and without shortcuts. We propose generic growing and stopping policies to minimize human efforts spent on the optimal depth search. Our experiments show that by applying the same policy to different tasks, AutoGrow can always discover network depth effectively and achieve state-of-the-art accuracy on various datasets of MNIST, FashionMNIST, SVHN, CIFAR10, CIFAR100 and ImageNet. Comparing to Neural Architecture Search (NAS) that often designs a gigantic search space and consumes tremendous resources, AutoGrow lies at the other end of the research spectrum: it focuses on efficient depth discovery and reduces the growing and searching time to a level similar to that of training a single DNN. Thus, AutoGrow is able to scale up to large datasets such as ImageNet. Our study also reveals that previous Network Morphism is sub-optimal for increasing layer depth. Finally, we demonstrate that AutoGrow enables the training of deeper plain networks, which has been problematic even using Batch Normalization.

1. Introduction

Layer depth is one of the decisive factors of the success of Deep Neural Networks (DNNs). For example, image classification accuracy keeps improving as the depth of network models grows [16, 30, 33, 11, 14]. Although shallow networks cannot ensure high accuracy, DNNs composed of too many layers may suffer from over-fitting and convergence difficulty in training. How to obtain the optimal depth for a DNN still remains mysterious. For instance, ResNet-152 [11] uses 3, 8, 36 and 3 residual blocks under output sizes of $56 \times 56$, $28 \times 28$, $14 \times 14$ and $7 \times 7$, respectively, which don’t show an obvious quantitative relation. In practice, people usually reply on some brute-force trials and tests to obtain the depth of a network: they first design a DNN with a specific depth and then train and evaluate the network on a given dataset; Finally, they change the depth and repeat the procedure until the accuracy meets the requirement. Besides the high computational cost induced by the iteration process, such trial & test iterations must be repeated whenever dataset changes. In this paper, we propose AutoGrow that can automate depth discovery over different datasets meanwhile achieving state-of-the-art accuracy.

Previously, VggNet [30] and DropIn [31] explored the methods of adding new layers onto shallower DNNs; Network Morphism [36, 35, 5] increased the layer depth meanwhile preserving the function of the shallow net. Their objective was enabling or accelerating the training of deeper DNNs, but we focus on automating depth discovery. More differently, all the above works need to manually determine the number and the locations of new layers, and new layers are usually added only once. In contrast, AutoGrow automatically learns the number and the locations of new layers and activates multiple growing/morphing.

Neural Architecture Search (NAS) [39, 40] aims to search DNNs by exploring a gigantic search space. Such an approach, however, often requires tremendous computing resources. To speedup the search process, Network Mor-
phism [36, 35, 5] is integrated into NAS by using a random sampler [8] or a reinforcement learning agent [3, 4]. However, NAS is naturally slow and it is not yet able to support large-scale tasks like ImageNet. AutoGrow lies at the other end of the research spectrum, where search space is limited to depth but the searching time is as short as training a single DNN, demonstrating substantially enhanced scalability. We also explored the integration of Network Morphism into AutoGrow for growing layers. Interestingly, simple random initialization outperforms complex Network Morphism. We visualize the optimization trajectory in the parameter space and show that the initialization given by Network Morphism is inadequate for training a deeper net. This sheds light that Network Morphism may give sub-optimal signals for NAS.

Figure 1 illustrates a simple example of AutoGrow. It starts from the shallowest backbone network and gradually grows sub-modules\(^1\); the growth stops once a stopping policy is satisfied. We study multiple growing policies and surprisingly find that a simple Periodic Policy surpasses complicated Morphing Policy based on Network Morphism. Unlike previous wisdom [8, 3, 4], we find that it is more effective to grow sub-modules before a shallow net converges. This is because a fully converged shallow net is an inadequate initialization for training deeper net. To tackle this, we avoid full convergence during the growing by using (1) a constant large learning rate; (2) random initialization of new sub-modules; and (3) a short interval between growths. Our contributions are:

- We propose AutoGrow\(^2\) to automate DNN layer growing and depth discovery. With the same set of hyper-parameters, it adapts network depth to various datasets including MNIST, FashionMNIST, SVHN, CIFAR10 and CIFAR100. AutoGrow can also discover shallower DNNs when the dataset is a subset.
- AutoGrow demonstrates high efficiency and scales up to ImageNet, because the layer growing is as fast as training a single DNN. On ImageNet, it discovers a new ResNets with better trade-off between accuracy and computation complexity.
- We study Network Morphism in AutoGrow and visualize the optimization trajectory in the parameter space. Our experiments imply that Network Morphism is sub-optimal to train deeper networks.
- AutoGrow is able to train deeper plain DNNs, which has been problematic even with Batch Normalization.

2. Related Work

Neural Architecture Search (NAS) [39] and neural evolution [25, 1, 32, 21, 28] can search network architectures including layer depth, however, with very long search time. For example, on CIFAR10 dataset, NAS used 22, 400 Nvidia K40 GPU hours [39] and neural evolution [28] took 60, 000 GPU hours [8] to perform the search. To accelerate NAS, one-shot models [29, 26, 2], DARTS [22] and NAS with Transferable Cell [40, 20] were proposed. The search time reduces dramatically but is still long from practical perspective. For example, 2, 000 Nvidia P100 GPU hours were used to search a transferable cell (a.k.a sub-module) [40]. It is also very challenging to deploy these methods to larger datasets such as ImageNet. Moreover, DNN depth in one-shot models, DARTS and Transferable Cell based NAS has to be predefined. In contrast, we target to learn the depth of DNNs and our AutoGrow can scale up to ImageNet thanks to its short depth learning time, which is as efficient as training a single DNN.

Network Morphism [36, 35, 5, 8, 3, 4] refers to approaches which morph a smaller DNN to a wider or deeper DNN meanwhile keeping the same loss function of the smaller DNN. It was proposed to enable accelerate the training of wider or deeper DNNs. Network Morphism is orthogonal to our work and we integrate it into our AutoGrow framework for each growth. The corresponding experiments show that simple random initialization outperforms Network Morphism. We hypothesize that Network Morphism gives an inadequate initialization but noises help to escape it. We visualized optimization trajectory and proved our hypothesis is true.

In addition to architecture search which requires to train lots of DNNs from scratch, there are also many studies on learning neural structures within a single training. Structure pruning [37, 18, 17, 27, 12, 24, 23, 7, 13, 10] and growing [9, 38, 27, 7] were proposed for different goals, such as efficient inference [37, 18, 17, 24, 23, 7, 13, 10], lifelong learning [38] and model adaptation [9, 27]. However, those works fixed the network depth and limited structure learning within the existing layers. Optimization over a DNN with fixed depth is easier as the skeleton architecture is known. AutoGrow performs in a scenario where the DNN depth is unknown hence we need to seek for the optimal depth. A growing sub-network was utilized for semi-supervised learning [34], and the new network fully converged before a next growth; AutoGrow targets on supervised learning and grows the network before the new network converges. We find the growth ahead of convergence is important to avoid accuracy loss and early stop of the growth.

AdaNet [6] can adapt neural networks to different datasets. It designs a pool of sub-networks and selects the final network as a combination of some sub-networks to minimizes the loss function. The number of combinations explodes exponentially with the pool size so that the search time can be very long. As a consequence, the largest problem reported in AdaNet is only a binary classification simplified from CIFAR10.

---

\(^1\)A sub-module can be one or more layers, e.g., a residual block.
\(^2\)Code: https://github.com/wenwei202/autogrow.
3. AutoGrow – A Depth Growing Algorithm

Figure 1 gives an overview of the proposed AutoGrow. In this paper, we use network, sub-networks, sub-modules and layers to describe the architecture hierarchy. A typical sub-network is composed of sub-modules, which have the same output size. A typical sub-module (e.g. a residual block) is an elementary building block composed of a few layers. In this section, we rigorously formulate a generic version of AutoGrow which will be materialized later. A deep convolutional network \( g(X_0) \) is a cascade of sub-networks by composing functions as

\[
g(X_0) = l \left( f_{M-1} \left( \cdots f_1 \left( f_0 (X_0) \right) \cdots \right) \right),
\]

where \( X_0 \) is an input image, \( M \) is the number of sub-networks, \( l(\cdot) \) is a loss function following a classifier, and \( f_i(X_i) \) is a sub-network that operates on an input image or a feature tensor \( X_i \in \mathbb{R}^{c_i \times h_i \times w_i} \) and outputs \( X_{i+1} \). Here, \( c_i \) is the number of channels, and \( h_i \) and \( w_i \) are spatial dimensions. \( f_i(X_i) \) is a simplified notation of \( f_i(X_i; \mathcal{W}_i) \), where \( \mathcal{W}_i \) is a set of sub-modules’ parameters within the \( i \)-th sub-network. Thus \( \mathcal{W} = \{ \mathcal{W}_i : i = 0 \ldots M - 1 \} \) denotes the whole set of parameters in the DNN. A sub-network \( f_i(X_i; \mathcal{W}_i) \) includes one or multiple sub-modules, and a sub-module contains one or more layers. To facilitate growing, the following properties are supported within a sub-network: (1) the first sub-module usually reduces the size of input feature maps, e.g., using pooling or convolution with a stride; and (2) all sub-modules in a sub-network maintain the same output size. As such, our framework can support popular networks, including VggNet-like plain networks [30], GoogLeNet [33], ResNets [11] and DenseNets [14]. In this paper, we select ResNets and VggNet-like nets as representatives of DNNs with and without shortcuts, respectively. Algorithm 1 describes the AutoGrow algorithm. In brief, AutoGrow starts with the shallowest net where every sub-network has only one sub-module for spatial dimension reduction. AutoGrow loops over all sub-networks in sequence. For each sub-network, AutoGrow keeps stacking a new sub-module until the growth cannot improve the accuracy further. The details of our method will be materialized in the following subsections.

3.1. Seed Shallow Networks and Sub-modules

On MNIST, FashionMNIST, SVHN, CIFAR10 and CIFAR100, we explore growing depth for four types of DNNs:

1. Basic3ResNet: the same ResNet used for CIFAR10 in [11], which has 3 residual sub-networks with output spatial sizes of \( 32 \times 32 \), \( 16 \times 16 \) and \( 8 \times 8 \), respectively;
2. Basic4ResNet: a variant of ResNet with basic residual blocks \(^3\) used for ImageNet in [11]. The difference is that there is only a \( 3 \times 3 \) convolution layer before the first residual. There are 4 sub-networks with output spatial sizes of \( 32 \times 32 \), \( 16 \times 16 \), \( 8 \times 8 \) and \( 4 \times 4 \), respectively;
3. Plain3Net: a VggNet-like plain net by removing shortcuts in Basic3ResNet;
4. Plain4Net: a VggNet-like plain net by removing shortcuts in Basic4ResNet.

In plain DNNs, a sub-module is a stack of convolution, Batch Normalization and ReLU; in residual DNNs, a sub-module is a residual block. In AutoGrow, a sub-network is a stack of all sub-modules with the same output spatial size. Unlike [11] which manually designed the depth, AutoGrow starts from a seed architecture in which each sub-network having only one sub-module and automatically learns the number of sub-modules.

On ImageNet, we apply the same backbones in [11]

\[^3\text{A basic residual block contains two convolutions and one shortcut.}\]
as the seed architectures. A seed architecture has only one sub-module under each output spatial size. For a ResNet using basic residual blocks or bottleneck residual blocks [11], we respectively name it as Basic4ResNet or Bottleneck4ResNet. Plain4Net is also obtained by removing shortcuts in Basic4ResNet.

3.2. Sub-module Initializers

Here we explain how to initialize a new sub-module \( W \) by \( \text{initializer}(W) \) in Algorithm 1. Network Morphism changes DNN architecture meanwhile preserving the loss function via special initialization of new layers, that is,

\[
g(X_0; W) = g(X_0; W \cup W) \forall X_0. \tag{2}
\]

A residual sub-module shows a nice property: when stacking a residual block and initializing the last Batch Normalization layer as zeros, the function of the shallower net is preserved but the DNN is morphed to a deeper net. Thus, Network Morphism can be easily implemented by this zero initialization (ZeroInit).

In this work, all layers in \( W \) are initialized using default randomization, except for a special treatment of the last Batch Normalization layer in a residual sub-module. Besides ZeroInit, we propose a new AdamInit for Network Morphism. In AdamInit, we freeze all parameters except the last Batch Normalization layer in \( W \), and then use Adam optimizer [15] to optimize the last Bath Normalization for maximum 10 epochs till the training accuracy of the deeper net is as good as the shallower one. After AdamInit, all parameters are jointly optimized. We view AdamInit as a Network Morphism because the training function is similar after AdamInit. We empirically find that AdamInit can usually find a solution in less than 3 epochs. We also study random initialization of the last Batch Normalization layer using uniform (UniInit) or Gaussian (GauInit) noises with a standard deviation 1.0. GauInit obtains the best result based on our experiments.

3.3. Growing Policies

A growing policy refers to \( \text{meetGrowingPolicy()} \) in Algorithm 1. Two growing policies are studied here:

1. Morphing Growth: \( \text{meetGrowingPolicy()} \) returns true when the improvement of validation accuracy is less than \( \tau \) in the last \( K \) epochs. In Morphing Growth, \( \text{AutoGrow} \) only grows when current network converges under an optimizer. Note that similar policies have been used for efficient NAS based on Network Morphism [8, 3, 4].

2. Periodic Growth: \( \text{meetGrowingPolicy()} \) always returns true, that is, the network always grows every \( K \) epochs. Therefore, \( K \) is also the growing period. In the best practice of \( \text{AutoGrow} \), \( K \) is small such that it grows before current network converges.

Our experiments show that Periodic Growth outperforms Morphing Growth. We hypothesize that a fully converged shallower net is an inadequate initialization to train a deeper net. We will perform experiments to prove this hypothesis and visualize optimization trajectory to illustrate it.

3.4. Stopping Policies

A stopping policy denotes \( \text{meetStoppingPolicy()} \) in Algorithm 1. Two stopping policies are studied:

1. Morphing Stop: \( \text{meetStoppingPolicy()} \) returns true when a sub-network just grew \( J \) epochs ago but the validation accuracy improves less than \( \tau \). Morphing Stop works with Morphing Growth, indicating that the last growth is meaningless and the sub-network reaches the maximum depth.

2. Periodic Stop: \( \text{meetStoppingPolicy()} \) returns true when the validation accuracy improves less than \( \tau \) in the last \( J \) epochs.

Hyper-parameters \( \tau, J \) and \( K \) control the operation of \( \text{AutoGrow} \) and can be easily setup and well generalized. \( \tau \) denotes the significance of accuracy improvement for classification. We simply set \( \tau = 0.05\% \) in all experiments. \( J \) represents how many epochs to wait for an accuracy improvement before stopping the growth of a sub-network. It is more meaningful to consider stopping when the new net is trained to some extent. As such, we set \( J \) to the number of epochs \( T \) under the largest learning rate when training a baseline. \( K \) means how frequently \( \text{AutoGrow} \) checks the polices. In Morphing Growth and Morphing Stop, we simply set \( K = T \). In Periodic Growth and Periodic Stop, \( K \) is set to a fraction of \( T \) to enable faster growth before convergence; more importantly, \( K = 3 \) is very robust to all networks and datasets.

4. Experiments

In this paper, we use Basic3ResNet-2-3-2, for instance, to denote the model architecture which contains 2, 3 and 2 sub-modules in the first, second and third sub-networks, respectively. Sometimes we simplify it as 2-3-2 for convenience. \( \text{AutoGrow} \) always starts from the shallowest depth of 1-1-1 and uses the maximum validation accuracy as the metric to guide growing and stopping. All DNN baselines are trained by SGD with momentum 0.9 using staircase learning rate. The initial learning rate is 0.1 in ResNets and 0.01 in plain networks. On ImageNet, baselines are trained using batch size 256 for 90 epochs, between which learning rate is decayed by 0.1x at epoch 30 and 60. In all other smaller datasets, baselines are trained using batch size 128 for 200 epochs and learning rate is decayed by 0.1x at epoch 100 and 150.
Morphing
ResNet-32
Learning rate decay
(a) (b)

Figure 2: An optimization trajectory comparison between (a) Network Morphism and (b) training from scratch.

### 4.1. Suboptimum of Network Morphism

In this section, we study Network Morphism itself and its integration into our AutoGrow as Network Morphism can be used to gradually morph the depth of DNNs [36, 35, 5, 8, 3, 4]. When studying Network Morphism, we take the following steps: 1) train a shallower ResNet to converge, 2) stack residual blocks on top of each sub-network to morph to a deeper net, 3) use ZeroInit or AdamInit to initialize last Batch Normalization layers, and 4) train the deeper net in a standard way. We compare the accuracy difference (“Δ”) between Network Morphism and the deep net trained from scratch. Table 1 summaries our results. Network Morphism has a lower accuracy (negative “Δ”) in all the cases.

We hypothesize that a converged shallower net may not be an adequate initialization. Figure 2 visualizes and compares the optimization trajectories of Network Morphism and the training from scratch. In this figure, the shallower net is Basic3ResNet-3-3-3 (ResNet-20) and the deeper one is Basic3ResNet-5-5-5 (ResNet-32) in Table 1. The initializer is ZeroInit. The visualization method is extended from [19]. Points on the trajectory are evenly sampled every a few epochs. To maximize the variance of trajectory, we use PCA to project from a high dimensional space to a 2D space and use the first two Principle Components (PC) to form the axes in Figure 2. The contours of training loss function and the trajectory are visualized around the final minimum of the deeper net. When projecting a shallower net to a deeper net space, zeros are padded for the parameters not existing in the deeper net. We must note that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss remains decreasing in the high dimension while it appears in an opposite way in the 2D space. The sharp detour at the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection. It is possible that the loss increase along the trajectory does not truly represent the situation in high dimensional space, as the trajectory is just a projection.

**Table 1: Network Morphism tested on CIFAR10.**

| net backbone | shallower | deeper | initializer | accu % | Δ* |
|--------------|-----------|--------|-------------|--------|----|
| Basic3ResNet | 3-3-3     | 5-5-5  | ZeroInit    | 92.71  | -0.77 |
|              |           |        | AdamInit    | 92.82  | -0.66 |
| Basic3ResNet | 5-5-5     | 9-9-9  | ZeroInit    | 93.64  | -0.27 |
|              |           |        | AdamInit    | 93.53  | -0.38 |
| Basic4ResNet | 1-1-1-1   | 2-2-2-2| ZeroInit    | 94.96  | -0.37 |
|              |           |        | AdamInit    | 95.17  | -0.16 |

* Δ = (accuracy of Network Morphism) – (accuracy of training from scratch)

“Morphing” in Figure 2(a) indicates that the shallower net converges to a point that the deeper net struggles to escape. In contrast, Figure 2(b) shows that the trajectory of the direct optimization in the deeper space smoothly converges to a better minimum.

To further validate our hypothesis, we integrate Network Morphism into AutoGrow and refer to it as m-AutoGrow with “m-” denoting “Morphing.” More specific, we take ZeroInit or AdamInit as sub-module initializer and “Morphing Growth” and “Morphing Stop” policies in Algorithm 1. To recap, in this setting, AutoGrow trains a shallower net till it converges, then grows a sub-module which is initialized by Network Morphism, and repeats the same process till there is no further accuracy improvement.

In the interval of $K$ epochs – $\text{train}(g(X_0), K)$, “staircase” learning rate is used. The learning rate is reset to 0.1 at the first epoch, and decayed by 0.1× at epoch $\frac{K}{2}$ and $\frac{3K}{4}$. The results are shown in Table 2 by “staircase” rows, which illustrate that m-AutoGrow can grow a DNN multiple times and finally find a net. However, there are two problems: 1) the final accuracy is lower than training the found net from scratch, as indicated by “Δ”; 2) the depth learning stops too early with a relatively shallower net, while a deeper net beyond the found depth can achieve a higher accuracy as we will show in Table 6. These problems provide a circumstantial evidence of the hypothesis that Network Morphism gives a bad initialization. Thus, AutoGrow cannot receive signals to continue growing after a limited number of growths. Figure 3(a) visualizes the trajectory of m-AutoGrow corresponding to row “2-3-6” in Table 2. Along the trajectory, there are many trials to detour and escape an initialization from a shallower net.

### 4.2. Ablation Study for AutoGrow Design

Based on the findings in Section 4.1, we propose the following modifications to further enhance AutoGrow and refer it as p-AutoGrow, with “p-” denoting “Periodic”:

| dataset | learning rate | initializer | found net† | accu % | Δ* |
|---------|--------------|-------------|------------|--------|----|
| CIFAR10 | staircase     | ZeroInit    | 2-3-6      | 91.77  | -0.16 |
|         | staircase     | AdamInit    | 3-4-3      | 92.21  | -0.59 |
| CIFAR100| staircase    | ZeroInit    | 2-2-4      | 92.23  | 0.16 |
|         | staircase    | AdamInit    | 3-4-4      | 92.60  | -0.41 |
|         | staircase    | UnInit      | 3-4-4      | 92.93  | -0.08 |
|         | staircase    | UnInit      | 2-4-3      | 93.12  | 0.55 |

† Basic3ResNet

* Δ = (accuracy of m-AutoGrow) – (accuracy of training from scratch)
1. Use a large constant learning rate for growing, i.e., 0.1 for residual networks and 0.01 for plain networks.
2. Use random initialization (UniInit or GauInit) as noises to escape from an inadequate initialization;
3. Grow rapidly before a shallower net converges by taking Periodic Growth and Stop policies with a small $K$.

We first test the impact of the proposed modifications on $m$-AutoGrow. As shown in Table 2, by replacing the staircase learning rate with a constant learning rate, the accuracy of $m$-AutoGrow and therefore “$\Delta$” improves; further replacing Network Morphism (ZeroInit or AdamInit) with a random initializer (UniInit or GauInit) results in a bigger gain. Overall, combining a constant learning rate with GauInit performs the best. Figure 3(b) visualizes the trajectory corresponding to row “2–4–3” in Table 2, which is much smoother compared to Figure 3(a). Thus, constant learning rate and GauInit are adopted in the remaining experiments, unless we explicitly specify them.

Our ablation study results for $p$-AutoGrow are summarized in Tables 3-5. Table 3 analyzes the impact of the growing period $K$. In general, $K$ is a hyper-parameter to trade off speed and accuracy: a smaller $K$ takes a longer learning time but discovers a deeper net, or vice versa. Our results validate the preference of a faster growth (i.e. a smaller $K$). On CIFAR10, the accuracy becomes plateaus at $K=3$; further reducing $K$ produces a deeper net while the accuracy gain is marginal. In the following robustness test for $p$-AutoGrow, we simply select $K=3$. Figure 3(c)(d) visualize the trajectories of $p$-AutoGrow with $K=50$ and 3. The 2D projection gives limited information to reveal the advantages of $p$-AutoGrow comparing to $m$-AutoGrow in Figure 3(b), although the trajectory of our final $p$-AutoGrow in Figure 3(d) is plausibly more similar to the one of training from scratch in Figure 2(b). Moreover, our quantitative results in Table 3 show that $p$-AutoGrow overcomes the very-early stop issue of $m$-AutoGrow in Table 2.

For sanity check, we perform the ablation study of initializers for $p$-AutoGrow. The results in Table 4 further validate our wisdom on selecting GauInit. The motivation of Network Morphism is to start a deeper net from a loss function that has been well optimized in a shallower net so as not to restart the deeper net from scratch [36, 35, 5, 8, 3, 4]. In all our experiments, we find this is sure even with random initialization. Figure 4 plots the convergence curves and learning process for “42–42–42” in Table 3. Even with GauInit, the loss and accuracy rapidly recover and no
At last, we perform the ablation study on the initial depth of the seed network. Table 5 demonstrates that a shallowest DNN works as well as a deeper seed. This implies that AutoGrow can appropriately stop regardless of the depth of the seed network. As the focus of this work is on depth automation, we prefer starting with the shallowest seed to avoid a manual search of a seed depth.

### 4.3. Adaptability of AutoGrow

To verify the adaptability of AutoGrow, we use an identical configuration (p-AutoGrow with $K = 3$) and test over 5 datasets and 4 seed architectures. Table 6 includes the results of all 20 combinations. Figure 5 compares AutoGrow with manual search which is obtained by training many baselines from scratch. The results lead to the following conclusions and contributions:

1. In Table 6, AutoGrow adapts layer depth across all scenarios without any tuning meanwhile achieving state-of-the-art accuracy. Manual design needs $m \cdot n \cdot k$ trials, where $m$ and $n$ are respectively the numbers of datasets and sub-module categories, and $k$ is the number of trials per dataset per sub-module category;

2. For ResNets, a discovered depth (‘●’ in Figure 5) falls at the location where accuracy saturates. This means AutoGrow discovers a near-optimal depth: a shallower depth will lose accuracy while a deeper one gains little. The final accuracy of AutoGrow is as good as training the discovered net from scratch as indicated by “Δ” in Table 6.

3. For plain networks, there are large positive “Δ”s in Table 6. It implies that baselines fail to train very deep...
Table 8: The adaptability of AutoGrow to dataset sizes

| Dataset Size | Found Net | Top-1 | Top-5 | Δ | Top-1 (AutoGrow) - Top-1 (Manual) |
|--------------|-----------|-------|-------|---|----------------------------------|
| Basic4ResNet on CIFAR10 | 6-6-6-6 | 92.79 | 91.94 | 0.93 | 62.53 |
| Bottleneck4ResNet | 6-6-6-17 | 93.91 | 93.65 | 0.26 | 66.36 |
| Plain4Net | 6-6-6-6 | 90.08 | 89.76 | 0.32 | 69.79 |

Table 9: Scaling up to ImageNet

| Dataset Size | Found Net | Top-1 | Top-5 | Δ | Top-1 (AutoGrow) - Top-1 (Manual) |
|--------------|-----------|-------|-------|---|----------------------------------|
| Basic4ResNet on CIFAR10 | 6-6-6-6 | 93.91 | 93.65 | 0.26 | 66.36 |
| Bottleneck4ResNet | 6-6-6-17 | 93.91 | 93.65 | 0.26 | 66.36 |
| Plain4Net | 6-6-6-6 | 90.08 | 89.76 | 0.32 | 69.79 |

Table 10: The efficiency of AutoGrow

| Dataset Size | GPUs | Growing Time | Fine-Tuning Time |
|--------------|------|--------------|-----------------|
| Basic4ResNet-12-12-11 | GTX 1080 Ti | 56.7 hours | 157.9 hours |
| Bottleneck4ResNet-6-6-6-17 | TITAN V | 45.3 hours | 114.0 hours |
| Plain4Net-6-6-6-6 | GTX 1080 Ti | 11.7 hours | 29.7 hours |

Figure 7: The comparison between AutoGrow and manual design [11] on ImageNet. The area of a marker is proportional to the size of the model. “basic” (“bottleneck”) refers to ResNets with basic (bottleneck) residual blocks.

Figure 8: The convergence curves and growing process on ImageNet for (a) Basic4ResNet-9-3-6-4 and (b) Plain4Net-6-6-6-6 in Table 10.

4. K can be used to trade off accuracy and model size. As shown in Figure 5, AutoGrow discovers smaller DNNs when increasing K from 3 (“•”) to 50 (“○”). Interestingly, the accuracy of plain networks even increases at K = 50. This implies the possibility of improving accuracy by tuning K although we stick to K = 3 for generalizability study. Table 7 shows the accuracy improvement of plain networks at larger K, which is close to the corresponding ResNets’ accuracy. Figure 6 visualizes loss surfaces around minima by AutoGrow and baseline. Intuitively, AutoGrow finds wider or deeper minima with less chaotic landscapes.

5. In Table 6, AutoGrow achieves different accuracies when using different sub-modules. The accuracy is limited by sub-module design, not by the AutoGrow framework.

Table 8 summarizes the adaptability of AutoGrow to the sizes of dataset. In each set of experiments, dataset is randomly down-sampled to 100%, 75%, 50% and 25%. For a fair comparison, K is divided by the percentage of dataset such that the number of mini-batches between growths remains the same. As expected, our experiments show that AutoGrow adapts to shallower networks when the sizes are smaller.

4.4. Scaling to ImageNet and Efficiency

In small datasets, we set J in Periodic Stop policy as the number of epochs used for training a baseline under the largest learning rate. In ImageNet, we shrink J to one third (i.e., J = 10) for earlier stop and faster evaluation. We explore AutoGrow with K = 2 and K = 5 for both plain networks and ResNets. The results are shown in Table 9. AutoGrow automatically finds a depth and the accuracy is higher than training the found net from scratch. The larger K = 5 enables AutoGrow to obtain a smaller DNN to trade-off accuracy and model size (computation) and the smaller K = 2 achieves higher accuracy. The comparison of AutoGrow and manual design [11] in Figure 7 shows that AutoGrow achieves better trade-off between accuracy and computation (measured by floating point operations).

Table 10 summarizes the breakdown of wall-clock time in AutoGrow. The growing/searching time is as efficient as (often more efficient than) fine-tuning the single discovered DNN. The scalability of AutoGrow comes from its intrinsic features that (1) it grows quickly with a short period K and stops immediately if no improvement is sensed; and (2) the network is small at the beginning of growing. Figure 8 plots the growing and converging curves for two DNNs in Table 10.
References

[1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. IEEE transactions on Neural Networks, 5(1):54–65, 1994. 2

[2] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In International Conference on Machine Learning, pages 549–558, 2018. 2

[3] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018. 2, 4, 5, 6

[4] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu. Path-level network transformation for efficient architecture search. arXiv preprint arXiv:1806.02639, 2018. 2, 4, 5, 6

[5] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641, 2015. 1, 2, 5, 6

[6] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. Adanet: Adaptive structural learning of artificial neural networks. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 874–883. JMLR. org, 2017. 2

[7] X. Dai, H. Yin, and N. K. Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. arXiv preprint arXiv:1711.02017, 2017. 2

[8] T. Elsken, J.-H. Metzen, and F. Hutter. Simple and efficient architecture search for cnns. In Workshop on Meta-Learning (MetaLearn2017) at NIPS, 2017. 2, 4, 5, 6

[9] J. Feng and T. Darrell. Learning the structure of deep convolutional networks. In Proceedings of the 6th international conference on computer vision, pages 2749–2757, 2015. 2

[10] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi. Morpnet: Fast & simple resource-constrained structure learning of deep networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1586–1595, 2018. 2

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016. 1, 3, 4, 8

[12] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In Proceedings of the IEEE International Conference on Computer Vision, pages 1389–1397, 2017. 2

[13] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2752–2761, 2018. 2

[14] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017. 1, 3

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014. 4

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012. 1

[17] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2554–2564, 2016. 2

[18] H. Li, A. Kadav, I. Durandalovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710, 2016. 2

[19] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In Advances in Neural Information Processing Systems, pages 6391–6401, 2018. 5

[20] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In Proceedings of the European Conference on Computer Vision (ECCV), pages 19–34, 2018. 2

[21] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436, 2017. 2

[22] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055, 2018. 2

[23] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In Proceedings of the IEEE International Conference on Computer Vision, pages 2736–2744, 2017. 2

[24] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE international conference on computer vision, pages 5058–5066, 2017. 2

[25] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrazad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In Artificial Intelligence in the Age of Neural Networks and Brain Computing, pages 293–312. Elsevier, 2019. 2

[26] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268, 2018. 2

[27] G. Philipp and J. G. Carbonell. Nonparametric neural networks. arXiv preprint arXiv:1712.05440, 2017. 2

[28] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 2902–2911. JMLR. org, 2017. 2

[29] S. Saxena and J. Verbeek. Convolutional neural fabrics. In Advances in Neural Information Processing Systems, pages 4053–4061, 2016. 2

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014. 1, 3
[31] L. N. Smith, E. M. Hand, and T. Doster. Gradual dropin of layers to train very deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4763–4771, 2016. 1

[32] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2):99–127, 2002. 2

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015. 1, 3

[34] G. Wang, X. Xie, J. Lai, and J. Zhuo. Deep growing learning. In Proceedings of the IEEE International Conference on Computer Vision, pages 2812–2820, 2017. 2

[35] T. Wei, C. Wang, and C. W. Chen. Modularized morphing of neural networks. arXiv preprint arXiv:1701.03281, 2017. 1, 2, 5, 6

[36] T. Wei, C. Wang, Y. Rui, and C. W. Chen. Network morphism. In International Conference on Machine Learning, pages 564–572, 2016. 1, 2, 5, 6

[37] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In Advances in neural information processing systems, pages 2074–2082, 2016. 2

[38] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. arXiv preprint arXiv:1708.01547, 2017. 2

[39] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016. 1, 2

[40] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8697–8710, 2018. 1, 2