ENHANCING ACCURACY AND PARAMETER EFFICIENCY OF NEURAL REPRESENTATIONS FOR NETWORK PARAMETERIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we investigate the fundamental trade-off regarding accuracy and parameter efficiency in neural network weight parameterization using predictor networks. We present a surprising finding where the predicted model not only matches but also surpasses the original model's performance through the reconstruction objective (MSE loss) alone. Remarkably this improvement can be compound incrementally over multiple rounds of reconstruction. Moreover, we extensively explore the underlying factors for improving weight reconstruction under parameter-efficiency constraints and propose a novel training scheme that decouples the reconstruction objective from auxiliary objectives such as knowledge distillation that leads to significant improvements compared to state-of-the-art approaches. Finally, these results pave the way for more practical scenarios, where one needs to achieve improvements in both model accuracy and predictor network parameter-efficiency simultaneously.

1 INTRODUCTION

Recently, neural network weight space exploration and manipulation have gained an increase in popularity as an additional step to improve model performance after traditional model training, fine-tuning or compression. Examples of these methods range from weight manipulation strategies such as weight merging to improve model performance without fine-tuning (Matena & Raffel, 2022) to weight generation approaches that directly predict network parameters like Neural Representation for Neural Networks (NeRN) (Ashkenazi et al., 2022) or diffusion-based D2NWG (Soro et al., 2024). NeRN extends the concept of neural representations popularized in Neural Radiance Fields (NeRF) (Mildenhall et al., 2021) to the domain of neural network weight space learning that has increasingly become an essential foundation for optimizing model performance in various applications. This innovative approach shifts from representing data (e.g., images or videos) to representing neural network parameters as functions, introducing a structured, coordinate-driven framework for weight generation. Specifically, NeRN employs implicit neural representation (INRs) (Sitzmann et al., 2020) to map kernel or layer indices directly to their corresponding weights, offering a compact and continuous functional encoding of network parameters. By transforming weights into a function, NeRN opens the door to exciting possibilities such as dynamic parameter generation, model storage, and novel forms of network adaptation. Despite its conceptual appeal, NeRN has practical limitations: reconstructed weights fail to match the original model's performance, and its potential usability in applications like model compression, fine-tuning, or transfer learning remains under-explored.

In this work, we address these limitations by advancing the understanding and improving NeRN-style weight parametrization through novel progressive-training and decoupled training strategies, enabling effective reparametrization. We demonstrate that better-performing models can be obtained through a weight reconstruction-only objective and these improvements can be compounded over multiple repetitions. By iterating over the reconstruction process multiple times, where each round builds upon the previously reconstructed weights, we establish an a recursive setup with nested improvements in each prediction round. NeRN utilizes a multi-objective loss function where the reconstruction loss serves as the primary objective, while distillation losses (feature-level and logit-level) play a critical complementary role in improving reconstruction fidelity. These distillation losses help stabilize the learning process and enable the predictor network to better approximate the original model's behavior. However, we identified a key challenge: the multi-objective loss function leads to contradictory training signals during the reconstruction process, resulting in limited performance gains. Therefore, we propose a new training scheme that decouples the learning objectives into two phases: a reconstruction phase and a distillation phase, ensuring that each learning objective has the desired impact. Remarkably, our approach results in significant improvements compared to NeRN for reconstruction fidelity and network compression. Moreover, the proposed separation provides greater flexibility in the distillation step by allowing the use of powerful networks to be involved during the weight refinement. All these improvements and insights led to several usage scenarios that were not possible or practical before, for example, obtaining a better-performing model through a predictor network that is much smaller than the original network, or iterative improving a given model through the reconstruction process. The proposed approach provides a unique, even surprising, perspective for achieving model performance improvement that is different from existing weight manipulation approaches such as stochastic weight averaging (Guo et al., 2023a; Izmailov et al., 2018). We also achieve storage compression via the predictor network, which is orthogonal and composable with existing model quantization, model pruning (Lee et al., 2019; Liu et al., 2018; Gao et al., 2021; Wang et al., 2021; He & Xiao, 2023) and knowledge distillation approaches (Chen et al., 2020; Gou et al., 2021; Chen et al., 2017; Beyer et al., 2022).

2 PREDICTING MODEL WEIGHTS USING NEURAL REPRESENTATIONS

There is a growing interest in predicting the weights of pre-trained models not only to enhance the memory efficiency of model storage but also to improve the throughput of model inference. Existing solutions range from building high-fidelity auxiliary models (Knyazev et al., 2023) to learning parameter-efficient approximators (Guo et al., 2023b). However, in this paper, we focus on neural representations learning based on their flexibility and potential for producing effective, yet parameter-efficient, approximations for deep networks. The NeRN framework first explored the idea of training INRs and demonstrated its utility in compressing CNNs. At its core, NeRN uses a 5-layer multilayer perceptron (MLP) G_ϕ with fixed hidden layer size to learn a mapping from an input tuple (layer ℓ, filter f, channel c) to the corresponding $k \times k$ kernel in the original CNN model F_θ. Note that the output size of the MLP also remains fixed at the largest kernel size and smaller kernels in the CNN are sampled from the middle while fully-connected or normalization layers are excluded based on their comparatively negligible parameter size. Since there is no inherent smoothness in the ordering of filters in a CNN, permutation strategies are introduced to rearrange filters in the original model based on similarity, ensuring stable training of NeRN.

The central component of NeRN training is the reconstruction loss aimed at quantifying the disparity between the original network weights and those recovered using the predictor. Regardless of the capacity of G_ϕ, one might expect that the reconstruction loss should reliably converge to a meaningful approximation. However, in practice, it has been found that NeRN is prone to training instabilities when the auxiliary model is smaller than the original network. To mitigate these issues, the authors introduced additional loss terms inspired by existing knowledge distillation methods. Note that, while the reconstruction loss does not require access to training data, the distillation loss does, making it impractical for scenarios where access to the training data is not available. The overall objective function for NeRN can be expressed as

$$\mathcal{L}_{objective} = \mathcal{L}_{recon} + \alpha \mathcal{L}_{KD} + \beta \mathcal{L}_{FMD}, \text{with}$$

$$\mathcal{L}_{recon} = \frac{1}{|\mathbf{W}|} \|\mathbf{W} - \hat{\mathbf{W}}\|_2, \;\; \mathcal{L}_{FMD} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \sum_{l} \|\mathbf{a}_i^l - \hat{\mathbf{a}}_i^l\|_2, \;\; \mathcal{L}_{KD} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} KL(\mathbf{a}_i^{\text{out}}, \hat{\mathbf{a}}_i^{\text{out}}), \tag{1}$$

where $\mathbf{W} = [\mathbf{w}^0, \mathbf{w}^1, ..., \mathbf{w}^L]$, represents a list of convolutional weight vectors for layer ℓ in the original network, and $\hat{\mathbf{W}}$ denotes the corresponding weights of the reconstructed weights. The terms $\mathbf{a}_i^\ell$ and $\hat{\mathbf{a}}_i^\ell$ denote the L_2 normalized feature maps generated from the i-th sample in the minibatch $\mathcal{B}$ at layer ℓ for the original and reconstructed networks respectively. Additionally, the logit distillation loss $\mathcal{L}_{KD}$ employs the Kullback-Leibler divergence to compare the output logits $\mathbf{a}_i^{\text{out}}$ and $\hat{\mathbf{a}}_i^{\text{out}}$ from the original and reconstructed networks for each sample i in the minibatch $\mathcal{B}$.

3 PROPOSED APPROACH

In this section, we take a closer look at learning neural representations for pre-trained neural networks. Specifically, we explore the role of different training objectives in realizing an effective parameterization. Based on our findings, we propose a new training scheme that circumvents the undesirable trade-off between accuracy and model compression rate, and simultaneously improve on both aspects.

3.1 IS THE RECONSTRUCTION OBJECTIVE ALL YOU NEED?

A well-known limitation of existing approaches used for weight prediction is that they trade off accuracy to achieve parameter efficiency. This non-trivial compromise in model performance can be a critical bottleneck for practical applications. While existing approaches attempt to recover the lost performance through the use of additional objectives, e.g., distillation as in NeRN, they lead to increased reconstruction errors, albeit providing improvements in the accuracy. As this seems counter-intuitive, it naturally raises the question: What is the relationship between reconstruction error and expected model performance?

If we assume that reconstruction error, e.g., mean-squared error (MSE), is indeed an indication for performance, a straightforward strategy to improve performance would be to increase the capacity of the predictor network, allowing it to overfit to the original model weights. To this end, we first empirically analyze how well we can recover the original network's performance, as we continually reduce the reconstruction error by increasing the predictor network capacity. Note that in this analysis, we are not concerned about parameter efficiency, and the neural representations are trained solely with the reconstruction objective. Interestingly, by using only the reconstruction loss and increasing the predictor network capacity, we empirically find weight parameterizations with non-zero reconstruction errors that not only recover the true performance but even surpass it as shown in Figure 1.

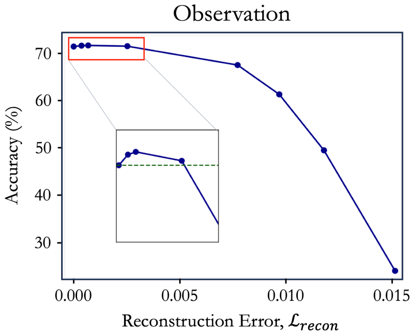


Figure 1: The first dot is the original network accuracy, the rest show results from predictors with different hidden layer sizes in descending order from left to right: 750, 680, 510, 360, 320, 280, and 220.

How does reconstruction-only objective lead to better networks? A well-known property of the MSE loss is that it tends to have a smoothing effect on the reconstructed weights (Domingos, 2012). We hypothesize that the performance improvement is linked to such an effect. To quantify this, we compare the weights of the reconstructed network (based on neural representations learned using only the reconstruction objective) with those of the original network, in terms of changes in the ratio of singular values of their weight matrices measured as follows: Let $\mathbf{M}$ be the weight matrix of a given convolutional layer, shaped as $\mathbf{M} \in \mathbb{R}^{n \times m}$. Here n represents the number of output channels (c_{out}), and m denotes the product of the number of input channels (c_{in}) and the size of the filters ($k \cdot k$). Using singular value decomposition (SVD) on this matrix, we obtain $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ is a diagonal matrix with singular values $\mathbf{s} = [\sigma_1, \sigma_2, \ldots, \sigma_k]$ on the diagonal, sorted in descending order, and $k = \min(n, m)$. The S_{ratio} is then calculated as $S_{ratio} = \sum_{i=1}^{\lfloor k/2 \rfloor} \sigma_i^2 / \sum_{i=1}^{k} \sigma_i^2$. This ratio measures the proportion of the total variance (energy) of the matrix $\mathbf{M}$ that is captured by the first half of the singular values. A higher ratio indicates that more variance can be explained by fewer dominant components that tend to be lower-frequency in nature, whereas the smaller singular values are often more related to noise (e.g., as demonstrated in SVD-based image denoising methods (Guo et al., 2015)). As illustrated in Figure 2, a clear trend emerges upon close inspection of the layer-wise S_{ratio} differences between the reconstructed weights and the original weights. We see the reconstructed weights have a higher S_{ratio}, particularly in the later layers, which indicates that a smoothing effect has been applied to the reconstructed weights that align with our hypothesis regarding reconstruction loss.

To better understand the relationship between weight components and model performance, we find interesting connections to prior works that explore neural networks' simplicity or spectral bias (Cao et al., 2019; Huh et al., 2021; Rahaman et al., 2019; Yoshida & Miyato, 2017), where the model tends to capture lower frequency components. In many cases, intentionally inducing reduction of high-frequency information through truncation/pruning (Chen et al., 2024; Sharma et al., 2023), weight smoothing/averaging (Cheng et al., 2023; Jean & Wang, 1994) or even activation regularization during training (Khan et al., 2019; Bu et al., 2023) can improve model's generalization performance. To dive deeper into the mechanism behind the observed performance improvement and verify our hypothesis, we have carried out an experiment to explicitly test the relationship between weight frequency manipulation and performance by applying low-pass filters on weight, which leads to model test performance improvement, see Appendix A for details.

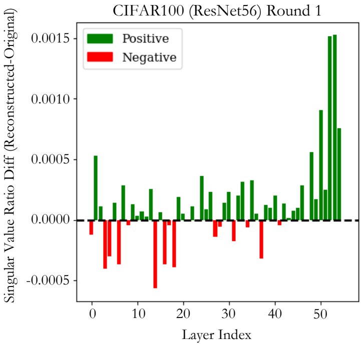
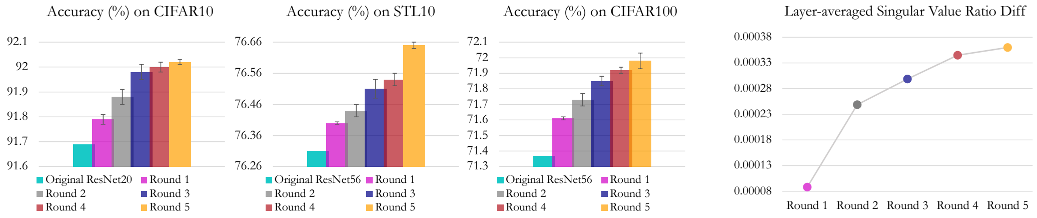


Figure 2: Layer-wise difference in singular value ratios between the reconstructed network and the original network.

Building upon our observation of model improvement induced by the weight reconstruction objective, we ask: *Can multiple rounds of neural representation learning further improve the performance of the reconstructed network?* To this end, we extend our previous experiment by training multiple generations of network parameterizations, where the first-round predictor reconstructs the original network, the second-round predictor recovers the reconstructed network from round 1, and so on.



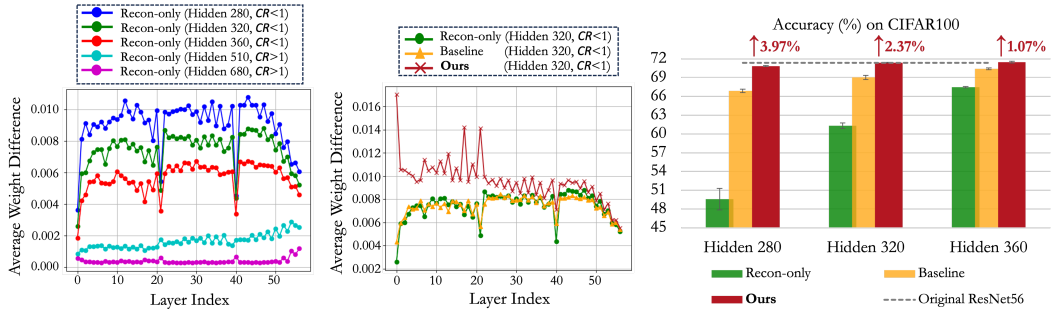
(a) (b)

Figure 3: (a) Incrementally improving performance through progressive training. The colored bar in each round represents the average accuracy across three runs. (b) Layer-averaged difference in singular value ratios increases as the rounds progress in ResNet56 on CIFAR100.

Such a process resembles a recursive setup where the new predictor is built on top of the previously reconstructed weights. This simple procedure leads to further improvements over the original network's performance, albeit producing relatively higher reconstruction errors due to smoothing. As shown in Figure 3(a), this progressive training produces consistent improvements in accuracy, surpassing the original network's performance across all architectures and datasets. We further investigate the singular value ratio (S_{ratio}) analysis at each round. Figure 3(b) presents the layer-averaged difference in S_{ratio} for the last half of the layers in ResNet56 on CIFAR100, i.e., $\frac{2}{L}\sum_{l=n/2}^{L}\left(S_{ratio}^{\text{round}}[l] - S_{ratio}^{\text{original}}[l]\right)$, where l denotes each layer. Compared to the original network weights, S_{ratio} increases with each round of reconstruction and levels off at 5. Notably, while the slope remains positive across rounds, the slope becomes less steep. This suggests that the weight smoothing effect diminishes, leading to only limited improvement. Once weights are sufficiently smoothed, we do not witness further improvements by using additional rounds of training (performance does not drop either).

3.2 DISTILLATION IMPROVES COMPRESSION, BUT ONLY WITH LOSS DECOUPLING

So far, we inspected the behavior of the reconstruction loss, and demonstrated its surprising efficacy in enhancing model performance. Despite the observed performance improvement, we did not take parameter-efficiency into account for our analysis. However, in practice, an important motivation for using weight prediction networks is to achieve signification reduction in memory requirements for model storage, while not trading-off the performance unreasonably. While NeRN originally leveraged with such a compression objective, their accuracy trade-off makes them a less preferred choice over other model compression (or reduction) strategies in practice. In this section, we show how the compression capability of neural representations can be enhanced. To this end, we take a closer look at the widely adopted distillation objective and how it interacts with the reconstruction loss. By doing so, we address the critical need to strike a balance between model complexity and efficiency, paving the way for more practical and resource-efficient neural networks.



(a) (b) (c)

Figure 4: (a) Average weight difference between predicted weights by Recon-only and original weights with varying hidden sizes. (b) Comparison among Recon-only ($\mathcal{L}_{recon}$), NeRN ($\mathcal{L}_{recon} + \mathcal{L}_{KD} + \mathcal{L}_{FMD}$), and ours (only $\mathcal{L}_{KD}$ in the second phase). (c) Evaluation of the reconstruction performance for each method. ↑ represents our performance improvement over the NeRN.

To begin with, we analyze the layer-wise weight differences between the reconstructed and original networks at varying predictor network sizes (Figure 4(a)). Note, we use the term "*CR*" to denote the ratio of the learnable size of the predictor $S(P)$ to that of the original network $S(O)$. For instance, if the predictor network has Q learnable parameters in order to recover an original network with P parameters, then *CR* $\times 100 = (Q/P) \times 100\%$. As expected, smaller-sized predictor networks (i.e., higher compression) exhibit relatively large gaps and this can be attributed to the insufficient representation power. On the other hand, increasing the capacity of the predictor network leads to improved reconstruction performance, as depicted by the green bar in Figure 4(c). However, insights from Section 3.1 indicate that, unless we increase the predictor capacity even further and perform progressive training, the reconstruction-only training cannot match the true performance. Hence, to recover the lost performance while also enabling parameter-reduction (i.e., *CR* < 1), NeRN incorporates an additional distillation objective ($\mathcal{L}_{KD} + \mathcal{L}_{FMD}$) during training. As illustrated by the orange bar in Figure 4(c), the predictor network's performance can be significantly improved with the guidance of this distillation process. This can be attributed to the fact that the arbitrary perturbations in the reconstructed network induced by the distillation losses can make non-trivial changes to the decision rules, thus impacting the generalization performance of the network. Hence, additional guidance in terms of the prediction probabilities provides valuable task-relevant information.

While the original NeRN approach has proven effective, we observe that the role of the distillation objective is to primarily supplement the reconstruction loss. For instance, Figure 4(b) illustrates the layer-wise discrepancies between the reconstructed weights in the Recon-only and the NeRN methods compared to the original weights for the case of Hidden 320 (number of neurons in each layer of the MLP for neural representations). It is apparent that the training process of NeRN appears to be predominantly driven by the reconstruction loss, which limits effectiveness at high compression factors. This indicates that a limited predictor capacity (*CR* < 1) cannot accurately recover the original weights. While one can further emphasize distillation terms in equation 1 by increasing α and β, we find that it leads to training instabilities and the resulting network is far inferior (as shown in Figure 6). This highlights the complementary nature of the two objectives, as well as the challenges involved in combining them in practice.

To address this critical limitation, we propose to employ the two objectives in distinct training stages. Initially, we train the predictor network solely based on the reconstruction objective (Recon-only) and optionally perform progressive training when the predictor network capacity is high. In the second phase, we fine-tune the predictor stage 1 using only the distillation objective, $\mathcal{L}_{KD}$. As shown in Figure 4(b), this allows the predictor to explore solutions that can in principle be different from the original network, but adhere to similar decision boundaries. Interestingly, we find that this two-stage optimization leads to significant differences in the early layers of the network, while still matching the later layers. This is intuitive, as it is well known that the decision rules typically emerge in the later layers of a deep network. While the larger differences in the early layers may seemingly compromise reconstruction fidelity, this separate training strategy facilitates a more effective integration of distillation into the network parametrization, as evidenced by significant improvements (red bars in Figure 4(c)). As we show later, this decoupling of the training objectives not only demonstrates greater resilience to variations in the predictor network size, but also recovers or even surpasses the performance of the original network with predictors that are $> 40\%$ smaller than the original network, which NeRN cannot achieve.

3.3 IMPROVING COMPRESSION-PERFORMANCE TRADE-OFF VIA STRONG TEACHERS

In Section 3.1, and 3.2, we considered the performance or the compression objective independently. However, a natural next question is whether one can improve the compression vs. performance trade-off, and obtain additional improvements in both aspects. The proposed decoupled training offers flexibility in the second phase after the initial reconstruction objective is accomplished. One particular benefit of such a decoupling is that it enables the use of other high-performing models for guiding the distillation phase. We argue that ***by leveraging guidance from a high-performing teacher, one can further improve the efficacy of decoupled training, thereby improving on the performance-compression trade-off.*** In other words, through the proposed strategies, one can achieve non-trivial improvements to model accuracy for a fixed predictor network capacity, or easily push past the original network's performance via progressive reconstruction. This flexibility of our proposed approach goes beyond the decoupled training, as every component we have introduced can be combined with each other or with other methods (e.g., model quantization, pruning, and standard knowledge distillation). To help put everything together, we provide a summary of our findings and components of the proposed approach in Figure 5.

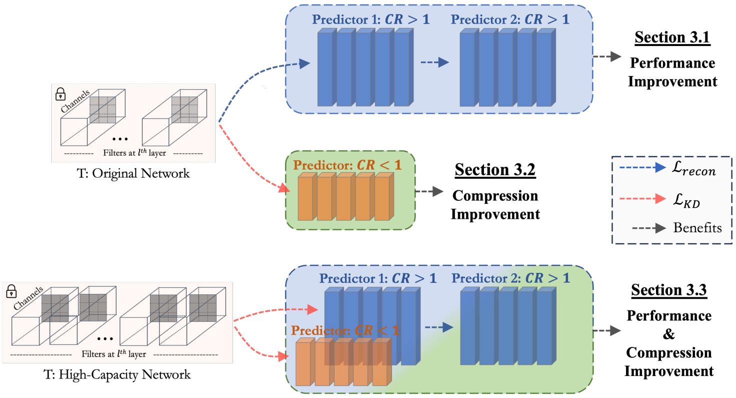


Figure 5: Summary of training schemes and their benefits: Section 3.1 explores reconstruction-only setups for performance improvement. Section 3.2 introduces decoupled training for better storage compression. Finally, we show how a high-capacity teacher enhances both compression and accuracy.

4 EXPERIMENTS

We present experiment results to support the observations made in Section 3 and highlight the practical usage scenarios made possible with our proposed approach. We use benchmark datasets, including CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), STL-10 (Coates et al., 2011), and ImageNet (Deng et al., 2009) and models based on ResNet architectures (He et al., 2016) . We use the test split for model evaluations, the exceptions are no split for the reconstruction task and training split for the knowledge distilation. Beyond evaluating the in-distribution performance of the reconstructed models, we also assess their robustness using out-of-distribution (OOD) datasets such as CIFAR10-C, CIFAR100-C, and ImageNet-R, and two popular adversarial attacks, FGSM and I-FGSM (Goodfellow et al., 2014). See detailed training setups in the Appendix G.

4.1 ENHANCING PERFORMANCE THROUGH PROGRESSIVE RECONSTRUCTION - $CR > 1$

Firstly, we discovered that the reconstructed model using only the reconstruction loss performs better in testing when weights are predicted by a large predictor ($CR > 1$). The reconstruction loss alone enabled the model to improve its performance slightly, with gains ranging from 0.1% to 0.3%. Interestingly, additional rounds of weight prediction further enhanced performance beyond the initial reconstructed model, ultimately achieving gains of up to 0.6%. Table 1 presents the results of the reconstructed models at each round. In *Round 1*, the model reconstructs the original network, and subsequent rounds predict the previous round's weights. This process continues progressively through multiple generations until performance improvements plateau. As each round progresses, the predicted weights deviate more from the original solution, as indicated by the increased reconstruction loss. Furthermore, the solution found in each round does not compromise on OOD generalization and adversarial robustness metrics. In a few cases, we also notice that non-trivial gains in robustness are possible. This suggests that our progressive training does not lead to undesirable overfitting behavior.

Table 1: Evaluation performance of large predictor networks via progressive reconstruction. The reported results are computed across three runs.

CIFAR10	Original ResNet20	Hidden 300 ($CR\!>\!1$)				
		Round 1	*Round 2*	*Round 3*	*Round 4*	*Round 5*
Accuracy (↑, %)	91.69%	**91.79**±**0.02**	**91.88**±**0.04**	**91.98**±**0.03**	**92.00**±**0.02**	**92.02**±**0.01**
$\mathcal{L}_{recon}$	-	0.00332	0.00414	0.00500	0.00547	0.00590
OOD (↑, %)	**70.49**	69.45±0.48	69.32±0.51	68.82±0.19	68.85±0.06	68.61±0.06
FGSM (↑, %)	76.41	**77.16**±**0.25**	77.03±0.12	77.08±0.13	77.08±0.02	77.07±0.08
I-FGSM (↑, %)	75.02	75.56±0.17	75.49±0.22	75.59±0.08	**75.69**±**0.01**	75.63±0.06

STL10	Original ResNet56	Hidden 680 ($CR\!>\!1$)				
		Round 1	*Round 2*	*Round 3*	*Round 4*	*Round 5*
Accuracy (↑, %)	76.31%	**76.40**±**0.004**	**76.44**±**0.02**	**76.51**±**0.03**	**76.54**±**0.02**	**76.65**±**0.01**
$\mathcal{L}_{recon}$	-	0.00120	0.00117	0.00125	0.00128	0.00166
FGSM (↑, %	39.28	**39.36**±**0.16**	39.27±0.10	39.30±0.05	39.32±0.06	39.19±0.02
I-FGSM (↑, %)	36.12	36.13±0.11	36.24±0.06	**36.26**±**0.13**	36.17±0.03	36.11±0.02

CIFAR100	Original ResNet56	Hidden 680 ($CR\!>\!1$)				
		Round 1	*Round 2*	*Round 3*	*Round 4*	*Round 5*
Accuracy (↑, %)	71.37	**71.61%**±**0.01**	**71.73%**±**0.04**	**71.85%**±**0.03**	**71.92%**±**0.02**	**71.98%**±**0.05**
$\mathcal{L}_{recon}$	-	0.00068	0.00082	0.00088	0.00095	0.0010
OOD (↑, %)	44.70	44.47±0.47	**44.75**±**0.31**	44.50±0.26	44.29±0.03	44.47±0.21
FGSM (↑, %)	44.69	44.83±0.26	44.80±0.21	45.02±0.11	**45.23**±**0.08**	45.18±0.17
I-FGSM (↑, %)	39.31	40.91±0.38	40.82±0.28	41.04±0.14	**41.21**±**0.08**	41.04±0.08

4.2 ACHIEVING GREATER COMPRESSION WITH DECOUPLED TRAINING - $CR\!<\!1$

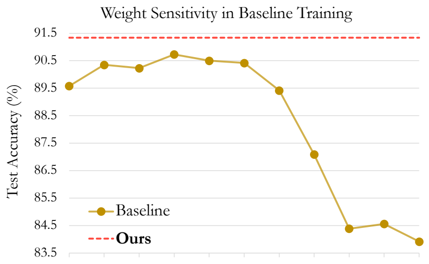


Figure 6: Impact of hyperparameters on model performance in NeRN training.

We identified that the NeRN training process appears to be predominantly driven by the reconstruction loss, thus the compression ability induced by $\mathcal{L}_{FMD}$ and $\mathcal{L}_{KD}$ might be limited in Section 3.2. Although increasing the distillation weights could improve compression, we found that it ends up with training instability and significantly inferior network performance ($\alpha, \beta = 1$). In Figure 6, we gradually vary the penalty of the distillation loss term while fixing the reconstruction term's weight at 1. When this parameter is aptly chosen, it is possible to improve over the NeRN; however, the sensitivity of this hyperparameter to the dataset and model architecture choices makes it challenging to choose reliably in practice.

By adopting the proposed two-stage training approach, where each loss is optimized separately, we not only observe further performance gains compared to even the "oracle" hyper-parameter value (i.e., rank different choices based on the test accuracy itself), it entirely alleviates the sensitivity of this challenging optimization process. Consequently, the proposed network parameterization behaves robustly in both compression and knowledge distillation use-cases, and produces significantly superior results over NeRN. We found that even starting from an inferior point (24.20% from Recon-only with Hidden 220 on CIFAR100 in Table 2), performance can be significantly recovered to 69.31% with only $\mathcal{L}_{KD}$ in the second phase, representing a 9% improvement over NeRN. Furthermore, we achieve a *CR* of approximately 57% (Hidden 360) while surpassing the original network's performance. For ImageNet, our approach shows only a 3% drop, while NeRN shows an 8% drop, compared to the original performance when the *CR* is 15%.

Table 2: Evaluation performance of small predictor networks via decoupled training. The reported results represent the mean values over three runs except for ImageNet experiments. Recon-only ($\mathcal{L}_{recon}$), NeRN ($\mathcal{L}_{recon} + \mathcal{L}_{KD} + \mathcal{L}_{FMD}$), and **Ours** ($\mathcal{L}_{KD}$ only in the second phase).

Method		Recon-only / NeRN / **Ours**		
CIFAR10	Original ResNet20	Hidden 120 ($CR\times100 \approx 27\%$)	Hidden 140 ($CR\times100 \approx 35\%$)	Hidden 180 ($CR\times100 \approx 53\%$)
Accuracy (↑, %)	91.69	75.75 / 87.99 / **90.75**	85.64 / 89.67 / **91.34**	90.03 / 91.26 / **91.75**
OOD (↑, %)	70.49	50.50 / 65.00 / **68.84**	60.08 / 67.19 / **69.95**	66.34 / 69.75 / **70.48**
FGSM (↑, %)	76.41	59.76 / 72.73 / **75.38**	70.26 / 74.96 / **75.83**	74.78 / 76.01 / **76.27**
I-FGSM (↑, %)	75.02	58.87 / 71.48 / **73.77**	69.05 / 73.58 / **74.21**	73.39 / 74.44 / **74.83**
CIFAR100	Original ResNet56	Hidden 220 ($CR\times100 \approx 24\%$)	Hidden 280 ($CR\times100 \approx 36\%$)	Hidden 360 ($CR\times100 \approx 57\%$)
Accuracy (↑, %)	71.37	24.20 / 60.94 / **69.31**	49.55 / 66.87 / **70.84**	67.48 / 70.39 / **71.46**
OOD (↑, %)	44.70	13.01 / 38.59 / **43.76**	27.08 / 42.00 / **44.61**	40.21 / 44.21 / **45.14**
FGSM (↑, %)	43.69	14.65 / 40.30 / **45.35**	30.76 / 43.51 / **44.95**	43.28 / 44.82 / **45.01**
I-FGSM (↑, %)	39.31	14.14 / 38.73 / **42.61**	29.38 / 41.41 / **41.95**	40.74 / **41.78** / 41.12
STL10	Original ResNet56	Hidden 280 ($CR\times100 \approx 36\%$)	Hidden 320 ($CR\times100 \approx 46\%$)	Hidden 360 ($CR\times100 \approx 57\%$)
Accuracy (↑, %)	76.31	69.41 / 74.99 / **76.02**	73.36 / 75.64 / **76.25**	74.81 / 75.74 / **76.26**
FGSM (↑, %)	39.28	36.27 / 39.69 / **39.82**	38.89 / **39.83** / 39.28	38.53 / **39.77** / 39.21
I-FGSM (↑, %)	36.12	33.60 / 36.33 / **36.61**	35.44 / **36.27** / 35.96	35.30 / **36.40** / 35.96
ImageNet	Original ResNet18	Hidden 700 ($CR\times100 \approx 15\%$)	Hidden 1024 ($CR\times100 \approx 31\%$)	Hidden 1372 ($CR\times100 \approx 55\%$)
Accuracy (↑, %)	69.76	51.10 / 61.91 / **66.48**	65.69 / 67.32 / **68.68**	68.81 / 68.87 / **69.32**
OOD (↑, %)	33.07	19.87 / 25.59 / **30.25**	28.36 / 30.90 / **32.17**	31.72 / 32.54 / **32.82**
FGSM (↑, %)	57.83	39.99 / 50.19 / **53.94**	53.82 / 55.67 / **56.69**	56.96 / 57.14 / **57.40**
I-FGSM (↑, %)	57.15	38.42 / 49.63 / **53.28**	53.23 / 55.06 / **56.09**	56.32 / 56.43 / **56.76**

4.3 DISTILLATION-DRIVEN COMPRESSION AND PERFORMANCE BOOST - *CR* < 1 & *CR* > 1

In Sections 4.1 and 4.2, we explored two distinct avenues: improving model performance with large predictors (*CR* > 1) and achieving greater compression with small predictors (*CR* < 1), respectively. In this section, we seek to simultaneously improve on both of the objectives. Leveraging the flexibility of our decoupled training approach, we can utilize the superior guidance provided by a high-capacity teacher network to enhance parameter efficiency and produce high-fidelity representations. To this end, we employ ResNet50 as a teacher network, which has a size of 90.43MB with 78.48% accuracy on CIFAR100, and the reconstructed network (ResNet56) with a size of 3.25MB. As the teacher network is used only during the training in the second phase, the computational overhead and larger parameter of the teacher do not affect either the predictor network or the reconstructed model.

Firstly, we present the results of a parameter-efficient predictor guided by the teacher network in Table 3. For a fair comparison, the NeRN model is also trained using $\mathcal{L}_{KD}$ under the same teacher's supervision, as $\mathcal{L}_{FMD}$ is applicable only to architectures that are identical between the student and the teacher. The results in the table support the evidence that a high-performing teacher network can improve the efficiency of the predictor. For instance, in the case of Hidden 280, our method's performance 'with guidance' achieves an accuracy of 72.06%, surpassing both the 'without guidance' case and the original network (71.37%), as well as NeRN (66.21%).

Interestingly, we observe that increasing the predictor size, particularly when parameter efficiency is not a critical factor, can lead to significant performance gains. This is likely due to the ability of a larger predictor to capture higher-fidelity representations of the teacher model. As shown in Table 4, our method achieves superior performance levels over NeRN. Notably, our best-performing model achieves an accuracy of $\mathbf{73.95\%}$, outperforming the conventional KD approach, a student

Table 3: Evaluation performance of **parameter-efficient predictor networks with guidance** from a high-performing teacher network (ResNet50). We compare the effect of including the distillation objective to both NeRN ($\mathcal{L}_{recon} + \mathcal{L}_{KD}$) and the proposed approaches ($\mathcal{L}_{KD}$ only in the second phase). In each case, we show the results for *without guidance* / *with guidance* from a teacher.

CIFAR100	Original ResNet56	Hidden 220 (*CR*×100 ≈ 24%)		Hidden 280 (*CR*×100 ≈ 36%)		Hidden 360 (*CR*×100 ≈ 57%)	
		NeRN	Ours	NeRN	Ours	NeRN	Ours
Accuracy (↑, %)	71.37	60.94 / 58.30	69.31 / **70.25**	66.87 / 66.21	70.84 / **72.06**	70.39 / 70.94	71.46 / **72.91**
OOD (↑, %)	44.70	38.59 / 35.45	43.76 / **44.66**	42.00 / 41.13	44.61 / **46.20**	44.21 / 44.37	45.14 / **47.12**
FGSM (↑, %)	43.69	40.30 / 37.01	45.35 / **46.69**	43.51 / 42.65	44.95 / **47.32**	44.82 / 45.53	45.01 / **48.24**
I-FGSM (↑, %)	39.31	38.73 / 35.68	42.61 / **44.29**	41.41 / 40.66	41.95 / **44.29**	41.78 / 42.61	41.12 / **44.59**

(ResNet56) is trained from scratch using the guidance of ResNet50 teacher network with the $\mathcal{L}_{KD}$ loss, achieving 73.60%. Building on the observation, and aligning with the idea presented in Section 4.1 that additional training complexity can further improve performance, we proceed with one round of progressive reconstruction targeting our best-performing model (**73.95%**). This resulted in a further improvement to **74.15%**.

Table 4: Evaluation performance of **large predictor networks with guidance** from a high-performing teacher network (ResNet50). We compare the effect of including the distillation objective on both, NeRN ($\mathcal{L}_{recon} + \mathcal{L}_{KD}$) and the proposed approaches ($\mathcal{L}_{KD}$ only in the second phase). In each case, we show the results for *without guidance* / *NeRN with guidance* / *Ours with guidance*.

Method	Recon-only / NeRN / **Ours**			
CIFAR100	Original ResNet56	Hidden 510 (*CR*>1)	Hidden 680 (*CR*>1)	Hidden 750 (*CR*>1)
Accuracy (↑, %)	71.37	71.45 / 72.02 / **73.41**	71.61 / 71.80 / **73.95**	71.56 / 71.89 / **73.82**
OOD (↑, %)	44.70	44.33 / 45.18 / **47.62**	44.47 / 45.16 / **47.61**	44.93 / 44.66 / **47.74**
FGSM (↑, %)	43.69	44.32 / 44.02 / **48.75**	44.83 / 44.10 / **49.19**	44.58 / 44.74 / **49.22**
I-FGSM (↑, %)	39.31	40.24 / 39.82 / **45.24**	40.91 / 39.94 / **45.26**	40.35 / 40.55 / **45.53**

4.4 MODEL COMPRESSION COMPARISON

In this section, we present comparative results with network quantization, applying post-training static quantization using the 'fbgemm' backend (Khudia et al., 2021) with the int8 approach. Note that the core contribution of this study lies in effective reparameterization via progressive and decoupled training strategies. While decoupled training enhances model compression, our objectives differ from quantization methods. We believe that combining reparameterization with quantization could advance both model fine-tuning and efficient inference. To strengthen our hypothesis,we first perform a direct comparison of our method with quantization and find the resulting accuracy to be superior as shown in Table 5. For example, in the case of ResNet56 on CIFAR100, the performance of the quantized ResNet56 is reduced by a significant margin, with a 1.72% drop. Remarkably, our model with the same level of size reduction, experiences only a 0.5% drop, while NeRN shows lower performance than the quantized model. Moreover, our predictor can be further compressed to achieve an even smaller model size by leveraging quantization techniques, as demonstrated in the last column. This further strengthens our hypothesis that these two methods can provide complementary benefits.

5 RELATED WORKS

Weight space generation and manipulation Recent advancements in weight generation utilize transformer (Knyazev et al., 2023), and diffusion model (Soro et al., 2024) for predicting model weights. These networks focus on representing the distribution of weights or parts of the overall network, while our approach focuses on accurate reconstruction of the source network. Furthermore, Weight space manipulation provides a direct way to alter model behavior and comes in a variety of flavors. The recent development of ever larger models (Shoeybi et al., 2019) makes fine-tuning

Table 5: Comparison with int8 quantization method. Our approach outperforms both NeRN and the quantization method on complex datasets and architectures like ResNet56 on CIFAR100.

	Method	Original ResNet20	Quantized ResNet20	NeRN Hidden 140	**Ours** Hidden 140	Quantized NeRN	Quantized **Ours**
CIFAR10	Size	1.06MB	0.37MB	0.36MB	0.36MB	0.11MB	0.11MB
	Accuracy (%)	91.69	91.38	89.67±0.28	**91.34±0.04**	77.96	**89.34**
	Method	Original ResNet56	Quantized ResNet56	NeRN Hidden 280	**Ours** Hidden 280	Quantized NeRN	Quantized **Ours**
CIFAR100	Size	3.25MB	1.17MB	1.17MB	1.17MB	0.32MB	0.32MB
	Accuracy (%)	71.37	69.65	66.87±0.29	**70.84±0.09**	49.48	**51.72**

existing models increasingly challenging, as a result, post-training model merging (Matena & Raffel, 2022; Tam et al., 2023; Ilharco et al., 2022) are becoming increasingly popular that combines existing available models for performance enhancement.

Implicit neural representations (INR) (Sitzmann et al., 2020; Tancik et al., 2020) were initially designed for representing low-dimensional data (e.g., 2D or 3D) with complex and potential high-frequency signals. Recently, INR has then been utilized for a variety of domains, e.g., from uncovering correlation in scientific data (Chitturi et al., 2023) to estimating human pose (Yen-Chen et al., 2021). We use INR for predicting filter weights for model reconstruction.

Knowledge distillation and pruning has been widely adopted for reducing model size while preserving model performance. Knowledge distillation (Chen et al., 2020; Gou et al., 2021; Chen et al., 2017; Beyer et al., 2022) utilizes a more capable teacher network to transfer prediction behavior to the student network. Pruning techniques (Lee et al., 2019; Liu et al., 2018; Gao et al., 2021; Wang et al., 2021; He & Xiao, 2023) aim to remove non-essential or potentially duplicated functionality in the network and reduce the overall parameter counts.

Semantic representation of neural networks encode meaningful, interpretable features aligned with human-understandable concepts Unterthiner et al. (2020); Schürholt et al. (2021); Peebles et al. (2022); Schürholt et al. (2022); Lim et al. (2023); Navon et al. (2023); Herrmann et al. (2024); Schürholt et al. (2024); Kofinas et al. (2024); Zhou et al. (2024b;a). Our implicit representations encode information in a distributed and flexible manner, capturing complex patterns and relationships.

6 DISCUSSION AND FUTURE WORK

In this work, we identified effective strategies that significantly improve the accuracy of the reconstructed model and compression ratio for predictor networks through exploring various trade-offs in the parameterization of model weights with neural representation. While effective, one area of the limitations is that the current predictor only works with CNN architectures, restricting its usage. Moreover, despite the flexibility of the proposed protocols that can be combined or re-applied, the necessary additional steps incur more training runs which can lead to a significant increase in computation cost and complexity. However, the increased effort may be worth it to support edge applications where the benefits are multiplied by the number of deployed instances. Still, to help address these challenges, we need methods that can predict weights for diverse architectures and are ideally more efficient when the target model grows in size and complexity. Another interesting direction that is worthy of further investigation is the relationship between the weight smoothing and the model's generalization ability. Could we directly alter the original weights to achieve a similar effect without the need to train a predictor model? Or can we potentially use that insight during training as a regularization that directly improves models' generalizability? We believe that our research on several key areas—such as how weight parameterization converges, the data requirements, the interplay between different loss components, and the challenges of distillation in reparameterized models—offers valuable insights that can significantly improve model training and deployment practices.

REFERENCES

Maor Ashkenazi, Zohar Rimon, Ron Vainshtein, Shir Levi, Elad Richardson, Pinchas Mintz, and Eran Treister. Nern: Learning neural representations for neural networks. In *The Eleventh International Conference on Learning Representations*, 2022.

Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10925–10934, 2022.

Qingwen Bu, Dong Huang, and Heming Cui. Towards building more robust models with frequency bias. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4402–4411, 2023.

Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

Defang Chen, Jian-Ping Mei, Can Wang, Yan Feng, and Chun Chen. Online knowledge distillation with diverse peers. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3430–3437, 2020.

Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. *Advances in neural information processing systems*, 30, 2017.

Lei Chen, Joan Bruna, and Alberto Bietti. How truncating weights improves reasoning in language models. *arXiv preprint arXiv:2406.03068*, 2024.

Zhen Cheng, Fei Zhu, Xu-Yao Zhang, and Cheng-Lin Liu. Average of pruning: Improving performance and stability of out-of-distribution detection. *arXiv preprint arXiv:2303.01201*, 2023.

Sathya R Chitturi, Zhurun Ji, Alexander N Petsch, Cheng Peng, Zhantao Chen, Rajan Plumley, Mike Dunne, Sougata Mardanya, Sugata Chowdhury, Hongwei Chen, et al. Capturing dynamical correlations using implicit neural representations. *Nature Communications*, 14(1):5852, 2023.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

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. Ieee, 2009.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Shangqian Gao, Feihu Huang, Weidong Cai, and Heng Huang. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9270–9280, 2021.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

Hao Guo, Jiyong Jin, and Bin Liu. Stochastic weight averaging revisited. *Applied Sciences*, 13(5): 2935, 2023a.

Qiang Guo, Caiming Zhang, Yunfeng Zhang, and Hui Liu. An efficient svd-based method for image denoising. *IEEE transactions on Circuits and Systems for Video Technology*, 26(5):868–880, 2015.

Yangyang Guo, Guangzhi Wang, and Mohan Kankanhalli. Pela: Learning parameter-efficient models with low-rank approximation. *arXiv preprint arXiv:2310.10700*, 2023b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Vincent Herrmann, Francesco Faccio, and Jürgen Schmidhuber. Learning useful representations of recurrent neural network weight matrices. *arXiv preprint arXiv:2403.11998*, 2024.

Andrew G Howard. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Jack SN Jean and Jin Wang. Weight smoothing to improve network generalization. *IEEE Transactions on neural networks*, 5(5):752–763, 1994.

Salman H Khan, Munawar Hayat, and Fatih Porikli. Regularization of deep neural networks with spectral dropout. *Neural Networks*, 110:82–90, 2019.

Daya Khudia, Jianyu Huang, Protonu Basu, Summer Deng, Haixin Liu, Jongsoo Park, and Mikhail Smelyanskiy. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

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

Boris Knyazev, Doha Hwang, and Simon Lacoste-Julien. Can we scale transformers to predict parameters of diverse imagenet models? In *International Conference on Machine Learning*, pp. 17243–17259. PMLR, 2023.

Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, Efstratios Gavves, Cees GM Snoek, and David W Zhang. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

N Lee, T Ajanthan, and P Torr. Snip: single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*. Open Review, 2019.

Derek Lim, Haggai Maron, Marc T Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. *arXiv preprint arXiv:2312.04501*, 2023.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pp. 25790–25816. PMLR, 2023.

William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International conference on machine learning*, pp. 5301–5310. PMLR, 2019.

Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493, 2021.

Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35:27906–27920, 2022.

Konstantin Schürholt, Michael W Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. *arXiv preprint arXiv:2406.09997*, 2024.

Pratyusha Sharma, Jordan T Ash, and Dipendra Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. *arXiv preprint arXiv:2312.13558*, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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.

Bedionita Soro, Bruno Andreis, Hayeon Lee, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. *arXiv preprint arXiv:2402.18153*, 2024.

Derek Tam, Mohit Bansal, and Colin Raffel. Merging by matching models in task subspaces. *arXiv preprint arXiv:2312.04339*, 2023.

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.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020.

Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14913–14922, 2021.

Less Wright. Ranger-a synergistic optimizer. *GitHub Repos. Available online at: https://github.com/lessw2020/Ranger-Deep-Learning-Optimizer*, 2019.

Lin Yen-Chen, Pete Florence, Jonathan T Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin. inerf: Inverting neural radiance fields for pose estimation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1323–1330. IEEE, 2021.

Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36, 2024a.

Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Neural functional transformers. *Advances in neural information processing systems*, 36, 2024b.

A Frequency and Singular value Modulation

In Section 3.1, we observed interesting behavior of the weight through the lens of singular value ratio (S_{ratio}) analysis, where higher S_{ratio} values correlated with improved performance of the reconstructed model, as shown in Figure 2. We hypothesize that the reconstruction process may implicitly induce a weight smoothing effect, which relates to a reduction in high-frequency or noise components, and in turn, improves the model's generalization. To establish a closer and tangible connection between model performance and frequency or singular value-based modulation, we consider two simple, complementary approaches that directly apply manipulation on model weights: (1) reducing the high-frequency components of the weight through frequency-based modulation, and (2) downscaling the low singular values that only contribute minimally to the overall weight structure. We find that both procedures can independently lead to improved model performance without any additional training, though these heuristics require carefully tuned parameter on test data, which makes them less feasible for practical application. Our goal for these experiments is to further investigate why the reconstruction objective alone can improve reconstructed models' performance.

Frequency-based weight modulation: First, we simply suppress the high-frequency components using an exponential low-pass filter. Specifically, given a weight tensor $w \in \mathbb{R}^{c_{out} \times c_{in} \times k \times k}$, to focus on frequency patterns across the spatial dimensions, the weight tensor is reshaped to $c_{out} \times c_{in} \times (k \times k)$. We then apply the FFT along the spatial dimensions, transforming the tensor into its frequency representation. To reduce the high-frequency components, we leverage a Gaussian low-pass filter defined as: $H(f) = \exp(-0.5(f/D_0)^2)$ where f is the frequency index, and D_0 represents the cutoff frequency. A smaller D_0 leads to greater suppression of high-frequency components. After low-pass filtering, we apply the inverse FFT to return the frequency-modified weights back into the spatial domain. We apply this module to one block of ResNet56 while keeping the other blocks fixed, considering only the convolutional layers and excluding fully connected layers, batch normalization layers, and others. Figure 7 presents the model performance on test set (CIFAR100) using these modified weights (solid blue line) compared to the original model without frequency filtering (dotted red line).

Interestingly, suppressing the high-frequency components with an appropriate cutoff frequency (D_0) improves test accuracy. This trend is observed across all blocks of ResNet56, with the first block showing the most significant improvement. Note that, selecting the optimal cutoff frequency for each layer is non-trivial in practice, especially without access to test data. Our proposed progressive optimization automatically performs this smoothing effect, eliminating the need for manual hyperparameter tuning in different layers.

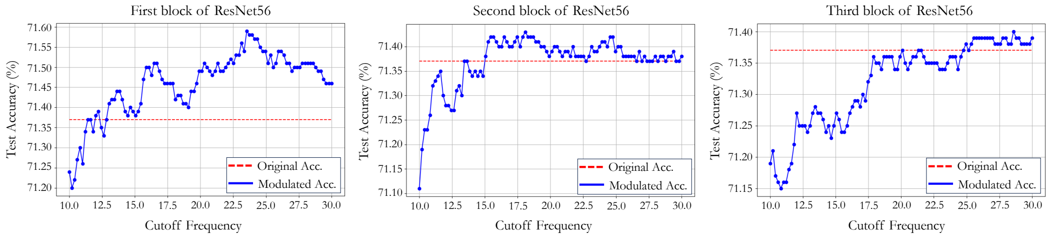


Figure 7: Evaluation of the original model's performance with varying cutoff frequencies applied in each block of ResNet56 on CIFAR100.

Singular value-based weight modulation: Here, we perform singular value modulation by scaling down the less significant singular values of a given layer, as these components contribute less to the overall matrix. Specifically, we select the last 15 singular values to be modulated, with each singular value multiplied by a weight factor (≤ 1). We then reconstruct the weights using the modified singular values and evaluate the model performance on the CIFAR100 test set. We apply this modulation to one block of ResNet56 while keeping the other blocks fixed, considering only the convolutional layers and excluding fully connected layers, batch normalization layers, and others.

Similar to frequency-based modulation, our results indicate that using appropriately modulated singular values yields a model that outperforms the original model. While direct modulation of singular values is feasible, conducting a hyperparameter search for optimal scaling factors can be challenging due to the varying preferences for weights across different layers. Our proposed progressive training entirely alleviates the need for this extensive hyperparameter tuning in different layers.

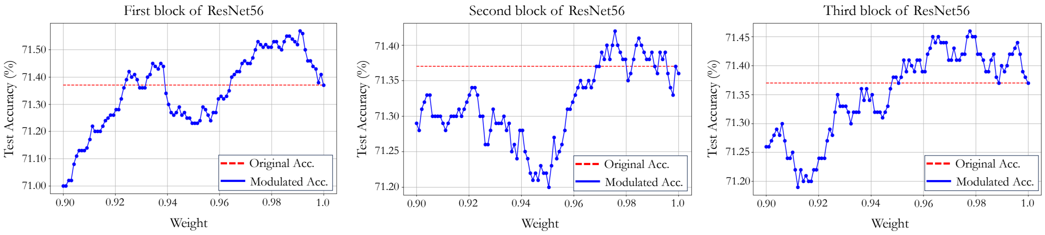


Figure 8: Evaluation of the original model's performance with varying weights (multipliers) applied in each block of ResNet56 on CIFAR100.

Exploring potential link between frequency and singular value-based modulations: While both frequency-based modulation and singular value-based modulation have independently demonstrated improvements in model performance with a naive hyperparameter approach, it is important to note that these methods are not inherently connected. This is because modulating singular values (especially smaller ones) does not guarantee the removal of high-frequency components.

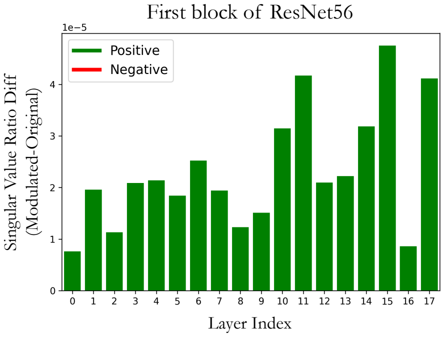


Figure 9: Layer-wise S_{ratio} analysis.

Given this distinction, the relationship between these two concepts remains questionable. To further investigate whether a meaningful link exists, we hypothesize that *the model with smoothed weights (after low-pass filtering) should exhibit larger* S_{ratio} *values than the original weight without low-pass filtering*, based on previous observations. To test this hypothesis, we conduct an additional analysis. First, we extract the best-performing model from Figure 7, where only the first block was modulated, achieving an accuracy of 71.59% at $D_0 = 23.53$. We then display layer-averaged singular value ratio difference between smoothed weights and original weights. Interestingly, the results demonstrate a similar trend to that observed in Figure 2 and show that the frequency-modulated model after low-pass filtering exhibits a higher S_{ratio} compared to the original weights before filtering. This suggests a potential correlation between the two concepts, although further research is required to confirm a definitive link.

B ROLE OF THE FEATURE MAP DISTILLATION (FMD) LOSS

Our analysis aligns with the finding in (Ashkenazi et al., 2022), that there is an apparent drop in the performance of the baseline NeRN when the FMD loss is omitted (e.g., 2% drop at $CR \approx 24\%$ for CIFAR100) as shown in Table 6. We make the following observation for proposed decoupled training: 1) $\mathcal{L}_{KD}$ is essential to our optimization and superior to using $\mathcal{L}_{FMD}$. 2) Decoupled training with only $\mathcal{L}_{KD}$ is highly effective as shown in Table 7, and it outperforms NeRN trained with $\mathcal{L}_{KD} + \mathcal{L}_{FMD}$. 3) Adding $\mathcal{L}_{FMD}$ to decoupled training does not provide significant performance gains.

Table 6: NeRN performance with/without $\mathcal{L}_{FMD}$.

Method		$\mathcal{L}_{recon} + \mathcal{L}_{KD} + \mathcal{L}_{FMD}$ / $\mathcal{L}_{recon} + \mathcal{L}_{KD}$	
CIFAR100	Original ResNet56	Hidden 220 ($CR \times 100 \approx 24\%$)	Hidden 280 ($CR \times 100 \approx 36\%$)
Accuracy (↑, %)	71.37	60.94%$\pm$0.39 / 58.94%$\pm$0.63	66.87%$\pm$0.87 / 66.03%$\pm$0.13

Table 7: Performance of our approach.

Method	$\mathcal{L}_{FMD}$ / $\mathcal{L}_{KD}$ / $\mathcal{L}_{KD} + \mathcal{L}_{FMD}$		
CIFAR100	Original ResNet56	Hidden 220 (*CR* ×100 ≈ 24%)	Hidden 280 (*CR* ×100 ≈ 36%)
Accuracy (↑, %)	71.37	67.53%±0.09 / 69.31%±0.03 / 69.31%±0.09	70.31%±0.09 / 70.84%±0.09 / 70.94%±0.09

C Decoupled Training with Noise Inputs

In this section, we explore the adaptability of our decoupled training in scenarios where the original task data is unavailable. This investigation aims to address the challenge of operating in a completely data-free environment. We employ uniformly sampled noise as input data, denoted as $X \sim U[-1, 1]$. Remarkably, even in the absence of meaningful data, our proposed approach demonstrates significant performance enhancement, with improvements of approximately 2 to 3%.

Table 8: Reconstruction performance of **Ours** ($\mathcal{L}_{KD}$ only in the second phase) with noise input data.

CIFAR10 Method (In-Filter)	Original ResNet20	Hidden 140		
		Recon-only	NeRN	**Ours**
Size	1.03MB	0.36MB	0.36MB	0.36MB
Acc. (↑, %)	91.69%	85.64%±0.39	86.31%±0.11	**87.25%±0.02**

CIFAR100 Method (In-Filter)	Original ResNet56	Hidden 320		
		Recon-only	NeRN	**Ours**
Size	3.25MB	1.48MB	1.48MB	1.48MB
Acc. (↑, %)	71.37%	61.31%±0.45	63.92%±0.11	**64.39%±0.01**

D Additional results with MobileNets

We also explore the effectiveness of the proposed approach with architectures other than ResNet variants. We present results using a lightweight network, MobileNet (Howard, 2017) in Table 9. The results demonstrate that our approach not only outperforms NeRN but is also applicable to lightweight architectures.

Table 9: Reconstruction performance with MobileNet.

CIFAR100 Method (In-Filter)	Original ResNet20	Hidden 50		
		Recon-only	NeRN	**Ours**
Acc. (↑, %)	63.71%	59.85%±0.21	61.58%±0.08	**62.90%±0.01**

E *CR* rates in Tables

Here, we provide the exact CR values used in all experiments.

Hidden Size	120	140	180	300			
CR w.r.t ResNet20	0.27	0.35	0.53	1.28			
Hidden Size	220	280	320	360	510	680	750
CR w.r.t ResNet56	0.24	0.36	0.46	0.57	1.06	1.83	2.20
Hidden Size	700	1024	1372				
CR w.r.t ResNet18	0.15	0.31	0.55				

F LOSS LANDSCAPE ANALYSIS

Understanding the landscape of the loss function in the weight space provides valuable insights into the optimization process and the behavior of neural networks. Here, we conduct a comprehensive loss landscape analysis to compare the learned weights from each method. As shown in Figure 10, the weights found by our decoupled training lie on the periphery of the most desirable solutions. This suggests that decoupled training enables the predictor to explore better optima by exclusively learning from the predictive knowledge of the original network.

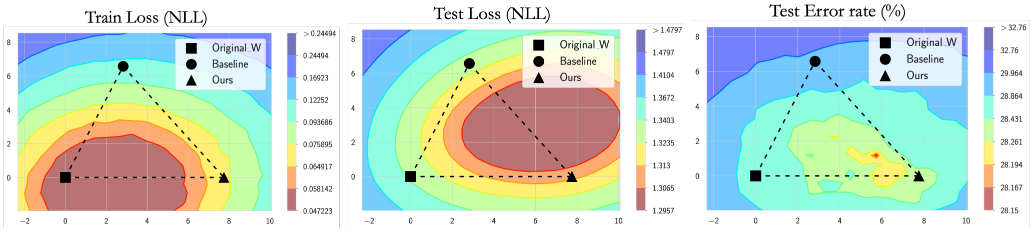


Figure 10: Loss landscape analysis with comparisons among weights from the original networks, NeRN ($\mathcal{L}_{recon} + \mathcal{L}_{KD} + \mathcal{L}_{FMD}$), and **Ours** ($\mathcal{L}_{KD}$ only in the second phase)

We also visualize the loss landscape by interpolating the weights among the original network, and the first and last rounds of the reconstructed model. As shown in Figure 11, all three weights belong to the same local extrema in the loss landscape. This observation is expected since the reconstruction loss constrains all weights to be close to the original model. For the testing loss/error, the reconstruction process appears to enhance the generalization performance.

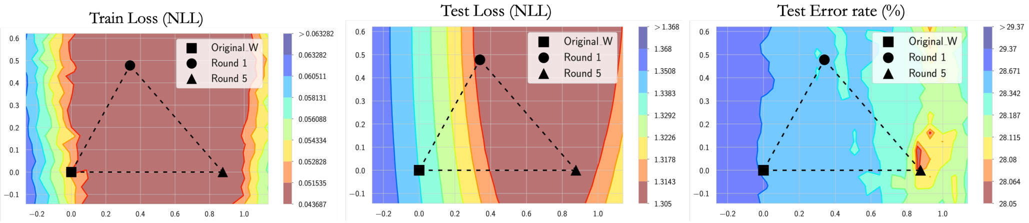


Figure 11: Loss landscape analysis comparing weights from the original network with those from the reconstructed models in the first and last rounds.

G DATASETS AND TRAINING DETAILS

G.1 DATASET DESCRIPTION

- **CIFAR-10 (Krizhevsky et al., 2009):** The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 testing images. Each image is labeled with one of the following classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck.
- **CIFAR-100 (Krizhevsky et al., 2009):** Similar to CIFAR-10, the CIFAR-100 dataset contains 60,000 32x32 color images, but organized into 100 classes, with 600 images per class. The dataset is divided into 50,000 training images and 10,000 testing images. Each image is labeled with one of the 100 fine-grained classes, which are grouped into 20 coarse-grained superclasses.
- **STL-10 (Coates et al., 2011):** The STL-10 dataset comprises 10,000 labeled 96x96 color images, with 5,000 images for training and 5,000 for testing. The dataset contains images from 10 different classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. Unlike CIFAR, STL-10 also includes a pre-defined unlabeled dataset for unsupervised learning tasks.
- **ImageNet-1K (Deng et al., 2009):** ImageNet-1K is a large-scale dataset consisting of over 1.2 million high-resolution images across 1,000 different classes. It is widely used for image

classification, object detection, and other computer vision tasks. The dataset is divided into training (1.28 million images), validation (50,000 images), and test sets (100,000 images). Each image is labeled with one of the 1,000 object categories.

G.2 TRAINING DETAILS

Training of NeRN: We follow the same settings outlined in Ashkenazi et al. (2022). The baseline method employs a Multi-layer Perceptron (MLP) with 5 layers as a predictor, with varying hidden sizes. Training is conducted using the ranger optimizer (Wright, 2019) with a learning rate of $5e-3$. The number of epochs for training is 350 for CIFAR-10 and STL-10, 450 for CIFAR-100, and 16×10^4 iterations for ImageNet experiments. Similar to minibatch sampling in standard stochastic optimization, during each training step, it predicts all reconstructed weights but optimizes only on a mini-batch of them. The weights batch method employed is a random weighted batch, using weighted sampling with a probability of $1 - p_{uni}$, where $p_{uni} = 0.8$, and a batch size of 4096 for CIFAR-10, CIFAR-100, and STL-10 datasets. For ImageNet, the experiment was conducted with a minibatch size of 2^{16}. Hyperparameters α and β in learning objectives are set to $1e-5$ for CIFAR-100 and STL-10 datasets, and to $1e-4$ to CIFAR-10, and $1e-6$ for ImageNet. Based on empirical observations, increasing the hyperparameter values during training to emphasize the distillation process causes the method to experience highly unstable training, often resulting in convergence failure. Therefore, we opted to use the same values as suggested by the authors.

When training predictors, there are two types of permutation-based smoothness: In-Filter and Cross-Filter. Both approaches do not show significant difference in terms of accuracy. The order of weights in the original network remains unchanged; this smoothness only affects the order in which the predictor processes the kernels. In all experiments, In-Filter smoothness was used for CIFAR-10, CIFAR-100, and STL-10 datasets, while Cross-Filter smoothness was employed for the ImageNet dataset.

Training of Progressive-Reconstruction Training: We adhere to the same settings as full training in the NeRN method, including the number of epochs, batch size, learning rate, and other parameters. To isolate and illustrate the reconstruction's pure effect, the predictor is trained only with the reconstruction loss, $\mathcal{L}_{recon}$. For the next round of progressive-reconstruction training, we select the best-performing models from the previously reconstructed network, determined across three trials with different random seeds, as the target network. If the performance does not surpass that of the target network, we conclude the round. To elucidate the training protocols, we present the results of all three trials conducted on the CIFAR-100 dataset. As observed in the trend of improvement, the gap in enhancement diminishes as the rounds progress.

Table 10: Evaluation performance of large predictor networks via progressive reconstruction. We report all results in three trials. The colored box represents the target performance for the next round.

CIFAR100 Method	# Trial	Original ResNet56	Hidden 680 (*CR*>1)				
			Round 1	*Round 2*	*Round 3*	*Round 4*	*Round 5*
Accuracy (↑, %)	1	71.37	71.62	71.73	71.80	71.91	71.92
Accuracy (↑, %)	2	-	71.59	71.69	71.89	71.96	72.05
Accuracy (↑, %)	3	-	71.63	71.79	71.86	71.91	71.99
mean±std			**71.61%**±0.01	**71.73%**±0.04	**71.85%**±0.03	**71.92%**±0.02	**71.98%**±0.05

Training of Decoupled Training: We fine-tune predictors in the second phase for 100 epochs for CIFAR-10, CIFAR-100, and STL-10, and 10^5 iterations for ImageNet, but even with a much smaller number of epochs/iterations, we observe comparable performance. We employ either Adam (Kingma & Ba, 2014) or Ranger (Wright, 2019) optimizers, and in most cases, both yield similar performance. Additionally, in the second phase with $\mathcal{L}_{KD}$, we empirically observed that applying weights to the $\mathcal{L}_{KD}$ with $\alpha < 1$ sometimes improves convergence, leading to better performance. Therefore, we set α to 0.01 in our experiment.

G.3 CODE REPRODUCIBILITY

We plan to release our code upon acceptance of the paper.

H SINGULAR VALUE RATIO COMPARISON

We extend our investigation to the solution obtained in the final round of progressive-reconstruction using the singular value ratio analysis. Figure 12 presents the layer-wise S_{ratio} differences between the reconstructed weights and the original weights. We also observe that the reconstructed weights in the final round exhibit a higher S_{ratio} compared to the original weights. Notably, compared to the first-round solution on CIFAR100 (Figure 2), the final-round solution displays a wider range of S_{ratio} differences in the positive area. This indicates that each round of progressive training promotes the weight smoothing effect, with an increase in S_{ratio}, leading to performance improvement.

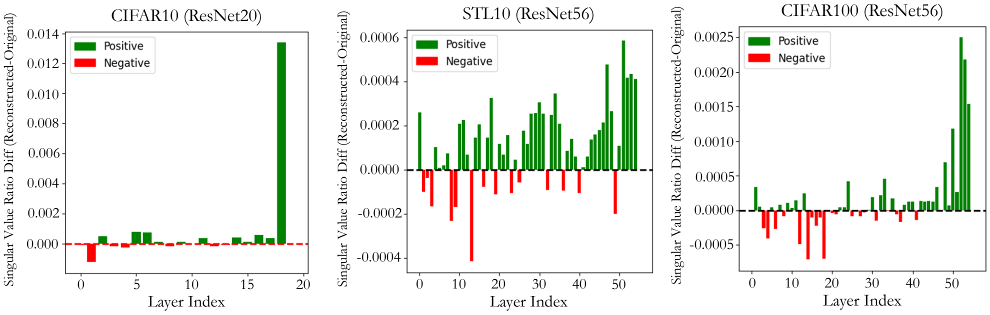


Figure 12: Layer-wise difference in singular value ratios between the reconstructed network of the last round and the original network.

I INTERPOLATION BETWEEN TWO SOLUTIONS

We further examine individual pairs of weights by directly interpolating between the original weight w_o and the reconstructed weight in each round, $\bar{w}$. Let w_o and $\bar{w}_i$ represent the original weights and the reconstructed weights at the i-th round, respectively. For each value of α in the range $[0, 1.0]$, we generate plots showing the test accuracy (on the right y-axis) and the corresponding reconstruction error (on the left y-axis) for the interpolated weights $f((1-\alpha)w_o + \alpha\bar{w}_i)$. Figure 13 illustrates the performance at intermediate points across different values of α. The plots reveal a gradual increase in both distance and accuracy as the rounds progress.

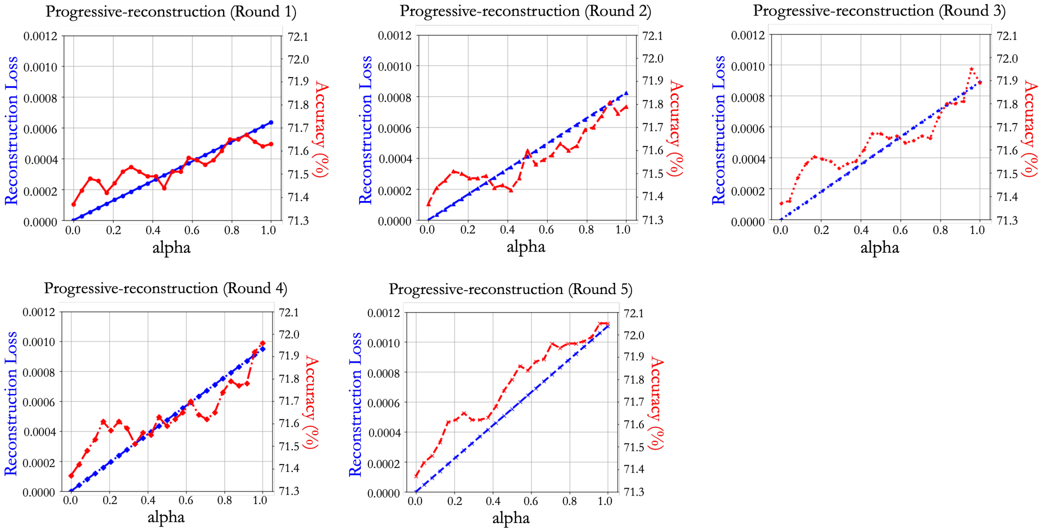


Figure 13: Analysis of interpolation in a 1-D parameter space between the original weight and the reconstructed weight in each round.

Table 11: Performance of the model across different conditions. The results are reported as mean ± standard deviation where applicable.

Original	**750**	**680**	**510**	**360**	**320**	**280**	**220**
71.37%	71.56 ± 0.05	71.61 ± 0.01	71.45 ± 0.07	67.48 ± 0.10	61.31 ± 0.45	49.55 ± 1.72	24.20 ± 1.56

J RECONSTRUCTED MODEL PERFORMANCE WITH DIFFERENT PREDICTORS

As a sanity check we include the reconstructed model performance with different predictors that are repeat trained with different initialization for each latent space size. In the Table, each column indicate a model configuration, the original is the original weight, and the numbers indicate the latent dimension size for the predictor INR.