Neural Metamorphosis

Xingyi Yang and Xinchao Wang*

National University of Singapore
xyang@u.nus.edu, xinchao@nus.edu.sg
https://adamdad.github.io/neumeta/

Abstract. This paper introduces a new learning paradigm termed **Neural Metamorphosis** (**NeuMeta**), which aims to build self-morphable neural networks. Contrary to crafting separate models for different architectures or sizes, NeuMeta directly learns the continuous *weight manifold* of neural networks. Once trained, we can sample weights for any-sized network directly from the manifold, even for previously unseen configurations, without retraining. To achieve this ambitious goal, NeuMeta trains neural implicit functions as hypernetworks. They accept coordinates within the *model space* as input, and generate corresponding weight values on the manifold. In other words, the implicit function is learned in a way, that the predicted weights is well-performed across various models sizes. In training those models, we notice that, the final performance closely relates on smoothness of the learned manifold. In pursuit of enhancing this smoothness, we employ two strategies. First, we permute weight matrices to achieve intra-model smoothness, by solving the Shortest Hamiltonian Path problem. Besides, we add a noise on the input coordinates when training the implicit function, ensuring models with various sizes shows consistent outputs. As such, NeuMeta shows promising results in synthesizing parameters for various network configurations. Our extensive tests in image classification, semantic segmentation, and image generation reveal that NeuMeta sustains full-size performance even at a 75% compression rate.

Keywords: Weight Manifold · Morphable Neural Network · Implicit Neural Representation

1 Introduction

The world of neural networks is mostly dominated by the *rigid* principle: Once trained, they function as static monoliths with immutable structures and parameters. Despite the growing intricacy and sophistication of these architectures over the decades, this foundational approach has remained largely unchanged.

This inherent rigidity presents challenges, especially when deployed in new scenarios unforeseen during the network's initial design. Each unique scenario calls for a new model of distinct configuration, involving repeated design, training, and storage processes. Such an approach is not only resource-intensive, but also limits the model's prompt adaptability in rapidly changing environments.

* Corresponding author.

In our study, we embark on an ambitious quest to design neural networks that can, once trained, be continuously morphed for various hardware configurations. Particularly, our goal is to move beyond the confines of fixed and pre-trained architectures, and create networks that readily generalize to unforeseen sizes and configurations during the training phase.

Indeed, this problem has been considered in slightly different setup, , employing strategies like flexible models [2,56] and network pruning techniques [10,11,31]. The former ones are designed to self-adapt to various subnetwork configurations, whereas the latter ones aim to eliminate redundant connections, achieving models that are streamlined yet robust. Nevertheless, these solutions have their own challenges: flexible models are confined to the their training configurations, and pruning methods compromise performance and often require further retraining. Most importantly, they are still building numerous rigid models, without the ability to be continuously morphed.

To this end, we present a new learning paradigm, termed **Neural Metamorphosis** (`NeuMeta`). That is, instead of interpreting neural networks not as discrete entities, we see them as points sampled from a continuous and high-dimensional *weight manifold*. This shift allows us to learn the manifold as a whole, rather than handling isolated points. As such, `NeuMeta`, as its name implies, can smoothly morphs one network to another, with similar functionality but different architecture, such as width and depth. Once done, we can generate the weights for arbitrary-sized models, by sampling directly from this manifold.

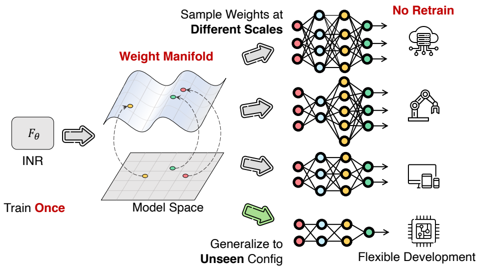


Fig. 1: Pipeline of **Neural Metamorphosis**.

At the heart of our paradigm is the use of Implicit Neural Representation (INR) as hypernetworks to fit the manifold. Intuitively, the INR acts as an indexing function of weights: upon receiving the network configuration and weight coordinates as inputs, it produce the corresponding weight values. In the training phase, the INR is assigned two goals: it approximates the weights of the pre-trained network, while simultaneously minimizing the task loss across a variety of randomly sampled network configurations. During the testing phase, the INR receives the weight coordinates for the desired network configuration, outputting values to parameterize the target network. This implementation is, by nature, different from existing methods that builds continuous neural networks that rely on integral operations and explicit continuous weight function [28,49].

Nonetheless, our ambitious effort comes with great challenges, causing the simplistic solutions to fail. A primary difficulty arises from the inherently non-smooth properties of network's weights, which hinders the fitting of the INR.

To overcome this, we put forth two strategic solutions. The first involves the permutation of weight matrices to enhance the **intra-network smoothness**.

Recognizing the flaws of prior attempts, we formulate it as a *multi-objective Shortest Hamiltonian Path problem* (mSHP). By solving this problem within individual network cliques, we enhance the smoothness of the network's weights.

The second strategy, aimed at **cross-network smoothness**, involves introducing random noise into the input. During the INR training, we maintain output consistency of the main model, irrespective of this input noise. During testing, the expected output value around this sampled coordinates is employed as the predicted weight value. This strategy moves away from a rigid grid, allowing for greater flexibility in generating new networks. Together, these strategies simplify the process of modeling the weight manifold, thereby enhancing the robustness of our approach.

We evaluated `NeuMeta` on various tasks including image classification, semantic segmentation, and generation. Our findings reveal that `NeuMeta` not only matches the performance of original model but also excels across different model sizes. Impressively, it maintains full-size model performance even with a 75% compression rate. Remarkably, `NeuMeta` extrapolates unseen weights. In other words, it can generates parameters for network sizes outside its training range, accommodating both larger and smaller models.

This paper's contributions are summarized as follows:

- We introduce Neural Metamorphosis, a new learning paradim that leverage INRs to learn neural networks' continuous weight manifold. Once trained, this INR can generate weights for various networks without retraining.
- We introduce dual strategies to improve both *intra-network* and *cross-network* smoothness of the weight manifold. This smoothness is key to generate high-performing networks.
- The proposed method undergoes thorough evaluations across multiple domains, including image classification, segmentation and generation, underscoring their versatility and robustness in varied computational setup.

2 Related Work

Efficient Deep Neural Networks. In recourse-limited applications, the efficiency of neural networks becomes a critical concern. Researcher have explored structure pruning methods [16,34] that trim non-essential neurons to reduce computation. Another development is the flexible neural networks [2,3,14,21,55,56], which offer modifiable architectures. These networks are trained on various subnetwork setups, allowing for dynamic resizing. In our context, we present a new type of flexible model that directly learns the continuous weight manifold. This allows it to generalize to configurations that haven't been trained on, an infeasible quest for existing approaches.

Continuous Deep Learning. Beyond traditional neural networks that discretize weights and outputs, continuous deep learning models represent these elements as continuous functions [4, 49]. The concept extends to neural networks with infinite hidden width, modeled as Gaussian processes [38]. Further

Method	Continuous	HyperNet	Resizable	Checkpoint-Free	Generalize to Unseen
Structure Prune [36]	✗	✗	✓*	✓	✓
Network Transform [5, 52, 54]	✗	✗	✓	✓	✓
Flexiable NN [2, 56]	✗	✗	✓	✓	✗
Continuous NN [45, 49]	✓	✗	✓	✓	✗
Weight Generator [23]	✗	✓	✓	✗	✓
NeuMeta (Ours)	✓	✓	✓	✓	✓

Table 1: Comparing methods for building neural networks but can be resized. Structural pruning ($*$) only reduces network size. Network Transform manipulates weights to construct a functionally identical version of the source network. Flexible Models are training-dependent and fail with unseen networks. Existing Continuous NNs are only valid for specific operators. Weight generators need extensive training checkpoints. Our NeuMeta uniquely learns a continuous weight manifold with INR hypernet. As such we can generalize to any neural operator and unseen configurations beyond training.

endeavors replace matrix multiplication with integral operations [45, 49]. Neural Ordinary Differential Equations (Neural ODEs) [4] build models by defining dynamical system with differential equations, fostering a continuous transformation of data. Our method diverges from those above, using an INR to create a continuous weight manifold. This allows for continous weight sampling and introduces a new type of continuous network.

Knowledge Transfer. Pre-trained neural networks have become a cornerstone for advancing deep learning, enabling rapid progress in downstream tasks due to their transferable learned features [5, 52]. Techniques like network transform [5, 52, 54] and knowledge distillation [20, 53, 57] adapt these features to fit new architectures and more compact networks. Our approach also transfer knowledge, but instead of doing a one-to-one transfer, we derive a continuous manifold of weights from a trained neural network. This enables a one-to-many knowledge transfer, which can create multiple networks of various sizes.

HyperNetworks. HyperNetworks optimize neural architectures by generating weights via auxiliary networks [15]. To accommodate weights for new models or architectures, they are trained on a vast range of checkpoints, learning a weight distribution [23, 40, 46], facilitating multitask learning [37, 42], continual learning [39], fewshot learning [47] and process implicit function [6].

Unlike typical hypernetworks producing fix-sized weights, our method uses an INR as hypernetwork that learns to predict *variable-sized* weights, offering dynamic, on-demand weight adaption. Similarly, [1] uses INR as hypernet, but their approach is confined to fixed-size weight prediction. [42] predicts connections between layers without accounting for various weight sizes.

We provide a comparative analysis of the aforementioned methods in Table 1.

3 Implicit Representation on Weight Manifold

In this section, we introduce the problem setup of NeuMeta and present our solution. In short, our idea is to create a neural implicit function to predict weight for many different neural networks. We achieve this goal by posing the principle of smoothness in this implicit function.

3.1 Problem Definition

Let's imagine the world of neural networks as a big space called $\mathcal{F}$. In this space, every neural network model $f_{\mathbf{i}} \in \mathcal{F}$ is associated with a set of weight $\mathbf{W}_{\mathbf{i}} = \{w_{(\mathbf{i},\mathbf{j})}\}$. Such a model $f_{\mathbf{i}}$ is uniquely identified by its configuration $\mathbf{i} \in \mathcal{I}$, such as width (channel number) and depth (layer number). Furthermore, each weight element within the network is indexed by $\mathbf{j} \in \mathcal{J}$, indicating its specific location, including aspects like the layer index and channel index. The combination of configurations and indices, $\mathcal{I} \times \mathcal{J}$ forms the *model space*, uniquely indexing each weight. We say the all weights values that makes up a good model on a dataset D lies on a *weight manifold* $\mathcal{W}$. We also assume we have access to a pretrained model $f(\cdot; \mathbf{W}_{\text{original}})$. Our goal is to learn the weight manifold $\mathcal{W}$.

Definition 1. (Neural Metamorphosis) *Given a labeled dataset D and a pretrained model $f(\cdot; \mathbf{W}_{original})$, we aim to develop a function $F : \mathcal{I} \times \mathcal{J} \to \mathcal{W}$ that maps any points in the model space to its optimal weight manifold. This is achieved by minimizing the expected loss across full $\mathcal{I} \times \mathcal{J}$.*

$$\min_{F} \mathbb{E}_{\forall(\mathbf{i},\mathbf{j})\in\mathcal{I}\times\mathcal{J}}\left[\mathcal{L}_{\text{task}}(f_{\mathbf{i}}(\mathbf{W}^*_{\mathbf{i}}); D)\right], \quad s.t.\mathbf{W}^*_{\mathbf{i}} = \{w^*_{(\mathbf{i},\mathbf{j})}\}, w^*_{(\mathbf{i},\mathbf{j})} = F(\mathbf{i},\mathbf{j}), \tag{1}$$

where $\mathcal{L}_{\text{task}}$ denotes the task-specific loss function. In other words, F give us the best set of weights for any model setup in $\mathcal{I} \times \mathcal{J}$, rather than fitting a single or a set of neural networks [2,56]. In context neural network, our F, which inputs coordinates and outputs values, is known as implicit neural representation (INR) [48]. Consequently, we choose INR as our F, offering a scalable and continuous method learn this mapping.

Connecting to Continuous NN. `NeuMeta` can be viewed as a method to build Continuous NNs, by representing weight values as samples from a continuous weight manifold. Here, we would like to see how it differs from existing methods. For example, in continuous-width NNs [45, 49], linear operations are typically defined by the Riemann integral over inputs and weights:

$$f(x) = \mathbf{x} \cdot \mathbf{w} = \sum_{j}(\mathit{\Delta}_j x_j W(j)) \approx \int_0^1 x(j)W(j)\delta j, \tag{2}$$

where $\mathbf{x}$ and $\mathbf{w}$ represent discrete input and weight vectors. j is the continuous-valued channel index. $W(j)$ is a continuous weight function, and $\mathit{\Delta}_j$ is the width of the sub-interval between sampled points for integral.

Our method offers three key advantages over this traditional approach:

- Integral-Free. `NeuMeta` requires no integral.
- Learned Continuous Sampling. Our method jointly learns the continuous weight function and the sampling interval $w_j = \mathit{\Delta}_j W(j)$, rather that learning $W(j)$ along. This enables us to generate continuous-width NN on-fly, a feat unachievable with discrete learned sampling [49].
- INR Parameterization. INR offers a generalized form to model the continuous function[1].

[1] Prior designs using kernel [49] or piece-wise functions [45] can be considered special cases of INR, as detailed in our supplementary material.

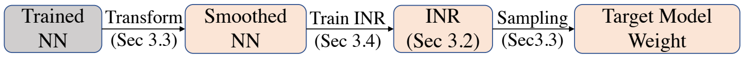


Fig. 2: Diagram of `NeuMeta` and our content organization.

Challenge and Solution. Our effort, while ambitious, presents distinct challenges. First, an INR design for neural network weight is largely unexplored. Second, it is essential to train on limited samples from the weight manifold and then generalize to unseen ones. Our solution, as depicted in Figure 2, includes an INR-based architecture in Section 3.2 and a strategy for learning on a smooth weight manifold in Sec 3.3. The training process is discussed in Sec 3.4 .

3.2 Network Architecture

At the core of `NeuMeta`, we employ an INR model, $F(\cdot;\theta):\mathbb{R}^k\rightarrow\mathbb{R}^d$, to parameterize the weight manifold. This function, based on a multi-layer perceptron (MLP), transforms model space into weight values. In our implementation, we set the parameter $k=6$. For convolutional networks, the dimension $d=K\times K$, the maximum kernel size, whereas for non-convolutional setups, $d=1$.

Considering a generalized network with L layers, each layer with an input-output channel size of $(C_{\text{in}},C_{\text{out}})$. Each weight element, $w_{(\mathbf{i},\mathbf{j})}$, is associated with a unique index within the network. This index is represented as a coordinate pair $(\mathbf{i},\mathbf{j})$, with $\mathbf{i}=(L,C_{\text{in}},C_{\text{out}})$ denoting the network structure and $\mathbf{j}=(l,c_{\text{in}},c_{\text{out}})$ indicating the its specific layer, input, and output channel number. To ensure the same coordinate system is applicable to all $(\mathbf{i},\mathbf{j})$, these raw coordinates undergo normalization, typically rescaling them with a constant N

$$\mathbf{v}=\left[\frac{l}{L},\frac{c_{\text{in}}}{C_{\text{in}}},\frac{c_{\text{out}}}{C_{\text{out}}},\frac{L}{N},\frac{C_{\text{in}}}{N},\frac{C_{\text{out}}}{N}\right], \tag{3}$$

Similar to prior technique [35], the normalized coordinates undergo a transformation through sinusoidal position embedding, to extract its Fourier features.

$$\gamma_{\text{PE}}(\mathbf{v})=\Big[\sin(2^0\pi\mathbf{v}),\cos(2^0\pi\mathbf{v}),\dots,\sin(2^{L-1}\pi\mathbf{v}),\cos(2^{L-1}\pi\mathbf{v})\Big]$$

These encoded Fourier features, $\gamma_{\text{PE}}(\mathbf{v})$, serve as inputs to the MLP, yielding the weights:

$$w_{(\mathbf{i},\mathbf{j})}=F(\gamma_{\text{PE}}(\mathbf{v});\theta)=\frac{\text{MLP}(\gamma_{\text{PE}}(\mathbf{v});\theta)}{C_{\text{in}}}. \tag{4}$$

In equation (4), the output of the MLP is scaled by the number of input channels C_{in}, ensuring that the network's output maintains scale invariance relative to the size of the input channel [13, 18].

To handle lots of parameters with INR, we adopting a block-based approach [43, 51]. Instead of a single large INR, weights are divided into a grid, with each segment controlled by a separate MLP network. The full architecture will be mentioned in the supplementary material.

In our framework, the weights for standard neural network operations are defined as follows:

Linear Operation. For linear operations, we obtain the scalar weight as the element-wise average of $w_{(\mathbf{i},\mathbf{j})}$.
Convolution Operation. For convolution layers, weights $w_{(\mathbf{i},\mathbf{j})}$ are reshaped into $k \times k$. If the kernel size k is smaller than the K, only the central $k \times k$ elements are utilized.
Batch Normalization Layer. For batch normalization, we use a re-parameterization strategy [8], integrating BN weights into adjacent linear or convolution layers. This method integrates BN operations into a unified framework.

3.3 Maintaining Manifold Smoothness

A critical design within our paradigm is ensuring the weight manifold remains smooth. We, in this section, discuss why this smoothness is crucial for the model's performance, and outline our strategy for achieving this local smoothness.
Intra-Model Smoothness. Modern neural networks heavily rely on their ability to model smooth signals [41] to ensure convergence. Yet, empirical evidence suggests that the weight matrices are typically non-smooth. To enable our INR to reconstruct weights, we must find strategies that promote smoothness.

To address this challenge, previous studies have explored the concept of *weight permutation* [1, 49]. It is often likened to the Traveling Salesman Problem (TSP) [27]. However, such an approach, while seemingly straightforward, overlooks the crucial *inter-dependencies* within and between weight matrices.

Let's consider a weight matrix $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$and measure its smoothness using *total variation*, denoted as $TV(\mathbf{W})$. It is defined as the sum of variations along both channels: $TV(\mathbf{W}) = TV_{\text{in}}(\mathbf{W}) + TV_{\text{out}}(\mathbf{W})$. In fact, applying the TSP formulation presents 3 problems:

- **(P1) Loop VS Non-Loop**: Unlike TSP, which necessitates returning to the starting point, ensuring 2D weight matrix smoothness doesn't require looping back. Instead, it is better to be considered as a *Shortest Hamiltonian Path* (SHP) [12] problem, allowing for an arbitrary starting channel.
- **(P2) Breaking Smoothness for the Connecting Layer**: Unlike isolated weight matrices, neural networks consist of connected layers, creating complex inter-layer relationships. This is illustrated in Figure 3, where permutations in one layer necessitate corresponding reversals in adjacent layers to maintain the network's functional equivalence. For example, with an activation function $\sigma(\cdot)$ and a valid permutation pair P and P^{-1} (where $PP^{-1} = I$), the following equation holds:

$$\mathbf{W}_i P \sigma(P^{-1}\mathbf{W}_{i-1}\mathbf{X}) = \mathbf{W}_i \sigma(\mathbf{W}_{i-1}\mathbf{X}) \tag{5}$$

  As a result, P^{-1} may affect the adjacent layers, with increased TV for $\mathbf{W}_i P$.
- **(P3) Breaking Smoothness for the Other Dimension**: A permutation enhancing smoothness in output channel, might introduce non-smooth patterns in the input channel, thus reducing the overall smoothness.

Luckily, we find that the computation of the TV measurement renders (P3) infeasible, implying our focus should be directed towards (P1) and (P2).

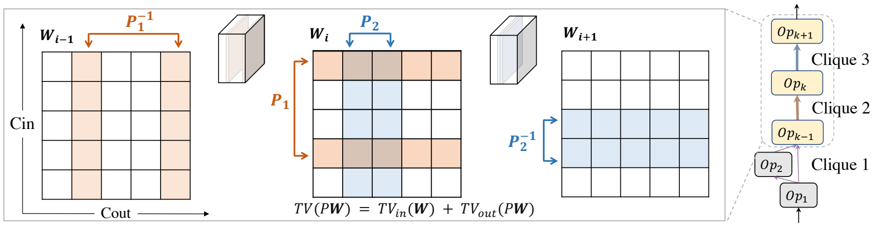


Fig. 3: Intra-model smoothness via permutation equivalence. Our approach involves permuting weights to minimize total variance within each neural clique graph, thereby enhancing global smoothness.

Proposition 1. (Axis Alignment) [2] *Let* $\mathbf{W}$ *be a given matrix and* P *be a permutation. The application of a permutation in one dimension of* $\mathbf{W}$ *does not influence the total variation in the orthogonal dimension.*

$$TV(\mathbf{W}P) = TV_{\text{in}}(\mathbf{W}P) + TV_{\text{out}}(\mathbf{W}) \tag{6}$$

$$TV(P\mathbf{W}) = TV_{\text{in}}(\mathbf{W}) + TV_{\text{out}}(P\mathbf{W}) \tag{7}$$

Hence, to tackle global smoothness, we address challenges P1 and P2. We consider a neural network as a dependency graph $G = (V, E)$ [10], where each node $v_i \in V$ represents an operation with weight $\mathbf{W}_i$ and each edge $e_{ij} \in E$ indicates inter-connectivity between v_i and v_j. Each *graph clique* $C = (V_C, E_C) \subset G$ is full-connected, representing a group of operation is connected. As a results, each C corresponds to a unique permutation matrix P. Our objective is to determine all P in a way that minimizes the total variation across the whole network.

Luckily, based on the Proposition 1, this complex optimization can be broken down into multiple independent optimizations, each on a clique. We define this as a multi-objective Shortest Hamiltonian Path ($mSHP$) problem:

$$\arg\min_{P} \sum_{e_{ij} \in E_C} \Big(TV_{\text{out}}(P\mathbf{W}_i) + TV_{\text{in}}(\mathbf{W}_j P^{-1}) \Big) \tag{8}$$

To address each mSHP problem, we transform it into a TSP problem by adding a dummy node. This new node has edges with zero-distance to all others in the clique. We then solve TSP using a 2.5-opt local search [50]. The resulting permutation P^* is applied to all weight matrices within the clique. This promotes the weight smoothness and preserves the functionality of the network.

Since each individual mSHP problem is only correlated to one clip graph, we can solve the optimal P in a relative small scale, very efficently. In fact, with ≤ 20 cliques per network, the total computation time is < 4 sec.

Cross-Model Smoothness. Another crucial challenge is to perverse the generalization behavior of the INR with different network configurations, which means, a small perturbation in the configuration, will eventually not affect the main model's performance. We address this by adding coordinate variation in the INR's learning process.

[2] Proof in the supplementary material.

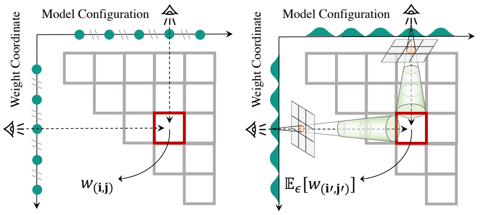

Model Configuration
Weight Coordinate
$w_{(\mathbf{i},\mathbf{j})}$
Model Configuration
Weight Coordinate
$\mathbb{E}_{\epsilon}[w_{(\mathbf{i}',\mathbf{j}')}]$

Fig. 4: Cross-model smoothness via coordinate perturbation. Unlike the predict weights in discrete grid **(Left)**, our INR predicts weight as the expectation within a small neighborhood **(Right)**.

During training, rather than using fixed coordinates and model sizes as in Equation 4, we introduce slight variations to the input coordinates. Specifically, we add a small perturbation $\boldsymbol{\epsilon}$ to the input coordinate $(\mathbf{i}', \mathbf{j}') = (\mathbf{i}, \mathbf{j}) + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is drawn uniformly from $\mathrm{U}(-\mathbf{a}, \mathbf{a})$. This strategy aims to minimize the expected loss $\mathbb{E}_{\boldsymbol{\epsilon}\in\mathrm{U}(-\mathbf{a},\mathbf{a})}[\mathcal{L}]$.

For model evaluation, we sampling weight from a small neighborhood, as illustrated in Figure 4. We compute this by averaging the weights obtained from multiple input, each perturbed by different $\boldsymbol{\epsilon} \in \mathrm{U}(-\mathbf{a}, \mathbf{a})$:

$$\bar{w}_{(\mathbf{i},\mathbf{j})} = \mathbb{E}_{\boldsymbol{\epsilon}\in\mathrm{U}(-\mathbf{a},\mathbf{a})}[w_{(\mathbf{i}',\mathbf{j}')}] \approx \frac{1}{K}\sum_{K} F(\gamma_{\mathrm{PE}}(\mathbf{v}');\theta) \tag{9}$$

This is implemented by inferring the INR $K = 50$ times with varied sampled inputs $\mathbf{v}'$ and then computing the average of these weights to parameterize the main network. This approach is designed to enhance the stability and reliability of the INR under different configurations.

3.4 Training and Optimization

Our approach optimizes the INR, denoted as $F(\cdot;\theta)$, to accurately predict weights for the main network of different configurations. It pursues two primary goals: approximating the weights of the pretrained network $f(\cdot;\mathbf{W}_{\mathrm{original}})$, and minimizing task-specific loss across a range of randomly sampled networks. As such, the optimization leverages a composite loss function, divided into three distinct components: task-specific loss, reconstruction loss, and regularization loss.

Task-specific Loss. Denoted as $\mathcal{L}_{\mathrm{task}}(y, \hat{y}(\mathbf{W}))$, this measures the difference between actual labels y and predictions $\hat{y}$, based on weights $\mathbf{W}$ from the INR.

Reconstruction Loss. This element, expressed as $\mathcal{L}_{\mathrm{recon}} = ||\mathbf{W}_{\mathrm{original}}||_2^2||\mathbf{W} - \mathbf{W}_{\mathrm{original}}||^2$, assesses how close the INR-derived weights $\mathbf{W}$ to the ideal weights $\mathbf{W}_{\mathrm{original}}$, weighted by the magnitude $||\mathbf{W}_{\mathrm{original}}||_2^2$.

Regularization Loss. Symbolized as $\mathcal{L}_{\mathrm{reg}} = ||\mathbf{W}||^2$. This introduces L2 norm regularization on the predicted weights, to prevent overfitting by controlling the complexity of the derived model [26,33].

We minimize the composite objective by sampling different points on the model space

$$\min_{\theta} \mathbb{E}_{\mathbf{i},\mathbf{j},\boldsymbol{\epsilon}}[\mathcal{L}] = \min_{\theta} \mathbb{E}_{\mathbf{i},\mathbf{j},\boldsymbol{\epsilon}}[\mathcal{L}_{\mathrm{task}} + \lambda_1\mathcal{L}_{\mathrm{recon}} + \lambda_2\mathcal{L}_{\mathrm{reg}}] \tag{10}$$

This loss function ensuring not only proficiency in the primary task through precise weight, but also bolstering model robustness via regularization. During training, we iteratively evaluates various combinations $(\mathbf{i}, \mathbf{j})$, striving to minimize the expected loss. The loss function is backpropagated from the main network to the INR as follows:

$$\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}_{\text{task}}}{\partial W}\frac{\partial W}{\partial \theta} + \lambda_1 \frac{\partial \mathcal{L}_{\text{recon}}}{\partial \theta} + \lambda_2 \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \theta} \tag{11}$$

This equation represents the gradient of the loss with respect to θ.

4 Experiments

In this section, we present our experimental analysis and various applications of `NeuMeta`, spanning classification, semantic segmentation, and image generation. To substantiate our design choices, we conduct different ablation studies. Additionally, we delve into exploring the properties of the learned weight manifold.

4.1 Experimental Setup

Datasets and Evaluation. We evaluate the proposed method on 3 tasks across 6 different visual datasets. For image classification, we select 4 dataset: MNIST [30], CIFAR10, CIFAR100 [25] and ImageNet [7]. Training includes horizontal flip augmentation, and we report top-1 accuracy. We also report the training time and the final size of stored parameters to evaluate our savings.

In semantic segmentation, we utilize PASCAL VOC2012 [9], a standard dataset for object segmentation tasks. We utilize its augmented training set [17], incorporating preprocessing techniques like random resize crop, horizontal flip, color jitter, and Gaussian blur. Performance is quantified using mean Intersection-over-Union (mIOU) and F1 score, averaged across 21 classes.

For image generation, we employ MNIST and CelebA [32]. A vanilla variational auto-encoder (VAE) fits the training data, with evaluation based on reconstruction MSE and negative log-likelihood (NLL).

Implementation Details. Our INR utilizes MLPs with ReLU activation, comprising five layers with residual connections and 256 neurons each. We employ a block-based INR approach, where each parameter type, such as weights and biases, is represented by a separate MLP. The positional embedding frequency is set to 16. Optimization is done using Adam [22] with a 1e-3 initial learning rate and cosine decay, alongside a 0.995 exponential moving average. During training, we maintain the balance between different objectives with $\lambda_1 = 1$ and $\lambda_2 = 1e{-}4$. In each training batch, we sample one network configuration, update a random subset of layers in the main network, computing gradients for the INR, to speed up training. The configuration pool, created by varying the channel number of the original network, utilizes a *compression rate* $\gamma = 1 - \frac{\text{sampled channel number}}{\text{full channel number}}$. We randomly sample a network width with compress rate $\gamma \in [0, 0.5]$ for training. For example, a 128-channel layer will have its width sampled from $64 \sim 128$.

For classification tasks, we apply LeNet [29] on MNIST, ResNet20 [19] on CIFAR10 and CIFAR100, and ResNet18 and ResNet50 on ImageNet, using batch sizes of 128 (MNIST, CIFAR) and 512 (ImageNet). We the INR train for 200 epochs. For segmentation task, we use the U-Net [44] with the ResNet18 backbone. The details are mentioned in the supplementary material.

Baselines. Our NeuMeta model is benchmarked against three family of methods: Structure Pruning, Flexible Models, and Continuous-Width Models.

- **Individually Trained Model.** Each models is trained separately.
- **Structure Pruning.** We evaluate against pruning methods that eliminate channels based on different criteria. This includes Weight-based pruning (removing neurons with low ℓ_1/ℓ_2-norm), Taylor-based method (using gradients related to output), Hessian-based pruning (using Hessian trace for sensitivity), and Random pruning (random channel removal).
- **Flexible Model.** We compare with the Slimmable network [56], which trains subnetworks of various sizes within the main model for dynamic test-time resizing. We train the model on $\{0\%, 25\%, 50\%\}$ compressed ratio, and also test on 75% compressed setup.
- **Continuous-Width NN.** Comparison is made with the Integral Neural Network [49], which uses kernel representation for continuous weight modeling. For comparison, we focus on the uniform sampling method, as its learned sampling technique does not support resizing. We train the model on $[0\%, 50\%]$ compress rate range, but also test on other values, like 75%.

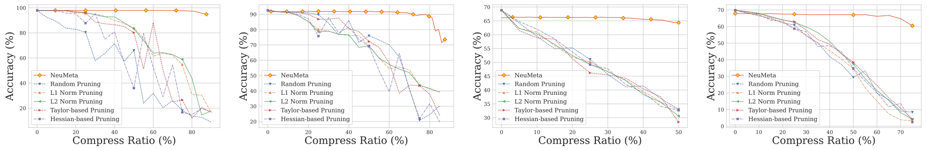
We ensure that all compression is applied uniformly for all methods, guaranteeing that all models compared have the exactly same cost for inference.

4.2 Enhancing Efficiency by Morphing the Networks

Image Classification. As depicted in Figure 5, in the realm of image classification, NeuMeta consistently surpasses existing pruning-based methods in accuracy across MNIST, CIFAR10, CIFAR100, and ImageNet datasets at various compression ratios. It is worth-noting that, pruning-based methods show a marked accuracy decrease, approximately 5% on ImageNet and 6% on CIFAR100, when the compression ratio exceeds 20%. Conversely, NeuMeta retains stable performance up to 40% compression. However, a minor performance reduction is noted in our full-sized model, highlighting a limitation in the INR's ability to accurately recreate the complex pattern of network weights.

Table 2 compares NeuMeta with Slimable NN and INN, including *Oracle* results of independently trained models for reference. We stick to the same model size for all method, to ensure the comparision is fair. Remarkably, NeuMeta often surpasses even these oracle models on large compress rate. This success is attributed to the preserved smoothness across networks of varying sizes, which inadvertently enhances smaller networks. Our approach outperforms both Slimable NN and the kernel representation in INN. Notably, at an untrained compression ratio of $75\%^{\dagger}$, other methods significantly underperform.

Furthermore, when evaluating total training time and parameter storage requirements, our approach demonstrates improved efficiency. Unlike the exhaustive individual model training and storage approach, other methods achieve some



(a) LeNet on MNIST **(b)** R20 on CIFAR10 **(c)** R20 on CIFAR100 **(d)** R18 on ImageNet

Fig. 5: Accuracy comparison of `NeuMeta` versus different structure pruning methods on MNIST, CIFAR10, CIFAR100 and ImageNet. Our method consistently outperforms pruning-based methods. R18 and R20 are short for ResNet18 and ResNet20.

	ResNet20 on CIFAR10					
Method	$\gamma = 0\%$	$\gamma = 25\%$	$\gamma = 50\%$	$\gamma = 75\%^{\dagger}$	Total Train Cost	Stored
	Acc	Acc	Acc	Acc	(GPU hours)	Params
Individual	92.60	90.65	89.57	87.04	5.3	0.67M
Slimable [56]	90.44	90.44	88.41	18.56	1.6	0.35M
INN [49]	91.33	90.50	89.24	71.70	1.8	0.27M
Ours	**91.76**	**91.32**	**90.56**	**89.56**	**1.3**	**0.20M**

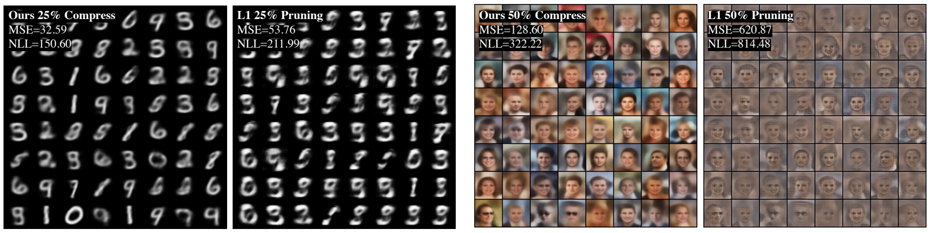
	ResNet20 on CIFAR100					
Method	$\gamma = 0\%$	$\gamma = 25\%$	$\gamma = 50\%$	$\gamma = 75\%^{\dagger}$	Total Train Cost	Stored
	Acc	Acc	Acc	Acc	(GPU hours)	Params
Individual	68.83	66.37	64.87	61.37	5.5	0.70M
Slimable [56]	64.44	64.01	63.38	1.59	1.5	0.37M
INN [49]	65.86	65.53	63.35	27.60	1.9	0.28M
Ours	**66.07**	**66.23**	**65.36**	**62.62**	**1.4**	**0.20M**

Table 2: Accuracy comparison of ResNet20 on CIFAR10 and CIFAR100 at different compression ratios. † The 75% compression ratio wasn't applied in training.

level of savings. However, Slimable NN's separate storage for BN parameters still renders it less efficient. Our method achieves the least storage size by storing a few MLPs instead of the original parameters, thus reducing the overall parameter count even below that of a single model.

Semantic Segmentation. For semantic segmentation on the PASCAL VOC2012 dataset, `NeuMeta` demonstrates superior performance in Table 8. It surpasses the Slimmable network that requires hard parameter sharing, especially at an untrained 75% compression rate. On this setup, we show a significant improvement of 20 mIOU. However, for complex tasks like segmentation, a slight performance drop is observed at smaller compression rate. It is attributed to the INR's limited representation ability. More results is provided in the supplementary.

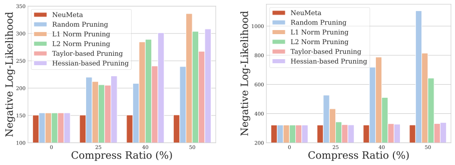
Image Generation. We implement `NeuMeta` to generate images on MNIST and CelebA, using VAE. Since Slimable NN and INN haven't been previously adapted for VAE before, we only compare with the pruning method, in Figure 7. Our approach demonstrated superior performance in terms of lower negative log-likelihood (NLL) across various compression ratios. For example, we visualize the generated results of when compressed by 25% for MNIST and 50% for CelebA in Figure 6. Compared with the ℓ_1-based pruning, our method significantly improved reconstruction MSE from 53.76→32.58 for MNIST and from 620.87→128.60 for CelebA. Correspondingly, the NLL was reduced by 61.33 for MNIST and 492.26 for CelebA.



(a) Results on MNIST

(b) Results on CelebA

Fig. 6: VAE Visualizations on MNIST and CelebA Datasets on the same compress rate. Lower NLL and MSE indicates better performance.



(a) MNIST Compress Rate *vs.* NLL

(b) CelebA Compress Rate *vs.* NLL

Fig. 7: Comparative analysis of compress rate and NLL on different datasets. Lower NLL indicates better performance.

Method	25%		50%		75%†	
	mIOU	F1	mIOU	F1	mIOU	F1
Individual	84.70	90.63	83.14	89.59	82.79	89.36
Slimmable [56]	81.09	88.14	80.92	88.03	61.19	72.78
Ours	**81.94**	**88.75**	**81.93**	**88.74**	**81.94**	**88.75**

Fig. 8: Comparison of different methods across compressed ratio for U-Net. † The 75% compression ratio wasn't seen in training.

4.3 Exploring the Properties for NeuMeta

As we represent the weights as a smooth manifold, we investigate its effects on network. Specifically, we compare `NeuMeta` induced networks, with individually trained models and models distilled [20] from full-sized versions.

NeuMeta promote feature similarity. We analyzed the last layer features of ResNet20 trained on CIFAR10, particularly from `layer3.2`, using linear central kernel alignment (CKA) [24] score between each resized and the full-sized model. The result is shown in Figure 9 (Top). It reveals higher feature map correlations across models compared to other methods, indicating that `NeuMeta` encourages similar network representations across different sizes.

NeuMeta as Implicit Knowledge Distillation. We also report the the pairwise output KL divergence in Figure 9 (Bottom), a key metric in knowledge distillation [20]. Individually trained models show higher divergence, whereas both KD and `NeuMeta` result in reduced divergence. These results imply that `NeuMeta` not only aligns internal representations but also ensures consistent network outputs, as an implicit form of distillation.

4.4 Ablation Study

Weight Permutation. To validate the effectiveness of our permutation strategy, we analyzed its impact on CIFAR10 accuracy. The comparison of Exp 2 and 4 in Table 11 demonstrates a significant 11.51 accuracy increase due to our permutation strategy. Detailed comparisons of our mSHP-based method with the TSP solution from [49] are presented in the supplementary material. It shows

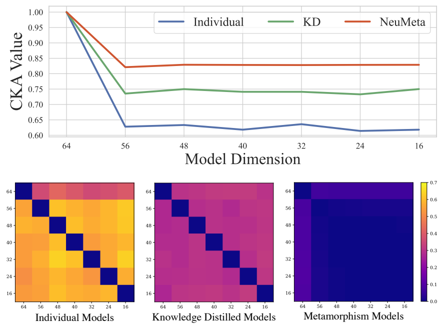


Fig. 9: Similarity Analysis Between Models. **(Top)** the CKA comparison between the full model and various other models of different sizes. **(Bottom)** heatmap of the output KL divergence for each pair of models.

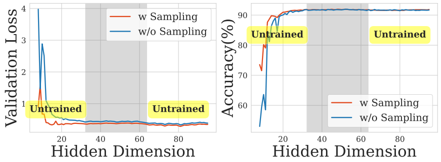


Fig. 10: Ablation study with or without manifold sampling.

No.	Weight Permutation	λ_1	λ_2	**Accuracy**
1	✗	0	1e-4	73.56
2	✗	1	1e-4	80.33
3	✓	1	0	64.37
4	✓	1	1e-4	**91.84**
5	✓	10	1e-4	91.73
6	✓	100	1e-4	91.47

Fig. 11: Ablation study for weight permutation and objective hyperprameters on CIFAR10 ResNet20.

that our mSHP-based solution achieved lower weight total variation score, indicating superior with-in model smoothness.

Objective. We verify different terms in our training objective in Eq 10. From Exp 1, 4-6 in Table 11, we find the optimal reconstruction weight $\lambda_1 = 1$ yields the best performance. Comparing Exp 3 and 4, we observe a performance boost with a weight penalty term at $\lambda_2 = 1e - 4$.

Manifold Sampling. Figure 10 evaluates our manifold sampling method with ResNet20 on CIFAR10. Sampling from the weight manifold neighborhood consistently improves performance, especially in untrained model sizes.

5 Conclusion

This paper presents Neural Metamorphosis (`NeuMeta`), a novel paradigm that builds self-morphable neural networks. Through the training of neural implicit functions to fit the continuous weight manifold, `NeuMeta` can dynamically generate tailored network weights, adaptable across a variety of sizes and configurations. A core focus of our approach is to maintain the smoothness of weight manifold, enhancing the model's fitting ability and adaptability to novel setups. Experiments on image classification, generation and segmentation indicate that, our method maintain robust performance, even under large compression rate.

Acknowledgement

This project is supported by the National Research Foundation, Singapore, under its AI Singapore Programme (AISG Award No: AISG2-RP-2021-023).

References

1. Ashkenazi, M., Rimon, Z., Vainshtein, R., Levi, S., Richardson, E., Mintz, P., Treister, E.: Nern: Learning neural representations for neural networks. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), https://openreview.net/pdf?id=9gfir3fSy3J 4, 7
2. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020), https://arxiv.org/pdf/1908.09791.pdf 2, 3, 4, 5
3. Chavan, A., Shen, Z., Liu, Z., Liu, Z., Cheng, K.T., Xing, E.P.: Vision transformer slimming: Multi-dimension searching in continuous optimization space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4931–4941 (2022) 3
4. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Advances in neural information processing systems **31** (2018) 3, 4
5. Chen, T., Goodfellow, I.J., Shlens, J.: Net2net: Accelerating learning via knowledge transfer. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016), http://arxiv.org/abs/1511.05641 4
6. De Luigi, L., Cardace, A., Spezialetti, R., Zama Ramirez, P., Salti, S., Di Stefano, L.: Deep learning on implicit neural representations of shapes. In: International Conference on Learning Representations (ICLR) (2023) 4
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., 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) 10
8. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: Making vgg-style convnets great again. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 13733–13742 (2021) 7
9. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision **88**(2), 303–338 (Jun 2010) 10
10. Fang, G., Ma, X., Song, M., Mi, M.B., Wang, X.: Depgraph: Towards any structural pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16091–16101 (2023) 2, 8
11. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=rJl-b3RcF7 2
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979) 7
13. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256. JMLR Workshop and Conference Proceedings (2010) 6
14. Grimaldi, M., Mocerino, L., Cipolletta, A., Calimera, A.: Dynamic convnets on tiny devices via nested sparsity. IEEE Internet of Things Journal **10**(6), 5073–5082 (2022) 3
15. Ha, D., Dai, A., Le, Q.: Hypernetworks (2016) 4

16. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016), http://arxiv.org/abs/1510.00149 3
17. Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., Malik, J.: Semantic contours from inverse detectors. In: 2011 international conference on computer vision. pp. 991–998. IEEE (2011) 10
18. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015) 6
19. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 10
20. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015) 4, 13
21. Hou, L., Huang, Z., Shang, L., Jiang, X., Chen, X., Liu, Q.: Dynabert: Dynamic bert with adaptive width and depth. Advances in Neural Information Processing Systems **33**, 9782–9793 (2020) 3
22. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR). San Diega, CA, USA (2015) 10
23. Knyazev, B., Drozdzal, M., Taylor, G.W., Romero Soriano, A.: Parameter prediction for unseen deep architectures. Advances in Neural Information Processing Systems **34**, 29433–29448 (2021) 4
24. Kornblith, S., Norouzi, M., Lee, H., Hinton, G.: Similarity of neural network representations revisited. In: International conference on machine learning. pp. 3519–3529. PMLR (2019) 13
25. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009) 10
26. Krogh, A., Hertz, J.: A simple weight decay can improve generalization. Advances in neural information processing systems **4** (1991) 9
27. Lawler, E.L., Lenstra, J.K., Kan, A.R., Shmoys, D.B.: The traveling salesman problem: a guided tour of combinatorial optimization. The Journal of the Operational Research Society **37**(5), 535 (1986) 7
28. Le Roux, N., Bengio, Y.: Continuous neural networks. In: Artificial Intelligence and Statistics. pp. 404–411. PMLR (2007) 2
29. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. vol. 86, pp. 2278–2324 (1998), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665 10
30. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010), http://yann.lecun.com/exdb/mnist/ 10
31. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017), https://openreview.net/forum?id=rJqFGTslg 2
32. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (ICCV) (December 2015) 10
33. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017) 9

34. Ma, X., Fang, G., Wang, X.: Llm-pruner: On the structural pruning of large language models. Advances in neural information processing systems **36**, 21702–21720 (2023) 3
35. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021) 6
36. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11264–11272 (2019) 4
37. Navon, A., Shamsian, A., Chechik, G., Fetaya, E.: Learning the pareto front with hypernetworks. In: International Conference on Learning Representations (2021), https://openreview.net/forum?id=NjF772F4ZZR 4
38. Neal, R.M.: Bayesian learning for neural networks, vol. 118. Springer Science & Business Media (2012) 3
39. von Oswald, J., Henning, C., Grewe, B.F., Sacramento, J.: Continual learning with hypernetworks. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=SJgwNerKvB 4
40. Peebles, W., Radosavovic, I., Brooks, T., Efros, A.A., Malik, J.: Learning to learn with generative models of neural network checkpoints. arXiv preprint arXiv:2209.12892 (2022) 4
41. Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., Courville, A.: On the spectral bias of neural networks. In: International Conference on Machine Learning. pp. 5301–5310. PMLR (2019) 7
42. Raychaudhuri, D.S., Suh, Y., Schulter, S., Yu, X., Faraki, M., Roy-Chowdhury, A.K., Chandraker, M.: Controllable dynamic multi-task architectures. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10955–10964 (2022) 4
43. Reiser, C., Peng, S., Liao, Y., Geiger, A.: Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14335–14345 (2021) 6
44. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. pp. 234–241. Springer (2015) 11
45. Roux, N.L., Bengio, Y.: Continuous neural networks. In: Meila, M., Shen, X. (eds.) Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS 2007, San Juan, Puerto Rico, March 21-24, 2007. JMLR Proceedings, vol. 2, pp. 404–411. JMLR.org (2007), http://proceedings.mlr.press/v2/leroux07a.html 4, 5
46. Schürholt, K., Knyazev, B., Giró-i Nieto, X., Borth, D.: Hyper-representations as generative models: Sampling unseen neural network weights. Advances in Neural Information Processing Systems **35**, 27906–27920 (2022) 4
47. Sendera, M., Przewięźlikowski, M., Karanowski, K., Zięba, M., Tabor, J., Spurek, P.: Hypershot: Few-shot learning by kernel hypernetworks. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 2469–2478 (2023) 4
48. Sitzmann, V., Martel, J., Bergman, A., Lindell, D., Wetzstein, G.: Implicit neural representations with periodic activation functions. Advances in neural information processing systems **33**, 7462–7473 (2020) 5

49. Solodskikh, K., Kurbanov, A., Aydarkhanov, R., Zhelavskaya, I., Parfenov, Y., Song, D., Lefkimmiatis, S.: Integral neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 16113–16122 (June 2023) 2, 3, 4, 5, 7, 11, 12, 13
50. Stattenberger, G., Dankesreiter, M., Baumgartner, F., Schneider, J.J.: On the neighborhood structure of the traveling salesman problem generated by local search moves. Journal of Statistical Physics **129**, 623–648 (2007) 8
51. Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P.P., Barron, J.T., Kretzschmar, H.: Block-nerf: Scalable large scene neural view synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8248–8258 (2022) 6
52. Wei, T., Wang, C., Rui, Y., Chen, C.W.: Network morphism. In: International conference on machine learning. pp. 564–572. PMLR (2016) 4
53. Yang, X., Ye, J., Wang, X.: Factorizing knowledge in neural networks. In: European Conference on Computer Vision. pp. 73–91. Springer (2022) 4
54. Yang, X., Zhou, D., Liu, S., Ye, J., Wang, X.: Deep model reassembly. Advances in neural information processing systems **35**, 25739–25753 (2022) 4
55. Yu, J., Huang, T.S.: Universally slimmable networks and improved training techniques. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1803–1811 (2019) 3
56. Yu, J., Yang, L., Xu, N., Yang, J., Huang, T.S.: Slimmable neural networks. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=H1gMCsAqY7 2, 3, 4, 5, 11, 12, 13
57. Yuan, L., Tay, F.E., Li, G., Wang, T., Feng, J.: Revisiting knowledge distillation via label smoothing regularization. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 3903–3911 (2020) 4