PRANC: Pseudo RAndom Networks for Compacting deep models

Parsa Nooralinejad
University of California, Davis

Ali Abbasi
Vanderbilt University

Soroush Abbasi Koohpayegani[*]
University of California, Davis

Kossar Pourahmadi Meibodi [*]
University of California, Davis

Rana Muhammad Shahroz Khan[*]
Vanderbilt University

Soheil Kolouri
Vanderbilt University

Hamed Pirsiavash
University of California, Davis

Abstract

We demonstrate that a deep model can be reparametrized as a linear combination of several randomly initialized and frozen deep models in the weight space. During training, we seek local minima that reside within the subspace spanned by these random models (i.e., 'basis' networks). Our framework, PRANC, enables significant compaction of a deep model. The model can be reconstructed using a single scalar 'seed,' employed to generate the pseudo-random 'basis' networks, together with the learned linear mixture coefficients. In practical applications, PRANC addresses the challenge of efficiently storing and communicating deep models, a common bottleneck in several scenarios, including multi-agent learning, continual learners, federated systems, and edge devices, among others. In this study, we employ PRANC to condense image classification models and compress images by compacting their associated implicit neural networks. PRANC outperforms baselines with a large margin on image classification when compressing a deep model almost 100 *times. Moreover, we show that PRANC enables memory-efficient inference by generating layer-wise weights on the fly. The source code of PRANC is here:* https://github.com/UCDvision/PRANC

1. Introduction

The prevailing notion is that larger deep models yield improved accuracy. Yet, it remains unclear if the better generalization of larger models stems from the increased complexity of the architecture or more parameters. Moreover, among numerous good local minima in the loss function, training finds one. In this paper, we introduce a fresh approach: viewing a deep model as a linear combination within the weight space of several randomly initialized and frozen models. During learning, our goal shifts to finding a minimum that exists within the subspace defined by these initial models. Our findings highlight the potential to significantly compact deep models by retaining only the seed value of the pseudo-random generator and the coefficients for weight combination.

This efficient reparameterization benefits AI and ML applications by reducing deep model size for easier storage or communication. In modern neural networks with millions to billions of parameters, storage, and communication become costly. This issue worsens in low-bitrate environments due to physical constraints or adversarial disruption. For instance, underwater applications might have as low as 100 bits per second bandwidth, then, transferring ResNet18's 11M parameter model takes more than 40 days in such conditions. Moreover, in distributed learning with many agents, high-bandwidth WiFi networks still face congestion issues.

Going beyond communications, loading or storing these large models on edge devices poses another significant challenge. Edge devices often come with small memories unsuitable for storing large neural networks and may want to run the model less frequently (on-demand). Hence, they may benefit from compacting a deep model to fewer parameters to construct the model layer-by-layer or even kernel-by-kernel on-demand to run each inference. This will result in significantly less I/O cost.

One may compact the model by distilling it into a smaller model [18], pruning the model parameters [29], quantizing the parameters [26], or sharing the weights as much as possible [40, 6]. More recently, dataset distillation [48] is proposed. It can be seen as an alternative to model compression since one can store or communicate the distilled dataset and then train the model again when needed. However, most of

[*] Equal contribution

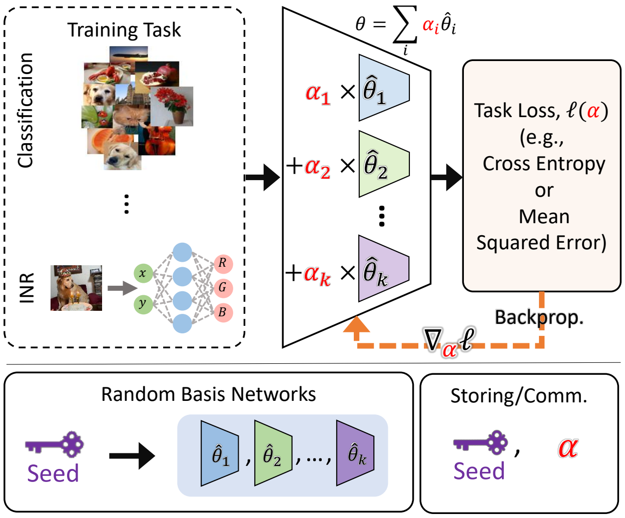

Training Task
Classification
INR
x
y
R
G
B
$\theta = \sum_i \alpha_i \hat{\theta}_i$
$\alpha_1 \times \hat{\theta}_1$
$+\alpha_2 \times \hat{\theta}_2$
$+\alpha_k \times \hat{\theta}_k$
Task Loss, $\ell(\alpha)$ (e.g., Cross Entropy or Mean Squared Error)
Backprop.
$\nabla_\alpha \ell$
Random Basis Networks
Seed
$\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_k$
Storing/Comm.
Seed, α

Figure 1. We restrict the deep model to be a linear combination 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.

these methods are limited to small reduction factors, *e.g.*, less than 30×. Also, knowledge distillation methods reduce the model architecture to a smaller one with fewer layers, which may limit the future application of that model, *e.g.*, for future fine-tuning or lifelong learning, which needs the deeper architecture.

We are interested in compacting a deep model by a considerable factor (*e.g.*, 100×) 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. The model can then be succinctly represented by the seed (a single scalar) to generate the pseudo-random basis models and the linear mixture coefficients (See Figure 1). Our method can be seen as a novel reparameterization of a deep model without changing its architecture. This enables us to study the effect of increasing the size of the architecture (both depth and width) without changing the number of optimized parameters (see Figure 3).

In addition to efficiency, our proposed method provides secure communication and storage, which is of significant interest in applications concerning cybersecurity and privacy. Briefly, our 'basis' models are generated with pseudo-random generators with a specific 'seed.' This seed could be privately shared between authenticated entities. Given the minimal self-correlation of pseudo-random sequences, a slight seed change produces a drastically different set of 'basis' models, making the publicly shared linear mixture coefficients useless to unauthorized parties. This design choice facilitates secure communication and storage, especially in cybersecurity or privacy-sensitive applications.

Theoretically, overparametrization is vital in contemporary neural networks, enhancing their representational power and simplifying optimization due to an improved loss landscape [33, 30]. Solutions of these over-parameterized systems often form a positive-dimension manifold [8], with larger systems lacking non-global minima [34]. Considering the abundance of good solutions, we examine if we can confine the solution search to low-dimensional subspaces defined by random vectors in the weight space (i.e., the 'basis' networks). Our experiments confirm the possibility of finding good solutions in very low-dimensional random subspaces in the weight space of overparametrized networks, urging further theoretical investigations.

Contributions: Below are our specific contributions:

- Introducing PRANC, a simple but effective network reparameterization framework that is memory-efficient during both the learning and reconstruction phases,
- Assessing the effectiveness of PRANC in compacting image recognition models for various benchmark datasets and model architectures, showing higher accuracy with a much fewer parameters compared to extensive recent baselines,
- Demonstrating the effectiveness of PRANC for image compression by compacting implicit neural representations for both natural and medical images,
- Showcasing the potential of PRANC in applications requiring encrypted communication of models (or data represented via models).

2. Related work

Random networks: Some prior works [35, 31, 7, 12] have shown that randomly initialized networks have a subnetwork that competes with the original network in accuracy. Some recent papers like [50] introduced an application for using this fact in continual learning. Instead of finding subnetworks in a randomly generated network (i.e., masking), we seek a linear combination of a small set of randomly generated networks, denoted as *basis* models, that can solve the task.

Model compression: Model compression is not a new topic. HashedNet [6] uses weight grouping with a hash function to reduce the number of learnable parameters. It can be seen as a specific case of our method where the random models are binary with an equal number of ones and each weight of the original model is one only in one of the random models. [6] experiments with MLP on small datasets. We reproduce HashedNet for our setting and show that our method outperforms it. Similar to HashedNet, Weight Fixed Network (WFN) [40] compresses the model by minimizing the entropy and number of unique parameters in a network. WFN preserves the model's accuracy with a $10\times$ reduction in storage size. Instead of hard-sharing the weights in HasedNet, [45] uses soft sharing. Although all these methods reduce the number of parameters, they all need to keep the index of each element to reconstruct the network. Han *et al*. [15] use pruning, quantization, and Huffman coding to achieve compression rates generally less than $50\times$. More recent approaches like MIRACLE [16] and weightless [37] have shown promising results with much higher compression rates (*e.g*., $+400\times$). However, they use large architectures, *e.g*. VGG which has 150M parameters, so even after $400\times$ compression, there are still more than 300K parameters (more than a dense ResNet-20). We show that we can reduce the number of required parameters keeping the network architecture intact.

Model pruning and quantization: Compressing a model can be defined as reducing the number of bytes required to store a deep model. Several papers like XNOR-NET [36] and EWGS [26] use weight/activation (W/A) quantization for reducing the size of a network. Although W/A Quantization has proven to be an effective approach for reducing network size while maintaining accuracy, it is mainly designed for optimizing the computation for network inference. Another approach that is used for compressing a model is pruning the set of less important 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 [20], Dynamic pruning [29, 38], ChipNet [43], Pruning at initializing [17], Wang et.al. [46], and Collaborative Compression (CC) [28]. Once again, most of these methods use sparsity factors of $20\times$ or less, which is lower than our goal in this paper. We compare our method, PRANC, with existing works that provide extreme compression rates (+99% pruning rate), e.g., DPF[29], STR[23], and SuRP[19]. Lastly, there are some prior works that decompose model filters as a linear combination of some basis filters [14, 2]. The goal of such methods is to reduce the computation and not necessarily the number of parameters. We focus on an extremely small number of parameters that cannot be achieved by such methods.

Data compression - core set: Another approach to recreating 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 obtain a core set of the dataset. These methods include: Dataset Distillation (DD) [48], which regresses images and learning rate, Flexible Dataset Distillation (FDD) [4], which regresses pseudo-labels for real images, soft labeling dataset distillation (SLDD) [41], 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) [52], with differentiable Siamese augmentation (DSA) [51], and Dataset distillation by matching training trajectories (DDMT) [5] took a step further and devised seed-independent approaches. These methods often rely on a second-order optimization, which is computationally expensive and limits their application.Moreover, the size of data required for storage in these methods is proportional to the size of input images. We show that PRANC provides better accuracy with a much fewer regressed parameters on the same architectures compared to the mentioned approaches.

Image compression: Some popular codecs like JPEG are based on hand-crafted modules to compress an image. Another line of image compression methods is learning-based approaches. These approaches usually train an auto-encoder on a large population of images [3, 32, 25, 9, 13] and store the code. Our method of using INR is also learning-based but is different from the above techniques since the model is learned on a single image (to overfit) rather than on a population of images. Hence, it may not suffer from the biases of the training data. COIN [10] is probably the closest to our method, which overfits an INR and stores all the parameters. We are different since we compact the INR by reparametrizing it as a linear combination of random networks and storing the coefficients.

3. Method

We are interested in training a deep model with a 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 works (*e.g*., model compression, pruning, or quantization) that aim to reduce the inference computation or improve the generalization while preserving the accuracy. 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 a 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. Although basis models are not necessarily orthogonal to each other, their pairwise dot product is 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

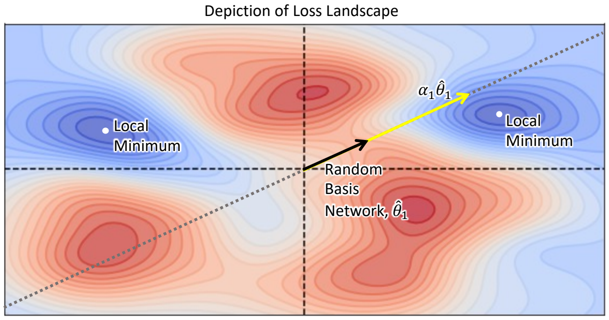

Depiction of Loss Landscape
$\alpha_1\hat{\theta}_1$
Local Minimum
Local Minimum
Random Basis Network, $\hat{\theta}_1$

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

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. 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 k 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 much 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 an infinite number of solutions for θ that are as good as θ^*, so we may search for one with a smaller residual error when projected to the span of the basis models. Hence, we search for a solution that minimizes the task loss in the basis models' span by optimizing:

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

Note that at the test time, after reconstructing the model by linear combination, the inference for PRANC is exactly the same as the standard dense model.

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 backpropagation to calculate $\frac{dL}{d\theta}$ and then multiply that with the basis models' matrix to get:

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

Memory efficiency in training: Note that the matrix of basis models $\hat{\theta}$ is very large, so keeping that in the memory is not efficient. Hence, we divide this matrix into multiple smaller chunks, and at each iteration, we generate each chunk using a pseudo-random generator at the GPU itself, perform the multiplication, discard the chunk, and go to the next chunk. This method reduces the memory footprint by a large factor at the cost of generating the whole random basis once per iteration, which is very efficient in modern GPUs. Choosing chunks of 100 alpha values for ResNet18 consumes almost 4.4GB (i.e., $11M \times 4 \times 100$) of GPU memory which is reasonable.

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 footprint of the reconstruction becomes negligible (i.e., $d+1$).

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. 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.

Distributed learning: In order to train the model on multiple GPUs, we use a simple distributed learning algorithm to increase m, the number of basis models. We divide m basis models between g GPUs so that each GPU works on 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 parameters of the BatchNorm layer are not tuned by PRANC. For the simplicity of this work, we assume that we can communicate those parameters and include them in the budget. This makes sense since the number of BatchNorm parameters is relatively small compared to the number of weight parameters.

4. Application

We test our framework on two different applications.

Image classification networks: In this setting, we parameterize an image recognition model, e.g., ResNet-20 for CIFAR-10, by our PRANC framework and optimize αs instead of the model weights. This results in a compact model that can be stored and communicated very efficiently.

Image compression using implicit neural networks: We also test our framework on compressing an implicit neural network (INR) that is over-fitted to a single image [42]. Such an INR inputs the coordinate of the pixel and returns the color value. Hence, one can store or communicate the INR model instead of the original image. We parameterize a standard INR model [42] using the PRANC framework so that we optimize the αs instead of the weights of the INR model. Our method outperforms JPEG compression on two standard datasets and two evaluation metrics.

5. Experiments on image classification

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

5.1. Comparison with model pruning methods:

For communicating a sparse model with a sparsity rate of more than 2×, it is required to transmit two numbers per parameter: the value of weight and its index in the network. Therefore, even if a model pruning method uses a pruning factor of 99% (100× reduction in size) since it should transmit the indices alongside the values, the actual reduction size will be smaller than 100×. DPF [29], STR[23], LAMP[27], RiGL[11], and SuRP[19] are the SOTA methods that use a large sparsity $(+50\times)$ and maintain a reasonable accuracy. We used their code on CIFAR-10 and CIFAR-100 along with ResNet-20 and ResNet-56 architectures and compared them with our method in Table 1. PRANC achieves consistently higher accuracy with fewer parameters. Please note that all these methods excluded BatchNorm layers from their pruning process. Therefore we also excluded them from the parameter count. For ResNet-20, the number of BatchNorm parameters is 2,752 and for ResNet-56 it is 8,128.

5.2. Comparison with model distillation methods:

One of the critical baselines for our work is model distillation. However, the number of parameters we use is very small compared to any existing CNN architecture. Even LeNet[24] (one of the smallest CNN architectures), has more than 60,000 parameters. To compare PRANC with model distillation, we trained a ResNet18 on CIFAR-10 and distilled its knowledge to a LeNet model. On the other hand, we compressed a ResNet20 model using PRANC with 10,000 αs and a ResNet56 model with merely 5,000 αs and compared their accuracies. As shown in Table 2, PRANC-compressed architectures require almost 5× fewer parameters while achieving higher accuracies with a significant gap (81.48% vs. 74.1%).

5.3. Comparison with dataset distillation methods:

In Table 3, we report the accuracy and the number of parameters for PRANC in comparison with various dataset distillation methods. Most of these methods are based on meta-learning approaches that involve a high computational cost and memory footprint at the training time, so they are limited in the depth of the model. Moreover, they need to do a few gradient descent steps in constructing the model. Nonetheless, the number of required parameters in dataset distillation methods is proportional to the size of the input image. For instance, in distilling CIFAR-10 to 10 images only, we need to store at least $10 \times 32 \times 32 \times 3$ parameters. In order to be comparable with the SOTA Dataset Distillation methods, we use AlexNet (which is a modified version that is described in [48]) on CIFAR-10. For CIFAR-100 and tinyImageNet, we use depth-3 and depth-4 128-width ConvNet architectures described in [5], respectively. Note that some dataset distillation methods do not

Table 1. Comparison of our model with SOTA pruning methods, DPF [29], STR[23], LAMP[27], RiGL[11], and SuRP[19]. "Pr." denotes the pruning rate. Also, when the network is pruned, we have to keep two numbers for each weight: the weight itself and its position in the model. Note that we excluded the number of BatchNorm parameters in this table since that is constant for all the models. This number is 2,752 for Resnet-20 and 8,128 for ResNet-56.

Method	**Data**	**Arch.**	**# Params exc. BatchNorm**	**Accuracy**
Baseline (Pr. 0%)	C10	R20	269,722	88.92
DPF(Pr. 98.2%)	C10	R20	4,920×2	41.86
RiGL(Pr. 99.62%)	C10	R20	1026×2	50.9
LAMP(Pr. 99.62%)	C10	R20	1026×2	51.24
SuRP (Pr. 99.62%)	C10	R20	1026×2	54.22
STR (Pr. 95.5%)	C10	R20	12,238×2	75.99
Ours	**C10**	**R20**	**1,000**	**64.59**
Ours	**C10**	**R20**	**10,000**	**81.48**
Baseline (Pr. 0%)	C10	R56	853,018	91.64
DPF (Pr. 98.43%)	C10	R56	13,414×2	47.66
SuRP (Pr. 98.73%)	C10	R56	10,834×2	66.65
STR (Pr. 98.4%)	C10	R56	13,312×2	67.77
Ours	**C10**	**R56**	**5,000**	**76.87**
Baseline (Pr. 0%)	C100	R20	275,572	60.84
DPF (Pr. 96.13%)	C100	R20	10,770×2	12.25
SuRP (Pr. 97.48%)	C100	R20	6,797×2	14.46
STR (Pr. 96.12%)	C100	R20	10,673×2	13.18
Ours	**C100**	**R20**	**5,000**	**32.33**
Baseline (Pr. 0%)	C100	R56	858,868	64.32
DPF (Pr. 97.8%)	C100	R56	19,264×2	19.11
SuRP (Pr. 98.72%)	C100	R56	10,919×2	14.59
STR (Pr. 97.8%)	C100	R56	18,881×2	25.98
Ours	**C100**	**R56**	**5,000**	**32.97**

Table 2. Comparison with model distillation. PRANC outperforms a LeNet distilled from ResNet-18 on CIFAR-10. 2,752 and 8,128 are the number of BatchNorm parameters that we exclude from the coefficients but need to consider them as parameters.

Method	**Arch.**	**# Params**	**Acc.**
Distilled from R18	LeNet	62,006	74.1%
Ours	**R56**	**5,000 + (8,128)**	**76.87%**
Ours	**R20**	**10,000 + (2,752)**	**81.48%**

require a 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 as the central part of our idea is not prohibitive.

Table 3. Comparison with dataset distillation 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. "Trained model" is the upper bound of our method since one can optimize all weights and transmit/store them. Our method outperforms the baselines with a large margin and a much fewer parameters.

Method	Task	Arch	# Params	Acc.
Trained model	C10	AlexNet	1,756,426	84.8
FDD [4]	C10	AlexNet	397,000	43.2
SLDD [41]	C10	AlexNet	308,200	60.0
DD [48]	C10	AlexNet	307,200	54.0
DM [52]	C10	AlexNet	30,720	26.0
DSA [51]	C10	AlexNet	30,720	28.8
DC [53]	C10	AlexNet	30,720	28.3
CAFE [47]	C10	AlexNet	30,720	30.3
CAFE+DSA [47]	C10	AlexNet	30,720	31.6
DDMT [5]	C10	AlexNet	30,720	46.3
Ours	C10	AlexNet	**17,000**	**76.69**
Trained model	C100	3-128-Conv	504,420	56.2
FDD [4]	C100	3-128-Conv	317,200	11.5
DM [52]	C100	3-128-Conv	307,200	11.4
DSA [51]	C100	3-128-Conv	307,200	13.9
DC [53]	C100	3-128-Conv	307,200	12.8
CAFE+DSA [47]	C100	3-128-Conv	307,200	14.0
DDMT [5]	C100	3-128-Conv	307,200	24.3
Ours	C100	3-128-Conv	**15,000**	**25.57**
Trained model	tinyIN	4-128-Conv	857,160	37.6
DM [52]	tinyIN	4-128-Conv	2,457,600	3.9
DDMT [5]	tinyIN	4-128-Conv	2,457,600	8.8
Ours	tinyIN	4-128-Conv	**15,000**	**12.02**

5.4. Large-Scale dataset and models:

So far, we have provided evidence that our method can outperform recent works in dataset distillation, model pruning, and model compression in terms of the number of parameters vs. accuracy. 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 models. We evaluate our method on ImageNet100 with ResNet-18 architecture. Table 4 shows the results. Our method achieves 61.08% accuracy with less than 1% parameters, while the standard ResNet-18 model achieves 82.1% accuracy with more than 11M parameters. Since ResNet-18 is a huge model, we use one of the unique capabilities of PRANC, which is creating the network on the fly. We only used a single NVIDIA 3090 GPU and trained our model for 200 epochs, using Adam optimizer and step scheduler with $\gamma = 0.5$ for every 50 epochs and an initial learning rate of 0.001. Also, we distributed our budget of α vector across the layers, *i.e.,* we used 4,000 coefficients for each layer of the convolutional encoder and 20,000 coefficients for our classifier (giving us a total of 100,000 coefficients).

Table 4. Result of our method on ImageNet-100 dataset and ResNet-18. 19,200 is the total number of parameters of all BatchNorm layers in our model

Method	# Params	Acc.
trained model	11,227,812	**82.1%**
HashedNet [6]	110,000 + (19,200)	52.96%
Ours	**100,000 + (19,200)**	**61.08%**
Ours	**200,000 + (19,200)**	**67.28%**

5.5. Ablation study:

In PRANC, k, the number of basis models, is a hyperparameter. One question is: "How will k affect the performance?" Moreover, it is arguable that "why do we try to find a linear combination that is accurate on the task? why not try to regress an entire already trained network?" Also, "Can k be more important than the architecture? *i.e.,* can we use large k with a small network and still get high accuracy?" In this section, we answer these questions.

Sensitivity to seed: Since one of the applications of PRANC is in federated learning, it is worth discussing its pseudo-encryption ability. We experimented with changing the seed at the reconstruction time. On CIFAR-10 with AlexNet, with a minor change in seed, the accuracy of the reconstructed model dropped from **74.0%** to **9.4%**, which is close to chance. This is expected as the new basis models are not correlated with the ones that are used in training. This fact means that the seed can act as a mutual key between the agents and even if α is intercepted, reconstructing the model is nearly impossible. Therefore, in federated learning applications which deal with safe communication of deep learning models, PRANC can be used as both **compression** and **encryption** method.

Regressing θ^* directly: 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 optimal 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 5 shows that the accuracy of this baseline using 10,000 parameters is close to chance.

Effect of varying k vs. architecture: We perform an ablation study to understand the effect of the number of basis models, k vs. the architecture of the model. One can argue that sometimes it is better to design a better architecture rather than increasing the number of αs. Here, we change k

Table 5. Results of regressing a pretrained model using 10,000 basis models. C10 and C100 denote CIFAR-10 and CIFAR-100 and tinyIN denotes tiny ImageNet datasets

Dataset	Architecture	Full Acc.	Regress. Acc.
C10	AlexNet	84.8 %	10.0%
C10	LeNet [24]	73.5%	12.74%
C100	3-128-Conv	56.2 %	1.14%
C100	AlexNet	50.7%	1.0%
tinyIN	4-128-Conv	37.6 %	0.5%

from 1,000 to 20,000 for LeNet, AlexNet, ResNet-20, and ResNe-56 on CIFAR-10 and plot the accuracy in Figure 3. All experiments have been done in the same setup with 400 epochs with Adam optimizer. As expected, the accuracy increases as we increase the number of basis functions. However, as we can see, the architecture has more effect on the accuracy compared to k.

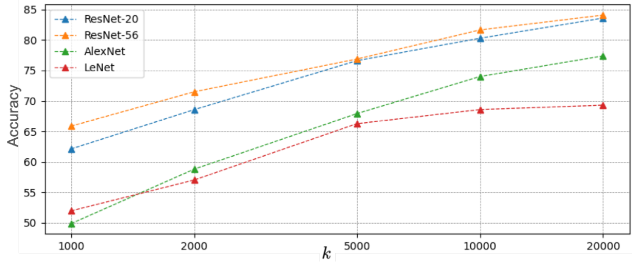


Figure 3. Illustration of impact of k in accuracy of different architectures on CIFAR-10. The accuracy improves by increasing the number of basis models. However, the architecture plays more critical role in the accuracy compared to the number of basis.

6. Experiments on image compression

As another application of PRANC, here we show that we can compress an image by compacting the implicit neural network (INR) that is overfitted to the image. The INR model inputs the coordinates of a pixel and outputs the color of that pixel. We will store or communicate the seed of the pseudo-random generator and α values only. We use [42] as our INR model. The INR is a 4-layer MLP (3 hidden layers) model with 512 neurons at each layer and a LayerNorm [1] after each layer. We encode the pixel coordinate in the input to 512 dimensions using Fourier mapping and use three neurons in the output for RGB images (the color values) and one neuron for X-ray images.

We use a half-precision floating point for αs to further reduce the storage cost. We use a different set of αs for each layer of the network.

To reduce the memory footprint, we divide the base network matrix $\hat{\theta}$ into smaller chunks and generate and then discard each chunk at the GPU for each iteration of the optimization. Similar to stochastic gradient descent, we randomly sample a subset of pixels to be optimized at each iteration, leading to faster convergence due to more frequent parameter updates.

We evaluate our model on three different datasets: **Kodak** dataset [22] that contains 24 non-compressed images of size 512 × 768, **CLIC-mobile** test set [44] that contains 178 high-resolution images (e.g., 1512 × 2016), and 64 randomly sampled images from **NIH Chest X-ray** dataset [49] that consists of $100,000$ de-identified chest X-rays images of size 1024×1024.

Baselines. We compare our method with the following hand-crafted image codecs: JEPG, JPEG2000, and WebP. Our goal is to show that PRANC is a general framework that performs well when simply applied to INR compression out of the box. Hence, we do not compare it with more advanced codecs like BPG and VTM since they are highly engineered and include components like entropy coding. Those results are presented in the supplementary material. We also compare with learning-based image compression methods in the supplementary (BMS, MBT, and CST). Note that, unlike PRANC, these methods require a large dataset of images to pre-train their auto-encoder. This limits the applications and also may degrade the results for out-of-distribution data points. For instance, unlike PRANC, models that are trained on a training set may not be suitable for medical data since they may not be truthful to abnormalities specific to patients not represented in the training data.

COIN [10] is probably the closest to our method which trains an INR using SIREN [39] and stores all the parameters. Since COIN does not use Fourier mapping, for a fair comparison, we produce a similar baseline, called 'trained INR,' using our MLP network described above without the PRANC framework. For the trained INR, we reduce the width of the network to match the compression rate of our method and use half-precision floating points.

Results. We evaluate our model with two metrics: PSNR and MS-SSIM. Note that we fix the network architecture and vary the number of α values to get different bit-per-pixel (bpp) values for each image. We report the results for the Kodak dataset in Figure 4 and CLIC-mobile dataset on Table 6. Our method outperforms JPEG and INR. We report the results of the chest X-ray in Table 7. An example image is shown in Figure 6 for the Kodac dataset and in Figure 7 for the X-ray dataset.

Ablation. Since we reconstruct the model weights with α values, one can vary the size of the architecture without changing the number of parameters (α values), hence, in PRANC, we can increase both width and depth without changing the bit-rate. We keep the number of parameters $k = 102K$ and vary both the network width and depth of the MLP model. Results on the Kodak dataset are shown in Table 8. Interestingly, we can improve the performance by increasing the model depth while keeping the number of

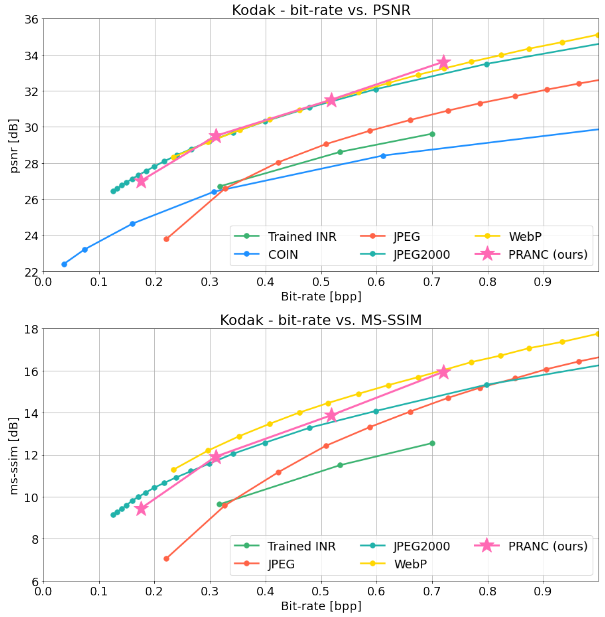


Figure 4. **Kodak Dataset Image Compression:** Our method outperforms JPEG and 'trained INR' on both PSNR and MS-SSIM evaluations at various bitrates. Note that, unlike the other baselines, our method is learned on a single image and is not handcrafted, except for the architecture of the INR model, which is a simple MLP.

Table 6. **CLIC-mobile Dataset Image Compression:** PRANC outperforms JPEG and JPEG2000 with smaller bpp on this dataset.

Model	bpp	PSNR	MS-SSIM
WebP	0.185	30.07	0.940
JPEG2000	0.126	29.40	0.918
JPEG	0.195	24.82	0.836
Trained INR	0.125	26.93	0.864
PRANC (ours)	0.119	29.71	0.920

Table 7. **Chest X-ray Dataset Image compression:** We compare PRANC with JPEG for X-ray images. Our method is better than JPEG with a lower bpp. Since unlike auto-encoders, PRANC does not use any population-based training, it may be better suited for medical images since it may preserve out-of-distribution artifacts which are important for diagnosis purposes. However, we leave studying this for future work.

Model	PRANC	JPEG
bpp	0.152	0.168
PSNR	36.28	34.25
MS-SSIM	0.972	0.921

Table 8. **Effect of increasing width/depth of the model:** We can increase both the depth and width of the MLP model without changing the number of parameters. When changing the depth, we redistribute the number of α values uniformly among layers to keep the total number constant. More details are in the Supp.

Width	128	256	512	1024
PSNR	30.00	32.05	31.5	31.32
MS-SSIM	0.937	0.963	0.959	0.961

Depth	3	4	5
PSNR	30.78	31.5	32.38
MS-SSIM	0.978	0.959	0.965

learnable parameters constant.

Sorting α values. In Figure 5, we show reconstructed images with a subset of α values with the largest absolute values. Since we have a different set of αs for each layer of MLP in image compression, we sort absolute values and select the top $p\%$ of each layer with higher values. We vary p and visualize the reconstructed images for each p value. In the supplementary material, for the image classification setting, we show that in reconstructing the ResNet model using a partial set of α values, choosing larger $|\alpha|$ values leads to much better accuracy compared to choosing a random subset.

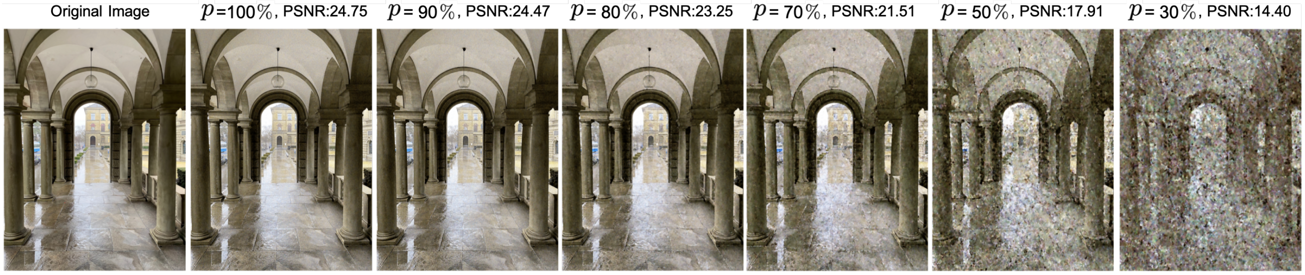
Implementation Details. For each image, we train α values with $10k$ iterations on Kodak and $5k$ iterations on CLIC-mobile dataset. Each iteration processes 25% of the pixels of the image sampled randomly. Note that increasing the number of iterations cannot damage the model since the goal is to overfit to the image and generalization is not an issue. We use PyTorch Adam [21] optimizer with an initial learning rate of $1e-3$ and a Cosine scheduler. More details about the number of α values per layer for each bpp are in the supplementary material.

7. Future directions

PRANC can enable multiple future directions:

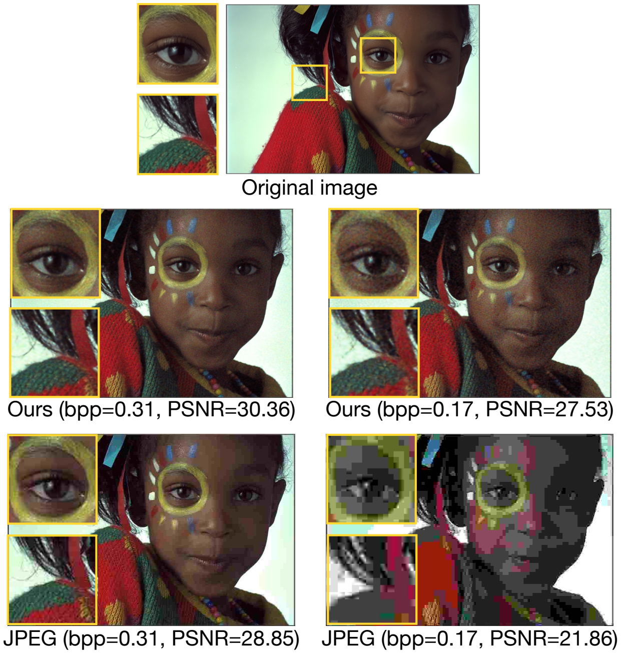
Generative models for memory-replay: Our method can be used to compact a generative model (e.g., GANs or diffusion models), where the α parameters may be stored in the agent or sent to another agent. Then, any agent can reconstruct the model in the future and draw samples from it that are similar to 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 one may learn them in decreasing importance order as a future work.



Original Image | p=100%, PSNR:24.75 | p= 90%, PSNR:24.47 | p= 80%, PSNR:23.25 | p= 70%, PSNR:21.51 | p= 50%, PSNR:17.91 | p= 30%, PSNR:14.40

Figure 5. **Effect of keeping $p\%$ of basis models with the highest absolute α values.** We get reasonable images with a smaller subset of basis models. One can reconstruct an approximate image upon receiving a partial set of α values, similar to progressive JPEG.



Original image

Ours (bpp=0.31, PSNR=30.36)

Ours (bpp=0.17, PSNR=27.53)

JPEG (bpp=0.31, PSNR=28.85)

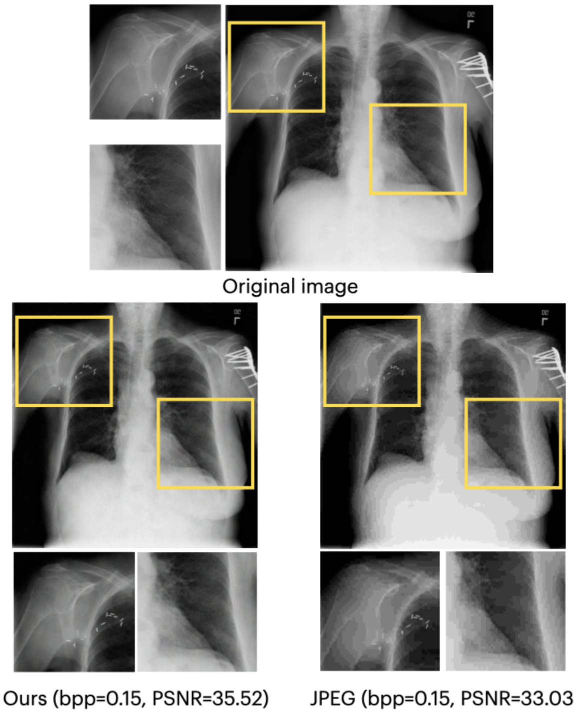
JPEG (bpp=0.17, PSNR=21.86)

Figure 6. **Kodak visualization.** We compare PRANC with JPEG on image 15 of the Kodak dataset. See Supp. for more examples.

8. Conclusion

We introduced a simple yet 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 small memory footprint at the learning or reconstruction stages. We perform extensive experiments on multiple image classification datasets with multiple architectures and also on image compression settings and show that our method achieves better accuracy with fewer 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: As discussed, some model parameters, e.g., BatchNorm layers, cannot be easily reparameterized 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 and included them in our budget. Lastly, PRANC is computationally expensive for a large number of basis, so is currently not suitable for training very large models.



Original image

Ours (bpp=0.15, PSNR=35.52)

JPEG (bpp=0.15, PSNR=33.03)

Figure 7. **Chest X-ray visualization**. We compare PRANC and JPEG on a Chest X-ray image. See Supp. for more examples.

Acknowledgement: This work was partially supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR00112190135 and HR00112090023, the United States Air Force under Contract No. FA8750-19-C-0098, and funding from NSF grants 1845216 and 1920079. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the United States Air Force, DARPA, or NSF.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 7

[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*, pages 7120–7129, 2017. 3

[3] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018. 3

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

[5] 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. 3, 5, 6

[6] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294. PMLR, 2015. 1, 2, 6

[7] 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

[8] Yaim Cooper. Global minima of overparameterized neural networks. *SIAM Journal on Mathematics of Data Science*, 3(2):676–691, 2021. 2

[9] JCMSA Djelouah and Christopher Schroers. Content adaptive optimization for neural image compression. In *Proceedings of the CVPR*, 2019. 3

[10] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 3, 7

[11] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020. 5

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

[13] Tiansheng Guo, Jing Wang, Ze Cui, Yihui Feng, Yunying Ge, and Bo Bai. Variable rate image compression with content adaptive optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 122–123, 2020. 3

[14] 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*, pages 1580–1589, 2020. 3

[15] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 3

[16] Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. *arXiv preprint arXiv:1810.00440*, 2018. 3

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

[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

[19] Berivan Isik, Tsachy Weissman, and Albert No. An information-theoretic justification for model pruning. In *International Conference on Artificial Intelligence and Statistics*, pages 3821–3846. PMLR, 2022. 3, 5

[20] 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

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 8

[22] Kodak. Kodak Dataset. http://r0k.us/graphics/kodak/, 1991. 7

[23] 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. 3, 5

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

[25] Jooyoung Lee, Seunghyun Cho, and Seung-Kwon Beack. Context-adaptive entropy model for end-to-end optimized image compression. *arXiv preprint arXiv:1809.10452*, 2018. 3

[26] 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*, pages 6448–6457, 2021. 1, 3

[27] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020. 5

[28] 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*, pages 6438–6447, 2021. 3

[29] 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, 5

[30] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *Applied and Computational Harmonic Analysis*, 59:85–116, 2022. 2

[31] 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*, pages 6682–6691. PMLR, 2020. 2

[32] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018. 3

[33] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. 2

[34] Quynh Nguyen, Mahesh Chandra Mukkamala, and Matthias Hein. On the loss landscape of a class of deep neural networks with no bad local valleys. In *International Conference on Learning Representations*, 2019. 2

[35] 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*, pages 11893–11902, 2020. 2

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

[37] Brandon Reagan, Udit Gupta, Bob Adolf, Michael Mitzenmacher, Alexander Rush, Gu-Yeon Wei, and David Brooks. Weightless: Lossy weight encoding for deep neural network compression. In *International Conference on Machine Learning*, pages 4324–4333. PMLR, 2018. 3

[38] 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

[39] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020. 7

[40] Christopher Subia-Waud and Srinandan Dasmahapatra. Weight fixing networks. In *European Conference on Computer Vision*, pages 415–431. Springer, 2022. 1, 2

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

[42] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. 5, 7

[43] 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

[44] George Toderici, Wenzhe Shi, Radu Timofte, Lucas Theis, Johannes Balle, Eirikur Agustsson, Nick Johnston, and Fabian Mentzer. Workshop and challenge on learned image compression (clic2020), 2020. 7

[45] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017. 3

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

[47] 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

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

[49] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017. 7

[50] 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

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

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

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