Learning to Compose SuperWeights for Neural Parameter Allocation Search

Piotr Teterwak* — Boston University — piotrt@bu.edu
Soren Nelson* — Physical Sciences Inc — snelson@psicorp.com
Nikoli Dryden — ETH Zürich — nikoli.dryden@inf.ethz.ch
Dina Bashkirova — Boston University — dbash@bu.edu
Kate Saenko — Boston University — saenko@bu.edu
Bryan A. Plummer — Boston University — bplum@bu.edu

Abstract

Neural parameter allocation search (NPAS) automates parameter sharing by obtaining weights for a network given an arbitrary, fixed parameter budget. Prior work has two major drawbacks we aim to address. First, there is a disconnect in the sharing pattern between the search and training steps, where weights are warped for layers of different sizes during the search to measure similarity, but not during training, resulting in reduced performance. To address this, we generate layer weights by learning to compose sets of SuperWeights, which represent a group of trainable parameters. These SuperWeights are created to be large enough so they can be used to represent any layer in the network, but small enough that they are computationally efficient. The second drawback we address is the method of measuring similarity between shared parameters. Whereas prior work compared the weights themselves, we argue this does not take into account the amount of conflict between the shared weights. Instead, we use gradient information to identify layers with shared weights that wish to diverge from each other. We demonstrate that our SuperWeight Networks consistently boost performance over the state-of-the-art on the ImageNet and CIFAR datasets in the NPAS setting. We further show that our approach can generate parameters for many network architectures using the same set of weights. This enables us to support tasks like efficient ensembling and anytime prediction, outperforming fully-parameterized ensembles with 17% fewer parameters[1].

1. Introduction

Parameter sharing is used to increase the computational efficiency and/or accuracy of neural networks (*e.g.*, [2, 11, 17, 20, 23, 26, 29]). Instead of using hand-crafted heuristics to obtain a good parameter sharing strategy, Plummer *et al.* [23] introduced a task they called Neural Parameter Allocation Search (NPAS), which automatically determines where and how to share parameters in any neural network. A key challenge in this task is learning when sharing can occur between layers that may perform different operations (*e.g.*, convolutional and fully connected) and have different sizes. Plummer *et al.* addressed this issue by using bilinear interpolation to warp the shared parameters to the size of each layer in the network (illustrated in Figure 1(a)). However, this was computationally costly as each layer would be need to be re-warped on every forward pass. In addition, this approach also made training the network more chal-

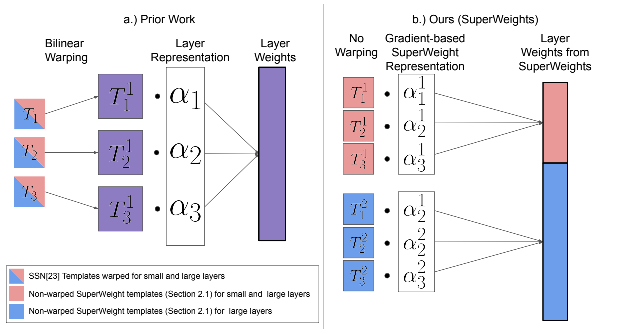


Figure 1. **Comparison to prior work.** **(a)** Prior work in Neural Parameter Allocation Search [23] would search for a good parameter sharing strategy by comparing the linear weights learned by each layer over a set of shared templates that warped to the size required by each layer using bilinear interpolation only when searching for a good parameter sharing strategy. In contrast, **(b)** illustrates our SuperWeight Networks, which creates SuperWeights that are concatenated together to form a layer's weights. Each SuperWeight is only shared by layers of a minimize size, helping us to avoid warping the layers in the search step and ensuring there is no disconnect between search and training stages.

*Equal Contribution, work done while Soren was at Boston University
[1]Code available: https://github.com/piotr-teterwak/SuperWeights

lenging as the effective receptive field for a set of parameters would vary as its size was changed.

To address these issues, we present SuperWeight Networks, a method for NPAS that effectively automates parameter sharing between diverse architectures. A key contribution of our work is introducing the concept of a SuperWeight, which are groups of parameters that can be thought of as a feature detector that captures a unique pattern (*e.g.*, an edge detector). Instead of warping our parameters to fit layers of varying size as in prior work [23], we create SuperWeights that are small enough to be shared between layers. Then, as shown in Figure 1(b), we compose them together to create a layer's weights. Thus, our search task transforms into locating instances where two layers find the same SuperWeight useful.

To find effective sharing strategies between our SuperWeights, we introduce a new mechanism for measuring similarity between parameters. Specifically, Plummer *et al.* [23] measured similarity using the value of some of the shared parameters themselves. However, two layers could potentially have a conflict, where they both wish to alter the parameters in opposing directions, but effectively cancel their gradients out. Such conflicts have been observed in parameter sharing settings in prior work (*e.g.*, [38]), and we found the likelihood that such conflicts increases as the number of layers increase. In other words, these networks may contain similar parameters, but the gradients from these layers would be very dissimilar. Thus, we propose a gradient analysis approach for determining where SuperWeights can be reused.

To further improve our model's parameter efficiency, we also take advantage of template mixing methods from prior work (*e.g.*, [2, 23, 26]). We construct SuperWeights using a weighted linear combination of templates made up of trainable parameters which we call Weight Templates. This creates a hierarchical representation for neural network weight generation, where we begin by combining Weight Templates to create SuperWeights, then concatenating together SuperWeights to create the layer weights. This template mixing can boost performance when reusing the same parameters many times, as each combination of templates can use a unique set of coefficients. We can share parameters within a layer of a single network (Section 3) or between different members of an ensemble (Section 4 and 4.2). This allows our method to be used for efficient ensembling and anytime prediction, in addition to the NPAS task.

Our SuperWeight formulation has some similarities to Slimmable Networks [35, 37], which trained multiple networks of varying widths to support a range of inference times using a hand-crafted sharing strategy. In contrast, our task is to automatically search for a good sharing strategy, including within a single network. In addition, one goal of NPAS methods like ours is to support training networks with fewer computational resources, whereas Slimmable Networks use additional resources to train each subnetwork. Several other tasks also used parameter sharing to make more efficient networks, including those applied to neural architecture search (*e.g.*, [15, 34, 36, 40, 43]), mixture-of-experts models [24, 27, 30, 42], or modular and self-assembling networks [1, 6, 7]. However, these tasks are orthogonal to each other as they focus on optimizing the network architecture, whereas we aim to optimize the sharing strategy between layers. Thus, they often can be combined.

In summary, our contributions are as follows:

- We propose SuperWeight Networks, a new technique for automated parameter sharing across layers that composes reusable SuperWeights across layers in a network to support diverse architectures.
- We introduce a new approach to computing parameter similarity that uses gradient information rather than the values of the parameters themselves, enabling us to find more effective sharing strategies that consistently improve performance over the state-of-the-art.
- We demonstrate that our approach is also effective at searching for sharing strategies over ensembles of diverse models. This enables us to surpass the performance fully-parameterized ensembles on CIFAR while still using 17% fewer parameters as well as obtain state-of-the-art performance on anytime inference.

2. Learning to Share with SuperWeight Networks

Given a parameter budget B and layers $\ell_{1,\ldots,S}$, the goal of Neural Parameter Allocation Search (NPAS) is to generate the weights of the layers of each member that maximize task performance. Plummer *et al.* [23] only explored settings where all the layers ℓ_i belonged to the same network, but in this paper these layers can be separated into M target architectures, each of which can be generated and make independent predictions. To address this task, we propose SuperWeight Networks, an automated approach for sharing parameters with a hierarchical weight construction construction process. At the lowest level, we perform a linear combination of Weight Templates that contain trainable parameters to generate our SuperWeights (discussed in Section 2.1.1). Then we concatenate together SuperWeights to create weights for a layer (described in Section 2.1.2). An overview of this process is illustrated in Figure 2.

To find a good parameter sharing strategy, we use a two step search-and-refine process. This begins by identifying what layers may share trainable parameters effectively, which we group together into SuperWeight Clusters (Section 2.2.1). Then we determine what layers can share SuperWeights effectively (Section 2.2.2). In both these stages

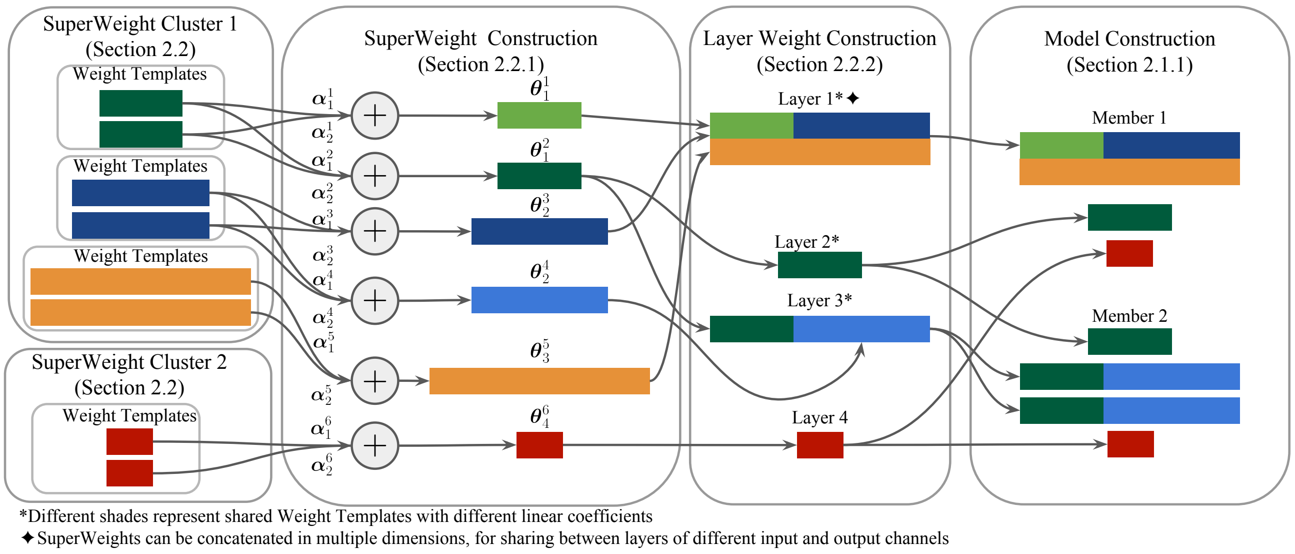

SuperWeight Cluster 1 (Section 2.2)
Weight Templates
Weight Templates
Weight Templates
SuperWeight Cluster 2 (Section 2.2)
Weight Templates
SuperWeight Construction (Section 2.2.1)
Layer Weight Construction (Section 2.2.2)
Layer 1*✦
Layer 2*
Layer 3*
Layer 4
Model Construction (Section 2.1.1)
Member 1
Member 2
*Different shades represent shared Weight Templates with different linear coefficients
✦SuperWeights can be concatenated in multiple dimensions, for sharing between layers of different input and output channels

Figure 2. **SuperWeight Networks overview. Left:** Weight Templates are clustered into SuperWeight Clusters; layers can only share SuperWeights from Weight Templates in the same cluster (Section 2.2) **Center Left:** SuperWeights are constructed from linear combinations of Weight Templates (see Eq. (1)) using the SuperWeight Clusters learned in Section 2.2.1. **Center Right:** The SuperWeights assigned to a layer are then concatenated to generate layer weights. We automatically identify when we should use the same linear coefficients or different coefficients for generating a layer's SuperWeight using the procedure in Section 2.2.2. **Right:** These layer weights are then assembled one or more ensemble members, which can have diverse architectures.

we will leverage gradient information to measure how similar the parameters are (and, therefore, how much they would benefit from parameter sharing).

2.1. Generating a Neural Network using SuperWeights

In this section we shall describe how to generate weights for a layer in a neural network, which will provide context for our search-and-refine procedure for creating our SuperWeight Clusters in Section 2.2. Traditional methods of (hard) parameter sharing directly reuse weights (*e.g.*, [4, 8, 12, 41]), but this limits layer weight diversity as shared layers then encode the same learned function. Instead, we take inspiration from recent work in cross-layer parameter sharing (*e.g.*, [2, 23, 26]) that performs template mixing. In these works, parameters and layer weights are decoupled; each layer's weights are a linear combination of parameter matrices called templates.

2.1.1 Network Generation Process

Consider our generation process outlined in Figure 2, which we shall step through from right-to-left. In the rightmost column describing model construction, we see seven layers across two different networks. The first layer of "Member 1" is generated by concatenating together three SuperWeights (labeled as "Layer Weight Construction" in the third column). These SuperWeights are generated by a linear combination of Weight Templates ("SuperWeight Construction"), whereas the Weight Templates may share parameters between any SuperWeights within the same SuperWeight Cluster.

Now going from left-to-right through the generation process: our first step is to create a set of Weight Templates $T^g_{1,\ldots,N}$ with the same dimensions as the kth SuperWeight $\theta^{(k)}_g$. Following [23], these Weight Templates are created by splitting the trainable parameters into templates and allocating them to SuperWeights in a round robin fashion. Then we generate our SuperWeight $\theta^{(k)}_g$ via a linear combination using coefficients $\alpha^{(k)}$, *i.e.*,

$$\theta^{(k)}_g = \sum_{i=1}^{N} \alpha^{(k)}_i T^g_i \tag{1}$$

Multiple SuperWeights can share g while having different coefficients $\alpha^{(k)}$ (represented as different color shades for a SuperWeight, *e.g.*, green, in the second column of Figure 2). To create a layer's weights (third column of Figure 2), one or more SuperWeights are generated using Eq. (1) and then concatenated together (additional details in Section 2.1.2). These layers are stacked together to create a neural network (last column of Figure 2). *Note that templates and coefficients are learned jointly via gradient descent.*

Plummer *et al.* [23] also shared parameters via template mixing between layers of different sizes, but they would directly reshape the available parameters into templates of the target layer size. This means that it would be very difficult to accurately represent subnetworks within a larger network. For example, let us assume we are generating the weights for two networks that are identical except that network X is twice as wide as network Y. The approach from Plummer *et al*. would either have to generate layers for X and Y independently, or we would generate the layers of X, and then slice out the weights that would fit network Y. We found that the first option was difficult to optimize that resulted in lower performance in our experiments. The second option would reduce diversity, as network X would contain exactly the same weights as network Y. Instead, our SuperWeight Networks can directly optimize any subnetworks, like in the second option in our example above, but while minimizing any diversity loss (details Section 2.2).

2.1.2 Determining SuperWeight Sizes

The target sizes for our SuperWeights are based on layer shapes. Consider the case where we are generating weights for a set of layers $\ell_{1,...,S}$, where each layer is a different size. We create an initial set of SuperWeights by sorting the layers in order from smallest to largest. The first SuperWeight we would create would be the same size as the smallest layer ℓ_i, and every layer within that SuperWeight Cluster that was the same size as ℓ_i would initially share the same SuperWeight. Then, when we generate the weights for the next layer ℓ_{i+1}, we assume we have the SuperWeight from layer ℓ_i. Thus, the SuperWeights for layer ℓ_{i+1} need only generate the additional weights required beyond those provided by the SuperWeights from ℓ_i.

For example, if ℓ_i needed 100K parameters and ℓ_{i+1} needed 400K, then the new SuperWeights would generate $400 - 100 = 300$K weights. If ℓ_{i+1} is larger than ℓ_i in only a single dimension, then only a single new SuperWeight is generated. If it's larger in both input and output channels, ℓ_{i+1} generates two new SuperWeights. The first SuperWeight extends the size in a input channels dimension, and the second is concatenated in the output channel dimension. See Layer 1 in the third column of Figure 2 for a graphical illustration. Any subsequent layers would follow the same process, but would have all the weights from the previous layers (*i.e*., ℓ_{i+2} would have 400K weights from ℓ_{i+1}, 100K of which originally came from ℓ_i).

2.2. Finding Where to Share via Search-and-Refine with Gradient Similarity

There are two components of Figure 2 that we did not discuss in the network generation details found in Section 2.1. First, our SuperWeight Cluster creation process, which separates SuperWeights into groups that share trainable parameters (shown in the first column of the figure). In this setting we find what layers can effectively share trainable parameters (described in Section 2.2.1). This can be thought of as trying to learn a coarse policy where we identify layers that find any sharing detrimental.

After Plummer *et al*. [23] found a good sharing strategy, they would train their model using soft-sharing, where every layer has their own coefficients $\alpha^{(k)}$ when performing a combination of templates (*e.g*., similar to using Eq. 1). However, our approach differs in that we consider the case that some SuperWeights may find hard-sharing across layers beneficial (*e.g*., the dark green SuperWeight being reused in Layers 2 and 3 in Figure 2). Thus, after we begin training our model, we first perform hard-sharing across all layers in the the same SuperWeight Cluster (*i.e*., the same set of coefficients $\alpha^{(k)}$ are used for every SuperWeight). After a few epochs, we refine our sharing policy so that layers that find hard-sharing challenging can begin using soft-sharing instead, *i.e*., they would begin using their own coefficients, but still share Weight Templates, as illustrated with the two SuperWeights of varying blue shades in Figure 2 (described in Section 2.2.2).

2.2.1 Searching for a Coarse Parameter Sharing Policy

Our first step in finding a good sharing strategy is to determine what layers can effectively share parameters to create Superweight Clusters. This provides our coarse sharing policy that we shall refine in Section 2.2.2. There are two key differences with our search step from prior work [23]. First, our SuperWeight construction allows us to avoid warping the shared templates as discussed in the Introduction, meaning that how we will perform the search step will be using the same mechanisms as when we train our model. Second, rather than using the coefficients learned by each layer $\alpha^{(k)}$ to measure similarity between parameters, we measure similarly by comparing the gradients a SuperWeight is receiving from the layers that share it. The intuition behind this approach is that layers with conflicting gradients are playing a gradient tug-of-war, hurting optimization. In other words, if the sum of gradients from two layers is close to zero, no learning occurs.

More formally, given a set of layers $\ell_{1,...,S}$, our goal is to determine which layers can share parameters. At this stage, we treat all layers are belonging to the same SuperWeight Cluster and are using hard-sharing of SuperWeights across layers. This means that every set of Weight Templates has exactly one set of coefficients. For example, if a cluster has three layers that all need a SuperWeight of 100K dimensions, then each layer would use a shared Weight Template that produces the same 100K dimension SuperWeight (additional details found in Section 2.1.2).

Thus, we can infer that if the gradients of the loss w.r.t. the layers sharing SuperWeights are misaligned, then the model will have optimization difficulties. Thus, we compute gradient similarity over SuperWeights to determine what layers may be grouped. However, some layers may be composed of multiple SuperWeights, only some of which may share with other layers in the initial cluster. For example, Layer 2 in the third column of Figure 2 has one SuperWeight, but Layer 3 has two SuperWeights (only one of which is shared with Layer 2). $\upsilon_{i,j}$ refers to the set of SuperWeights shared by layers i and j. W_i refers to the instantiated weights of layer i. Then the gradient similarity between W_i and W_j would be computed as:

$$\psi_{SW} = \cos\left(\frac{\partial \mathcal{L}}{\partial W_i}\frac{\partial W_i}{\partial \upsilon_{i,j}}, \frac{\partial \mathcal{L}}{\partial W_j}\frac{\partial W_j}{\partial \upsilon_{i,j}}\right) > \tau, \tag{2}$$

where τ is a hyperparameter representing the minimum threshold for which two layers that share $\upsilon_{i,j}$ should remain in the same cluster, and $\mathcal{L}$ is the loss.

After training for a few epochs, we create our SuperWeight Clusters by placing layers sharing the same SuperWeight into a priority queue by the cosine similarity with respect to the gradients of the shared coefficients aggregated over an epoch. We pop the first two layers, ℓ_i, ℓ_j off the queue and check whether their cosine similarity exceeds the threshold τ, *i.e.*, it satisfies Eq. 2. If they do not satisfy Eq. 2, we split any ungrouped layers into their own individual groups (*e.g.*, 4 ungrouped layers would result in 4 additional single layer groups). Otherwise, if they do Eq. 2 satisfy and both layers belong to an existing group, we merge their groups. If only one layer is in an existing group, then we add the new layer into that group's set. The final set of layer groups is referred to as our SuperWeight Cluster. We set τ via grid search on a validation set (we found $\tau = 0.1$ worked well in our experiments). Please refer to the supplementary for pseudocode for our priority-queue assignment procedure. Following [23], we reinitialize our model and re-train from scratch with our new SuperWeight Clusters, where we refine our sharing policy as described in the next section.

2.2.2 Learning when SuperWeights should share Weight Template coefficients

After we obtain a set of SuperWeights Clusters from Section 2.2.1, we start training the network by only using hard-parameter sharing between SuperWeights. In other words, each layer that shares a SuperWeight will begin by using the same set of coefficients $\alpha^{(k)}$ used to combine weight templates for the first E epochs of training (we found $E = 10$ worked well). After that, we analyze which layers may benefit from refining the sharing policy by decoupling linear coefficients combining Weight Templates. In this way, unlike Plummer *et al.* [23] which only used soft-sharing, we allow a combination of hard and soft-sharing.

To search for a refined sharing policy, we use the same priority-queue assignment procedure as described in Section 2.2.1. In other words, we place the layers sharing the same SuperWeight into a priority queue. Then, we iterate over pairs of layers ℓ_i, ℓ_j and check whether their cosine similarity exceeds some threshold β, *i.e.*,

$$\psi_{coef.} = \cos\left(\frac{\partial \mathcal{L}}{\alpha^{k,j}}, \frac{\partial \mathcal{L}}{\alpha^{k,i}}\right) > \beta, \tag{3}$$

where $\alpha^{k,i}$ is the shared coefficient corresponding to layer ℓ_i. Layers that satisfy Eq. 3 will be grouped together. Any layers that do not satisfy Eq. 3 with any other layer will remain in their own group. After we have identified the groups of similar layers, each obtains their own copy of the coefficients for that SuperWeight $\alpha^{(k)}$ and we resume training. This creates a new SuperWeight for each group of layers whose gradients point in a similar direction, allowing for layer specialization.

3. Single Network Search Results

We first compare our SuperWeight Networks (SWN) to prior work on the Neural Parameter Allocation Search (NPAS) task in Section 3, whereas in Section 4 we will explore a new setting where we apply our NPAS approach to tasks that can be implemented using an ensemble of models that we generate from a set of shared parameters.

Datasets and metrics. We evaluate our method on three standard benchmarks: CIFAR-10 [19], CIFAR-100 [19], and ImageNet [5]. We evaluate the performance of a model based on its top-1 accuracy given a parameter budget. Additional details on our experimental setup and hyperparameters can be found in the supplementary material.

3.1. Results

Table 1 compares different strategies for creating a parameter sharing strategy for image classification. Table 1(a) reports the performance of prior work reproduced using the author's code. Comparing the last lines of Table 1(a) representing the state-of-the-art NPAS approach [23] and Table 1(b) that reports our full method, we see we obtain a consistent boost over prior work. Notably, we receive around a 1% boost on the ImageNet dataset. Overall, we find that when a sharing strategy is most needed, *i.e.*, in the lower budget settings, we observe a larger boost to performance (increasing to an almost 2% gain on ImageNet).

When comparing the effect of the different components of our SuperWeight Networks in Table 1(b), find that each contributes meaningfully to the final model performance. The gradient similarity function provides the most benefit on the ImageNet dataset, where we see a 0.5% gain to top-1 accuracy. In addition, also compare to using only perform the search step from Section 2.2.1, but skip the refine-

		CIFAR-10 [19]		CIFAR-100 [19]		ImageNet [5]	
	Param Budget %	1%	10%	1%	10%	5%	10%
(a)	Baseline [39]	93.54±0.19	95.73±0.17	71.49±0.29	77.43±0.24	66.89±0.22	68.12±0.18
	Single Cluster	93.39±0.17	95.56±0.10	71.30±0.26	77.19±0.29	66.27±0.36	67.55±0.28
	Random Cluster	93.48±0.31	95.62±0.11	70.97±0.50	76.82±0.40	66.81±0.41	67.95±0.43
	SSN [23]	94.74±0.16	95.83±0.10	74.66±0.29	78.17±0.27	67.69±0.19	70.39±0.22
(b)	SWN w/o Grad Sim	94.80±0.11	95.84±0.12	74.99±0.24	78.57±0.23	68.24±0.19	70.79±0.13
	SWN w/o Refine	94.81±0.13	95.95±0.14	75.49±0.20	78.26±0.21	68.11±0.23	70.69±0.16
	SWN (**Ours**)	**94.87**±0.14	**95.99**±0.11	**75.77**±0.34	**78.94**±0.26	**68.42**±0.21	**71.14**±0.18

Table 1. Comparing methods that searching for parameter sharing strategies using a WRN 28-10 [39] for CIFAR and WRN 50-2 for ImageNet averaged over three runs. Baseline reduces the width/number of layers to support a given parameter budget, which is reported as percentage of the original model's parameters. **(a)** reports the performance of our baseline methods from prior work reproduced using the author's code. **(b)** contains ablations of our SWN, where we report a consistent boost over the state-of-the-art. See Section 3 for discussion.

ment step in Section 2.2.2, (similar to the search strategy of SSNs [23]). Another way of thinking of this comparison is measuring the effect of hard-sharing SuperWeights across layer (w/o Refine) vs. using a mix of hard and soft sharing in our full model. Comparing the second and third lines of Table 1(b), we see that the refinement step is key to good performance in some settings, *e.g.*, we see more than a 0.5% gain on CIFAR-100 with a 10% budget. In the next section, we will explore applications of our work where we are tasked with finding a good parameter sharing strategy across an ensemble of models and architectures.

4. Multi-Network Search Experiments

In this paper we explore a new application of NPAS methods, where rather than searching for a good parameter sharing strategy over a single network, they must search over multiple architectures. We find that this work spans two different application areas: efficient ensembling (*e.g.*, [21,31,32]) and anytime inference (*e.g.*, [12,25,31,34,37]). In efficient ensembling the goal is to reduce the computational resources required to support a ensemble of models. These often make strong architectural assumptions, such as ensemble member homogeneity (*i.e.*, each member is the same architecture), which limits their use. For example, homogeneous ensembles are ill-suited to tasks like anytime prediction because one only has n options for computational complexity, where n is the number of ensemble members. In contrast, heterogeneous ensembles can select a subset of its ensemble members to provide a range of inference times (*e.g.*, a 4 member heterogeneous ensemble can adjust to $\binom{4}{1}+\binom{4}{2}+\binom{4}{3}+\binom{4}{4} = 15$ levels of inference latency). We demonstrate the flexibility and generalization power of our approach by addressing both tasks. Implementation details can be found in the supplementary.

Additional datasets and metrics. , We evaluate the robustness of our method on out-of-domain samples in addition to CIFAR-10 and CIFAR-100. Specifically, we report performance on CIFAR-100-C [14], which is the CIFAR-100 test set corrupted by distortions such as Gaussian blur and JPEG compression. We also supplement top-1 accuracy with calibration metrics [10]: Negative Log-Likelihood (NLL) and Expected Calibration Error (ECE).

4.1. Efficient Ensembling Results

Table 2(a) compares our SuperWeight Networks (SWN-HO) on the CIFAR-100 CIFAR-100-C, and CIFAR-10 with prior work in efficient ensembling. Note that all the methods boost performance over a single model without requiring additional model parameters. However, our SuperWeight Networks outperforms all other methods on CIFAR-100 when using 36.5M parameters. Table 2(a) also shows that heterogeneous SuperWeight Networks Ensembles (SWN-HE), consisting of a WRN-34-8, 28-12, 28-10, and 28-8, outperforms SWN-HO in over half of the metrics while also supporting many inference times.

Unlike methods like BatchEnsemble (BE) [31] and MIMO [12], which cannot change the number of parameters without making architecture adjustments to the widths and/or number of layers, our SuperWeight Networks can support any parameter budget without requiring architecture changes by adjusting the number of templates or the amount of sharing between layers. Thus, if the number of parameters are not a concern, our approach can increase our parameter budget to boost performance. This is illustrated in Table 2(b), where we outperform standard Deep Ensembles, which trains independent networks for ensemble members, while still retaining 17% fewer parameters.

Computational resources comparison. In addition to the number of parameters that we report in Table 2, the training time and inference time is also a key contributor to an efficient ensembling approach. Thus, our results for SWN-HO in Table 2(a) reduces inference time by using a smaller

			CIFAR-100 (clean)			CIFAR-100-C			CIFAR-10		
	Method	Params	Top-1 ↑	NLL ↓	ECE ↓	Top-1 ↑	NLL ↓	ECE ↓	Top-1 ↑	NLL ↓	ECE ↓
(a)	WRN-28-10	36.5M	79.8	0.875	8.6	51.4	2.70	23.9	96.0	0.159	2.3
	BE [31]	36.5M	81.5	0.740	5.6	**54.1**	2.49	19.1	96.2	0.143	2.1
	BE [31] + EnsBN	36.5M	81.9	n/a	2.8	**54.1**	n/a	19.1	96.2	n/a	1.8
	MIMO [12]	36.5M	82.0	0.690	2.2	53.7	2.28	12.9	96.4	0.123	1.0
	Thin Deep Ensembles	36.5M	81.5	0.694	**1.7**	53.7	2.19	11.1	96.3	**0.115**	**0.8**
	SWN-HO (**Ours**)	36.5M	82.2	0.702	2.7	52.9	**2.17**	10.3	96.3	0.120	**0.8**
	SWN-HE (**Ours**)	36.5M	**82.4**	**0.663**	3.0	53.0	**2.17**	**10.0**	**96.5**	**0.115**	**0.8**
(b)	Deep Ensembles	146M	82.7	**0.666**	**2.1**	54.1	2.27	13.8	**96.6**	**0.114**	1.0
	SWN-HO (**Ours**)	120M	**82.9**	**0.666**	2.2	**54.7**	**2.00**	**10.3**	**96.6**	0.119	**0.8**

Table 2. **Homogeneous ensembling comparison** on CIFAR-100 (clean) [19] CIFAR-100-C (corrupt) [14], and CIFAR-10 using WideResNets [39] averaged over three runs. **(a)** shows that our approach outperforms prior work in efficient ensembling [12,31]. **(b)** compares the performance of increasing the number of parameters (without changing the architecture) using our approach compared to Deep Ensembles, which trains 4 independent networks as members.

network (WRN-28-5, the same size used by Thin Deep Ensembles) so it has a similar inference time compared with work like MIMO [12], which uses a WRN-28-10. For a fair comparison to MIMO and BE, we also normalized our experiments by training time (using their learning schedules).

4.2. Anytime Inference Experiments

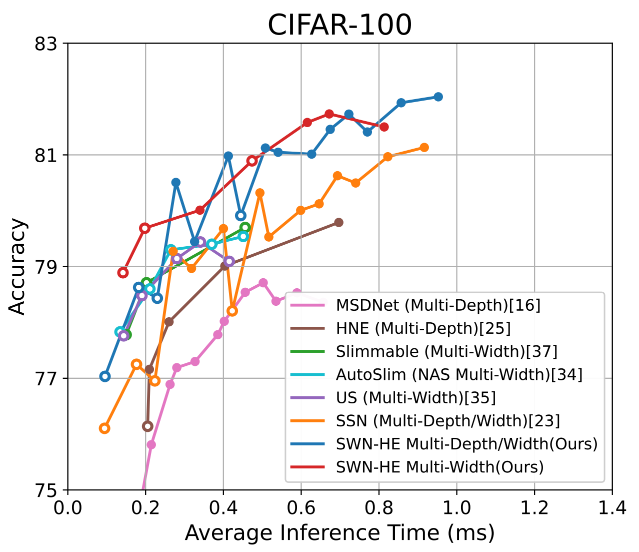
In anytime inference the goal is to make high-performing predictions within a given time budget. As the time budget increases, a good method will use the additional time to improve performance. A homogeneous ensemble, like those used in the experiments in Section 4.1, would only provide limited time budgets as each member has an identical computational complexity. Thus, their flexibility is limited because one has only M options for computational complexity, where M is the number of ensemble members. However, for a heterogeneous ensemble this limitation is removed with an ensemble since each member provides a different inference time resulting in $\binom{M}{1} + \binom{M}{2} + ... + \binom{M}{M}$ possible inference times to choose from. This results in a highly effective anytime inference model. We note that although we evaluate our individual ensemble members in series, our method is trivial to parallelize to increase inference speeds by using multiple GPUs. This is unlike many early-exit anytime inference methods (*e.g.*, [3, 16, 18, 22, 28, 33]), which are intrinsically serial.

We use two settings for our Heterogeneous SuperWeight Networks (SWN-HE) in our experiments: SWN-HE -Multi-Width, which trains a three member WRN [39] ensemble WRN-28-[7,4,3], and SWN-HE-Multi-Depth/Width, a four member ensemble WRN-28-[7,4] 16-[7,4]. SSNs [23] and Slimmable [37] use the same ensemble configurations. Other approaches use method-specific strategies (*e.g.*, HNE [25] trains a set of tree-nested ensembles). See supplementary for additional details, including experiments where we share parameters across different architecture families.

4.2.1 Anytime Inference Results

Figure 3 reports top-1 accuracy vs. average inference time using a single P100 GPU on CIFAR-10 and CIFAR-100 [19]. When comparing to other dynamic width methods such as Slimmable [37], Universally Slimmable [35] and AutoSlim [34] models, SuperWeight Networks perform on par or better than them for inference times they support, but our approach can provide a wider range of inference times that improve performance. We reiterate that other efficient ensembling methods such as BatchEnsemble [31] and MIMO [12] are not suitable for Anytime Inference, because each ensemble member has the exact same inference time. This results in a very limited set of possible inference times (see Figure 1). We also significantly outperform the tree-based ensemble HNE [25] on both datasets, as well as the early-exit model MSDNet [16]. Lastly, we use SuperWeight Networks to construct a dynamic width and depth network. Our main comparison is to adaptation of Shapeshifter Networks [23] to ensemble dynamic widths and depths. We show a consistent improvement over Shapeshifter Networks across inference times. These results demonstrate that SuperWeight Networks can share parameters across members of diverse architectures more effectively than other approaches.

Comparison of SuperWeight Clustering methods. Table 3 demonstrates the effectiveness of our SuperWeight Clustering approach described in Section 2.2. We provide four baselines: *Shared Coefficients*, which learns SuperWeight Clusters, but shares coefficients between all layers (*i.e.*, removing Section 2.2); *Single SuperWeight Cluster*, which allows layers to have their own coefficients, but does not learn clusters (*i.e.*, removing Section 2.2.1); *Depth-binning*, a heuristic where we group together layers of the same relative depth across network architectures; and *Coefficient Clustering* from prior work [23], which clusters coefficients



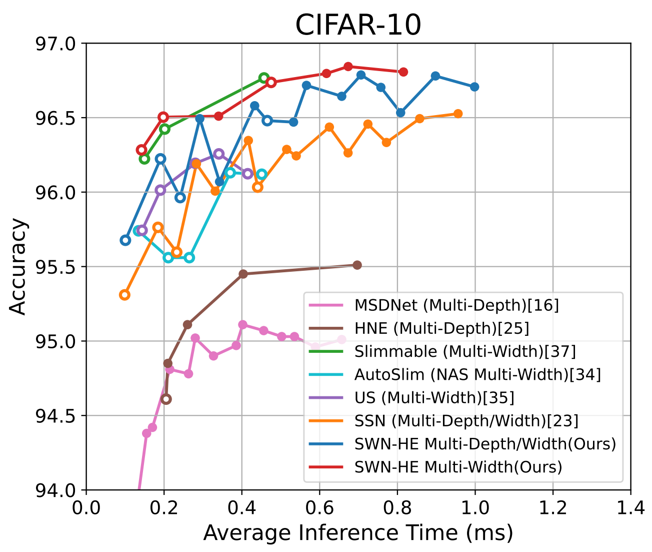


Figure 3. **Anytime Inference comparison** using CIFAR-100 and CIFAR-10 [19] top-1 accuracy vs. inference time (ms) averaged over 3 runs on a single P100 GPU. Most methods [16, 23, 34, 35, 37] use WRN backbones [39], except HNE [25] which modifies ResNet-50 [13].

Method	WRN-28-3	WRN-28-4	WRN-28-7	Full Ensemble
Shared Coefficients	77.3 ± 0.09	78.0 ± 0.15	79.7 ± 0.10	80.4 ± 0.06
Single SuperWeight Cluster	76.3 ± 0.37	77.6 ± 0.30	78.8 ± 0.20	81.1 ± 0.17
Depth-binning	76.8 ± 0.15	77.7 ± 0.17	79.2 ± 0.19	81.4 ± 0.15
Coefficient Clustering [23]	76.1 ± 0.19	77.0 ± 0.15	78.9 ± 0.15	80.7 ± 0.14
SWN-HE (**Ours**)	$\mathbf{78.9 \pm 0.26}$	$\mathbf{79.7 \pm 0.08}$	$\mathbf{80.9 \pm 0.03}$	$\mathbf{81.5 \pm 0.13}$

Table 3. **Multi-width SuperWeight Cluster creation** using top-1 accuracy on CIFAR-100 over three runs. See Section 4.2 for discussion.

α in Eq. (1) to group layers. We show that our approach outperforms these baselines. Notably, we find that Coefficient Clustering performs in par or worse than other baselines. In contrast, our gradient analysis approach (Section 2.2) takes into account the direction of change rather than just the current coefficient value. Thus, we obtain a 2% gain on individual models and a small boost to ensembling performance with our approach (Table 3). We show a visualization of SuperWeight cluster assignment in the supplementary.

5. Conclusion

We introduce SuperWeight Networks, a method for learning parameter sharing patterns in single models as well as model ensembles. Our automatic sharing improves single model performance by up to 4% compared to the baselines (Section 3). SuperWeight Networks also match performance of efficient ensembles in the low-parameter regime, compared to prior work (Section 4.1). When we add parameters, we outperform even deep ensembles on CIFAR with 17% fewer parameters (Section 4.1). Finally, SuperWeight Networks enables effective anytime inference (Section 4.2).

We believe that SuperWeight Networks are a promising step forward in parameter-efficiency. Future work will include more deeply exploring architecture diversity; [9] show that model architecture heterogeneity can be key to ensemble diversity on challenging tasks.

Broader Impacts and Limitations. Effective parameter sharing allows one to use less compute, potentially running networks in efficient modes and conserving energy. However, it can also be used to maximize the use of compute if it's available, using *more* energy with corresponding drawback. We urge readers to be aware of the carbon and energy footprint of the models they train.

Although learning the sharing pattern (SuperWeight Clusters and coefficient sharing) is relatively lightweight, it does add computation to the learning process. Nevertheless, the improved predictive performance makes this a reasonable trade-off.

Acknowledgements This material is based upon work supported, in part, by DARPA under agreement number HR00112020054. 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] Ferran Alet, Erica Weng, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Neural relational inference with fast modular meta-learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 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, 2, 3

[3] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 527–536. PMLR, 06–11 Aug 2017. 7

[4] 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. 3

[5] 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

[6] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *International Conference on Robotics and Automation*, 2017. 2

[7] Leslie Pack Kaelbling Ferran Alet, Tomas Lozano-Perez. Modular meta-learning. In *Conference on Robot Learning (CoRL)*, 2018. 2

[8] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016. 3

[9] Raphael Gontijo-Lopes, Yann Dauphin, and Ekin D Cubuk. No one representation to rule them all: Overlapping features of training methods. *arXiv preprint arXiv:2110.12899*, 2021. 8

[10] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017. 6

[11] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 1

[12] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *International Conference on Learning Representations*, 2021. 3, 6, 7

[13] 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*, pages 770–778, 2016. 8

[14] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019. 6, 7

[15] Shoukang Hu, Ruochen Wang, Lanqing Hong, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot nas with gradient matching. *arXiv preprint arXiv:2203.15207*, 2022. 2

[16] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018. 7, 8

[17] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning (ICML)*, 2021. 1

[18] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR, 2019. 7

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *University of Toronto*, 2009. 5, 6, 7, 8

[20] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations (ICLR)*, 2020. 1

[21] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015. 6

[22] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1891–1900, 2019. 7

[23] 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, 2, 3, 4, 5, 6, 7, 8

[24] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In *Advances in Neural Information Processing Systems*, volume 34, 2021. 2

[25] Adria Ruiz and Jakob Verbeek. Anytime inference with distilled hierarchical neural ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. 6, 7, 8

[26] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *International Conference on Learning Representations*, 2019. 1, 2, 3

[27] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. 2

[28] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from

deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016. 7

[29] Matthew Wallingford, Hao Li, Alessandro Achille, Avinash Ravichandran, Charless Fowlkes, Rahul Bhotika, and Stefano Soatto. Task adaptive parameter sharing for multi-task learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7561–7570, 2022. 1

[30] Xin Wang, Fisher Yu, Lisa Dunlap, Yi-An Ma, Ruth Wang, Azalia Mirhoseini, Trevor Darrell, and Joseph E Gonzalez. Deep mixture of experts via shallow embedding. In *Uncertainty in artificial intelligence*, pages 552–562. PMLR, 2020. 2

[31] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020. 6, 7

[32] Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. *Advances in Neural Information Processing Systems*, 33:6514–6527, 2020. 6

[33] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2369–2378, 2020. 7

[34] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 2, 6, 7, 8

[35] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019. 2, 7, 8

[36] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020. 2

[37] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. 2, 6, 7, 8

[38] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020. 2

[39] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016. 6, 7, 8

[40] Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris Holmes, Frank Hutter, and Yee Whye Teh. Neural ensemble search for performant and calibrated predictions. *arXiv preprint arXiv:2006.08573*, 2:3, 2020. 2

[41] 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. 3

[42] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, zhifeng Chen, Quoc V Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*, 2022. 2

[43] Zixuan Zhou, Xuefei Ning, Yi Cai, Jiashu Han, Yiping Deng, Yuhan Dong, Huazhong Yang, and Yu Wang. Close: Curriculum learning on the sharing extent towards better one-shot nas. *arXiv preprint arXiv:2207.07868*, 2022. 2