# Neural Similarity Learning

**Weiyang Liu**[1,*] **Zhen Liu**[2,*] **James M. Rehg**[1] **Le Song**[1,3]
[1]Georgia Institute of Technology [2]Mila, Université de Montréal [3]Ant Financial [*]Equal Contribution
wyliu@gatech.edu, zhen.liu.2@umontreal.ca, rehg@gatech.edu, lsong@cc.gatech.edu

## Abstract

Inner product-based convolution has been the founding stone of convolutional neural networks (CNNs), enabling end-to-end learning of visual representation. By generalizing inner product with a bilinear matrix, we propose the *neural similarity* which serves as a learnable parametric similarity measure for CNNs. Neural similarity naturally generalizes the convolution and enhances flexibility. Further, we consider the *neural similarity learning* (NSL) in order to learn the neural similarity adaptively from training data. Specifically, we propose two different ways of learning the neural similarity: static NSL and dynamic NSL. Interestingly, dynamic neural similarity makes the CNN become a dynamic inference network. By regularizing the bilinear matrix, NSL can be viewed as learning the shape of kernel and the similarity measure simultaneously. We further justify the effectiveness of NSL with a theoretical viewpoint. Most importantly, NSL shows promising performance in visual recognition and few-shot learning, validating the superiority of NSL over the inner product-based convolution counterparts.

## 1 Introduction

Recent years have witnessed the unprecedented success of convolutional neural networks (CNNs) in supervised learning tasks such as image recognition [20], object detection [47], semantic segmentation [40], etc. As the core of CNN, a standard convolution operator typically contains two components: a learnable template (*i.e.*, kernel) and a similarity measure (*i.e.*, inner product). One active stream of works [13, 63, 25, 61, 8, 53, 24, 59, 26] aims to improve the flexibility of the convolution kernel and increases its receptive field in a data-driven way. Another stream of works [39, 36] focuses on finding a better similarity measure to replace the inner product. However, there still lacks a unified formulation that can take both the shape of kernel and the similarity measure into consideration.

To bridge this gap, we propose the neural similarity learning (NSL) for CNNs. NSL first defines the neural similarity by generalizing the inner product with a parametric bilinear matrix and then learns the neural similarity jointly with the convolution kernels. A graphical comparison between inner product and neural similarity is given in Figure 1. With certain regularities on the neural similarity, NSL can be viewed as learning the shape of the kernel and the similarity measure simultaneously. Based on the neural similarity, we propose the *neural similarity network* (NSN) by stacking convolution layers with neural similarity. We consider two distinct ways to learn the neural similarity in CNN. First, we learn a static neural similarity which is essentially a (regularized) bilinear similarity. By having more parameters, the static neural similarity becomes a natural generalization of the standard inner product. Second and more interestingly, we also consider to learn the neural similarity in a dynamic fashion.

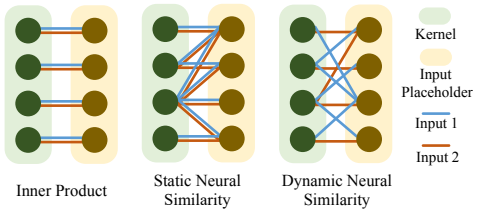



Figure 1: Bipartite graph comparison of inner product, static neural similarity and dynamic neural similarity. A line represents a multiplication operation and a circle denotes an element in a vector. Green color denotes kernel and yellow denotes input.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Specifically, we use an additional neural network module to learn the neural similarity adaptively from the input images. This module is jointly optimized with the CNN via back-propagation. Using the dynamic neural similarity, the CNN becomes a dynamic neural network, because the equivalent weights of the neuron are input-dependent. In a high-level sense, CNNs with dynamic neural similarity share the same spirits with HyperNetworks [18] and dynamic filter networks [28].

A key motivation behind NSL lies in the fact that inner product-based similarity is unlikely to be optimal for every task. Learning the similarity measure adaptively from data can be beneficial in different tasks. A hidden layer with dynamic neural similarity can be viewed as a quadratic function of the input, while a standard hidden layer is a linear function of the input. Therefore, dynamic neural similarity introduces more flexibility from the function approximation perspective.

NSL aims to construct a flexible CNN with strong generalization ability, and we can control the flexibility by imposing different regularizations on the bilinear similarity matrix. In this paper, we mostly consider the block-diagonal matrix with shared blocks as the bilinear similarity matrix in order to reduce the number of parameters. In different applications, we will usually impose domain-specific regularizations. By properly regularizing the bilinear similarity matrix, NSL is able to make better use of the parameters than standard convolutional learning and find a good trade-off between generalization ability and representation flexibility.

NSL is closely connected to a surprising theoretical result in [16] that optimizing an underdetermined quadratic objective over a matrix $\boldsymbol{W}$ with gradient descent on a factorization of this matrix leads to an implicit regularization for the solution (minimum nuclear norm). A more recent theoretical result in [5] further shows that gradient descent for deep matrix factorization tends to give low-rank solutions. Since NSL can be viewed as a form of factorization over the convolution kernel, we argue that such factorization also yields some implicit regularization in gradient-based optimization, which may lead to more generalizable inductive bias. We will give more theoretical insights in the paper.

While showing strong generalization ability in generic visual recognition, NSL is also very effective for few-shot learning due to its better flexibility. Compared to initialization based methods [14, 46], NSL can naturally make full use of the pretrained model for few-shot learning. Specifically, we propose three different learning strategies to perform few-shot recognition. Besides applying both static and dynamic NSL to few-shot recognition, we further propose to meta-learn the neural similarity. Specifically, we adopt the model-agnostic meta learning [14] to learn the bilinear similarity matrix. Using this strategy, NSL can benefit from the generalization ability of both the pretrained model and the meta information [14]. Our results show that NSL can effectively improve the few-shot recognition by a considerable margin.

Our main contributions can be summarized as follows:

- We propose the *neural similarity* which generalizes the inner product via bilinear similarity. Furthermore, we derive the *neural similarity network* by stacking convolution layers with neural similarity. Although this paper mostly discusses CNNs, we note that NSL can easily be applied to fully connected networks and recurrent networks.
- We propose both *static* and *dynamic* learning strategies for the neural similarity. To order to overcome the convergence difficulty of dynamic neural similarity, we propose hyperspherical learning [39] with identity residuals to stablize the training.
- We apply the neural similarity learning to generic visual recognition and few-shot recognition. For few-shot learning, we propose novel usages of NSL and significantly improve the current few-shot learning performance.

## 2 Related Works

**Flexible convolution**. Dilated (atrous) [61, 8] convolution has been proposed in order to construct a convolution kernel with large receptive field for semantic segmentation. [13, 25] improve the convolution kernel for high-level vision tasks by making the shape of kernel learnable and deformable. [39, 36] provide a decoupled view to understand the similarity measure and propose some alternative (learnable) similarity measures. Such decoupled similarity is shown to be useful for improving network generalization and adversarial robustness.

**Dynamic neural networks**. Dynamic neural networks have input-dependent neurons, which makes the network adaptively changing in order to deal with different inputs. HyperNetworks [18] uses a recurrent network to dynamically generate weights for another recurrent network, such that the weights can vary across many timesteps. Dynamic filter networks [28] generates its filters which

are dynamically conditioned on an input. These dynamic neural networks usually perform poorly in image recognition tasks and can not make use of any pretrained models. In contrast, the dynamic NSN performs consistently better than the CNN counterpart, and is able to take advantage of the pretrained models for few-shot learning. [11] investigates the input-dependent networks by dynamically selecting filters, while NSN uses totally different approach to achieve the dynamic inference.

**Meta-learning**. A classic approach [7, 50] for meta-learning is to train a meta-learner that learns to update the parameters of the learner's model. This approach has been adopted to learn deep networks [1, 32, 43, 51]. Recently, There are a series of works [46, 14] that address the meta-learning problem by learning a good network initialization. Specifically for few-shot learning, there are initialization-based methods [43, 46, 14, 10], hallucination-based methods [57, 19, 2] and metric learning-based methods [55, 52, 54]. Besides having very different formulation from the previous works, NSL also combines the advantages from the initialization-based methods and the generalization ability from the pretrained model.

# 3 Neural Similarity Learning

## 3.1 Generalizing Convolution with Bilinear Similarity

We denote a convolution kernel with size $C \times H \times V$ ($C$ for the number of channels, $H$ for the height and $V$ for the width) as $\tilde{\boldsymbol{W}}$. We flatten the kernel in each channel separately and then concatenate them to a vector: $\boldsymbol{W} = \{\tilde{\boldsymbol{W}}^F_{1,:,:}, \tilde{\boldsymbol{W}}^F_{2,:,:}, \cdots, \tilde{\boldsymbol{W}}^F_{C,:,:}\} \in \mathbb{R}^{CHV}$ where $\tilde{\boldsymbol{W}}^F_{i,:,:}$ is the flatten kernel weights of the $i$-th channel. Similarly, we denote an input patch of the same size $C \times H \times V$ as $\tilde{\boldsymbol{X}}$, and its flatten version as $\boldsymbol{X}$. A standard convolution operator uses inner product $\boldsymbol{W}^\top \boldsymbol{X}$ to compute the output feature map in a sliding window fashion. Instead of using the inner product to compute the similarity, we generalize the convolution with a bilinear similarity matrix:

$$f_{\boldsymbol{M}}(\boldsymbol{W}, \boldsymbol{X}) = \boldsymbol{W}^\top \boldsymbol{M} \boldsymbol{X} \tag{1}$$

where $\boldsymbol{M} \in \mathbb{R}^{CHV \times CHV}$ denotes the bilinear similarity matrix and is used to parameterize the similarity measure. In fact, if we requires $\boldsymbol{M}$ to be a symmetric positive semi-definite matrix, it shares some similarities with the distance metric learning [60]. Although we do not necessarily need to constrain the matrix $\boldsymbol{M}$, we will still impose some structural constraints on $\boldsymbol{M}$ in order to stablize the training and save parameters in practice. To avoid introducing too many parameters in the generalized convolution operator, we make the bilinear similarity matrix $\boldsymbol{M}$ to be block-diagonal with shared blocks (there are $C$ blocks in total):

$$f_{\boldsymbol{M}}(\boldsymbol{W}, \boldsymbol{X}) = \boldsymbol{W}^\top \begin{bmatrix} \boldsymbol{M}_s & & \\ & \ddots & \\ & & \boldsymbol{M}_s \end{bmatrix} \boldsymbol{X} \tag{2}$$

where $\boldsymbol{M} = \text{diag}(\boldsymbol{M}_s, \cdots, \boldsymbol{M}_s)$ and $\boldsymbol{M}_s$ is of size $HV \times HV$. Interestingly, the hyperspherical convolution [39] becomes a special case of this bilinear formulation when $\boldsymbol{M}$ is a diagonal matrix with a normalizing factor $\frac{1}{\|\boldsymbol{W}\|\|\boldsymbol{X}\|}$ being the diagonal. Since additional parameters are introduced to control the similarity measure, we are able to learn a similarity measure directly from data (*i.e.*, static neural similarity) or learn a neural predictor that can estimate such a similarity matrix from the input feature map (*i.e.*, dynamic neural similarity). In the paper, we mainly consider two structures for $\boldsymbol{M}_s$.

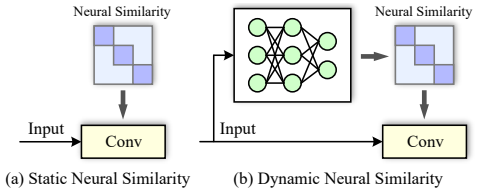
**Diagonal/Unconstrained neural similarity**. If we require $\boldsymbol{M}_s$ to be a diagonal matrix, then we end up with the diagonal neural similarity (DNS). DNS is very parameter-efficient and can be viewed as a weighted inner product or an element-wise attention. Besides that, DNS is essentially putting an additional spatial mask over the feature map, so it is semantically meaningful. If no constraint is imposed on $\boldsymbol{M}_s$, then we have the unconstrained neural similarity (UNS) which is very flexible but requires much more parameters.

## 3.2 Learning Static Neural Similarity

We first introduce a static learning strategy for the neural similarity. Specifically, we learn the matrix $\boldsymbol{M}_s$ jointly with the convolution kernel via back-propagation. An intuitive overview for static neural similarity is given in Figure 2(a). When $\boldsymbol{M}_s$ has been jointly learned after training, it will stay fixed in the inference stage. More interestingly, as we can see from Equation (1) that the neural similarity is incorporated into the convolution operator via a linear multiplication, we can compute an equivalent weights for the kernel in advance if the neural similarity is static. Therefore, we can view the new

kernel as $\boldsymbol{M}^\top \boldsymbol{W}$. As a result, when it comes to deployment in practice, the number of parameters used in static NSN is the same as the CNN baseline and the inference speed is also the same.

Learning static neural similarity can be viewed as a factorized learning of neurons. It also shares a lot of similarities with matrix factorization in the sense that the equivalent neuron weights $\hat{\boldsymbol{W}}$ is factorized into into two matrix $\boldsymbol{M}^\top$ and $\boldsymbol{W}$. Although the original weights and the factorized weights are mathematically equivalent, they have different behaviors and properties during gradient-based optimization [16]. Recent theories [16, 5, 33] suggest that an implicit regularization may encourage the gradient-based matrix factorization to give minimum nuclear norm or low-rank solutions. Besides that, we also have structural constraints to explicitly regularize the matrix $\boldsymbol{M}$. Furthermore, we can also view this static neural similarity convolution as a one-hidden-layer linear network. It has been shown that such over-parameterization can be beneficial to the generalization [29, 3, 44, 4].



(a) Static Neural Similarity (b) Dynamic Neural Similarity

Figure 2: Intuitive comparison between static neural similarity and dynamic neural similarity.

### 3.3 Learning Dynamic Neural Similarity

#### 3.3.1 Formulation

Besides the static neural similarity, we further propose to learn the neural similarity dynamically. The intuition behind is that the similarity measure should be adaptive to the input in order to achieve optimal flexibility. From a cognitive science perspective, it is also plausible to enable the network with dynamic inference [56, 31]. The difference between static and dynamic neural similarity is shown in Figure 2. Specifically, the dynamic neural similarity is generated dynamically using an additional neural network $\boldsymbol{M}_\theta(\cdot)$ with parameters $\theta$, namely $\boldsymbol{M}_s = \boldsymbol{M}_\theta(\boldsymbol{X})$. As a result, learning a dynamic neural similarity jointly with the network parameters is to solve the following optimization problem (without loss of generality, we simply use one neuron as an example in the following formulation):

$$\{\boldsymbol{W}, \theta\} = \arg\min_{\{\boldsymbol{W},\theta\}} \sum_i \mathcal{L}\big(y_i, \boldsymbol{W}^\top \boldsymbol{M}_\theta(\boldsymbol{X}_i)\boldsymbol{X}_i\big) \tag{3}$$

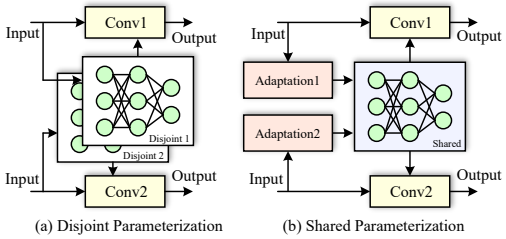
where $y_i$ is the ground truth value for $\boldsymbol{X}_i$, and $\mathcal{L}$ is some loss function. Both $\boldsymbol{W}$ and $\theta$ can be learned end-to-end using back-propagation. Note that, although $\boldsymbol{X}_i$ denotes the entire sample here, $\boldsymbol{X}_i$ will become the local patch of the input feature map in CNNs. For simplicity, we consider a one-neuron fully connected layer instead of a convolution layer. Due to the dynamic neural similarity, the equivalent weights $\boldsymbol{M}_\theta(\boldsymbol{X})^\top \boldsymbol{W}$ become a function of the input $\boldsymbol{X}$ and therefore construct a dynamic neural network. In fact, dynamic networks which generate the neuron weights entirely based on an additional neural network have poor generalization ability for recognition tasks [18]. In contrast, our dynamic NSN achieves a dedicate balance between generalization and flexibility by using neuron weights that are "semi-generated" (*i.e.*, part of the weights are statically and directly learned from supervisions, and the neural similarity matrix is generated dynamically from the input). Interestingly, we notice that hyperspherical convolution [39] can be viewed as a special case of dynamic neural similarity. One can see that its equivalent similarity matrix $\boldsymbol{M}_\theta(\boldsymbol{X}) = \text{diag}(\frac{1}{\|\boldsymbol{W}\|\|\boldsymbol{X}\|}, \cdots, \frac{1}{\|\boldsymbol{W}\|\|\boldsymbol{X}\|})$ also depends on the input feature map but does not have any parameter $\theta$.

**Hyperspherical learning with identity residuals**. In our experiments, we find that naively using a neural network to predict the neural similarity is very unstable during training, leading to difficultly in convergence (it requires a lot of tricks to converge). To address the training stability problem, we propose hyperspherical networks (SphereNet) [39] with identity residuals to serve as the neural similarity predictor. The convergence stability of hyperspherical learning over standard neural networks is discussed in [39, 37, 38, 36, 35, 34]. In order to further stablize the training, we learn the residual of an identity similarity matrix instead of directly learning the entire similarity matrix. Formally, the neural similarity predictor is written as $\boldsymbol{M}_\theta(\boldsymbol{X}) = \texttt{SphereNet}(\boldsymbol{X}; \theta) + \boldsymbol{I}$ where $\boldsymbol{I}$ is an identity matrix and $\texttt{SphereNet}(\boldsymbol{X}; \theta)$ denotes the hyperspherical network with parameter $\theta$ and input $\boldsymbol{X}$. To save parameters, we can use hyperspherical convolutional networks instead of hyperspherical fully-connected networks. One advantage of SphereNet is that each element of the output in SphereNet is bounded between $-1$ and 1 ($[0, 1]$ if using ReLU), making the similarity matrix bounded and well behaved. In contrast, the output is unbounded in a standard neural network, easily making some values of the similarity matrix dominantly large. Most importantly, SphereNet with identity residuals empirically yields not only more stable convergence but also stronger generalization.

### 3.3.2 Disjoint and Shared Parameterization in Neural Similarity Predictor

We mainly consider disjoint and shared parameterizations for the dynamic neural similarity predictor.

**Disjoint parameterization**. Disjoint parameterization treats every dynamic neural similarity independently. For each convolution kernel (*i.e.*, neuron), we use a disjoint neural network to predict the neural similarity matrix $\boldsymbol{M}_s$. A brief overview is given in Figure 3(a).




(a) Disjoint Parameterization (b) Shared Parameterization

Figure 3: Comparison between disjoint and shared parameterization for dynamic neural similarity predictor.

**Shared parameterization**. Assuming that there exists an intrinsic structure to predict the neural similarity from the input, we consider a shared neural network that produces the neural similarity matrix for different convolution kernels (usually convolution kernels of the same size). To address the dimension mismatch problem of the input feature map, we adopt an adaptation network (*e.g.*, convolution networks or fully-connected networks) to first transform the inputs to the same dimension. Note that, these adaptation networks are not shared across different kernels in general, but we can share those adaptation networks for the input feature map of the same size. An intuitive comparison between disjoint and shared parameterization is given in Figure 3 (Conv1 and Conv2 denote different convolution kernels). By sharing the neural similarity prediction networks across different kernels, the number of parameters used in total can be significantly reduced. Most importantly, this shared neural similarity network may be able to learn some meta-knowledge about the neural similarity.

### 3.4 Regularization for Neural Similarity

One of the largest advantages about the neural similarity formulation is that one can impose suitable regularizations on the neural similarity matrix $\boldsymbol{M}$ in different tasks. It gives us a way to incorporate our prior knowledge and problem understandings into the neural networks. The regularization on $\boldsymbol{M}$ controls the flexibility of the neural similarity. If we impose no constraints on $\boldsymbol{M}$, then it will have way too many parameters. Although it may be flexible enough, the generalization is not necessarily good. Instead we usually need to impose some constraints (*e.g.*, the block-diagonal with shared blocks, diagonal, etc.) in order to save parameters and improve generalization.

**Structural regularization**. As a typical example, requiring $\boldsymbol{M}$ to be a block-diagonal matrix with shared blocks is a strong structural regularization. Dilated convolution can be viewed as both structural and sparsity regularization on $\boldsymbol{M}_s$. In fact, more advanced structural regularizations could be considered. For instance, requiring $\boldsymbol{M}$ to be a symmetric or symmetric positive semi-definite matrix is also feasible (by using a Cholesky factorization $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^\top$ where $\boldsymbol{L}$ a learnable lower triangular matrix) and can largely limit the learnable class of similarity measures. Most importantly, structural regularizations may bring more geometric and semantic interpretability.

**Sparsity regularization**. Soft sparsity regularization on the matrix $\boldsymbol{M}_s$ can be enforced via a $\ell_1$-norm penalty. One can also impose a hard sparsity constraint to limit the non-zero values in $\boldsymbol{M}_s$, similar to [42]. It is also appealing to enforce sparsity-one pattern on $\boldsymbol{M}_s$, because it can construct efficient neural networks based on the shift operation in [59].

### 3.5 Joint Learning of Kernel Shape and Similarity

**Formulation**. NSL is also a unified framework for jointly learning the kernel shape and similarity measure. If we further factorize $\boldsymbol{M}_s$ to the multiplication of a diagonal Boolean matrix $\boldsymbol{D}$ and a similarity matrix $\boldsymbol{R}$, then the neural similarity can be parameterized as

$$f_M(\boldsymbol{W}, \boldsymbol{X}) = \boldsymbol{W}^\top \begin{bmatrix} \boldsymbol{DR} & & \\ & \ddots & \\ & & \boldsymbol{DR} \end{bmatrix} \boldsymbol{X} = \boldsymbol{W}^\top \cdot \underbrace{\begin{bmatrix} \boldsymbol{D} & & \\ & \ddots & \\ & & \boldsymbol{D} \end{bmatrix}}_{\text{Kernel Shape}} \cdot \underbrace{\begin{bmatrix} \boldsymbol{R} & & \\ & \ddots & \\ & & \boldsymbol{R} \end{bmatrix}}_{\text{Similarity Measure}} \cdot \boldsymbol{X} \tag{4}$$

where $\boldsymbol{D} = \text{diag}(d_1, \cdots, d_{HV})$ in which $d_i \in \{0, 1\}, \forall i$ is a Boolean value. $\boldsymbol{D}$ actually controls the shape of the kernel because it will spatially mask out some elements in the kernel. Specifically, because the diagonal of $\boldsymbol{D}$ is binary, some elements of $\boldsymbol{M}_s$ will become zero and therefore the kernel shape is controlled by $\boldsymbol{D}$. On the other hand, $\boldsymbol{R}$ still serves as the neural similarity matrix, similar

to the previous $\boldsymbol{M}_s$. $\boldsymbol{D}$ can also be viewed as masking out some elements of each column in $\boldsymbol{R}$. Interestingly, if we do not require the diagonal of $\boldsymbol{D}$ to be Boolean, then it will become a continuous spatial mask for the kernel shape.

**Optimization**. First of all, we only consider $\boldsymbol{D}$ to be static in both static and dynamic NSN. The optimization of $\boldsymbol{D}$ is non-trivial, because it is a Boolean matrix which is discretized and can not be optimized directly using gradients. Therefore, we use a heuristic approach to optimize $\boldsymbol{D}$. Specifically, we preserve a real-valued matrix $\boldsymbol{D}_r$ which is used to construct the Boolean matrix $\boldsymbol{D}$. We define $\boldsymbol{D}=\mathcal{I}(\boldsymbol{D}_r,\alpha)$ where $\mathcal{I}(v,\alpha)$ is an element-wise function that outputs 1 if $v>\alpha$ and 0 otherwise. $\alpha$ is a fixed threshold. We will update $\boldsymbol{D}_r$ with the following equation:

$$\{\boldsymbol{D}_r\}^{t+1}=\{\boldsymbol{D}_r\}^{t}-\eta\frac{\partial\mathcal{L}}{\partial\boldsymbol{D}} \tag{5}$$

where $\boldsymbol{D}_r$ is only computed in order to update $\boldsymbol{D}$. In both forward and backward passes, only $\boldsymbol{D}$ is used for computation, but $\boldsymbol{D}_r$ is used to generate $\boldsymbol{D}$. Essentially, the gradient w.r.t $\boldsymbol{D}$ serves as a noisy gradient for $\boldsymbol{D}_r$. Similar optimization strategy has also been employed in [22, 12, 42]. $\boldsymbol{R}$ is updated end-to-end using back-propagation. It is also easy to dynamically produce $\boldsymbol{D}$ with a neural network, but we do not consider this case for simplicity.

# 4 Neural Similarity Networks

After introducing the neural similarity learning of a single convolution kernel, we discuss how to construct a neural similarity network using this building block. In order to save parameters, we let all the convolution kernels of the same layer share the same neural similarity matrix, which means that we require the same convolution layer has the same similarity measure. We will empirically validate this design choice in Section 7.1. Stacking convolution layers with static (dynamic) neural similarity gives us static (dynamic) NSN. Note that, static NSN has the same number of parameters as standard CNN in deployment but yields better generalization ability. Compared to [28], dynamic NSN has better regularity on the convolution kernel and is also able to utilize the pretrained CNN models.

**Training from pretrained models**. In order to make use of the pretrained models, we can simply use the pretrained model as our backbone network (with all the weights loaded). Then we add the static or dynamic neural similarity modules to the convolutional kernels and train the neural similarity modules with backbone weights fixed until convergence. Optionally, we can finetune the entire network after the training of the neural similarity module. In contrast, the other dynamic networks [18, 28] are not able to take advantage of the pretrained models. Note that, it is not necessary for both static and dynamic NSN to be trained from pretrained models. They can also be trained from scratch (weights of both backbone and neural similarity module are optimized from random initialization) and still yield better result than the CNN baselines.

**Training and inference**. Similar to CNNs, both static and dynamic NSN can be trained end-to-end using mini-batch stochastic gradient descent. Apart from that the factorized form with $\boldsymbol{D}$ and $\boldsymbol{R}$ needs to be optimized using a heuristic approach, the training is basically the same as the standard CNN. In the inference stage, we can compute all the equivalent weights for static NSN in advance to speed up inference in practice. For dynamic NSN, the inference is also similar to the standard CNN with slightly more additional computations from the neural similarity module.

# 5 Theoretical Insights

## 5.1 Implicit Regularization Induced by NSL

As mentioned before, NSL can be viewed as a form of matrix multiplication where the weight matrix $\boldsymbol{W}$ is factorized as $\boldsymbol{M}^\top\boldsymbol{W}'$ ($\boldsymbol{W}'$ is the new weight matrix and $\boldsymbol{M}$ is the similarity matrix). Such factorization form not only provides more modeling and regularization flexibility, but it also introduces an implicit regularization (in gradient descent). The implicit regularization in matrix factorization is studied in [16]. We first compare the behavior of gradient descent on $\boldsymbol{W}$ and $\{\boldsymbol{W}',\boldsymbol{M}\}$ to observe the difference. We consider a simple example of a one-layer neural network with least square loss (*i.e.*, linear regression): $\min_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{W}):=\frac{1}{2}\sum_i\|\boldsymbol{y}_i-\boldsymbol{W}^\top\boldsymbol{X}_i\|_2^2$ where $\boldsymbol{W}\in\mathbb{R}^{n\times m}$ is the weight matrix for neurons, $\boldsymbol{y}_i\in\mathbb{R}^m$ is the target and $\boldsymbol{X}_i\in\mathbb{R}^n$ is the $i$-th sample. The behavior of gradient descent with infinitesimally small learning rate can be captured by the differential equation: $\dot{\boldsymbol{W}}_t+\nabla\mathcal{L}(\boldsymbol{W}_t)=0$ with an initial condition $\boldsymbol{W}_0$, where $\dot{\boldsymbol{W}}_t:=\frac{\mathrm{d}\boldsymbol{W}_t}{\mathrm{d}t}$. For NSL, the objective becomes $\min_{\{\boldsymbol{W}',\boldsymbol{M}\}}\mathcal{L}(\boldsymbol{W}',\boldsymbol{M}):=\frac{1}{2}\sum_i\|\boldsymbol{y}_i-\boldsymbol{W}'^\top\boldsymbol{M}\boldsymbol{X}_i\|_2^2$, so the corresponding differential equations

of gradient descent on $\boldsymbol{W}'$ and $\boldsymbol{M}$ are $\dot{\boldsymbol{W}}'_t + \nabla_{\boldsymbol{W}'}\mathcal{L}(\boldsymbol{W}'_t, \boldsymbol{M}) = 0$ and $\dot{\boldsymbol{M}}_t + \nabla_{\boldsymbol{M}}\mathcal{L}(\boldsymbol{W}'_t, \boldsymbol{M}) = 0$, respectively (with initial condition $\boldsymbol{W}'_0$ and $\boldsymbol{M}_0$). Therefore, the gradient flows of the standard update on $\boldsymbol{W}$ and the factorized NSL update on $\{\boldsymbol{W}', \boldsymbol{M}\}$ can be expressed as

$$\begin{aligned} \text{Standard Derivative: } \dot{\boldsymbol{W}}_t &= \sum\nolimits_i \boldsymbol{X}_i(\boldsymbol{y}_i - \boldsymbol{W}_t^\top \boldsymbol{X}_i)^\top = \sum\nolimits_i \boldsymbol{X}_i(\boldsymbol{r}_t^i)^\top \quad (\text{Define } \boldsymbol{r}_t^i = \boldsymbol{y}_i - \boldsymbol{W}_t^\top \boldsymbol{X}_i) \\ \text{NSL Derivative: } \dot{\boldsymbol{W}}_t &= \boldsymbol{M}_t^\top \dot{\boldsymbol{W}}'_t + \dot{\boldsymbol{M}}_t^\top \boldsymbol{W}'_t = \boldsymbol{M}_t^\top \boldsymbol{M}_t \sum\nolimits_i \boldsymbol{X}_i(\boldsymbol{r}_t^i)^\top + \sum\nolimits_i \boldsymbol{X}_i(\boldsymbol{r}_t^i)^\top \boldsymbol{W'}_t^\top \boldsymbol{W}'_t \end{aligned} \tag{6}$$

from which we observe that the gradient dynamics of the NSL update is very different from the gradient dynamics of the standard update. Therefore, NSL may introduce a regularization effect that is different from the standard update, and we argue that such implicit regularization induced by NSL is beneficial to the generalization power. [16] conjectures that optimizing matrix factorization with gradient descent implicitly regularizes the solution towards minimum nuclear norm. [5] extends the analysis of implicit regularization to deep matrix factorization (*i.e.*, multi-layer linear neural networks) and shows that multi-layer matrix factorization enhances an implicit tendency towards low-rank solution. [15, 27] show that gradient descent converges to the maximum margin solution in linear neural networks for binary classification of separable data. More interestingly, [5] argues that implicit regularization in matrix factorization may not be captured using simple mathematical norms.

### 5.2 Connection to Dynamical Systems

Classic dynamic neural unit (DNU) [17] receives not only external inputs but also state feedback signals from themselves and other neurons. A general mathematical model of an isolated DNU is given by a differential equation $\dot{\boldsymbol{x}}(t) = -\alpha \boldsymbol{x}(t) + f(\boldsymbol{w}, \boldsymbol{x}(t), \boldsymbol{u}),\ \boldsymbol{y}(t) = g(\boldsymbol{x}(t))$ where $\boldsymbol{x}$ is DNU's neural state, $\boldsymbol{w}_i$ is the weight vector, $\boldsymbol{u}$ is the external input, $f(\cdot)$ is the nonlinear activation and $g(\cdot)$ is DNU's output. As a dynamical system, the output of DNU depends on both the external input and the output time stamp. The neural state trajectory also depends on the equilibrium convergence property of DNU. Different from DNU, dynamic NSN does not have the state feedback and self-recurrence. Instead it realizes the dynamic output with a neural similarity generator that changes the equivalent weight matrix adaptively based on the input. However, it will be interesting to combine self-recurrence to NSL, since it can save parameters and strengthen the approximation power.

Recent work [9, 41, 49, 58] shows that many existing deep neural networks can be consider as different numerical schemes approximating an ordinary differential equation (ODE). NSN with certain similarity design is also equivalent to approximating ODEs. For example, $f_{\boldsymbol{M}} = \boldsymbol{W}^\top(\tilde{\boldsymbol{W}} + \boldsymbol{M})\boldsymbol{X} = X_m + \boldsymbol{W}^\top \boldsymbol{M}\boldsymbol{X}$ where $\boldsymbol{W}^\top \tilde{\boldsymbol{W}} = \text{Diag}(0, \cdots, 0, 1, 0, \cdots, 0)$ (1 lies in the center location) can be written as $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \cdot g_n(\boldsymbol{x}_n)$ (*i.e.*, ResNet) where $\boldsymbol{x}_n$ is the input feature map at depth $n$ and $g_n(\cdot)$ is the transformation at depth $n$. It is one step of forward Euler discretization of the ODE $\boldsymbol{x}_t = g(\boldsymbol{x}, t)$. Different neural similarity designs correspond to different iterative method for ODEs.

## 6 Discussions

**Connection and comparison to the existing works**. Static NSN is a direct generalization from the standard CNN, and can be viewed as a factorized learning (with optional regularizations) of convolution kernels. Dynamic NSN can be viewed as a non-trivial generalization of hyperspherical convolution [39] in the sense that hyperspherical convolution is also input-dependent and can be viewed as a special case of $\boldsymbol{M}$ being $\frac{1}{\|\boldsymbol{W}\|\|\boldsymbol{X}\|}\boldsymbol{I}$. Compared to dynamic filter networks [28], dynamic NSN achieves a better trade-off between flexiblity and generalization. Dynamic filter networks are very flexible since the weights are completely generated using another network, but it yields unsatisfactory image recognition accuracy. In contrast, dynamic NSN imposes strong regularizations on the weights and is less flexible than dynamic filter networks, but it has much stronger generalization ability while still being dynamic. When $\boldsymbol{M}$ has no constraints, our dynamic NSN will become essentially equivalent to the dynamic filter network. [11] proposes to dynamically select filters to perform inference, while NSL dynamically estimates a similarity measure.

**Dynamic NSN is a high-order function of input**. Dynamic NSN outputs $\boldsymbol{W}^\top \boldsymbol{M}_\theta(\boldsymbol{X})\boldsymbol{X}$. Assuming $\boldsymbol{M}_\theta(\boldsymbol{X})$ is a one-layer neural network, *i.e.*, $\boldsymbol{M}_\theta(\boldsymbol{X}) = \boldsymbol{W}'\boldsymbol{X}^\top$. Then the one-layer dynamic NSN is written as $\boldsymbol{W}^\top \boldsymbol{W}' \boldsymbol{X}^\top \boldsymbol{X}$ which is a quadratic function of $\boldsymbol{X}$. In general, $\boldsymbol{M}_\theta(\boldsymbol{X})$ is much more nonlinear, so one-layer dynamic NSN is naturally a high-order function of the input $\boldsymbol{X}$. Therefore, dynamic NSN has stronger approximation ability and flexibility than the standard convolution.

**Self-attention as a global dynamic neural similarity**. Since self-attention [62] is also a high-order function of input, it can also be viewed as a form of dynamic neural similarity. We define a novel global neural similarity that can reduce to self-attention in Appendix B.

# 7 Applications

## 7.1 Generic Visual Recognition

**Experimental settings**. For fair comparison, the backbone network architecture is the same in each experiment. We will mostly use VGG-like plain CNN architecture. Detailed structures for baselines and NSN are provided in Appendix A. For CIFAR10 and CIFAR100, we follow the same augmentation settings from [21]. For Imagenet 2012 dataset, we mostly follow the settings in [30]. Batch normalization, ReLU, mini-batch 128, and SGD with momentum 0.9 are used as default in all methods. For CIFAR-10 and CIFAR-100, we start momentum SGD with the learning rate 0.1. The learning rate is divided by 10 at 34K, 54K iterations and the training stops at 64K. For ImageNet, the learning rate starts with 0.1, and is divided by 10 at 200K, 375K, 550K iterations (finsihed at 600K).

**Different neural similarity predictor**. We consider two types of architectures: CNN and SphereNet [39] for the neural similarity predictor of dynamic NSN. We experiment on CIFAR-10 and DNS ($\boldsymbol{M}_s$ is diagonal) is used in NSN. Table 1 shows that SphereNet works better than standard CNN as a neural similarity predictor. It is because SphereNet has better convergence properties can can stablize NSN training. In fact, dynamic NSN can not converge if trivially applying CNN to the predictor, and we have to perform normalization (or sigmoid activation) to the predictor's final output to make it converge. In contrast, SphereNet can make dynamic NSN converge easily and perform better. Therefore, we will use SphereNet as the neural similarity predictor for dynamic NSN by default.

| Method | Error (%) |
|---|---|
| Baseline CNN | 7.78 |
| Dynamic NSN (CNN) | 7.04 |
| Dynamic NSN (SN) | **6.85** |

Table 1: Predictor Network.

**Joint learning of kernel shape and similarity**. We now evaluate how jointly learning kernel shape and similarity can improve NSN. We use CIFAR-10 in the experiment. For both static and dynamic NSN, we use DNS ($\boldsymbol{M}_s$ is a diagonal matrix). For dynamic NSN, we use SphereNet [39] as the neural similarity predictor. Table 2 show that joint learning $\boldsymbol{D}$ and $\boldsymbol{R}$ performs better than simply learning $\boldsymbol{M}_s$. However, to be simple, we will still learn a single $\boldsymbol{M}_s$ in the other experiments.

| Method | Error (%) |
|---|---|
| Baseline CNN | 7.78 |
| Static NSN | 7.15 |
| Static NSN (J) | 6.92 |
| Dynamic NSN | 6.85 |
| Dynamic NSN (J) | **6.64** |

Table 2: Joint learning.

**Shared v.s. disjoint dynamic NSN**. We evaluate the shared and disjoint parameterization for the neural similarity predictor. We use CIFAR-10 in the experiment. For both static and dynamic NSN, we use DNS. Table 3 shows that the shared similarity predictor performs slightly worse than the disjoint one, but the shared one saves nearly half of the parameters used in the disjoint one.

| Method | Error (%) |
|---|---|
| Baseline CNN | 7.78 |
| Dynamic NSN (Shared) | 7.20 |
| Dynamic NSN (Disjoint) | **6.85** |

Table 3: Predictor parameterization.

**CIFAR-10/100**. We comprehensively evaluate both static and dynamic NSN on CIFAR-10 and CIFAR-100. All dynamic NSN variants use SphereNet as neural similarity predictor. Both DNS and UNS are experimented for comparison. Because dynamic NSN uses slightly more parameters than the baseline CNN, we construct a new baseline CNN++ by making the baseline CNN deeper and wider such that the number of parameters is slightly larger than all variants of NSN. The results in Table 4 verify the superiority of both static and dynamic NSN. Our dynamic NSN outperforms both baseline CNN and baseline CNN++ by a considerable margin. Moreover, one can observe that dynamic NSN performs generally better than static NSN, showing that dynamic inference can be beneficial for the image recognition task. Both DNS and UNS perform similarly on CIFAR-10 and CIFAR-100, indicating that DNS is already flexible enough for the image recognition task.

| Method | CIFAR-10 | CIFAR-100 |
|---|---|---|
| Baseline CNN | 7.78 | 28.95 |
| Baseline CNN++ | 7.29 | 28.70 |
| Static NSN w/ DNS | 7.15 | 28.35 |
| Static NSN w/ UNS | 7.38 | 28.11 |
| Dynamic NSN w/ DNS | 6.85 | **27.81** |
| Dynamic NSN w/ UNS | **6.5** | 28.02 |

Table 4: Error (%) on CIFAR-10 & CIFAR-100.

**ImageNet-2012**. In order to be parameter-efficient, we evaluate the dynamic NSN with DNS on the ImageNet-2012 dataset. The backbone network is a VGG-like 10-layer plain CNN, so the absolute performance is not state-of-the-art. However, the purpose here is to perform apple-to-apple fair comparison. Using the same backbone network, dynamic NSN is significantly and consistently better than both baseline CNN and CNN++. Note that, baseline CNN++ is a deeper and wider version of baseline CNN. The results in Table 5 show that dynamic NSN yields strong generalization ability with the

| Method | Top-1 | Top-5 | # params |
|---|---|---|---|
| Baseline CNN | 42.72 | 19.11 | 8.90M |
| Baseline CNN++ | 42.11 | 18.98 | 9.71M |
| Dynamic NSN w/ DNS | **40.61** | **18.04** | 9.61M |

Table 5: Validation error (%) on ImageNet-2012.

same number of parameter, and most importantly, the experiments demonstrated that the dynamic inference mechanism can work well in a challenging large-scale image recognition task.

### 7.2 Few-Shot Learning

**Static NSN**. It is very natural to apply static NSN to the few-shot learning. Similar to the finetuning baseline, we first train a backbone network using the base class data. When it comes to the testing stage, we first finetune both the static neural similarity matrix and the classifier on the novel class data and then use the finetuned classifier to make prediction. Note that, in order to use a pretrained backbone, we need to initialize the neural similarity matrix with an identity matrix. Due to the strong regularity that we imposed to the mete-similarity matrix, static NSN is able to preserve rich information from the pretrained model while quickly adapting to the novel class data.

**Dynamic NSN**. Dynamic NSN is very suitable for the few-shot learning due to its dynamic nature. Its filters are conditioned on the input. Because dynamic NSN is able to learn a meta-information about the similarity measure, so its intermediate layers do not need to be finetuned in the testing stage. From a high-level perspective, dynamic NSN shares some similarities with MAML [14] in the sense that dynamic NSN learns to transform its filters with a projection matrix, while MAML transforms its filters using gradient updates during inference. We directly train the dynamic NSN on the base class data. In the testing stage, we first retrain the classifiers using the novel class data, and then classify the query image using the dynamic NSN and the retrained classifier.

**Meta-learned static NSN**. Inspired by MAML [14], we propose to meta-learn the neural similarity. We pretrain the network on the base classes with identity similarity and then meta-learned the neural similarity and classifiers similar to MAML. The meta-learned static NSN dynamically transforms its filters via projection using the gradients, similar to MAML. The meta-optimization is given by

$$\min_{\boldsymbol{M}} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}(f_{\boldsymbol{M}'}) \quad \text{s.t. } \boldsymbol{M}' = \boldsymbol{M} - \eta \nabla_{\boldsymbol{M}} \mathcal{L}_{\tau_i}(f_{\boldsymbol{M}}) \tag{7}$$

which aims to learn a good initialization for the static neural similarity matrix. During testing, the procedure exactly follows MAML [14] except that the meta-learned static NSN only updates the neural similarity matrix with gradients. The pretrained model is recently shown to perform well with certain normalization [10]. Meta-learned static NSN is able to take full advantage of the pretrained model, and can be viewed as an interpolation between the pretrained model and MAML [14]. In fact, dynamic neural similarity can be also meta-learned similarly, which is left for future investigation.

**Experiment on Mini-ImageNet**. The experimental protocol is the same as [46, 14]. Following [46], we use 4 convolution layers with 32 $3 \times 3$ filters per layer. Batch normalization [23], ReLU non-linearity and $2 \times 2$ pooling are used. For all the NSN variants, we use the best setup and hyperparameters. The results in Table 6 show that all of our proposed three few-shot learning strategies work reasonably well. The dynamic NSN outperforms the other competitive methods by a considerably large margin. Static NSN works better than most exisint methods. Meta-learned static NSN also shows obvious advantages over its direct competitor MAML. Moreover, we also compare with the recent state-of-the-art method LEO [48] which uses features from ResNet-28. Our dynamic NSN with the CNN-9 backbone achieves $77.44\%$ accuracy, which is comparable to LEO but ours has much fewer network parameters. This experiment further validates the strong generalization ability of all NSN variants.

| Method | Backbone | 5-shot Accuracy |
|---|---|---|
| Finetuning Baseline [46] | CNN-4 | $49.79 \pm 0.79$ |
| Nearest Neightbor Baseline [46] | CNN-4 | $51.04 \pm 0.65$ |
| MatchingNet [46] | CNN-4 | $55.31 \pm 0.73$ |
| ProtoNet [52] | CNN-4 | $68.20 \pm 0.66$ |
| MAML [14] | CNN-4 | $63.15 \pm 0.91$ |
| RelationNet [54] | CNN-4 | $65.32 \pm 0.70$ |
| Static NSN (ours) | CNN-4 | $65.74 \pm 0.68$ |
| Meta-learned static NSN (ours) | CNN-4 | $66.21 \pm 0.69$ |
| Dynamic NSN (ours) | CNN-4 | $\mathbf{71.26 \pm 0.65}$ |
| Discriminative k-shot [6] | ResNet-34 | $73.90 \pm 0.30$ |
| Tadam [45] | ResNet-12 | $76.7 \pm 0.3$ |
| LEO [48] | ResNet-28 | $\mathbf{77.59 \pm 0.12}$ |
| Dynamic NSN (ours) | CNN-9 | $77.44 \pm 0.63$ |

Table 6: Few-shot classification on Mini-Imagenet test set.

## 8 Concluding Remarks

We have proposed a general yet powerful framework to generalize traditional convolution with the *neural similarity*. Our framework can capture the similarity structure that lies in our data of interest, and regularizing the similarity to accommodate the nature of input dataset may yield better performance. Our experiments on image recognition and few-shot learning show the potential of our framework being flexible, generalizable and interpretable. This framework can be further applied to more applications, *e.g.*, semantic segmentation, and may inspire different threads of research.

## Acknowledgements

Weiyang Liu was supported in part by Baidu Fellowship and Nvidia GPU Grant. Le Song was supported in part by NSF grants CDS&E-1900017 D3SC, CCF-1836936 FMitF, IIS-1841351, CAREER IIS-1350983, DARPA Program on Learning with Less Labels.

## References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, 2016. 3

[2] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017. 3

[3] Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018. 4

[4] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*, 2018. 4

[5] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. *arXiv preprint arXiv:1905.13655*, 2019. 2, 4, 7

[6] Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartłomiej Świątkowski, Bernhard Schölkopf, and Richard E Turner. Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*, 2017. 9

[7] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche . . . , 1990. 3

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2018. 1, 2

[9] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NIPS*, 2018. 7

[10] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *ICLR*, 2019. 3, 9

[11] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network. *arXiv preprint arXiv:1811.11205*, 2018. 3, 7

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015. 6

[13] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017. 1, 2

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 2, 3, 9, 16

[15] Suriya Gunasekar, Jason D Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. In *NIPS*, 2018. 7

[16] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *NIPS*, 2017. 2, 4, 6, 7

[17] Madan Gupta, Liang Jin, and Noriyasu Homma. *Static and dynamic neural networks: from fundamentals to advanced theory*. John Wiley & Sons, 2004. 7

[18] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 2, 4, 6

[19] Bharath Hariharan and Ross Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *ICCV*, 2017. 3

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 8

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, 2016. 6

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 9

[24] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015. 1, 15

[25] Yunho Jeon and Junmo Kim. Active convolution: Learning the shape of convolution for image classification. In *CVPR*, 2017. 1, 2

[26] Yunho Jeon and Junmo Kim. Constructing fast network through deconstruction of convolution. In *NIPS*, 2018. 1

[27] Ziwei Ji and Matus Telgarsky. Gradient descent aligns the layers of deep linear networks. *arXiv preprint arXiv:1810.02032*, 2018. 7

[28] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *NIPS*, 2016. 2, 6, 7

[29] Kenji Kawaguchi. Deep learning without poor local minima. In *NIPS*, 2016. 4

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 8

[31] Samuel J Leven and Daniel S Levine. Multiattribute decision making in context: A dynamic neural network methodology. *Cognitive Science*, 20(2):271–299, 1996. 4

[32] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016. 3

[33] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *COLT*, 2018. 4

[34] Rongmei Lin, Weiyang Liu, Zhen Liu, Chen Feng, Zhiding Yu, James M. Rehg, Li Xiong, and Le Song. Compressive hyperspherical energy minimization. *arXiv preprint arXiv:1906.04892*, 2019. 4

[35] Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. Learning towards minimum hyperspherical energy. In *NIPS*, 2018. 4

[36] Weiyang Liu, Zhen Liu, Zhiding Yu, Bo Dai, Rongmei Lin, Yisen Wang, James M Rehg, and Le Song. Decoupled networks. *CVPR*, 2018. 1, 2, 4, 13

[37] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, 2017. 4

[38] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016. 4

[39] Weiyang Liu, Yan-Ming Zhang, Xingguo Li, Zhiding Yu, Bo Dai, Tuo Zhao, and Le Song. Deep hyperspherical learning. In *NIPS*, 2017. 1, 2, 3, 4, 7, 8, 13, 15

[40] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[41] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *ICML*, 2018. 7

[42] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. *arXiv preprint arXiv:1801.06519*, 2018. 5, 6

[43] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *arXiv preprint arXiv:1703.00837*, 2017. 3

[44] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018. 4

[45] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018. 9

[46] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. 2, 3, 9

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1

[48] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018. 9

[49] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018. 7

[50] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. 3

[51] Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *NeurIPS*, 2019. 3

[52] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017. 3, 9

[53] Yu-Chuan Su and Kristen Grauman. Learning spherical convolution for fast features from 360 imagery. In *NIPS*, 2017. 1

[54] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018. 3, 9

[55] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NIPS*, 2016. 3

[56] Yingxu Wang. The cognitive processes of formal inferences. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 1(4):75–86, 2007. 4

[57] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. *arXiv preprint arXiv:1801.05401*, 2018. 3

[58] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017. 7

[59] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *CVPR*, 2018. 1, 5

[60] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, 2003. 3

[61] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 1, 2

[62] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *ICML*, 2019. 7, 15

[63] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. *arXiv preprint arXiv:1811.11168*, 2018. 1

# Appendix

## A Experimental Details

| Layer | CNN-9 (CIFAR-10/100) | CNN-10 (ImageNet-2012) |
|---|---|---|
| Conv1.x | [3×3, 32]×3 | [7×7, 64], Stride 2<br>3×3, Max Pooling, Stride 2<br>[3×3, 64]×3 |
| Pool1 | 2×2 Max Pooling, Stride 2 | |
| Conv2.x | [3×3, 64]×3 | [3×3, 128]×3 |
| Pool2 | 2×2 Max Pooling, Stride 2 | |
| Conv3.x | [3×3, 128]×3 | [3×3, 256]×3 |
| Pool3 | 2×2 Max Pooling, Stride 2 | |
| Fully Connected | 256 | 512 |

Table 7: Our plain CNN architectures with different convolutional layers. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.

In CIFAR-10/100, our DNS predictor utilizes the structure of "Input - 64 hidden units (SphereConv, only x being normalized) - 9 output units (no ReLU)", and our UNS predictor uses the structure of "Input - 128 hidden units (SphereConv, only x being normalized) - 81 output units (no ReLU)". Note that, SphereConv comes from [39]. For DNS predictor, we will add an identity matrix to the output of the predictor to improve its initialization point. For UNS predictor, we simply use the output of the network as the neural similarity matrix. For CIFAR-10/100, we use the same training data augmentation as in [36].

On ImageNet-2012 dataset, the DNS predictor uses the structure of "Input - 32 hidden units (SphereConv, only input is normalized) - 9 output units (no ReLU)"

For meta-learning on Mini-ImageNet dataset, we use DNS for all experiments. For our non-MAML baseline and NSN models, we train the models on both training and validation set of Mini-ImageNet, while we train the MAML-trained static NSN model with the training set only.

For non-MAML training, we use Adam optimizer with $\text{lr} = 1e-3$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. For non-MAML testing, we finetune the model on query sets with SGD with $\text{lr} = 0.01$, $\text{momentum} = 0.9$, $\text{dampening} = 0.9$ and $\text{weight decay} = 0.001$ for 100 epochs.

In non-MAML static NSN experiments, We train the whole model from scratch and fix the static similarity matrices to be identity; during testing, we only finetune the matrices and the classifier. The (non-MAML) dynamic NSN experiments are similar excluding that we have no static similarity matrix anymore.

In MAML-trained static NSN experiments, we use the trained non-MAML static NSN as a pretrained model, and meta-train both the static similarity matrices and the classifier. For the MAML gradient steps on the support set, we first run 5 gradient steps on both the static similarity matrices and the classifier with $\text{step size} = 0.2$. Because MAML-trained static NSN has less capacity for finetuning on query sets, we run additional 20 gradient steps with the same step size but on the classifier only.

The CNN-9 network architecture of dynamic NSN on Mini-ImageNet is the same as the one we use on CIFAR-10/100.

Our code is publicly available at https://github.com/wy1iu/NSL. For all the missing experimental details, please refer to our code repository.

# B Local and Global Neural Similarity

## B.1 Formulation

The original dynamic neural similarity is performed in a local fashion, meaning that the similarity matrix operates on the local patch instead of the entire input feature map. We extend the original neural similarity from operating on the local patch to operating on the global input feature map. As a result, we call the original neural similarity as *local neural similarity* (LNS). Specifically, for the input feature map $\boldsymbol{X} \in \mathbb{R}^{m\times m\times c}$ with size $m \times m \times c$ and a convolution kernel $\boldsymbol{W} \in \mathbb{R}^{k\times k\times c}$ of size $k \times k \times c$ (stride 1 and dimension-preserving padding), the *global neural similarity* (GNS) for convolution is formulated as

$$F_{\boldsymbol{M}}^{\mathcal{G}} = \boldsymbol{W}_{\mathcal{G}}^{\top} \boldsymbol{M}_{\mathcal{G}} \boldsymbol{X}_F \tag{8}$$

where $F_{\boldsymbol{M}}^{\mathcal{G}}$ is a vector of size $mm\times 1$ which is different from the standard neural similarity (with stride 1 and dimension-preserving padding), $\boldsymbol{W}_{\mathcal{G}}$ is the block circulant matrix (a special case of Toeplitz matrix) for performing 2D convolution, $\boldsymbol{M}_{\mathcal{G}}$ is the neural similarity matrix, and $\boldsymbol{X}_F \in \mathbb{R}^{mmc\times 1}$ is flattened vector of the input feature map $\boldsymbol{X}$. The block circulant matrix $\boldsymbol{W}_{\mathcal{G}}^s$ converts the 2D convolution into a matrix multiplication. The GNS matrix $\boldsymbol{M}_{\mathcal{G}} \in \mathbb{R}^{mmc\times mmc}$ usually takes the following block-diagonal form with the same block matrix $\boldsymbol{M}_{\mathcal{G}}^s$:

$$\boldsymbol{M}_{\mathcal{G}} = \begin{bmatrix} \boldsymbol{M}_{\mathcal{G}}^s \in \mathbb{R}^{mm\times mm} & & \\ & \ddots & \\ & & \boldsymbol{M}_{\mathcal{G}}^s \in \mathbb{R}^{mm\times mm} \end{bmatrix} \in \mathbb{R}^{mmc\times mmc} \tag{9}$$

where there are $c$ matrices $\boldsymbol{M}_{\mathcal{G}}^s \in \mathbb{R}^{mm\times mm}$. Note that, if $\boldsymbol{M}_{\mathcal{G}}^s$ is a diagonal matrix, then it will serve as a role similar to a spatial attention mask for the input feature map (The spatial attention map is also shared across different channels of the input feature map if we require $\boldsymbol{M}_{\mathcal{G}}$ to be a block-diagonal matrix with sharing blocks).

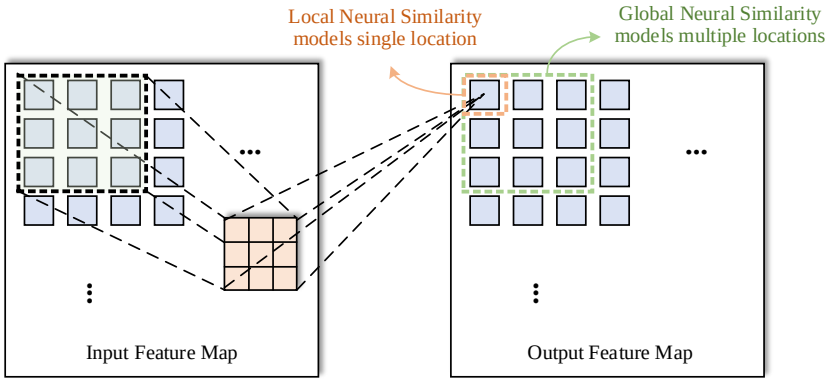



Figure 4: Comparison between neural similarity and generalized neural similarity.

**LNS vs. GNS**. The difference between neural similarity and global neural similarity lies in whether the convolution is taken into consideration. For the original neural similarity, although we apply it to convolution kernel, we do not consider the sliding window operation. Instead, we only consider the local inner product operation and combine the neural similarity matrix locally. For global neural similarity, we take the convolution into account and transform the original convolution operation to a matrix multiplication (using Toeplitz matrix). An intuitive comparison is given in Figure 4. From the computation perspective, GNS and LNS are not equivalent in general. For example, we consider the case where both $\boldsymbol{M}$ in LNS and $\boldsymbol{M}_{\mathcal{G}}$ in GNS are diagonal matrix. Although both similarity matrix share the same structure, the equivalent outputs are totally different. For LNS, each position in the output feature map is obtained with a weighted inner product. Diagonal $\boldsymbol{M}$ serves as the element-wise weighting factor for computing the inner product, and the same set of weighting factor will repeatedly be applied to every sliding window (with the same size of convolution kernel) in the input feature map. In contrast, Diagonal $\boldsymbol{M}_{\mathcal{G}}$ in GNS serves as a spatial attention mask for the entire input feature map. It is equivalent to first compute a Hadamard product between the input feature map and the spatial mask induced by $\boldsymbol{M}_{\mathcal{G}}$, and then perform standard 2D convolution with kernel $\boldsymbol{W}$

on the result. GNS and LNS are only equivalent when GNS only considers an input feature map of size $1 \times 1 \times c$ (*i.e.*, the input feature map contains only one spatial location). Both static GNS and dynamic GNS are similar to the corresponding variant in LNS.

**Self-attention as Dynamic GNS**. Dynamic GNS can be written as follows:

$$F_M^{\mathcal{G}} = \boldsymbol{W}_{\mathcal{G}}^{\top} \cdot \boldsymbol{M}_{\mathcal{G}}(\boldsymbol{X}; \theta) \cdot \boldsymbol{X}_F \tag{10}$$

where $\boldsymbol{M}_{\mathcal{G}}(\boldsymbol{X}; \theta)$ is a function dependent on $\boldsymbol{X}$. We show that self-attention [62] is a special case of dynamic GNS. We first resize the dimension of $\boldsymbol{X}_F$ in Eq. (10) to $mm \times c$ when multiplying with $\boldsymbol{M}_{\mathcal{G}}(\boldsymbol{X}; \theta)$. Then after the multiplication, we resize $\boldsymbol{X}_F$ back to $m \times m \times c$. We consider the case of $\boldsymbol{M}_{\mathcal{G}}(\boldsymbol{X}; \theta) = G_1(\boldsymbol{X}) G_2(\boldsymbol{X})^{\top}$ where $G_1(\boldsymbol{X})$ is a $1 \times 1$ convolution that transforms $\boldsymbol{X} \in \mathbb{R}^{m \times m \times c}$ to a new feature map with size $m \times m \times c$ and then resize the new feature map to $G_1(\boldsymbol{X}) \in \mathbb{R}^{mm \times c}$. $G_2(\boldsymbol{X})$ is also a combination of $1 \times 1$ convolution and a resize operation, same as $G_1(\boldsymbol{X})$. One can see that $\boldsymbol{M}_{\mathcal{G}}(\boldsymbol{X}; \theta) = G_1(\boldsymbol{X}) G_2(\boldsymbol{X})^{\top}$ is essentially a self-attention map. By multiplying the self attention map back to the feature map, we have exactly the same self-attention mechanism as in [62]. As a form of dynamic GNS, the self attention operation can be written as

$$F_M^{\text{self-attention}} = \boldsymbol{W}_{\mathcal{G}}^{\top} \cdot \text{Resize}\Big( G_1(\boldsymbol{X}) G_2(\boldsymbol{X})^{\top} \cdot \text{Resize}(\boldsymbol{X}_F, mm, c), mmc, 1 \Big) \tag{11}$$

**Connection to spatial transformer**. Dynamic GNS is also closely related to spatial transformer networks [24]. Spatial transformer contains localization network, grid generator, and sampler. In fact, the localization networks take the feature map as input and output parameters for grid generator. Then the grid generator and the sampler transform the feature map. The pipeline resembles the neural similarity learning, and can be viewed as a special case of GNS.

### B.2 Preliminary Experiments

We implement self-attention with our dynamic GNS in both standard CNN and SphereNet [39], and then evaluate them with image classification on CIFAR-10. To simplify evaluation, we only perform mild data augmentation on CIFAR-10 training set, unlike the main paper. We use the CNN-9 architecture in [39] for both standard CNN and SphereNet, but we use 128, 192 and 256 as the number of filters in Conv1.x, Conv2.x and Conv3.x. For more details, refer to our code repository. Table 8 shows the results of CNN and SphereNet with and without self-attention. We can see that self-attention does not seem to bring too many gains to the image classification task. However, we observe that using SphereNet can boost the advantages of self-attention and achieve considerable accuracy gain.

| Method | Accuracy (%) |
|---|---|
| CNN | 90.86 |
| CNN w/ self-attention | 90.69 |
| SphereNet | 91.31 |
| SphereNet w/ self-attention | **91.76** |

Table 8: CNN and SphereNet with self-attention (dynamic GNS) on CIFAR-10.

## C Significance of NSL for Meta-Learning

One of the key in MAML [14] is that it uses the gradient update to make the network parameter dynamically dependent on the input. Essentially, we can view it as a novel realization of dynamic neural networks except that the network parameters are dynamically changed following the gradient direction. Different from MAML, dynamic NSL realizes the dynamic neural network with an additional neural similarity predictor (*i.e.*, an additional neural network). Essentially, we learn the most suitable direction to update the network parameters adaptively based on the input. As a result, the biggest difference between MAML and dynamic NSL is how we make the network parameters dynamically dependent on the input. MAML uses the gradient information from the gallery set in testing stage, while dynamic NSL learns how to change the network parameters with a neural network during training. Empirically, we find that dynamic NSL outperforms MAML with a significant margin, partially validating that using a neural network to approximate the update of the network parameters yields better generalizability.