Revisiting NeRN: Optimizing Training Strategies for Weight Reconstruction

Hongjun Choi [1] **Jayaraman J. Thiagarajan** [2] **Ruben Glatt** [1] **Shusen Liu** [1]

Abstract

The Neural Representation for Neural Network (NeRN) framework introduced a promising paradigm shift by using Implicit Neural Representations (INRs) to parameterize network weights as coordinate-based functions, rather than modeling data directly. Despite its conceptual novelty, NeRN remains impractical due to two main limitations: (1) the reconstructed weights fail to match the original model's performance, and (2) broader applicability such as compression, fine-tuning, and transfer learning remains underexplored. In this work, we revisit INR-based weight reconstruction by examining the limitations of NeRN's training strategy and propose improvements through two training strategies: (1) a reconstruction-only objective that, under overparameterization, enables the predictor to match and even exceed the performance of the original model, and (2) a decoupled training scheme that separates reconstruction and distillation phases, allowing each to specialize in its respective objective, thereby improving training stability and parameter efficiency. Our results advance INR-based parameterization toward practical use, demonstrating high-fidelity weight recovery and improved generalization.

1. Introduction

Neural network weight space exploration has recently gained attention as a post-training step to improve model performance during or after training, fine-tuning, or compression (Izmailov et al., 2018; Sun et al., 2022; Knyazev et al., 2023; Schürholt et al., 2024). Examples of these methods range from weight manipulation strategies such as weight merging to improve model performance without fine-tuning (Matena & Raffel, 2022; Ainsworth et al., 2022) to weight generation approaches such as NeRN (Ashkenazi et al., 2022) and diffusion-based methods (Wang et al., 2024; Soro et al., 2024). Particularly, NeRN extends the concept of implicit neural representations (Sitzmann et al., 2020; Tancik et al., 2020), popularized by Neural Radiance Fields (NeRF) (Mildenhall et al., 2021) to the domain of neural network weight space learning that has increasingly become an essential foundation of optimizing model performance in various applications. NeRN maps kernel or layer indices to their corresponding weights, offering a structured, coordinate-driven framework for weight generation. This enables compact, continuous encodings of parameters and opens the door to exciting possibilities such as dynamic generation, model storage, and novel forms fo network adaptation. Despite its conceptual appeal, NeRN faces practical limitations that hinder its broader adoption: the 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 underexplored.

In this work, we address these limitations by advancing the understanding the role of training objectives and then suggest improving NeRN-based weight parameterization that enhances both accuracy and parameter efficiency of the predictor. NeRN employs a multi-objective function where the reconstruction loss serve as the primary objective, while distillation losses (feature-level and logit-level) play a critical complementary role in improving reconstruction fidelity. However, we identify that coupling reconstruction and distillation objectives can lead to contradictory training signals, ultimately limiting performance. To address this, we propose two training strategies that better harness complementary strengths. 1) *Progressive training:* We first demonstrate that reconstruction-only objective under an overparameterization regime can yield better-performing models. Interestingly, these improvements can be compounded through multiple rounds of reconstruction, with each round building on the previously reconstructed weights. 2) *Decoupled training:* By separating reconstruction and distillation into distinct phases, we ensure that each learning objective has the desired impact. Remarkably, our approach results in significant improvements compared to NeRN in both reconstruction fidelity and parameter efficiency, enabling previously impractical use cases, such as obtaining

[1]Lawrence Livermore National Laboratory, Livermore, CA, 94550 USA [2]Amazon AGI, San Francisco, CA, USA. Correspondence to: Hongjun Choi <hongjun.choi08@gmail.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

a better-performing model or a highly compact predictor. Importantly, our approach is a post-hoc strategy applied to pretrained models. As such, it fundamentally differs from prior weight manipulation techniques like stochastic weight averaging (Guo et al., 2023; Izmailov et al., 2018) or training-time weight smoothing methods like weight decay (Krogh & Hertz, 1991) and adversarial weight perturbation (Wu et al., 2020). Moreover, we demonstrate that our approach is orthogonal to model quantization approaches (Lee et al., 2019; Liu et al., 2018; Gao et al., 2021; Wang et al., 2021; He & Xiao, 2023), and can be combined with them for additional storage compression of the predictor network.

2. Neural Representations for Neural Networks

We begin by reviewing the NeRN framework and its training objectives. The NeRN introduced the use of INRs to reconstruct the weights of convolutional neural networks (CNNs). At its core, NeRN uses a 5-layer multilayer perceptron (MLP) G_ϕ to map input tuples (layer ℓ, filter f, channel c) to the corresponding $k \times k$ kernel in the pretrained CNN model F_θ. Note that fully-connected or normalization layers are excluded based on their comparatively negligible parameter size. To address the lack of structural smoothness across filters, NeRN applies permutation strategies that reorder filters based on similarity, improving training stability.

The central component of NeRN training is the reconstruction loss that measures the disparity between the original network weights and those recovered using the predictor. While this loss ideally yields faithful approximation regardless of predictor capacity, NeRN often suffers from instability when G_ϕ has fewer parameters than the original network. To mitigate this, NeRN incorporates auxiliary distillation losses inspired by the knowledge distillation (Hinton et al., 2015). Here the original network serves as the teacher model, and the predictor acts as the student network. Note that, while the reconstruction loss does not require access to training data, the distillation loss does, making it impractical when the training data is not available. The overall objective function can be expressed as:

$$\begin{aligned}
\mathcal{L}_{\text{objective}} &= \mathcal{L}_{\text{recon}} + \alpha\mathcal{L}_{\text{KD}} + \beta\mathcal{L}_{\text{FMD}} \qquad (1)\\
\mathcal{L}_{\text{recon}} &= \frac{1}{|\mathbf{W}|}\|\mathbf{W} - \hat{\mathbf{W}}\|_2\\
\mathcal{L}_{\text{FMD}} &= \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\sum_{l}\|\mathbf{a}_i^l - \hat{\mathbf{a}}_i^l\|_2\\
\mathcal{L}_{\text{KD}} &= \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\text{KL}(\mathbf{a}_i^{\text{out}}, \hat{\mathbf{a}}_i^{\text{out}})
\end{aligned}$$

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 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 KL 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

We first investigate the impact of reconstruction-only setup in Section 3.1, followed by the decoupled training scheme that separates reconstruction and distillation phases in Section 3.2. Finally, we further show that both accuracy and compression are significantly enhanced by leveraging a high-capacity teacher with our strategies, whereas the baseline NeRN fails to effectively capitalize on the teacher's guidance. A summary of the proposed components and key findings is illustrated in Figure 1.

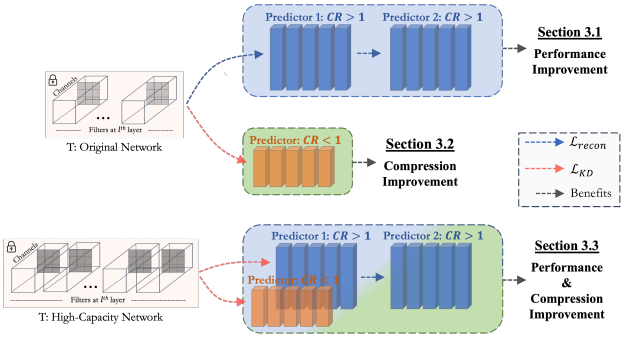

Figure 1. A summary of the explored training schemes and their benefits. *CR* denotes the ratio of the predictor's learnable parameter size to that of the original network. A value *CR* > 1 indicates a larger predictor, while *CR* < 1 indicates a smaller predictor than the original network.

3.1. Is the Reconstruction Loss All You Need?

A well-known limitation of weight prediction methods is the trade off between accuracy and parameter efficiency. While techniques like NeRN attempt to recover the lost performance via auxiliary objectives such as distillation, they often increase reconstruction errors, albeit providing improvements in the accuracy. This counterintuitive behavior motivates a deeper investigation into the relationship between reconstruction error and prediction accuracy.

If we assume that reconstruction error (e.g., $\mathcal{L}_{\text{recon}}$) is indeed an indication for performance, a straightforward strategy to improve performance would be to increase the capacity of the predictor, 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 (hidden layer sizes: 220, 280, 320, 360, 510, 680, 750). Note that here, we are not concerned about parameter efficiency, and the neural representations are trained solely with $\mathcal{L}_{\text{recon}}$. Interestingly, by using only the reconstruction loss and increasing the predictor network capacity, we empirically find that weight parameterizations with non-zero reconstruction errors (hidden layer sizes: 510, 680, 750) can not only recover but even surpass the original performance as shown in Figure 2 (Right).

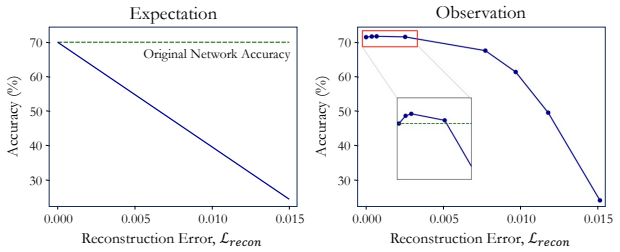

Figure 2. (Left) Starting with the original network, we expected to see accuracy decrease with increasing reconstruction error. (Right) However, we observe an unexpected initial increase in accuracy at low reconstruction errors.

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 learned weights (Domingos, 2012). We hypothesize that the performance improvement is linked to such effect.

To quantify this, we compare the reconstructed weights from a predictor trained only with $\mathcal{L}_{\text{recon}}$ to the original network's weights using the singular value ratios (S_{ratio}) analysis: 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 $S_{ratio} = \sum_{i=1}^{\lfloor k/2 \rfloor} \sigma_i^2 / \sum_{i=1}^{k} \sigma_i^2$, which captures the proportion of the total variance (energy) of the matrix explained by the first half of the singular values. A higher ratio indicates that most of the variance is captured by a few dominant, typically low-frequency components, 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 3, reconstructed weights exhibit consistently higher S_{ratio}, especially in deeper layers, suggesting a smoothing effect induced by $\mathcal{L}_{\text{recon}}$.

To better understand the link between weight components and model performance, we draw connections to prior studies on neural networks' simplicity or spectral bias (Cao et al., 2019; Huh et al., 2021; Rahaman et al., 2019; Yoshida & Miyato, 2017), which suggest models favor lower frequency components. Furthermore, techniques such as 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) have been shown to improve generalization by suppressing high-frequency information. To dive deeper into the mechanism behind the observed performance improvement and verify our hypothesis, we apply low-pass filtering to original weights and observe improved performance. Details are provided in Appendix A.

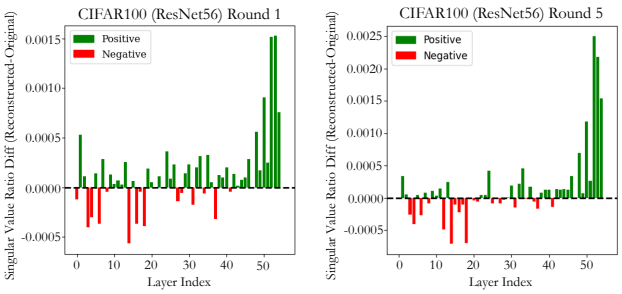

Figure 3. Layer-wise difference in singular value ratios between the reconstructed and the original network. Round 1 denotes the initial reconstruction, while Round 5 corresponds to 5th reconstruction using progressive training.

Building on the observed performance gains from the reconstruction loss, we explore a natural extension: *Can multiple rounds of neural representation learning further improve the performance?* To this end, we implement a recursive training strategy where each predictor reconstructs weights from the previous round (e.g., round 2 reconstructs round 1's output, and so on). Interestingly, 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 4(a), this progressive training produces consistent improvements in accuracy, surpassing the original network's performance across all datasets. To understand this, we further track the singular value ratio (S_{ratio}) analysis at each round. Figure 4(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. We observe a steady rise in S_{ratio} with each round, plateauing around round 5. This suggests that the weight smoothing effect diminishes, indicating that once weights are sufficiently smoothed, further rounds offer little benefit.

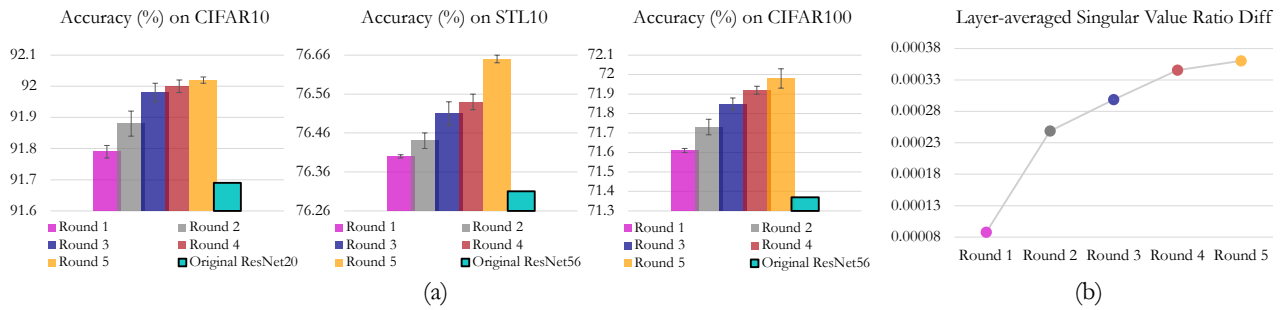

Figure 4. (a) Incrementally improving performance through progressive training. The colored bar in each round represents the average accuracy across three runs. (b) Layer-averaged difference in singular value ratios increases as the rounds progress in ResNet56 on CIFAR100.

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, our analysis did not consider parameter-efficiency. 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 neural representations were originally leveraged with such a compression objective (Ashkenazi et al., 2022), 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 to enhance the compression capability of neural representations by revisiting the role of the distillation and its interaction with the reconstruction loss. This paves the way for more practical and resource-efficient neural networks. To begin with, we analyze the layer-wise weight differences between the reconstructed and original networks at varying predictor network sizes (Figure 5(a)). Let "*CR*" 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 larger reconstruction errors due to limited capacity, while increasing capacity improves accuracy, as shown by the green bar in Figure 5(c). However, as shown in Section 3.1, reconstruction-only training struggles to fully recover performance unless predictor capacity is increased and progressive training is applied. Hence, to recover the lost performance while also enabling parameter-reduction (i.e., $CR < 1$), the baseline NeRN approach (Ashkenazi et al., 2022) incorporates an additional distillation objective ($\mathcal{L}_{KD} + \mathcal{L}_{FMD}$) during training. As illustrated by the orange bar in Figure 5(c), this guidance significantly improves the predictor's performance by injecting task-relevant information via soft targets. This can be attributed to the fact that the arbitrary perturbations in the reconstructed weights through distillation can make non-trivial changes to the decision rules, thus impacting the generalization performance of the network.

While the baseline NeRN has proven effective, it still relies heavily on the reconstruction loss, with the distillation objective playing only a supporting role, as evidenced by the minimal difference between Recon-only and NeRN in Figure 5(b)). While one can further emphasize distillation terms in (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 9 in Appendix E). This highlights both the complementary nature of the two objectives and practical difficulty of optimizing them jointly. To address this critical limitation, we propose a two-stage training scheme: 1) train the predictor network only based on $\mathcal{L}_{\text{recon}}$ (Recon-only) and optionally with progressive training, and 2) fine-tune it using only the distillation objective, $\mathcal{L}_{KD}$. As shown in Figure 5(b), 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 parameterization, as evidenced by significant improvements (red bars in Figure 5(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.

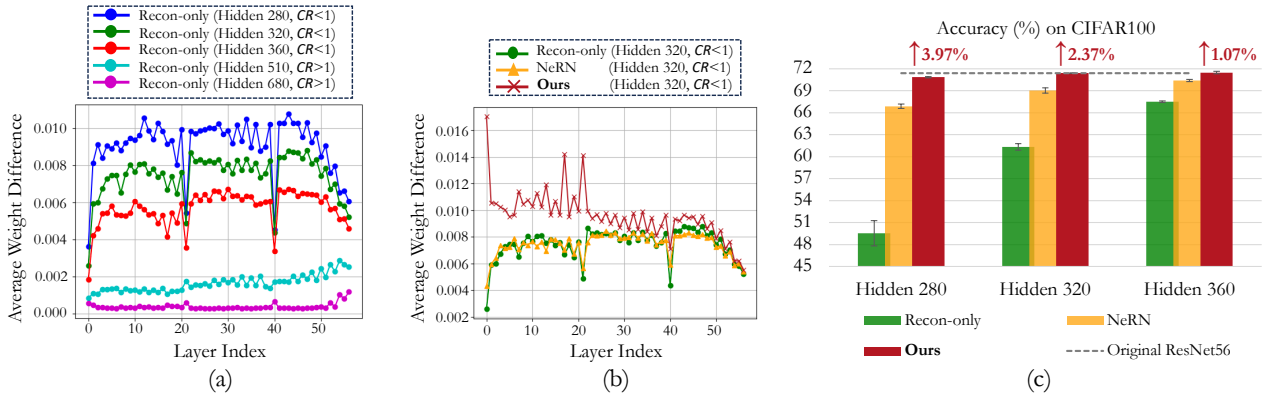

Figure 5. (a) Average weight difference between predicted weights by Recon-only and original weights with varying hidden sizes. (b) Comparison among Recon-only ($\mathcal{L}_{recon}$), NeRN ($\mathcal{L}_{recon} + \mathcal{L}_{KD} + \mathcal{L}_{FMD}$), and ours (only $\mathcal{L}_{KD}$ in the second phase). (c) Evaluation of the reconstruction performance for each method. ↑ represents our performance improvement over the NeRN.

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

3.3. Improving Compression-Performance Trade-off via Strong Teachers

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). For example, the original model can be from a previously pruned model, where the proposed method can further enhance its performance.

The limitation is mostly computational as every additional step would incur additional training.

4. Experiments

We present experiment results to support the observations 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). 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 K.

4.1. Enhancing Performance Through Progressive Reconstruction - *CR*>1

Firstly, we discovered that the reconstructed model using only the reconstruction loss outperforms the original network with 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 (e.g., smoothing effect). Importantly, this progressive training does not degrade OOD generalization or adversarial robustness metrics. In a few cases, it even leads to meaningful robustness gains. This suggests that our progressive training does not lead to undesirable overfitting behavior.

4.2. Achieving Greater Compression with Decoupled Training - *CR*<1

We identified that the baseline 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 not be fully exploited 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 ($\alpha, \beta = 1$) as seen in Figure 9 in Appendix E.

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 the 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 the 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 the NeRN shows an 8% drop, compared to the original performance when the *CR* is 15%.

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 both parameter efficiency and 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. As the teacher is only used 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 is trained with only $\mathcal{L}_{KD}$ under the same teacher, as $\mathcal{L}_{FMD}$ is applicable 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 the NeRN (66.21%). Additional results with varying hidden sizes are provided in Table 8 in Appendix F.

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

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). A lower *CR* indicates a smaller predictor size.

Method	Recon-only / NeRN / **Ours**			
CIFAR10	Original ResNet20	Hidden 120 ($CR\times100\approx27\%$)	Hidden 140 ($CR\times100\approx35\%$)	Hidden 180 ($CR\times100\approx53\%$)
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 ($CR\times100\approx24\%$)	Hidden 280 ($CR\times100\approx36\%$)	Hidden 360 ($CR\times100\approx57\%$)
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 ($CR\times100\approx36\%$)	Hidden 320 ($CR\times100\approx46\%$)	Hidden 360 ($CR\times100\approx57\%$)
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 ($CR\times100\approx15\%$)	Hidden 1024 ($CR\times100\approx31\%$)	Hidden 1372 ($CR\times100\approx55\%$)
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**

Table 3. Evaluation performance of **parameter-efficient predictor networks (Hidden size 280) with guidance** from a high-performing teacher network (ResNet50). We compare the effect of including the distillation objective to both the 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 280	
		NeRN	Ours
Accuracy (↑, %)	71.37	66.87 / 66.21	70.84 / **72.06**
OOD (↑, %)	44.70	42.00 / 41.13	44.61 / **46.20**
FGSM (↑, %)	43.69	43.51 / 42.65	44.95 / **47.32**
I-FGSM (↑, %)	39.31	41.41 / 40.66	41.95 / **44.29**

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 the 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 680 ($CR>1$)
Accuracy (↑, %)	71.37	71.61 / 71.80 / **73.95**
OOD (↑, %)	44.70	44.47 / 45.16 / **47.61**
FGSM (↑, %)	43.69	44.83 / 44.10 / **49.19**
I-FGSM (↑, %)	39.31	40.91 / 39.94 / **45.26**

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 that NeRN cannot reach. Notably, our best-performing model achieves an accuracy of **73.95%**, outperforming the conventional KD approach, a student (ResNet56) is trained from scratch using the guidance of ResNet50 teacher network with the $\mathcal{L}_{KD}$ loss, achieving 73.60%. Additional results with varying hidden sizes are provided in Table 9 in Appendix F.

5. Related Works

Weight space learning offers a direct approach to modifying model behavior and spans a wide range of techniques. With the growing scale of modern models (Shoeybi et al., 2019), traditional fine-tuning becomes increasingly impractical. As a result, post-training model merging methods (Matena & Raffel, 2022; Tam et al., 2023; Ilharco et al., 2022) have gained popularity by combining multiple pretrained models to enhance performance without additional training. Other

training-time strategies such as stochastic weight averaging (Izmailov et al., 2018; Guo et al., 2023), operate in the weight space to improve generalization by smoothing optimization trajectories. More recently, post-training methods have emerged that directly predict model weights using generative architectures, such as implicit neural representations (INRs) (Ashkenazi et al., 2022), transformers (Knyazev et al., 2023), and diffusion models (Soro et al., 2024). While INRs offer a promising framework, their practical success requires understanding and improving the optimization of NeRN, which is the focus of this work.

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 our work takes a step forward, the current predictor is limited to CNN architectures, which restricts its broader applicability. We discuss practical considerations for extending to other architectures in Appendix G, which also outlines directions for future work. Additionally, although our protocols are flexible and re-applicable, they require multiple training runs, increasing computational cost. 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? 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.

Acknowledgements should only appear in the accepted version.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 25-SI-001. LLNL-CONF-2019343.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning, particularly in weight space learning. We do not identify any immediate ethical concerns or societal risks specific to this work beyond those commonly associated with machine learning research.

References

Ainsworth, S. K., Hayase, J., and Srinivasa, S. Git re-basin: Merging models modulo permutation symmetries. *arXiv preprint arXiv:2209.04836*, 2022.

Ashkenazi, M., Rimon, Z., Vainshtein, R., Levi, S., Richardson, E., Mintz, P., and Treister, E. Nern: Learning neural representations for neural networks. In *The Eleventh International Conference on Learning Representations*, 2022.

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

Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., and Gu, Q. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*, 2019.

Chen, L., Bruna, J., and Bietti, A. How truncating weights improves reasoning in language models. *arXiv preprint arXiv:2406.03068*, 2024.

Cheng, Z., Zhu, F., Zhang, X.-Y., and Liu, C.-L. Average of pruning: Improving performance and stability of out-of-distribution detection. *arXiv preprint arXiv:2303.01201*, 2023.

Coates, A., Ng, A., and Lee, H. 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.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

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

Gao, S., Huang, F., Cai, W., and Huang, H. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9270–9280, 2021.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Guo, H., Jin, J., and Liu, B. Stochastic weight averaging revisited. *Applied Sciences*, 13(5):2935, 2023.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

Huh, M., Mobahi, H., Zhang, R., Cheung, B., Agrawal, P., and Isola, P. The low-rank simplicity bias in deep networks. *arXiv preprint arXiv:2103.10427*, 2021.

Ilharco, G., Ribeiro, M. T., Wortsman, M., Gururangan, S., Schmidt, L., Hajishirzi, H., and Farhadi, A. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

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

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

Khudia, D., Huang, J., Basu, P., Deng, S., Liu, H., Park, J., and Smelyanskiy, M. Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*, 2021.

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

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

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krogh, A. and Hertz, J. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.

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

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

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

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. On the spectral bias of neural networks. In *International conference on machine learning*, pp. 5301–5310. PMLR, 2019.

Schürholt, K., Mahoney, M. W., and Borth, D. Towards scalable and versatile weight space learning. *arXiv preprint arXiv:2406.09997*, 2024.

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

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

Soro, B., Andreis, B., Lee, H., Chong, S., Hutter, F., and Hwang, S. J. Diffusion-based neural network weights generation. *arXiv preprint arXiv:2402.18153*, 2024.

Sun, Y., Chen, Q., He, X., Wang, J., Feng, H., Han, J., Ding, E., Cheng, J., Li, Z., and Wang, J. Singular value fine-tuning: Few-shot segmentation requires few-parameters fine-tuning. *Advances in neural information processing systems*, 35:37484–37496, 2022.

Tam, D., Bansal, M., and Raffel, C. Merging by matching models in task subspaces. *arXiv preprint arXiv:2312.04339*, 2023.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems*, 33:7537–7547, 2020.

Wang, K., Tang, D., Zeng, B., Yin, Y., Xu, Z., Zhou, Y., Zang, Z., Darrell, T., Liu, Z., and You, Y. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.

Wang, Z., Li, C., and Wang, X. 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.

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

Wu, D., Xia, S.-T., and Wang, Y. Adversarial weight perturbation helps robust generalization. *Advances in neural information processing systems*, 33:2958–2969, 2020.

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

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 3 in the main text. 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 6 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.

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.

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. 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 6, 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 3 in the main text 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.

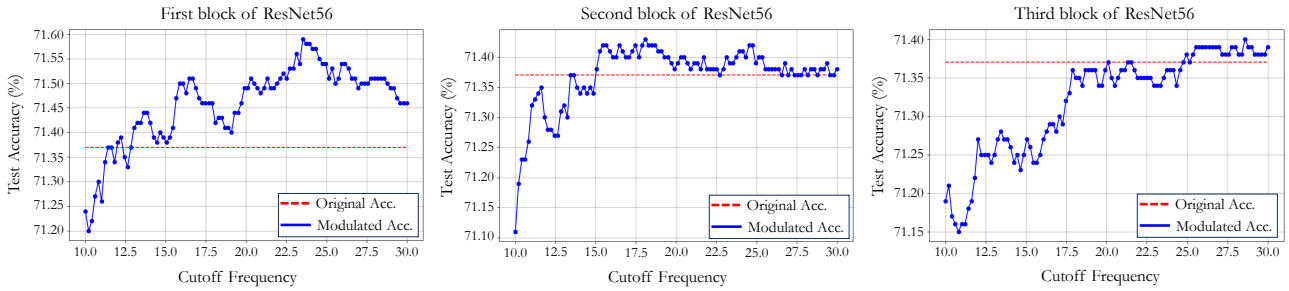

Figure 6. Evaluation of the original model's performance with varying cutoff frequencies applied in each block of ResNet56 on CIFAR100.

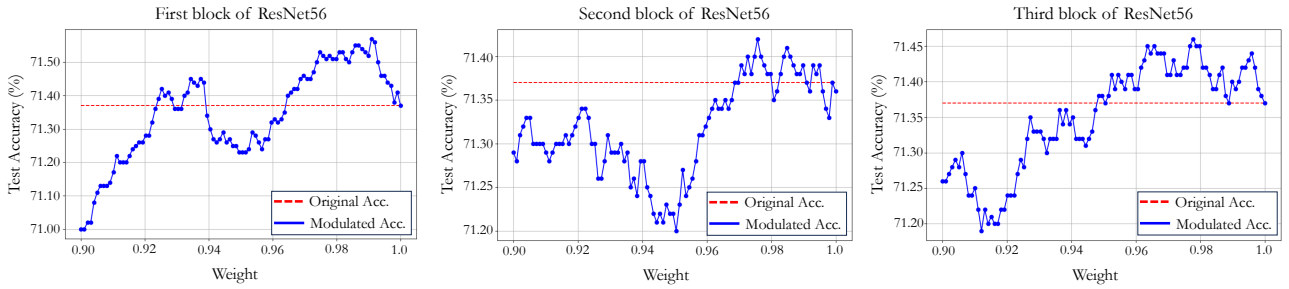

Figure 7. Evaluation of the original model's performance with varying weights (multipliers) applied in each block of ResNet56 on CIFAR100.

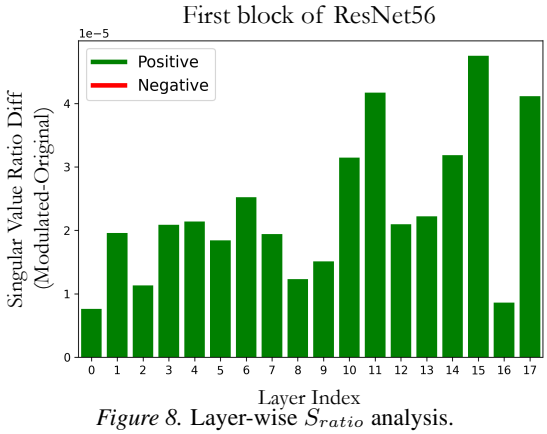

Figure 8. Layer-wise S_{ratio} analysis.

B. Downstream Tasks

To evaluate the generalization of our predicted weights, we designed a transfer learning experiment comparing the original weights (baseline pretrained on CIFAR-100) and our predicted weights (trained on CIFAR-100) when applied to the CINIC-10 dataset in Table 5. We tested two scenarios: 1) Fine-tuning all layers to fully adapt the model to the new task, and 2) Fine-tuning only the linear layer to highlight the generalizability of convolutional weights. In both cases, models were fine-tuned for 10 epochs. We confirmed that the performance trained from scratch for 150 epochs was achieved 76.88% on CINIC10 dataset. The results show that our predicted weights (Hidden 360, $CR < 1$) consistently outperform the original weights in both scenarios, achieving higher accuracy. Additionally, using predicted weights from progressive reconstruction (Round 5, Hidden 680, $CR > 1$) further improves generalization slightly. Overall, these results confirm that the proposed method generalizes well to downstream tasks.

C. Comparison with Distillation

We designed three experiments to highlight the advantages of our proposed training strategies. First, we compare our predictor network with a distilled network trained using conventional KD, i.e., a student network is trained from scratch using teacher guidance from ResNet50. The results show that our decoupled training achieves an accuracy of 73.95% (second row, 'Ours' column), outperforming the conventional KD approach, 73.60% (first row, 'Conventional KD' column) as shown in Table 6. However, the advantages of the proposed method are not limited to this performance gain; it also allows for further iterative improvement in each case. To verify this, we applied the progressive-reconstruction process on both the distilled network (73.60%, Conventional KD) and our best-performing model (73.95%, achieved through decoupled training from Table 9) for two rounds. Both result in improved performance beyond the target accuracy as shown in the third and fourth columns of the table with corresponding rows. This behavior demonstrates that our method is not constrained by the initial training techniques and can effectively refine an already optimized baseline. Additionally, to eval-

Table 5. Transferability performance. We compare the original model weights with predicted weights of our method under Hidden 360 and 680.

S→T	Tuning Layers	Original Weights	Predicted Weights Hidden 360	Predicted Weights Hidden 680
CIFAR100→ CINIC10	All Layers	80.22±0.05	**80.32±0.04**	**80.35±0.03**
CIFAR100→ CINIC10	Linear Layers	58.95±0.03	**58.98±0.04**	**59.20±0.04**

uate whether progressive reconstruction enhances smaller NeRNs, we used a compact predictor with 70.84% accuracy ('Ours' in Table 2 in the main text) as the target network. As shown in the last row of the table, applying a second round of progressive reconstruction (Hidden 680, $CR > 1$) improved performance beyond the target accuracy (70.84%) and brought it closer to the original accuracy (71.37%) as shown in the last row of the table. This suggests that smaller NeRNs can benefit from progressive reconstruction without directly targeting the original weights.

D. Model Compression Comparison

In this section, we compare our predictor with post-training static quantization using the 'fbgemm' backend (Khudia et al., 2021) with the int8 approach. It should be emphasized that the core contribution of this work is in effectively performing reparameterization through novel progressive-training and decoupled training strategies. While improved model compression is one of the key capabilities that decoupled training offers, the objective of our work are different from quantization methods. In fact, we believe that, by combining the benefits of neural network reparameterization and quantization methods, significant advances can be made both towards 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 7. 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 the 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.

E. Hyperparameter Sensitivity

To test reconstruction performance under varying configurations, 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 its performance, however, the sensitivity of this hyperparameter to the dataset and model architecture choices makes it challenging to choose reliably in practice.

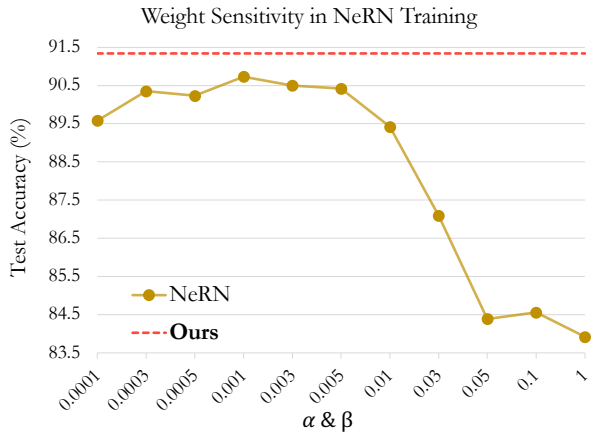

Figure 9. Impact of hyperparameters on model performance in NeRN training.

F. High-Performing Teacher Guidance

Here, we present extended results across different hidden sizes when the predictor is trained with guidance from a high-performing teacher. As shown in Table 8, our method consistently outperforms NeRN in both the *without guidance* and *with guidance* settings, particularly when using smaller predictors. Moreover, when parameter efficiency is not a constraint (e.g., $CR > 1$), our approach yields significant improvements over both the reconstruction-only and NeRN in Table 9.

G. Limitation of Applicability to Transformers

While our current work focuses on effectively performing NeRN's reparametrization on CNNs, we recognize the importance and value of extending NeRN to other architectures. Unfortunately, due to time constraints, we were unable to perform the extensive research to extend NeRN's application to transformers in this submission. However, we have carefully considered the challenges and requirements and will briefly discuss them here.

Generalizing NeRN to transformers would require signifi-

Table 6. Comparison with conventional knowledge distillation (denoted as "Conventional KD"). Our method achieves 73.95%, outperforming Conventional KD. Progressive training further boosts performance for both conventional KD and our best-performing model, with notable gains after two rounds of progression. Interestingly, even our smaller predictor (70.84%) benefits significantly from progressive training.

Original	Conventional KD	Round 1 Hidden 680, $CR > 1$	Round 2 Hidden 680, $CR > 1$
71.37%	73.60%	**73.88±0.04**	**73.99±0.01**
Original	Ours, Hidden 680 from Table 4	Round 1 Hidden 680, $CR > 1$	Round 2 Hidden 680, $CR > 1$
71.37%	73.95±0.09	**74.15±0.06**	**74.28±0.03**
Original	Ours, Hidden 280 from Table 2	Round 1 Hidden 680, $CR > 1$	Round 2 Hidden 680, $CR > 1$
71.37%	70.84	**71.13±0.01**	**71.25±0.004**

Table 7. Comparison with int8 quantization method. Our approach outperforms both the 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**

Table 8. 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 the 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 ($CR\times100 \approx 24\%$)		Hidden 280 ($CR\times100 \approx 36\%$)		Hidden 360 ($CR\times100 \approx 57\%$)	
		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**

Table 9. 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 the 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**

cant modifications to its current setup, primarily in designing a suitable coordinate system and addressing the larger size and complexity of weight matrices in transformers. For example, the coordinate system would need to capture aspects such as layer index, head index, weight type (query, key, value), and block indices for submatrices (if decomposed weight matrices into smaller submatrices). These additions would maintain NeRN's core principle of functional weight representation but would introduce new challenges during training due to the complex structure.

Additionally, although NeRN does not impose architecture-specific assumptions, it relies on INR, which typically as-

sumes that input coordinates represent a smooth and continuous space and that the learned function exhibits some level of smoothness. However, in transformers, the coordinates are discrete, and the corresponding target weight lacks continuity, making convergence and accuracy more challenging. NeRN addresses this for CNNs by introducing permutation-based smoothness, which promotes kernel smoothness by reordering pre-trained weights without altering their values. However, in transformers, the increased size and complexity of both the coordinate system and weight matrices would require further innovations to simplify the prediction task and ensure smooth learning.

H. 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 baseline NeRN when the FMD loss is omitted (e.g., 2% drop at *CR* $\approx 24\%$ for CIFAR100) as shown in Table 10. 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 11, and it outperforms the NeRN trained with $\mathcal{L}_{KD}+\mathcal{L}_{FMD}$. 3) Adding $\mathcal{L}_{FMD}$ to decoupled training does not provide significant performance gains.

I. 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%.

J. 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 13. The results demonstrate that our approach not only outperforms the NeRN but is also applicable to lightweight architectures.

K. Training Details

Training of NeRN: We follow the same settings outlined in (Ashkenazi et al., 2022). The NeRN 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 of NeRN, 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 NeRN 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.

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. Addi-

Table 10. 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%±0.39 / 58.94%±0.63	66.87%±0.87 / 66.03%±0.13

Table 11. Performance of our approach.

Method	$\mathcal{L}_{FMD}$ / $\mathcal{L}_{KD}$ / $\mathcal{L}_{KD} + \mathcal{L}_{FMD}$		
CIFAR100	Original ResNet56	Hidden 220 ($CR \times 100 \approx 24\%$)	Hidden 280 ($CR \times 100 \approx 36\%$)
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

Table 12. Reconstruction performance of **Ours** ($\mathcal{L}_{KD}$ only in the second phase) with noise input data.

CIFAR10	Original	Hidden 140		
Method (In-Filter)	ResNet20	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	Original	Hidden 320		
Method (In-Filter)	ResNet56	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**

Table 13. Reconstruction performance with MobileNet.

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

Table 14. 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	# Trial	Original	Hidden 680 (CR>1)				
Method		ResNet56	*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**

tionally, 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.