MixtureGrowth: Growing Neural Networks by Recombining Learned Parameters

Chau Pham[1*] Piotr Teterwak[1*] Soren Nelson[2*†] Bryan A. Plummer[1]
Boston University[1] Physical Sciences Inc[2]
{chaupham,piotrt,bplum}@bu.edu snelson@psicorp.com

Abstract

Most deep neural networks are trained under fixed network architectures and require retraining when the architecture changes. If expanding the network's size is needed, it is necessary to retrain from scratch, which is expensive. To avoid this, one can grow from a small network by adding random weights over time to gradually achieve the target network size. However, this naive approach falls short in practice as it brings too much noise to the growing process. Prior work tackled this issue by leveraging the already learned weights and training data for generating new weights through conducting a computationally expensive analysis step. In this paper, we introduce MixtureGrowth, a new approach to growing networks that circumvents the initialization overhead in prior work. Before growing, each layer in our model is generated with a linear combination of parameter templates. Newly grown layer weights are generated by using a new linear combination of existing templates for a layer. On one hand, these templates are already trained for the task, providing a strong initialization. On the other, the new coefficients provide flexibility for the added layer weights to learn something new. We show that our approach boosts top-1 accuracy over the state-of-the-art by 2-2.5% on CIFAR-100 and ImageNet datasets, while achieving comparable performance with fewer FLOPs to a larger network trained from scratch. Code is available at https://github.com/chaudatascience/mixturegrowth.

1. Introduction

Many tasks explore the trade-off between predictive performance and computational complexity, such as neural architecture search (NAS) [3, 12, 23, 39], knowledge distillation [16, 18], and parameter pruning [21, 44]. These methods result in a well-optimized model during inference but are often expensive to train (*e.g.*, [39]). Thus, some researchers have explored an alternative procedure where one begins with a small network and increases its size over time (*e.g.*, [14, 46]). By first training a smaller and cheaper model, this approach can also be used to learn a target architecture with fewer floating point operations (FLOPs) than training the target network from the start. The key challenge is deciding how to initialize any new layer weights after a growth step that avoids forgetting what the network has already learned while still leaving room for improvement. As illustrated in Fig. 1a, prior work explored methods that initialize new parameters through a computationally expensive analysis step that selects parameters that decrease the training loss [46] or increase gradient flow [14].

Our paper seeks to find a solution for a scenario where we have a fully trained model and wish to expand it to a bigger one. To this end, we propose MixtureGrowth, a novel network growing framework that generates new weights by learning to reuse and combine parameters from the smaller network. Our approach is inspired by recent success with template mixing methods [2, 31, 34, 41, 48], where layer weights are generated using a linear combination of shared parameter templates. Each layer obtains custom weights by using a unique set of linear coefficients, thereby increasing the expressiveness of the shared parameters. The shared templates and the coefficients are trained jointly without altering the loss functions or network architecture for a target task. When compared with traditional neural networks these methods have reported improved computational and parameter efficiency [2, 31, 41], reduced training time [2, 31], and improved task performance [31, 34, 41, 48].

A high-level overview of our approach is provided in Fig. 1b. Specifically, we postulate that since template mixing effectively shares parameters across layers, it can also be used to initialize new layer weights after a growth step. We investigate three research questions that were not explored in prior work. First, template mixing provides a new mechanism for generating new layer weights after growing. For example, let us consider increasing the width of a layer g times, resulting in most layers in the expanded network

*Equal contribution
†Work done while at Boston University

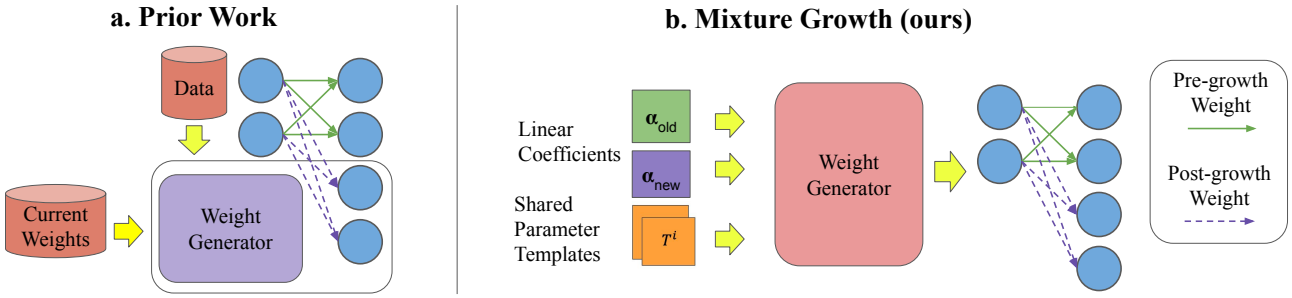


Figure 1. **A comparison of different strategies in growing neural networks. (a)** In prior work (*e.g.*, [14,46]), network growing methods use loss-based local measures to initialize new weights at growth time. These methods depend on performing analysis using the training data and the current weights of existing layers, requiring significant computational overhead at the growth step; **(b)** In contrast, we investigate a template-mixing based method to grow the network. Given a small trained network, we start off with training another equally-sized network. Then, parameters of these small networks are shared across layers by template mixing schemes, where additional layer weights are generated using new linear coefficients for existing templates.

having g^2 times the weights of the original network. Since the weights in the small network are generated using a linear combination of shared templates, we can generate the weights of the large network by initializing $g^2 - 1$ new sets of linear coefficients and concatenating the generated weights for the large layer (illustrated in Fig. 2). We find that careful initialization of these new linear coefficients is still necessary for the best performance, with some strategies, such as replicating the existing coefficients, having a detrimental impact on training.

The second research question we investigate is whether we should grow from a single small model, or if fusing two small models is better. We argue that training an extra small model provides more diversity in terms of learned features for the growth process. In a setting where we grow from a model that is half the width of the large one, then a small network would only take around 25% of the FLOPs of the large network making it relatively inexpensive.

The final research question we explore, following up on the previous question, is to find the optimal time to stop training the second network and start training a fully grown one. Given a fixed FLOPs budget, growing early gives more time to optimize all the layer parameters in the large network. However, this also limits the ability of the second network, thereby raising the question of when to grow. We find that our approach is relatively insensitive to the exact training stage at which a growth step occurs, and could be used to grow late with minimal impact on task performance.

In summary, our contributions are:

- We develop MixtureGrowth, a novel approach for growing a neural network over time by concatenating together weights of the old (small) network to fresh weights generated from newly initialized linear coefficients used to combine existing shared parameter templates.
- Our experiments show that MixtureGrowth is an effective growth method by outperforming the top-1 accuracy of prior work by 2-2.5% on CIFAR-100 and ImageNet, demonstrating the effectiveness of our approach.
- We analyze our approach for growing neural networks along axes such as sensitivity to growth point, the characteristics of learned features, and the effect of fusing two small models as opposed to growing from a single model.

2. Related Work

Growing Neural Networks has long been an active area of research; In early work, Ash *et al*. [1] added neurons in single-layer systems, which later was extended by Fahlman *et al*. [15] to the deep network regime. It became common to pre-train neural networks greedily stacking Restricted Boltzmann Machines [4, 20] to construct Deep Belief Networks [19]. This was simplified by Vincent *et al*. [43] with Stacked Denoising Autoencoders. More recently, Wen *et al*. [45] proposed depth growing policies to automatically grow networks until performance stops improving. Maile *et al*. [28] proposed initializing new weights to maximize orthogonality. However, these works aim to automate network size selection by progressive growth, as opposed to minimizing the cost of training for a given network size.

In more relevant work, Wu *et al*. [47] and Chen *et al*. [7] replicated weights of the network to progressively increase its width, while Net2Net [7] leveraged function-preserving transformations to transfer knowledge when growing. These approaches replicate weights such that the pre-growth function is preserved, and then perturb or add noise to break symmetry to grow. Firefly [46] also learned an overgrown network and then pruned uninformative weights. GradMax [14] and NeST [9] initialized weights to maximize gradient norms. Although these methods are more efficient than retraining from scratch, many of them still require a significant computational cost to analyze how the network should initialize the new network.

Template Mixing improves upon hard parameter sharing, which directly reuses parameters across layers [8, 50], by

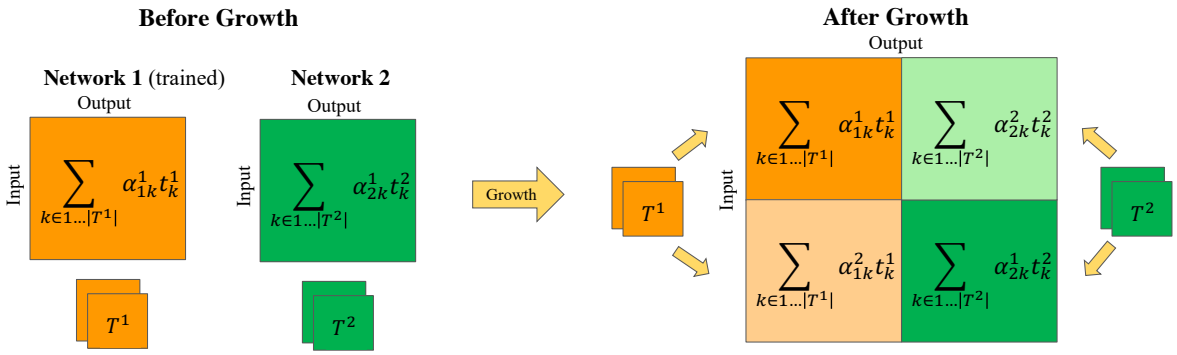


Figure 2. **Illustration of the growth process in MixtureGrowth models where $g = 2$ (*i.e.*, double the size):** We explore how to grow models that have weights constructed from linear combinations of templates, which we call Template Mixing Models (see Sec. 3.1). Before growth, assume that one already has a fully-trained model (denoted as *network 1*, left) and wishes to expand it. To this end, we first train another equally-sized model using a different set of templates (*network 2*) for some number of epochs. Then, we merge the weights of the two networks so that their weights are tiled in the corners along the diagonal. In this example, we form a model 4x the size and resume training on the fully grown model (**target model**, rightmost). Quadrants that share templates but have different linear combination coefficients are presented as different shades of the same color. The squares correspond to layer weights whose input and output correspond to the input and output dimensionality. Note that all templates and alpha coefficients are learnable parameters.

representing layer weights as a linear mixture of shared templates (*e.g.*, [2, 31, 34, 41, 48]). Savarese *et al.* [34] found they could use template mixing to boost performance by sharing across multiple layers within a single network. Isomorphic networks [33, 51] can push cross-layer parameter sharing further by constraining the network to have layers of identical shape and size. Plummer *et al.* [31] removed the need to have many identical layers by upsampling/downsampling templates when shared between layers of different shapes. Interestingly, these methods exhibit similarities to the concept of modular learning, where network layers are shared across multiple tasks, as explored in prior research [29, 30]. While modular learning relies on shared layer weights tailored to individual tasks, template mixing, in contrast, combines shared templates to dynamically generate layer weights. In this paper, we expand on the study of template mixing by asking new research questions not addressed in prior work. Specifically, prior work in template mixing assumes the network being trained retains the same size over time. However, in this work, we leverage the unique structure of template mixing to generate new weights when expanding the network during training.

Shrinking Neural Networks alters the size of a model during training typically aimed to increase its computational efficiency at test time. There are two broadly used methods for this: distilling and pruning neural networks. In neural network distillation, the goal is to transfer knowledge from a larger network to a smaller network. The technique, introduced in Hinton *et al.* [18], replaced one-hot labels with the predictions of a larger network (*i.e.* teacher network). In this way, knowledge from the teacher network is transferred to the smaller network called student network. Examples of subsequent work include methods that distill ensembles into a single model [36] and highlight the importance of the student and teacher seeing identical augmentations over long training periods [6]. In parameter pruning, the goal is to find and remove unimportant weights (*e.g.*, [27, 38, 42]), resulting in a more computationally efficient network at a cost of increases training time as it assumes one has trained the fully parameterized network until convergence. Contrary to these tasks, our goal is to reduce the training time of the large network by growing a neural network over time.

3. Growing via shared parameter initialization

Given a pre-trained small neural network architecture F^S with layers $\ell^S_{1,\dots,N}$, our goal is to generate the weights of a larger neural network architecture F^L with layers $\ell^L_{1,\dots,M}$. Following [14, 46], we evaluate settings where networks have the same number of layers (*i.e.*, $M = N$), even though our approach could be used to grow in depth as well [31, 34]. Instead, we vary in the width of each layer, *i.e.*, $|\ell^L_i| > |\ell^S_i|$, where the operator $|\cdot|$ returns the number of weights in a layer. In our experiments, set the size of the F^L to be twice that of F^S, although other settings are possible with minor changes to layer weight construction (*e.g.*, as in [31, 41]). The task then is to generate weights for F^L that enable it to converge quickly using what has been learned in F^S. Methods are ranked on both the computational efficiency of obtaining the fully trained model F^L as well as the quality of the final solution measured as task performance.

Prior work for growing neural networks relies on performing a computationally expensive per-layer analysis of the gradients or training loss to initialize any new param-

eters in F^L (*e.g.*, [14, 46]). We avoid this issue by learning how to generate new weights using a weight generator based on shared parameters. Note that we refer to *weights* as the matrix of numbers used by a layer operation, such as the filters used by convolutional layers, and *parameters* as the matrices being optimized using backpropagation. In traditional neural networks, weights and parameters typically refer to the same set of matrices, but in this paper, we generate layer weights using trainable parameters. The parameters are linear combinations of templates which are the results of a process we refer to as ***template mixing*** [31, 34, 41] which we briefly review in Sec. 3.1. Next, we discuss our strategy for learning more expressive parameter templates using model fusion (Sec. 3.2). Finally, we introduce a process of initializing any new weights needed by F^L by tiling weights generated using the templates initially learned from our small network F^S (Sec. 3.3).

3.1. Generating weights with template mixing

Template mixing networks (*e.g.*, [2, 31, 34, 41, 48]) generate layer weights by combining parameter templates and have shown they provide superior parameter efficiency and performance than traditional neural networks. We use the setting where each parameter template $T^{1,\dots,t}$ is the same size as its corresponding layer in the small neural network F^S (*i.e.*, $|T_i^k| = |\ell_i^S|$), although parameter templates of different sizes could be also used [41]. Thus, the weights of the i^{th} layer, donated as ℓ_i^S, are generated via,

$$\ell_i^S = \sum_{k \in 1,\dots,t} \alpha_i^k T_i^k, \tag{1}$$

where α_i^k is a trainable parameter that controls the contribution of template T_i^k in generating weights for layer ℓ_i^S.

We begin by pretraining our small network F^S using template mixing. Following [34], we share templates between any pair of layers of the same size (with a maximum grouping of two layers in a row that share templates) for the new weights. For example, if T_i and T_{i+1} are used to generate weights for identical layers ℓ_i^S and ℓ_{i+1}^S, then they refer to the same set of shared parameters, and they would not share templates with layer ℓ_{i+2}^S, even if was the same size. The total number of trainable parameters over a standard neural network only increases by the number of coefficients α_i^k used to generate the weights (resulting in a hundred or so extra parameters in our experiments during training). Both the coefficients α_i^k and the templates T^i are learned jointly via backpropagation from the task loss, with no additional loss terms over a standard neural network used during training. Note that this weight generation process does introduce some overhead during training, but this increase is negligible, accounting for just 0.03% of the FLOPs in a forward pass for a batch size of 64 samples [31].

3.2. Learning robust templates for growing

A primary difference between our MixtureGrowth and prior work in template mixing (*e.g.*, [2, 31, 34, 41, 48]) is that the architecture in our first training stage is not the final network architecture. Instead, we use a standard neural network procedure for learning a small network and then grow to a larger network, similar to prior work on network growth (*e.g.*, [14, 46]). Prior work in network growth would initialize new trainable parameters using a heuristic built from the small network. This could result in unfavorable initialization as the expressiveness of the large network is much more significant (due to having many more weights) than the small network. The methods of prior work in network growth could be seen as focused on trying to recover (some) of that additional expressiveness missing in the small network. In contrast, our approach assumes that many of the templates we have learned in F^S will generalize to the new weights needed after growing to the target network F^L.

We explore two variants of our method: one we grow directly from a single trained model and reuse the templates for new weights after growth, and another where we learn an extra model using an independent set of templates before growth. We call this second approach ***model fusion***, illustrated in Fig. 2. Our method is motivated by the idea that independently trained models learn substantially independent features [11, 26]. Thus, fusing two models creates a more robust feature representation, which in turn boosts performance. From a pair of small trained networks F^{S_1}, F^{S_2}, we use them to initialize and train our target network F^L, which we discuss in the next section.

3.3. Generating weights for the target network

After obtaining the pair of pre-trained small networks F^{S_1}, F^{S_2} using the approach in Sec. 3.2, the goal is to use these networks to construct the weights of the large network F^L. From F^S, we want to grow it g times larger to obtain F^L. Thus, most layers in the large network require g^2 times the weights. The only exception is the first and last layers, which are only g times the size as the dimensions of the input and outputs have not changed. Fig. 2 illustrates a tiling for the weights of each layer where $g = 2$.

Since our small networks have been trained using template mixing, we can generate new weights by using a new set of coefficients (α in Eq. 1). Therefore, if we need to generate g^2 tiles of weights for a layer, then we would learn g^2 tile-specific coefficients, but the templates would be shared with other tiles generated from the same small subnetwork. Of these g^2 tile-specific coefficients, we can generate $g^2 - 2$ new tiles by reusing our existing small subnetworks F^{S_1}, F^{S_2}, but we must initialize the new coefficients. We investigate multiple strategies for initializing these coefficients that are described below.

Random coefficients simply initializes new coefficients randomly from a standard normal distribution. By jointly optimizing all coefficients and shared templates, the neural network finds new low-loss regions after growth. However, it remains uncertain whether a random initialization would substantially increase diversity for network growth.

Coefficient copying is motivated by the idea that the entire subspace of weights spanned by a linear combination of templates does not correspond to low loss. Finding new low-loss combinations without performing a thorough analysis is challenging, so instead we copy the coefficients of the subnetworks F^{S_1}, F^{S_2} as the initial starting point for the new weights. These copied coefficients are not shared, so although they begin as the same values, they are changed independently while training F^L. This can be seen as providing a starting initialization that increases the response of each subnetwork g times in every layer after growing.

Orthogonal coefficients uses orthogonal coefficients when sharing templates to encourage diversity in the generating weights, introduced by Savarese *et al*. [34]. In other words, when each set of α coefficients can be considered a vector, all such α vectors that are associated with the same templates are initialized to be orthogonal following [35].

4. Experiments

Experimental Setup. We assume a trained network is available to grow. While our method starts with a small template mixing model, the baselines employ small networks of their own settings. For a fair comparison, all small networks have the same FLOPs (25% of the target network).

Metrics. We rank methods based on their top-1 classification accuracy averaged over three runs on all datasets. In addition, we report the number of floating point operations (FLOPs) required for training a model. This metric compares computational costs for different models regardless of their hardware. Thus, it is used widely to evaluate the efficiency of a network [13, 32, 40]. For our experiments, we normalize computational complexity across methods by specifying a model train in the same FLOPs to ensure a fair comparison. We use fvcore library[1] to compute FLOPs.

Baselines. We employ the following baseline methods.

- **Random** initializes any new weights randomly during growth. This refers to generating the weights directly, rather than randomly initializing the coefficients used in template mixing networks (described in Sec. 3.3).
- **Net2Net [7]** is based on function preserving initialization to initialize the new weights in a way such that the prediction is unchanged. This ensures the newly grown model can have the same ability as the small one preventing the drop in performance at growth point.
- **Firefly [46]** grows a network using pruning. Specifically, they overgrow the network, optimize it for a few steps, and then prune the network by using a Taylor Approximation of the loss function to estimate the importance score of new neurons. Note that our approach avoids this overhead by learning to generate new weights by reusing our shared parameters, making the growing step significantly more computationally efficient.
- **GradMax [14]** proposes an approach that seeks to maximize the gradients of new neurons. This is motivated by the observation that initialization with large gradients results in quick progress on the learning problem. GradMax maximizes the gradients of the loss with respect to new parameters. It solves this approximately with SVD at growth time, making the growth step computationally costly when compared with our approach. We use the author's code in our experiments[2], which we also use as our implementation of the Random and Firefly methods.

4.1. Datasets and implementation details

See the supplementary for details not described below.

CIFAR-10 and CIFAR-100 [25] contain 60K images of 10 and 100 categories, respectively. Each dataset is split into 50K images for training and 10K images for testing. We use the WideResnet architecture [49] to grow from a WRN-28-5 to WRN-28-10. We train the model for 200 epochs on a single GPU using stochastic gradient descent. Following prior work in network growing [14], Batch Normalization [22] is only used before each block for a fair comparison.

ImageNet [10] contains 1K categories with 1.2M images for training, 50K for validation, and 100K for testing. We train ResNet-50 models [17] for 90 epochs with a learning rate 0.1, decayed by 0.1 at 30, 60, and 80 epochs. We use stochastic gradient descent with a momentum 0.9, batch size 256 with cross entropy loss. Before growth, each layer has half the channels of the final network.

4.2. Results

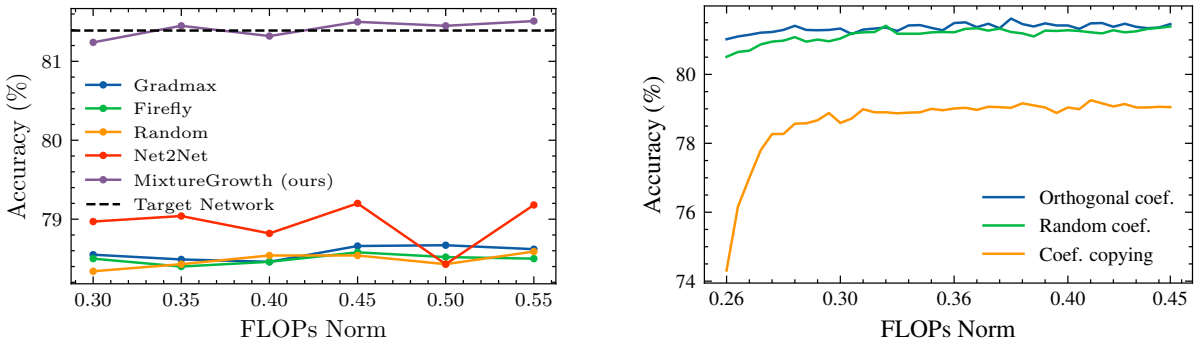
Tab. 1 reports the performance of growing a neural network on CIFAR-10, CIFAR-100, and ImageNet. We see that MixtureGrowth is able to get better performance compared with training the target architecture from scratch (*Target network*) with half of the FLOPs on CIFAR-100 dataset. As for ImageNet, our approach gets comparable performance with Target network using just one-third of the FLOPs. When considering the Target at the same FLOPs, it outperforms *Large Match* (*i.e*. the *Target Network* but trained with the same FLOPs as network growing baselines) by a significant margin. Thus, our approach is capable of utilizing the existing small models to obtain a high-performing model in less time than a standard network.

[1] https://github.com/facebookresearch/fvcore

[2] https://github.com/google-research/growneuron

	Method	CIFAR-10 [25]	CIFAR-100 [25]	FLOPs Norm	ImageNet [10]	FLOPs Norm
(a)	Large (Target network)	96.88%	81.39%	1.00X	76.49%	1.00X
	Large Match	96.62%	80.95%	0.45X	73.85%	0.35X
	Small	96.53%	80.29%	0.25X	72.48%	0.25X
	Small Match	96.60%	80.30%	0.45X	72.89%	0.35X
(b)	Random	95.61%	79.08%	0.45X	71.47%	0.35X
	Net2Net [7]	95.47%	79.20%	0.45X	72.29%	0.35X
	Firefly [46]	95.62%	78.90%	0.45X	71.30%	0.35X
	GradMax [14]	95.73%	79.05%	0.45X	71.73%	0.35X
	MixtureGrowth w/o fusion	96.05%	79.44%	0.45X	73.02%	0.35X
	MixtureGrowth (ours)	**96.66%**	**81.50%**	0.45X	**74.51%**	0.35X

Table 1. Network growing comparison using top-1 accuracy averaged over three runs. **(a)** Performance of training the target (post-growth) (*Large*) network architecture from scratch (WRN-28-10 for both CIFAR datasets and ResNet-50 for ImageNet), as well as the performance of the small (pre-growth) (*Small*) architecture (WRN-28-5 for both CIFAR datasets and a ResNet-50 with half the channels as the large network for ImageNet); **(b)** Performance of network growing methods, where our approach outperforms the current SOTA.



(a) **Growing methods by FLOPs**

(b) **Coefficient initializations**

Figure 3. Top-1 Accuracy on CIFAR-100. **(a)** Network growing methods under varying FLOPs. The dashed line represents the Target network's performance trained from scratch using a full training schedule. We find MixtureGrowth not only outperforms Random, FireFly [46], and GradMax [14], but also outperforms the target network with half the FLOPs; **(b)** Performance of different coefficient initialization methods (Sec. 3.3) from growth point. See Sec. 4.3 for discussion.

Prior work on network growth (Tab. 1b) struggles to boost performance after growing when compared to simply training a regular small network (Tab. 1a), denoted as *Small* and *Small Match* (*i.e.* the *Small* network but trained with the same FLOPs as network growing baselines). Note that compared to prior work, we explore larger network architectures in our experiments (*e.g.*, a ResNet-50 on ImageNet in our work vs. a VGG11 [37] evaluated in [14]), suggesting these prior methods find generalizing to larger networks challenging. In contrast, our method can allow the model to gain improvement over growth and even surpass the target model on CIFAR-100 with half of the FLOPs. When compared with other growing methods, our MixtureGrowth approach gets around 1% higher performance on CIFAR-10, 2.5% on CIFAR-100, and 2% on ImageNet.

Fig. 3a demonstrates the efficiency and effectiveness of our method compared with four baselines: Random [5], Net2Net [7], Firefly [46], and Gradmax [14] across various FLOPs budgets. Using WRN-28-10 as the target network, our method consistently outperforms the baselines with the same budget while being on par with or even outperforming the target network with half of the FLOPs.

Tab. 3 compares growing in a single step using MixtureGrowth to growing many times with the long growth schedule used by prior work [14]. In this setting, not only do we obtain a significant boost in performance compared with the current state-of-the-art, but do so in half the FLOPs. The additional FLOPs required by prior work are due to their growth overhead (discussed at the beginning of Sec. 4).

4.3. Discussion

How should we initialize linear coefficients? The first research question we explore is how to effectively initialize the coefficients used to generate new layer weights af-

	Second Network Training Time (FLOPs)							
Dataset	0.0	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CIFAR-100	79.24%	80.16%	80.46%	81.08%	80.9%	81.09%	**81.44%**	81.27
ImageNet	73.02%	73.74%	73.75%	74.09%	74.16%	**74.51%**	74.33%	74.31

Table 2. **Ablation on growth points** where we stop training the second network to train the full model. The first column is the extreme case where we grow without training the second network (*i.e*. MixtureGrowth w/out fusion). The rest shows MixtureGrowth's performance on varying growth points. *Second Network Training Time* indicates the proportion of full-training FLOPs that was used to train the second network, *e.g*., 0.5 means we train the second network half of its full train before growth. We find that late growth where we almost fully train the second network before growing gains the best performance (boldface). All runs report top-1 accuracy at the same 0.35X FLOPs.

ter growing. In Sec. 3.3 we describe three different methods for initializing these new coefficients. We report their performance in Tab. 4, where we find orthogonal initialization, which helps promote the most diversity in the generated weights, outperformed both random initialization and coefficient copying. Notably, random initialization also performs well and approaches the performance of orthogonal initialization. Note, for coefficient copying's results, we also tested adding some small random noise to break the symmetry, but it did not substantially affect performance.

To obtain insights into how coefficient initialization methods perform at the growth step, we plot top-1 accuracy starting from the growth step until the end in Fig. 3b. We find that both orthogonal and random can maintain task performance despite being provided with new weights at the moment of growth and start improving performance with the large network. However, coefficient copying struggles at the growth step as its performance drops significantly. We suspect this may be due to the redundancies caused by reusing and replicating the same coefficients, making it difficult to learn new informative weights.

Is growing by fusing two small models more effective than growing from a single model? The second question we explored was whether we could take advantage of having two small networks during growth. Having a pair of networks would improve robustness as discussed in Sec. 3.2. Tab. 1 compares growing with and without model fusion (last 2 rows), where we find that fusing two models gains a consistent boost after normalizing by FLOPs.

What is the right point during training to grow? The previous question sheds some light on the importance of having a second trained model. However, how long we should train the second network is remain open. If we train the second one for a long schedule, then there is not much budget left to optimize the parameters of the large network. Conversely, if we grow it too early, then the second network is not sufficiently trained, which in turn may lead to a drop in overall performance. From this perspective, we can consider MixtureGrowth w/out fusion as an extreme case of early growth where we do not train the second network but instead grow to a full network directly. Tab. 2 shows the performance of different growth points under the same FLOPs budget on CIFAR-100 and ImageNet datasets. We observe that late growth where the second network was almost fully trained gives the best performance. This confirms our assumption that growing with two networks is more robust.

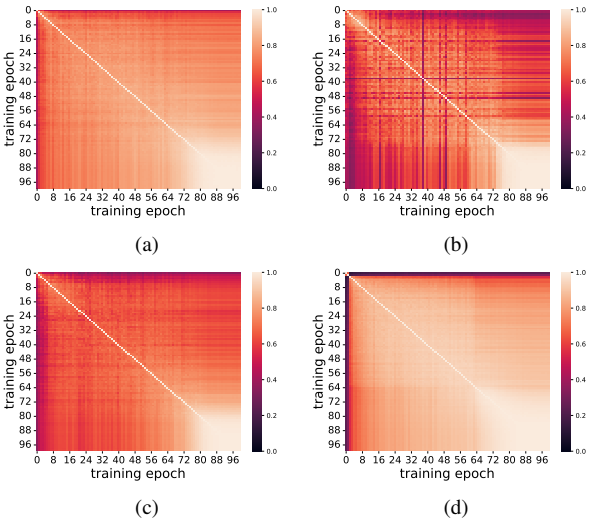


Figure 4. **CKA of different models at last convolutional layer over time. (a)** *Target Network:* A smooth pattern that becomes lighter along the diagonal toward the end of training when training from scratch; **(b)** *Random:* More dark and noisy pattern compared with the target when growing with random weights; **(c)** *MixtureGrowth w/out fusion:* Without model fusion, the pattern is somewhat similar to the Target, but the representations between epochs have smaller similarities; **(d)** *MixtureGrowth:* In our full model, we see a clear phase change at the growth point (epoch 63), suggesting rapid learning that occurs after fusing the 2 small models.

4.4. Feature analysis

We use Centered Kernel Alignment (CKA) [24] to analyze the similarity of features over training. For this analysis, we set up MixtureGrowth by training 2 small networks for 63 epochs, then training the fully grown network afterward until reaching 100 epochs in total. For Mixture-

	Method	Top-1 Acc.	FLOPs Norm
(a)	Random	79.12%	0.77X
	Firefly [46]	78.94%	0.77X
	GradMax [14]	79.24%	0.77X
(b)	**MixtureGrowth**	**81.50%**	0.45X

Table 3. **Network growing comparison on CIFAR-100**. **(a)** Using the long growing schedule used by the authors of GradMax; **(b)** MixtureGrowth with single growth for reference.

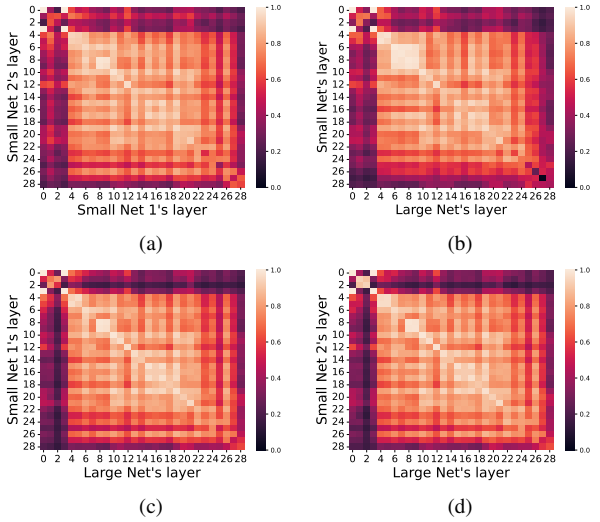


Figure 5. **CKA between layers of MixtureGrowth. (a)** MixtureGrowth: Small Network 1 vs. Small Network 2; **(b)** MixtureGrowth w/o model fusion: Large Network vs. Small Network 1; **(c)** MixtureGrowth with model fusion: Large Network vs. Small Network 1; **(d)** MixtureGrowth with model fusion: Large Network vs. Small Network 2; We analyze the similarity of representations for each pair of layers of a WRN-28-10 on CIFAR-100 dataset. See Sec. 4.4 for discussion.

Growth w/out fusion, only one small model was trained for 63 epochs, and then we grew. The target network refers to a full network trained from scratch for 100 epochs.

Fig. 4 shows how features evolve during training over time using CKA procedure described above. In Fig. 4d, which visualizes feature similarity, a very clear phase change in similarity is visible at the growth step at epoch 63. This indicates that learning at the growth point is very rapid, suggesting we effectively fuse two independent models in the growth step. Random weight initialization (Fig. 4b) reflects a similar change seen in Fig. 4d, but with more noisy patterns. In contrast, training the target network from scratch (Fig. 4a), and MixtureGrowth w/out fusion (Fig. 4c) have a more gradual transition from the beginning of training towards the end without clear quadrants like MixtureGrowth. Thus, learning during growth is not rapid, but gradual instead. This provides insight into the effectiveness of our approach, especially with low FLOP budgets.

Method	Top-1 Accuracy
Random coefficients	81.41%
Coefficient copying	79.05%
Orthogonal coefficients	**81.50%**

Table 4. **Ablation on coefficient initialization methods** (described in Sec. 3.3) for MixtureGrowth on CIFAR-100. Orthogonal initializations perform best, suggesting the importance of having independent instantiated weights.

Fig. 5 illustrates how networks have changed from the point of growth to the final representation with CKA. In Fig. 5a, we compare the two small networks right before growth. We observe that several layers across networks have similar features (light color on the diagonal, which represents highly similar features from the same layer), and some early and late layers differ (dark color). We find a similar pattern growing from a single model (*i.e.*, MixtureGrowth w/out fusion, Fig. 5b). This suggests that after growth (w/o fusion), the large network learns additional features that differ from the small net in a similar way as two independently trained small nets. In contrast, in MixtureGrowth w/fusion, from Fig. 5c and 5d, we see the full net shares more similarity with each of the small nets in the early and late layers than MixtureGrowth w/out fusion. Thus, little learning happens after fusing two small nets, and lightweight training after growth is sufficient.

5. Conclusion

In this paper, we propose MixtureGrowth, a new method to grow networks with weights constructed from linear combinations of shared templates that employs model fusion to boost diversity in our pretrained templates. We analyze our approach by answering several questions about the impact of linear coefficients initialization and finding a good growth point. We find that without model fusion, MixtureGrowth already achieves comparable or higher accuracy than the state-of-the-art. However, with fusion the benefits over prior work increases, resulting in a 2-2.5% improvement over the state-of-the-art on CIFAR-100 and ImageNet. We believe MixtureGrowth to represent an interesting step forward in learning to grow networks by leveraging template mixing to generate new weights.

Acknowledgements This material is based upon work supported, in part, by DARPA under agreement number HR00112020054 and the NSF under award DBI-2134696. Any opinions, findings, and conclusions or recommendations are those of the author(s) and do not necessarily reflect the views of the supporting agencies.

References

[1] Timur Ash. Dynamic node creation in backpropagation networks. *Connection science*, 1(4):365–375, 1989. 2

[2] Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Lcnn: Lookup-based convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1, 3, 4

[3] Pouya Bashivan, Mark Tensen, and James J DiCarlo. Teacher guided architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5320–5329, 2019. 1

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006. 2

[5] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 6

[6] Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10925–10934, 2022. 3

[7] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015. 2, 5, 6

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008. 2

[9] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019. 2

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5, 6

[11] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000. 4

[12] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3681–3690, 2019. 1

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 5

[14] Utku Evci, Max Vladymyrov, Thomas Unterthiner, Bart van Merriënboer, and Fabian Pedregosa. Gradmax: Growing neural networks using gradient information. *arXiv preprint arXiv:2201.05125*, 2022. 1, 2, 3, 4, 5, 6, 8

[15] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Advances in neural information processing systems*, 2, 1989. 2

[16] J Gou, B Yu, SJ Maybank, and D Tao. Knowledge distillation: A survey. corr. *arXiv preprint arXiv:2006.05525*, 2020. 1

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5

[18] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1, 3

[19] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009. 2

[20] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. 2

[21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021. 1

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 5

[23] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2387–2397, 2020. 1

[24] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019. 7

[25] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5, 6

[26] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017. 4

[27] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989. 3

[28] Kaitlin Maile, Emmanuel Rachelson, Hervé Luga, and Dennis George Wilson. When, where, and how to add new neurons to anns. In *International Conference on Automated Machine Learning*, pages 18–1. PMLR, 2022. 2

[29] Elliot Meyerson and Risto Miikkulainen. Beyond shared hierarchies: Deep multitask learning through soft layer ordering. In *International Conference on Learning Representations*, 2018. 3

[30] Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular deep learning. *arXiv preprint arXiv:2302.11529*, 2023. 3

[31] Bryan A. Plummer, Nikoli Dryden, Julius Frost, Torsten Hoefler, and Kate Saenko. Neural parameter allocation search. In *International Conference on Learning Representations (ICLR)*, 2022. 1, 3, 4

[32] Adria Ruiz and Jakob Verbeek. Anytime inference with distilled hierarchical neural ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. 5

[33] Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 3

[34] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *International Conference on Learning Representations*, 2019. 1, 3, 4, 5

[35] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013. 5

[36] Zhiqiang Shen and Marios Savvides. Meal v2: Boosting vanilla resnet-50 to 80imagenet without tricks. *arXiv preprint arXiv:2009.08453*, 2020. 3

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[38] Nikko Ström. Sparse connection and pruning in large dynamic artificial neural networks. In *Fifth European Conference on Speech Communication and Technology*, 1997. 3

[39] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1

[40] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 5

[41] Piotr Teterwak, Soren Nelson, Nikoli Dryden, Dina Bashkirova, Kate Saenko, and Bryan A. Plummer. Learning to compose superweights for neural parameter allocation search. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2024. 1, 3, 4

[42] Georg Thimm and Emile Fiesler. Evaluating pruning methods. In *Proceedings of the International Symposium on Artificial neural networks*, pages 20–25, 1995. 3

[43] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010. 2

[44] Huan Wang, Can Qin, Yue Bai, Yulun Zhang, and Yun Fu. Recent advances on neural network pruning at initialization. *arXiv e-prints*, pages arXiv–2103, 2021. 1

[45] Wei Wen, Feng Yan, Yiran Chen, and Hai Li. Autogrow: Automatic layer growing in deep convolutional networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 833–841, 2020. 2

[46] Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. *Advances in Neural Information Processing Systems*, 33:22373–22383, 2020. 1, 2, 3, 4, 5, 6, 8

[47] Lemeng Wu, Dilin Wang, and Qiang Liu. Splitting steepest descent for growing neural architectures. *Advances in neural information processing systems*, 32, 2019. 2

[48] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In *Advances in Neural Information Processing Systems*, 2019. 1, 3, 4

[49] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016. 5

[50] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014. 2

[51] Andrey Zhmoginov, Dina Bashkirova, and Mark Sandler. Compositional models: Multi-task learning and knowledge transfer with modular networks. *arXiv preprint arXiv:2107.10963*, 2021. 3