PRANC: PSEUDO RANDOM NETWORKS FOR COMPACTING DEEP MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compacting deep models has various applications where the communication and/or storage is expensive including multi-agent learning. We introduce a simple yet effective framework for compacting neural networks. In short, we train our network to be a linear combination of many pseudo-randomly generated frozen models. Then, one can reconstruct the model by communicating or storing the single 'seed' scalar used to generate the pseudo-random 'basis' networks along with the learned linear mixture coefficients. Our method, denoted as PRANC, learns almost $100\times$ fewer parameters than a deep model and still performs reasonably well on several datasets and architectures. PRANC enables 1) efficient communication of models between agents, 2) efficient model storage, and 3) memory-efficient inference by generating layer-wise weights on the fly. We test PRANC on CIFAR-10, CIFAR-100, tinyImageNet, and ImageNet-100 with various architectures like AlexNet, LeNet, ResNet18, ResNet20, and ResNet56 and demonstrate a massive reduction in the number of parameters while providing satisfactory performance on these benchmark datasets.

1 INTRODUCTION

Many artificial intelligence and machine learning applications would benefit from the efficient communication of deep models between agents. However, modern deep neural networks often have millions to billions of parameters, making model communication costly or even infeasible. The problem worsens when the agents need to communicate in environments with low bitrate constraints. Communication constraints could emerge from the physics of the environment, for instance, underwater or underground communications, or caused by an adversary as in communications denied environments. The long-range communication bandwidth for these applications could be as low as 100 bits per second. In such a low bitrate environment, transferring a ResNet18 model with 11M parameters takes more than five days. Moreover, in distributed learning applications with a large number of agents, even in high-bandwidth WiFi networks, many peer-to-peer communications may be limited by the congestion in the network (packets from different senders colliding with one another).

Going beyond communications, storing these large models on edge devices poses another significant challenge. Edge devices often come with small memories that are not suitable for storing large neural networks. Hence, such applications may benefit from compacting a deep model to fewer number of parameters, so that they can construct the model on-demand to run inference.

To solve this problem, one may compact the model by distilling it into a smaller model (Hinton et al., 2015), pruning the model parameters (Lin et al., 2020), or quantizing the parameters (Lee et al., 2021). More recently, dataset distillation has been proposed as an alternative method which reduces the size of the dataset to be used for training the model (Wang et al., 2018). However, most of these methods are limited to small reduction factors, e.g., less than $30\times$. Also, knowledge distillation methods reduce the model architecture to a smaller one with fewer number of layers, which may limit the future application of that model, e.g., for future fine-tuning or lifelong learning.

We are interested in compacting a deep model by a considerable factor (e.g., $100\times$) without changing its architecture. The core idea behind our approach is simple. We constrain our model to be a linear combination of a finite set of randomly initialized models, called *basis* models. Hence the problem boils down to finding the optimal linear mixture coefficients that result in a network that can solve the task effectively. Importantly, given that the basis models are all random, the agents can

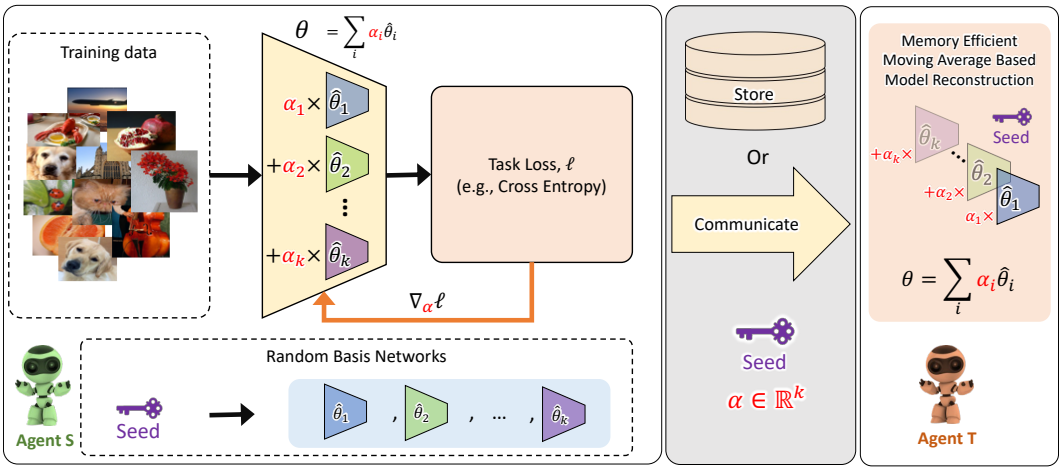

Figure 1: We restrict the deep model to be a linear combinations of k randomly initialized models. Since the number of models is much less than the size of the model, it is much less expensive to communicate or store the coefficients compared to the model or data itself. We tune α to minimize the loss of the task using standard backpropagation.

efficiently transmit them to one another by simply communicating the seed (a single scalar) of their pseudo-random generator. A receiving agent then can use the communicated seed to replicate all random basis models of the transmitting agent locally. Lastly, and in addition to the seed, the agents share their linear mixture coefficients to enable the receiving agent to replicate their model locally.

A naive way of learning the coefficients is to regress an already pretrained model parameters with simple MSE loss at the parameter space. However, this does not work well since the pretrained model may not be in the span of basis models. More importantly, MSE loss in the parameter space may not be correlated with the task loss. We introduce a very simple and efficient way of learning the coefficients that relaxes this optimization and looks for a model that is 'functionally' similar to the pretrained model rather than being close in the parameter space.

In addition to efficiency, our proposed method provides secured communication and storage, which is of significant interest in applications concerning cybersecurity and privacy. Briefly, our 'basis' functions are generated with pseudo-random generators with a 'seed.' The seed could be shared between the communicated agents privately. Pseudo-random generators are designed so that a random sequence has minimal correlation with its own minimally shifted version. Hence, a small change in the seed value at the reconstruction time will result in constructing a completely random model. This can enable a simple method for secured communication or storage of the models in applications with cybersecurity or privacy concerns.

Contributions: Our simple but effective method, denoted as PRANC, is efficient in computation and memory at the learning and reconstruction time. We perform experiments using various datasets and model architectures and show higher accuracy with much smaller number of parameters compared to the baselines. For instance, on CIFAR-10, an AlexNet model with 1.7M parameters achieves 83% accuracy. PRANC compacts this model to $10,000$ parameters only ($170\times$) and achieves 71% accuracy. This is much more efficient than the SOTA dataset distillation method (Cazenavette et al., 2022) that gets 60% with more than $300K$ parameters.

2 Prior Work

Some prior works (Ramanujan et al., 2020; Malach et al., 2020; Chen et al., 2021; Gallicchio & Scardapane, 2020) have shown that randomly initialized networks have a subnetwork that competes with the original network's accuracy. Some recent papers like (Wortsman et al., 2020) introduced an application for using this phenomenon in continual learning. Using random networks is appealing from multiple perspectives, for instance for model communication or for rapid adaptation. In this work, as opposed to finding subnetworks in a randomly generated network (i.e., masking), we seek a

linear combination of a relatively small set of randomly generated networks, denoted as *basis* models, that can reconstruct an accurate model. This approach can be used to transmit knowledge with very limited communication (which is one of the important bottlenecks in federated learning), can reduce the storage required for a model (that makes the algorithm useful for embedded devices or continual learners), and finally, can enable on-the-fly weight generation reducing the memory queries and accelerate the running time of a model. However, our main goal, in this paper, is to minimize the communication between agents when exchanging knowledge. For that purpose, current methods can be divided into two main categories described below.

2.1 Model Compression

Model compression is defined as reducing the number of bytes required to store a deep model. Several papers like XNOR-NET (Rastegari et al., 2016) and EWGS (Lee et al., 2021) use weight/activation (W/A) quantization for reducing the size of network. Although W/A Quantization has proven to be an effective approach for reducing the size of network while maintaining the accuracy, it is mainly designed for optimizing the computation for network inference. Besides, quantization has the limit of $(32\times)$ size reduction by reducing the 32-bit floating point to binary. Here, our goal is to reduce the required data by more than $(50\times)$. Another approach that is used for compressing a model is pruning that sets unimportant weights to zero which reduces the number of floating point operations (FLOPS) and can also reduce the amount of data required to store and communicate a network. These methods include: Neuron Merging (Kim et al., 2020), Dynamic pruning (Lin et al., 2020; Siems et al., 2021), ChipNet (Tiwari et al., 2021), Pruning at initializing (Hayou et al., 2020), Wang et.al. (Wang et al., 2020), and Collaborative Compression (CC) (Li et al., 2021). Once again, most of these methods use sparsity factors of 20 times or less, which is lower than our goal in this paper. Lastly, there is some prior work that decomposes model filters as a linear combination of some basis filters (Han et al., 2020; Bagherinezhad et al., 2017). The goal of such methods is to reduce the computation and not necessarily the number of parameters. We focus on extremely small number of parameters that cannot be achieved by such methods.

2.2 Data Compression - Core Set

Another approach to recreate an accurate network is to store or communicate its training dataset, and train a network in the target agent. Since most of the datasets are large, methods are proposed to synthesize metadata in the shape of images or obtaining a core set of the dataset. These methods include: Dataset Distillation (DD) (Wang et al., 2018), that regresses images and learning rate, Flexible Dataset Distillation (FDD) (Bohdal et al., 2020), that regresses pseudo-labels for real images, soft labeling dataset distillation (SLDD) (Sucholutsky & Schonlau, 2021), that generates pseudo-label and images. All these methods require the seed that initializes the network. Other methods including Dataset Condensation with distribution matching (DM) (Zhao & Bilen, 2021a), with differentiable Siamese augmentation (DSA) (Zhao & Bilen, 2021b), and Dataset distillation by matching training trajectories (DDMT) (Cazenavette et al., 2022) took a step further and devised seed-independent approaches. These methods often rely on a second-order optimization (similar to meta-learning approaches), which is computationally expensive and limits their application to large scale networks and large data. We show that our proposed method, PRANC, provides better accuracy with much fewer number of regressed parameter on same architectures compared to the mentioned approaches.

3 Method

We are interested in training a deep model with very small number of parameters so that it is less expensive to transfer the model from one agent to another or store it on an agent with small memory. This is in contrast to the goal of most prior work (e.g, model compression, pruning, or quantization) that aim to reduce the inference computation or improve the generalization. Hence, we introduce a compact representation assuming no change in the model size, number of non-zero parameters, or the precision of computation.

We assume that the deep model can be written as a linear combination of a set of randomly initialized models, called **basis**. Since we can use pseudo-random generator to generate the random models, one can communicate or store all random models by simply communicating or storing a single scalar that

is the seed of the pseudo-random generator. Note that the basis models are not necessarily orthogonal to each other, but their pairwise dot product should be close to zero since the number of samples (models) is much smaller than the dimensionality of each model. Then we optimize the weights of each base model so that their linear combination can solve the task (e.g, image classification.)

More formally, given a set of training images $\{x_i\}_{i=1}^N$ and their corresponding labels $\{y_i\}_{i=1}^N$, we want to train a deep model $f(.;\theta)$ with parameters $\theta \in \mathbb{R}^d$ so that $f(x_i;\theta)$ predicts y_i. The standard practice is to optimize θ by minimizing the empirical risk, $R(\theta) = \frac{1}{N}\sum_{i=1}^N L(f(x_i;\theta), y_i)$, where $L(\cdot,\cdot)$ is a discrepancy-measure, e.g., cross-entropy. However, in communicating such a model, we need to send a high-dimensional vector θ that contains d scalars.

To reduce the cost of communication, we assume a set of randomly initialized basis models with parameters $\{\hat{\theta}_j\}_{j=1}^k$. These basis models are generated using a known seed and are frozen throughout the learning process. Then we define:

$$\theta := \sum_{j=1}^{k} \alpha_j \hat{\theta}_j$$

where α_j is a scalar weight for the j'th basis model. Assuming that $k \ll d$, it will be less expensive to communicate or store α instead of θ.

To optimize α, one may first optimize for θ to find θ^* and then regress it by minimizing:

$$\arg\min_{\alpha} ||\theta^* - \sum_{j=1}^{k} \alpha_j \hat{\theta}_j||^2$$

However, since $k \ll d$, the optimum solution θ^* may be far from the span of the basis models, resulting in an inferior solution (also shown empirically in our experiments). We argue that there are infinite number of solutions for θ that are as good as θ^*, so we may search for one of those that has smaller residual error when projected to the span of the basis models. Hence, we simply search for a solution in the span of the basis models that minimizes the loss of the task by optimizing:

$$\arg\min_{\alpha} \sum_{i} L\Big(f(x_i; \sum_{j=1}^{k} \alpha_j \hat{\theta}_j), y_i\Big)$$

For large models, all basis models may not fit in the GPU memory, so at each iteration, we optimize the α for a random set of m basis models only using a random mini-batch of images. This is an instance of coordinate descent algorithm for α. Note that we use a single scalar seed to generate all the seeds for basis models randomly and store the seeds. Then, we use the generated seed for each basis model to generate that model. This method gives us random access to the basis models since we can generate any base model without traversing through all base models.

Optimization efficiency: Note that the optimization is very simple and efficient since $\frac{dL}{d\alpha} = \frac{dL}{d\theta}\frac{d\theta}{d\alpha}$ and $\frac{d\theta}{d\alpha_j} = \hat{\theta}_j$. Hence, we use standard back propagation to calculate $\frac{dL}{d\theta}$ and then simply multiply that with the matrix of basis models to get:

$$\frac{dL}{d\alpha} = \frac{dL}{d\theta}\hat{\theta}$$

Model reconstruction efficiency: Since basis models are generated using a pseudo-random generator, we can reconstruct the model using a simple running average of the basis models: generate each entry in $\hat{\theta}_j$, multiply it with α_j, add it to the running average, discard the entry and go to the next entry of $\hat{\theta}_j$. This way, the memory foot print of the reconstruction is very negligible (only one scalar more than the model size d.)

On-demand model reconstruction: In some applications, the agent may need to run the inference rarely, but does not have enough memory to hold the model in the memory. The device can store α, reconstruct each convolutional filter using the corresponding entries of the basis models, apply it to the input, and then discard the filter and go to the next filter. This process has a very small memory footprint as it needs to store α and just one filter at a time.

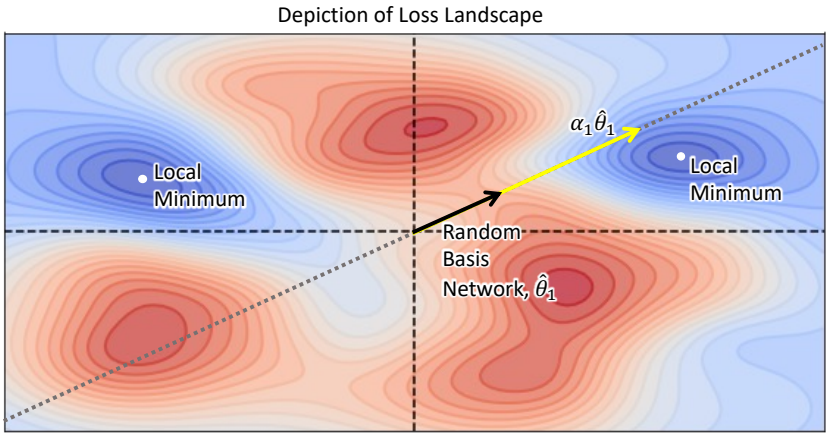

Figure 2: A simple illustration of the loss landscape of a model with two parameters and one basis model. None of the two local minimas may be in the span of the basis models, so we search for α to find a local minima in the span of the basis models.

Distributed learning: In order to train the model on multiple GPUs, we use a simple distributed learning algorithm to increase m. We divide m basis models between g GPUs so that each GPU works on $\frac{m}{g}$ basis models only. Then, we distribute α among GPUs. Each GPU calculates the partial weighted average over its basis models and distributes it to all GPUs. Then, all GPUs will have access to the complete weighted average and will use it to do backpropagation in standard distributed learning form and update their own set of α.

Batchnorm layer: We minimize the loss of the task by tuning the α instead of the model weights as done in standard learning. However, the μ and σ parameters of the Batchnorm layer are not tuned by optimizing the loss. They are directly derived from the data, so it is not straight forward to use our method for μ and σ. One can add a regression loss to estimate their data-driven value. However, for simplicity in this work, we assume that we can communicate those parameters and include them in the budget. This makes sense since the number of μ and σ are relatively small compared to the number of weight parameters. Note that we handle the other two parameters, γ and β, similar to regular weights since they are learned parameters.

4 Experimental Results

We report the results of PRANC on various datasets, architectures, and number of basis models.

Implementation details: We report the experimental results of our method on standard image classification datasets CIFAR-10 (Krizhevsky et al., 2014), CIFAR-100 (Krizhevsky et al., 2009), tinyImageNet (Le & Yang, 2015), and ImageNet-100 (a subset of ImageNet (Deng et al., 2009) with 100 classes.) For all experiments on CIFAR-10, CIFAR-100, and tinyImageNet, we used a single NVIDIA TITAN X (Pascal) GPU. For ImageNet-100, we used 4 NVIDIA GeForce RTX 3090 GPUs. We trained all models for 2000 epochs with learning rate of 10^{-3} for the first 1600 epochs and then 10^{-4} for 400 epochs. We updated $m = 500$ random entries of α at each epoch.

Comparison to Dataset Distillation methods: In Table 3, we report the accuracy and the number of paramteres for PRANC and various dataset distillation methods. Most these methods are based on meta-learning approaches that involve a large computational cost and memory foot print at the training time, so they are limited in the depth of the model. Also, they need to do a few gradient descend steps in constructing the model. Also, most dataset distillation methods need to optimize for at least one image per category, so they cannot reduce the number of parameters by a large factor (30,720 parameters corresponds to one image per class in CIFAR-10). For CIFAR-10, we use AlexNet (which is a modified version that is described in (Wang et al., 2018)), for CIFAR-100 and tinyImageNet, we use depth-3 and depth-4 128-width ConvNet architectures (Cazenavette et al., 2022) respectively. Note that some dataset distillation methods do not assume that they can communicate

the seed, so they solve a more challenging task since the distilled data should be able to tune any randomly initialized model. However, since we are focusing on reducing the cost of communication and storage, using a fixed seed, which is the central part of our idea, is not prohibitive.

Table 1: Comparison of our method with dataset distilation methods on various datasets and architectures. 3-128-Conv and 4-128-Conv represents 3-depth 128-width ConvNet and 4-depth 128-width ConvNet respectively. "seed" column shows which methods utilized a random generator with a constant seed. Note that the methods that do not have a constant seed are solving a more challenging task, which is not necessary in compacting the model for communication since communicating the seed is very cheap. "Transferring trained model" is somehow an upper-bound since it can optimize all weights. Our method outperforms the baselines with a large margin and much fewer number of parameters. We show the results with different number of basis models in Figure 3. Note that FDD optimizes the labels for a core-set of images, so we count the number of parameters in both images and labels since they all need to be communicated. We run our model 5 times and report the mean and standard deviation. We copy the mean and standard deviation for most baselines from the corresponding papers.

Method	Dataset	Architecture	# Params	Accuracy	seed?
Transferring trained model	CIFAR-10	AlexNet	1,756,426	$84.8 \pm 0.1\%$	
FDD (Bohdal et al., 2020)	CIFAR-10	AlexNet	397,000	$43.2 \pm 0.5\%$	✓
SLDD (Sucholutsky & Schonlau, 2021)	CIFAR-10	AlexNet	308,200	60%	✓
DD (Wang et al., 2018)	CIFAR-10	AlexNet	307,200	54%	✓
DM (Zhao & Bilen, 2021a)	CIFAR-10	AlexNet	30,720	$26.0 \pm 0.8\%$	
DSA (Zhao & Bilen, 2021b)	CIFAR-10	AlexNet	30,720	$28.8 \pm 0.7\%$	
DC (Zhao et al., 2020)	CIFAR-10	AlexNet	30,720	$28.3 \pm 0.5\%$	
CAFE (Wang et al., 2022)	CIFAR-10	AlexNet	30,720	$30.3 \pm 1.1\%$	
CAFE+DSA (Wang et al., 2022)	CIFAR-10	AlexNet	30,720	$31.6 \pm 0.8\%$	
DDMT (Cazenavette et al., 2022)	CIFAR-10	AlexNet	30,720	$46.3 \pm 0.8\%$	
Ours	CIFAR-10	AlexNet	**10,000**	$71.5 \pm 0.5\%$	✓
Transferring trained model	CIFAR-100	3-128-Conv	504,420	$56.2 \pm 0.3\%$	
FDD (Bohdal et al., 2020)	CIFAR-100	3-128-Conv	317,200	$11.5 \pm 0.4\ \%$	✓
DM (Zhao & Bilen, 2021a)	CIFAR-100	3-128-Conv	307,200	$11.4 \pm 0.3\%$	
DSA (Zhao & Bilen, 2021b)	CIFAR-100	3-128-Conv	307,200	$13.9 \pm 0.3\%$	
DC (Zhao et al., 2020)	CIFAR-100	3-128-Conv	307,200	$12.8 \pm 0.3\%$	
CAFE+DSA (Wang et al., 2022)	CIFAR-100	3-128-Conv	307,200	$14.0 \pm 0.3\%$	
DDMT (Cazenavette et al., 2022)	CIFAR-100	3-128-Conv	307,200	$24.3 \pm 0.3\%$	
Ours	CIFAR-100	3-128-Conv	**10,000**	$25.0 \pm 0.5\%$	✓
Transferring trained model	tinyImageNet	4-128-Conv	857,160	$37.6 \pm 0.4\%$	
DM (Zhao & Bilen, 2021a)	tinyImageNet	4-128-Conv	2,457,600	$3.9 \pm 0.2\%$	
DDMT (Cazenavette et al., 2022)	tinyImageNet	4-128-Conv	2,457,600	$8.8 \pm 0.3\%$	
Ours	tinyImageNet	4-128-Conv	**10,000**	$15.1 \pm 0.4\%$	✓

Comparison with model pruning methods: Most model pruning methods use a pruning factor less than 99% which leads to less than $50\times$ compression factor. Note that communicating a sparse model, we need to communicate the index and value for all non-zero entries. DPF (Lin et al., 2020) and STR(Kusupati et al., 2020) are two of the SOTA methods that use a large sparsity. We used their code on CIFAR-10 and CIFAR-100 along with ResNet-20 and ResNet-56 architectures and compare them with our method in Table 2. PRANC achieves consistently higher accuracy with fewer number of parameters.

Comparison with model distillation methods: We use model distillation as a baseline. In order to achieve a model comparable with our method in the number of parameters, we need to use a very small student model. Even a LeNet architecture has more than 60,000 parameters. On CIFAR-10, we train a ResNet18 and distill it to a LeNet model. We repot the results in Table 3. Our ResNet-20 model achieves better accuracy with significantly fewer number of parameters.

Sensitivity to seed: We did experiment with changing the seed at the reconstruction time. On CIFAR-10 with AlexNet, simply increasing the seed value by one, results in 9.4% accuracy only, which is close to the chance. This is expected as the new basis models are not correlated with the ones that are used in training. Therefore, in applications dealing with safe communication of deep

learning models, even when α is intercepted, the adversary will not be able to generate the target model without having the seed (i.e., the private key).

Table 2: Comparison of our model with 2 SOTA pruning methods, DPF (Lin et al., 2020) and STR (Kusupati et al., 2020). "Pr." denotes the pruning rate. Also, when network is pruned, we have to keep two numbers for each weight: the weight itself and its position in the model. Our method requires $\#\alpha + (\#\mu + \#\sigma)$ number of parameters to communicate the meand and std for the Batchnorm layers too.

Method	Dataset	Architecture	# Params	Accuracy
DPF (Pr. 98.2%)	CIFAR-10	ResNet-20	4,920×2	41.86 %
STR (Pr. 95.5%)	CIFAR-10	ResNet-20	12,238×2	75.99 %
Ours	**CIFAR-10**	**ResNet-20**	**1,000 + (1,376)**	**57.65 %**
Ours	**CIFAR-10**	**ResNet-20**	**10,000 + (1,376)**	**81.14%**
DPF (Pr. 98.43%)	CIFAR-10	ResNet-56	13,414×2	47.66%
STR (Pr. 98.4%)	CIFAR-10	ResNet-56	13,312×2	67.77%
Ours	**CIFAR-10**	**ResNet-56**	**6,000 + (4,064)**	**70.76%**
DPF (Pr. 96.13%)	CIFAR-100	ResNet-20	10,770×2	12.25%
STR (Pr. 96.12%)	CIFAR-100	ResNet-20	10,673×2	13.18%
Ours	**CIFAR-100**	**ResNet-20**	**5,000 + (1,376)**	**32.33%**
DPF (Pr. 97.8%)	CIFAR-100	ResNet-56	19,264×2	19.11%
STR (Pr. 97.8%)	CIFAR-100	ResNet-56	18,881×2	25.98%
Ours	**CIFAR-100**	**ResNet-56**	**5,000 + (4,064)**	**32.97%**

Table 3: Results of comparing with model distillation. Our method outperforms a LeNet distilled from ResNet-18 on CIFAR-10.

Method	Dataset	Architecture	# Params	Accuracy
Distilled from R18	CIFAR-10	LeNet	62,006	74.1%
Ours	**CIFAR-10**	**ResNet-20**	**10,000 + (1,376)**	**81.14%**

4.1 Regressing θ^* directly

As described earlier, we can first train a model to get θ^* and then optimize for α by regressing that solution using MSE loss in the parameter space. As shown in Fig. 2, this may not succeed since the optimum model may not be in the span of the basis models, and also the MSE loss in the parameter space is not necessarily correlated with the task loss. Table 4 shows that the accuracy of this baseline using 10,000 parameters is not much better than chance.

Table 4: Results of regressing a pretrained model using 10,000 basis models.

Dataset	Architecture	Full model Accuracy	Regression Accuracy	Ours Accuracy
CIFAR-10	AlexNet	$84.8 \pm 0.1\%$	10.0%	$71.5 \pm 0.5\%$
CIFAR-10	LeNet (LeCun et al., 2015)	73.5%	12.74%	64.3%
CIFAR-100	3-128-Conv	$56.2 \pm 0.3\%$	1.14%	$25.0 \pm 0.5\%$
CIFAR-100	AlexNet	50.7%	1.0%	31%
tinyImageNet	4-128-Conv	$37.6 \pm 0.4\%$	0.5%	$15.1 \pm 0.4\%$

4.2 ImageNet-100

Since our method is reasonably efficient in learning, particularly compared to meta-learning approaches that depend on second order derivatives of the network, we can evaluate it on larger scale

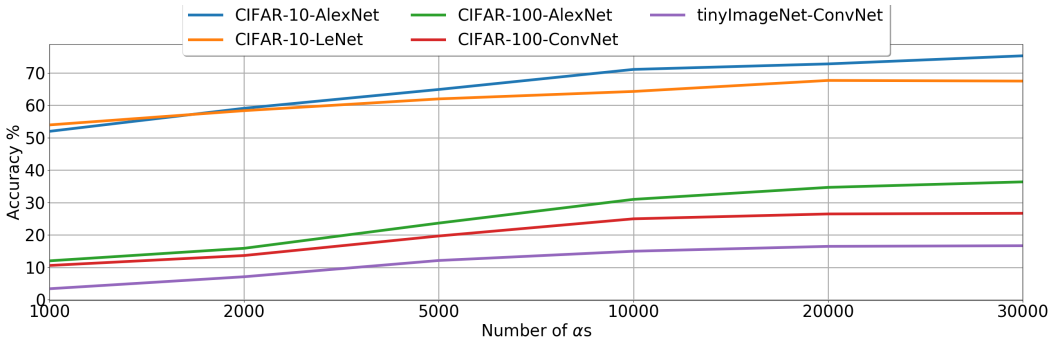

Figure 3: Illustration of impact of k in accuracy of different models trained on different architectures. The accuracy improves by increasing the number of basis models. As described in the text, this experiment is not very fair for the larger values of k since their α is updated less frequently.

models. We evaluate our method on ImageNet100 with ResNet-18 architecture. Table 5 shows the results. Our method achieves 34.8% Top-1 and 64.82% Top-5 accuracy with less than 30,000 parameters while the standard ResNet-18 model achieves 82.1% Top-1 and 95.16% Top-5 accuracy with more than 11M parameters. Our method suffers from a large drop in accuracy, however, considering that ImageNet100 is a difficult task with 1% chance accuracy, getting 34% Top-1 accuracy with such a compact set of parameters (only 30,000) is promising. We hope that our results will pave the way for the follow-up work to improve this accuracy further. Lastly, training our method for 2000 epochs on ResNet-18 and ImageNet100 took almost 82.5 GPU-hours.

Table 5: Result of our method on ImageNet-100 dataset and ResNet-18. With less than 30,000 parameters, our method achieves 34.8% accuracy which is much better than the chance level (1%).

Method	**Dataset**	**Architecture**	**# Params**	**Accuracy**
Transferring trained model	ImageNet-100	ResNet-18	11,227,812	**82.1%**
Ours	**ImageNet-100**	**ResNet-18**	**20,000 + (9,600)**	**34.8%**

5 Ablation Study:

Effect of varying k: We perform an ablation study to understand the effect of the number of basis models, k. We change k from 1,000 to 30,000 for CIFAR-10 (using AlexNet and LeNet), CIFAR-100 (using AlexNet and 3-128-ConvNet), and tinyImageNet (using 4-128-ConvNet) and plot the accuracy in Figure 3. As expected the accuracy increases as we increase the number of basis functions. In this experiment, we keep the total number of epochs and m constant, so as we increase the number of basis models, each α is updated fewer number of times. Hence, these results are not entirely fair: the model with more basis models can improve if we increase the number of updates.

Partial reconstruction of the model: Here we investigate whether we can progressively generate a model as a function of increasing the number of *basis* networks and if we can obtain networks that increasingly perform better on the task. We show that if no ordering is enforced on basis networks and their corresponding αs, then the constructed model would have very low accuracy until using a large number of basis models (See Figure 4). Interestingly, however, if we sort the basis networks according to their αs (in a decreasing absolute value), we observe acceptable performance progress as a function of increasing basis models, for different datasets and varying architectures. This is an interesting observation as it allows us to increase the performance of network progressively, and it suggests that the loss landscape is reasonably flat around the local minima since ignoring basis models with small α does not change the model accuracy much. We leave the theoretical analysis and closer investigation of this observation for future work.

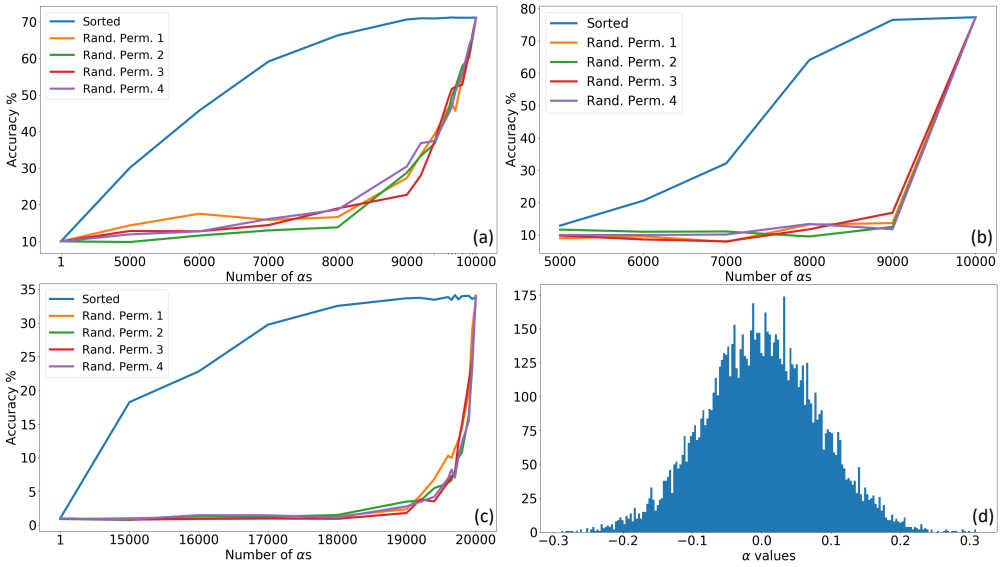

Figure 4: **(a,b,c)** Partial reconstruction of the model using random ordering and sorting of α values for (a) AlexNet on CIFAR10, (b) ResNet20 on CIFAR-10, and (c) ResNet-18 on ImageNet100. **(d)** Distribution of α values for ResNet20 on CIFAR10. Please see the ablation for more details.

6 Future Directions

We believe our ideas can enable multiple future directions:

Generative models for memory-replay: Our method can be used to compact a generative model (e.g., GAN), where the α parameters may be stored in the agent or sent to another agent. Then, any agent can reconstruct the model in the future and draw samples from it that are similar to the samples that were used earlier to train the model. This enables memory replay in lifelong learning in a single agent with limited memory or in multiple agents with limited communication.

Progressive compactness: In this method, we assumed a set of basis models with no specific ordering. However, one can optimize α so that the earlier indices of α can reconstruct an acceptable model. Then, depending on the communication or storage budget, the target agent can decide on how many α parameters it needs by trading off between accuracy and compactness. We showed that sorting α values is a first step in this direction, but as the future work, one can optimize α by simply calculating the loss for various subsets of αs and minimizing their summation.

7 Conclusion

We introduced a simple but effective method that can learn a model as a linear combination of a set of frozen randomly initialized models. The final model can be compactly stored or communicated using the seed of the pseudo-random generator and the coefficients. Moreover, our method has a very small extra computation or memory footprint at the learning or reconstruction stage. we perform extensive experiments on multiple image classification datasets and multiple architectures and show that our method achieves better accuracy with fewer number of parameters compared to SOTA baselines. We believe many applications including lifelong learning and distributed learning can benefit from our ideas. Hence, we hope this paper opens the door to studying more advanced compacting methods based on frozen random networks.

Limitations: For large models, since our method optimizes a subset of α at each iteration, it needs to run for many epochs to converge (82.5 GPU-hours for ImageNet100 using ResNet18). As discussed, some model parameters, e.g., μ and σ in the batch normalization layer, cannot be easily regressed using our method since they are calculated directly from data rather than minimizing the task loss. In this paper, we assumed we communicate them with no change.

Ethics statement: Our main idea is to compact deep models so that we can communicate and/or store them efficiently. AI agents that can benefit from these ideas may be used for various applications including military or surveillance. Moreover, the seed of the pseudo-random generator enables secure communication or storage of the deep models, which may have harmful societal impact at the hand of adversaries. We do acknowledge the existence of possible harmful applications, however, we believe studying such methods and releasing the results publicly may have benefits that outweigh the harms.

Reproducibility: We have explained all the details of our algorithm and experiments in the submission. Also, we have included our code in the supplementary material to enable easier reproduction of our experiments.

REFERENCES

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*, pp. 7120–7129, 2017. 3

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: Learn labels instead of images. *arXiv preprint arXiv:2006.08572*, 2020. 3, 6

George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. *arXiv preprint arXiv:2203.11932*, 2022. 2, 3, 5, 6

Xiaohan Chen, Jason Zhang, and Zhangyang Wang. Peek-a-boo: What (more) is disguised in a randomly weighted neural network, and how to find it efficiently. In *International Conference on Learning Representations*, 2021. 2

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*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848. 5

Claudio Gallicchio and Simone Scardapane. Deep randomized neural networks. *Recent Trends in Learning From Data*, pp. 43–68, 2020. 2

Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1580–1589, 2020. 3

Soufiane Hayou, Jean-Francois Ton, Arnaud Doucet, and Yee Whye Teh. Robust pruning at initialization. *arXiv preprint arXiv:2002.08797*, 2020. 3

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

Woojeong Kim, Suhyun Kim, Mincheol Park, and Geunseok Jeon. Neuron merging: Compensating for pruned neurons. *Advances in Neural Information Processing Systems*, 33:585–595, 2020. 3

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6(1):1, 2009. 5

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55(5), 2014. 5

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the International Conference on Machine Learning*, July 2020. 6, 7

Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015. 5

Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015. 7

Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6448–6457, 2021. 1, 3

Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6438–6447, 2021. 3

Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253*, 2020. 1, 3, 6, 7

Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pp. 6682–6691. PMLR, 2020. 2

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020. 2

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016. 3

Julien Niklas Siems, Aaron Klein, Cedric Archambeau, and Maren Mahsereci. Dynamic pruning of a neural network via gradient signal-to-noise ratio. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. 3

Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021. 3, 6

Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak K Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. *arXiv preprint arXiv:2102.07156*, 2021. 3

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020. 3

Kai Wang, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. Cafe: Learning to condense dataset by aligning features. *arXiv preprint arXiv:2203.01531*, 2022. 6

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018. 1, 3, 5, 6

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020. 2

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. *arXiv preprint arXiv:2110.04181*, 2021a. 3, 6

Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, pp. 12674–12685. PMLR, 2021b. 3, 6

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. *arXiv preprint arXiv:2006.05929*, 2020. 6