

# Subspace-Configurable Networks

Olga Saukh<sup>\*1,2</sup>, Dong Wang<sup>\*1</sup>, Xiaoxi He<sup>3</sup>, and Lothar Thiele<sup>4</sup>

<sup>1</sup>Graz University of Technology, Austria

<sup>2</sup>Complexity Science Hub Vienna, Austria

<sup>3</sup>University of Macao, China

<sup>4</sup>ETH Zurich, Switzerland

{saukh,dong.wang}@tugraz.at, hexiaoxi@um.edu.mo, thiele@tik.ee.ethz.ch

## Abstract

While the deployment of deep learning models on edge devices is increasing, these models often lack robustness when faced with dynamic changes in sensed data. This can be attributed to sensor drift, or variations in the data compared to what was used during offline training due to factors such as specific sensor placement or naturally changing sensing conditions. Hence, achieving the desired robustness necessitates the utilization of either an invariant architecture or specialized training approaches, like data augmentation. Alternatively, input transformations can be treated as a domain shift problem, and solved by post-deployment model adaptation. In this paper, we train a parameterized subspace of *configurable networks*, where an optimal network for a particular parameter setting is part of this subspace. The obtained subspace is low-dimensional and has a surprisingly simple structure even for complex, non-invertible transformations of the input, leading to an exceptionally high efficiency of subspace-configurable networks (SCNs) when limited storage and computing resources are at stake. We evaluate SCNs on a wide range of standard datasets, architectures, and transformations, and demonstrate their power on resource-constrained IoT devices, where they can take up to 2.4 times less RAM and be 7.6 times faster at inference time than a model that achieves the same test set accuracy, yet is trained with data augmentations to cover the desired range of input transformations. The source code is available online.\*

## 1 Introduction

In real-world applications of deep learning, it is common for systems to encounter environments that differ from those considered during model training. There are many reasons for this difference between training and post-deployment such as sensor drift and sensor variations, or domain shift in the data compared to what was used during offline training due to factors such as specific sensor placements, constraints or naturally changing sensing conditions. For example, these differences manifest as transformations of the input data, such as rotation, scaling, and translation. For a deep learning model to perform well in post-deployment settings, it must be capable of addressing these discrepancies and sustain high performance despite the differences between the training and the post-deployment environments. These discrepancies may arise either statically during the initial deployment or dynamically throughout the duration of deployment.

To address the above challenge, there are two primary approaches: designing robust, invariant models and employing domain adaptation techniques. Both strategies aim to mitigate the performance degradation resulting from the discrepancies between the source and the target domains.

---

\*Equal contribution

\*<https://github.com/osaukh/subspace-configurable-networks.git>

Invariant architectures focus on making the model robust, insensitive, or invariant to specific transformations of the input data. This can be achieved by various means, including training with data augmentation (Botev et al., 2022; Geiping et al., 2022), canonicalization of the input data (Jaderberg et al., 2015; Kaba et al., 2022), adversarial training (Engstrom et al., 2017), and designing network architectures that inherently incorporate the desired invariances (Biscione and Bowers, 2022; Blything et al., 2020; Kauderer-Abrams, 2018; Marcus, 2018). Domain adaptation, on the other hand, seeks to transfer the knowledge acquired from a source domain to a target domain, where the data distributions may differ. This approach leverages the learned representations or features from the source domain and fine-tunes or adapts them to better align with the target domain (Loghmani et al., 2020; Russo et al., 2017; Xu et al., 2018).

Aforementioned approaches can be seen as static or dynamic as to whether they make post-deployment changes to model weights. While static methods boil down to carefully designed architectures or training procedures optimized w.r.t. specific input transformations, dynamic approaches focus on updating the model with minimal overhead in terms of data samples, memory, and computational resources. Static methods either require a careful model design to account for the desired input transformations, which is not always a trivial task, or need sufficient model capacity to learn transformed data. Dynamic methods shift the overhead of adjusting the model to the post-deployment phase and implicitly assume that model adaptation is either a rare task or the corresponding overhead is comparably small. This paper proposes a dynamic low-overhead method to adapt a model to a transformed input during deployment.

**Contributions.** In contrast to a large body of literature on invariant architectures, configurable networks make invariances explicit. We parameterize the class of transformations of input data we are interested in and for which the network should be configurable. We then train *subspace-configurable networks* (SCNs) whose network weights lie within a subspace spanned by a small number of base models. If such a SCN is given a parameter vector corresponding to a specific input transformation, it chooses the corresponding weights of a high-accuracy model from this subspace. With this, SCNs factor out the desired class of invariances. We show that the weights of a family of classifiers operating on input data subject to a continuous transformation can be embedded in a subspace of network weight vectors with surprisingly few dimensions. The learned subspace is nicely structured: the weights optimized for a specific configuration of input data are a linear combination of the weights of other optimized solutions, i.e., the optimized weights lie within a *low-dimensional linear subspace* of the weight space.

We evaluate our SCN models by studying a wide range of real-world transformations: from reversible transformations such as 2D rotation and translation to complex irreversible transformations like 3D rotation-and-projection. We cover computer vision and audio signal processing domains and dedicated network architectures. To uncover configuration subspaces for a set of input transformations, SCNs leverage a hypernet-inspired architecture to learn optimal inference models for each specific transformation in the set (Figure 1 (left)). To offer additional insights, we visualize the relation between the input transformation parameters and the configuration vector in the configuration subspace for a number of transformations (an example of the configuration subspace for 2D rotation is shown in Figure 1 (middle)). Interestingly, the configuration parameter vectors form well-structured geometric objects, highlighting the underlying structure of the optimal parameter subspaces. If the inference network capacity is fixed, usually due to severe resource constraints of embedded devices, SCNs can quickly beat training with data augmentation (One4All) and match or outperform solutions trained for input transformation parameters optimized for each input transformation separately (One4One), see Figure 1 (right). Beside providing evidence for SCNs’ high performance for a number of transformations, we offer a practical methodology for model configuration on resource-constrained devices, as an efficient alternative to resource-hungry backpropagation. The contributions of this paper are summarized as follows:

- We design *subspace-configurable networks* (SCNs) to learn the configuration subspace and generate optimal networks for specific transformations. The approach presents a highly resource-efficient alternative to model adaptation through retraining and specifically targets embedded devices (Section 2).
- We evaluate SCNs on ten common real-world transformations, using five deep model backbones and five

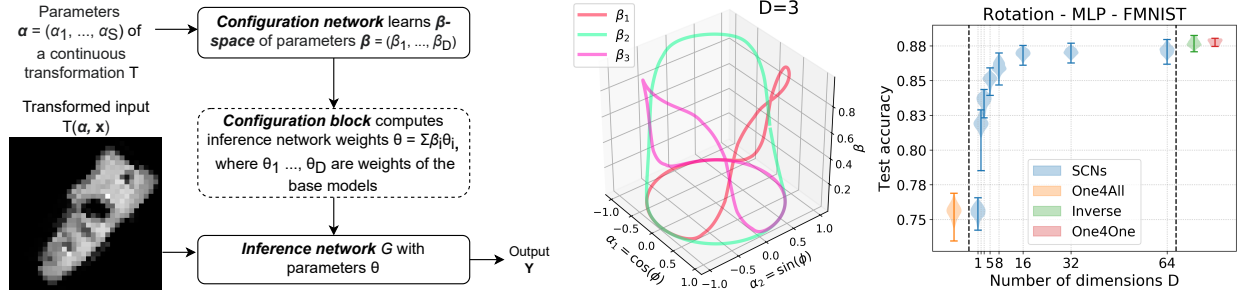


Figure 1: **Training subspace-configurable networks (SCNs)**, where an optimal network for a fixed transformation parameter vector is part of the subspace retained by few configuration parameters. **Left:** Given input transformation parameters  $\alpha$ , *e.g.*, a rotation angle for a 2D rotation, we train a *configuration network* which yields a  $D$ -dimensional configuration subspace ( $\beta$ -space) to construct an efficient inference network with weights  $\theta = \sum \beta_i \cdot \theta_i$ , where  $\theta_i$  are the weights of the *base models*, and  $\beta$  is a *configuration vector*. **Middle:** Optimal model parameters in the configuration subspace as functions of the rotation angle  $\alpha$  given by  $(\cos(\phi), \sin(\phi))$  for 2D rotation transformations applied to FMNIST (Xiao et al., 2017). Here SCN has three base models with parameters  $\theta_i$  and three configuration vectors  $\beta_i$  to compose the weights of the 1-layer MLP inference model. **Right:** Test accuracy of SCNs with  $D = 1..64$  dimensions compared to a single network trained with data augmentation (One4All), classifiers trained on canonicalized data achieved by applying inverse rotation transformation with the corresponding parameters (Inverse), and networks trained and tested on datasets where all images are rotated by a fixed degree (One4One). Note that dynamic adaptation using One4One is not implementable on resource-constrained platforms.\* Each violin shows the performance of the the models evaluated on all degrees with a discretization step of  $1^\circ$ , expect for One4One where the models are independently trained and evaluated on  $0, \pi/6, \pi/4, \pi/3, \pi/2$  rotated inputs.

standard benchmark datasets from computer vision and audio signal processing domains (Section 3).

- SCNs take transformation parameters as input, yet these parameters can be estimated from the input data. We design the corresponding algorithm which allows building a transformation-invariant architecture on top of SCNs (Section 2).
- In practical IoT scenarios the parameter supply can be replaced with a correlated sensor modality. We implemented SCNs on two IoT platforms and show their outstanding performance and remarkable efficiency in the context of fruit classification with RGB sensor and traffic sign classification from camera images (Section 4).

Section 5 concludes this paper. Further related work is discussed in Appendix C.

## 2 Subspace-Configurable Networks

### 2.1 Transformations and their parameterization

Let  $\mathbb{X} \times \mathbb{Y} = \{(x, y)\}$  be a dataset comprising labelled examples  $x \in \mathbb{X} \subset \mathbb{R}^N$  with class labels  $y \in \mathbb{Y} \subset \mathbb{R}^M$ . We apply a transformation  $T : \mathbb{R}^S \times \mathbb{R}^N \rightarrow \mathbb{R}^N$  parameterized by the vector  $\alpha = (\alpha_1, \dots, \alpha_S) \in \mathbb{A} \subseteq \mathbb{R}^S$  to each input example  $x$ . A transformed dataset is denoted as  $T(\alpha, \mathbb{X}) \times \mathbb{Y} := \{(T(\alpha, x), y)\}$ . For instance, let  $\mathbb{X}$  be a collection of  $P \times P$  images, then we have  $x \in \mathbb{R}^{P^2}$  where each dimension corresponds to the pixel intensity of a single pixel. Transformation  $T(\alpha, \mathbb{X}) : \mathbb{A} \times \mathbb{R}^{P^2} \rightarrow \mathbb{R}^{P^2}$  is modulated by pose parameters  $\alpha$ , such as rotation, scaling, translation or cropping. We assume that data transformations  $T(\alpha, \mathbb{X})$  preserve

\*One4One = *one* model for *one* parameter setting, *e.g.*, a fixed rotation degree. One4All = *one* model for *all* parameter settings, *i.e.*, a model trained with data augmentation for the considered continuous range of parameter values.

the label class of the input and represent a continuous function of  $\alpha \in \mathbb{A}$ , *i.e.*, for any two transformation parameters  $\alpha_1$  and  $\alpha_2$  there exists a continuous curve in  $\mathbb{A}$  that connects two transformation parameters. Note that by changing  $\alpha$  we transform all relevant data distributions the same way, *e.g.*, the data used for training and test. The set  $\{T(\alpha, x) \mid \alpha \in \mathbb{A}\}$  of all possible transformations of input  $x$  is called an *orbit* of  $x$ . We consider an infinite orbit defined by a continuously changing  $\alpha$ . Related literature (Gandikota et al., 2021; Weiler and Cesa, 2019; Weiler et al., 2018) primarily considers transformations that originate from a *group action*<sup>\*</sup> on  $\mathbb{X}$ . We do not make this assumption, and thus cover a broader set of transformations.

We consider a network  $\mathcal{G}$  to represent a function  $g : \mathbb{X} \times \mathbb{R}^L \rightarrow \mathbb{Y}$  that maps data  $x \in \mathbb{X}$  from the input space  $\mathbb{X}$  to predictions  $g(x, \theta) \in \mathbb{Y}$  in the output space  $\mathbb{Y}$ , where the mapping depends on the weights  $\theta \in \mathbb{R}^L$  of the network.  $E(\theta, \alpha)$  denotes the expected loss of the network  $\mathcal{G}$  and its function  $g$  over the considered training dataset  $T(\alpha, \mathbb{X})$ . Since the expected loss may differ for each dataset transformation parameterized by  $\alpha$ , we write  $E(\theta, \alpha)$  to make this dependency explicit. Optimal network parameters  $\theta_\alpha^*$  are those that minimize the loss  $E(\theta, \alpha)$  for a given transformation vector  $\alpha$ .

## 2.2 Learning configurable networks

The architecture of SCNs is sketched in Figure 1 (left). Excited by the hypernet (Ha et al., 2016) design, we train a configuration network  $\mathcal{H}$  with function  $h(\cdot)$  and an inference network  $\mathcal{G}$  with function  $g(\cdot)$  connected by a linear transformation of network parameters  $\theta = f(\beta)$  computed in the configuration block:

$$\theta = f(\beta) = \sum_{i=1}^D \beta_i \cdot \theta_i, \quad (1)$$

where  $\theta_i \in \mathbb{T} \subseteq \mathbb{R}^L$  for  $i \in [1, D]$  denote the static weights (network parameters) of the base models that are the result of the training process. The configuration network  $\mathcal{H}$  with the function  $h : \mathbb{R}^S \rightarrow \mathbb{R}^D$  yields a low-dimensional *configuration space* of vectors  $\beta \in \mathbb{R}^D$ , given transformation parameters  $\alpha \in \mathbb{A}$ . Along with learning the mapping  $h$ , we train the  $D$  *base models* with weights  $\theta_i \in \mathbb{R}^L$  to construct the weights of inference networks  $\theta = f(\beta)$ , *i.e.*,  $\theta_i$  are the base vectors of the corresponding linear subspace and  $\beta$  the coordinates of  $\theta$ . The whole SCN training process minimizes the expected loss  $E(\theta, \alpha) = E(f(h(\alpha)), \alpha)$  to determine the configuration network with function  $h$  and the base model parameters  $\theta_i$ . We use the standard categorical cross-entropy loss in all our experiments. During inference, a transformed input example  $\hat{x} = T(\alpha, x)$  is classified by the inference network  $\mathcal{G}$  with weights  $\theta$  according to Eq. (1) with  $\beta = h(\alpha)$ , *i.e.*,  $y = g(\hat{x}, f(h(\alpha)))$ .

Note that degenerated solutions, where  $\beta$  is constant for all  $\alpha$  are part of the solution space. In this case, SCN ends up representing a single model  $\mathcal{G}$  for all transformation parameters  $\alpha \in \mathbb{A}$ , which is essentially the One4All model, *i.e.*, a model trained with data augmentation over all transformation parameters  $\alpha \in \mathbb{A}$ . To avoid degenerated cases, we enhance the cross-entropy loss function with a regularization term as a squared cosine similarity  $\cos^2(\beta^{(1)}, \beta^{(2)})$  between the configuration vector  $\beta^{(1)}$  for a randomly chosen  $\alpha^{(1)}$  applied to transform the current batch, and a vector  $\beta^{(2)}$  obtained from  $\mathcal{H}$  for another randomly sampled  $\alpha^{(2)} \in \mathbb{A}$ . The applied regularization (with a weighting factor of 1.0) improves the performance of SCNs by reinforcing them to construct unique dedicated inference networks for different transformation parameters  $\alpha$ .

For a continuous transformation  $T(\alpha)$ , we provide theoretical results that help to understand the structure of the  $\beta$ -space using continuity arguments of the optimal solution space.

## 2.3 Continuity of the learned subspaces

Figure 1 (middle) exemplifies a learned  $\beta$ -space for the 2D rotation transformation learned by SCN for  $D = 3$  with a MLP inference network architecture trained on FMNIST. Transformation parameters  $(\alpha_1, \alpha_2) = (\cos(\phi), \sin(\phi))$  with  $\phi = 0..2\pi$  yield 3-dimensional  $\beta$  vectors  $(\beta_1, \beta_2, \beta_3)$  with each  $\beta_i$  being in charge of a

<sup>\*</sup>A group action of a set of transformations  $\mathbb{U}$  with the identity element  $\epsilon$  on a set  $\mathbb{X}$  is a mapping  $\sigma : \mathbb{U} \times \mathbb{X} \rightarrow \mathbb{X}$ , that satisfies that  $\sigma(\epsilon, x) = x$  and  $\sigma(u, \sigma(v, x)) = \sigma(uv, x)$ ,  $\forall u, v \in \mathbb{U}$  and  $\forall x \in \mathbb{X}$ .

specific contiguous region of the  $\alpha$ -space. Transitions between regions are covered by models that are a linear combination of optimal solutions for other  $\alpha$  values. This structural property of the  $\beta$ -space is independent of the dataset and architecture we used to train SCNs, as shown in the next section. Moreover, we observe a continuous change of  $\beta$  as we change  $\alpha$ .

Another observation we make from Figure 1 (right) is that SCNs match the high performance of the baselines already for small number of dimensions  $D$  of the linear subspace. In other words, the solution subspace spanned by only  $D$  base models learned by SCN pays no penalty for its very simple structure.

We provide theoretical results that help to understand the structure of the  $\beta$ -space using continuity arguments of the optimal solution space. Informally, the following theorem shows under certain conditions that for every continuous curve connecting two transformation parameters in  $\mathbb{A}$ , there exists a corresponding continuous curve in the network parameter space  $\mathbb{T}$ . These two curves completely map onto each other where the network parameters are optimal for the corresponding data transformations. In particular, the curve in the network parameter space  $\mathbb{T}$  is continuous.

To simplify the formulation of the theorems (see Appendix A.1 for the respective proofs), we suppose that the set of admissible parameters  $\theta \in \mathbb{T}$  is a bounded subspace of  $\mathbb{R}^L$  and all optimal parameter vectors (weights)  $\theta_\alpha^*$  are in the interior of  $\mathbb{T}$ .

**Theorem 2.1 (Continuity).** *Suppose that the loss function  $E(\theta, \alpha)$  satisfies the Lipschitz condition*

$$|E(\theta, \alpha^{(2)}) - E(\theta, \alpha^{(1)})| \leq K_\alpha \|\alpha^{(2)} - \alpha^{(1)}\|_2 \quad (2)$$

for  $\alpha^{(1)}, \alpha^{(2)} \in \mathbb{A}$ , and  $E(\theta, \alpha)$  is differentiable w.r.t. to  $\theta$  and  $\alpha$ . Then, for any continuous curve  $\alpha(s) \in \mathbb{A}$  with  $0 \leq s \leq \hat{s}$  in the parameter space of data transformations there exists a corresponding curve  $\theta(t) \in \mathbb{T}$  with  $0 \leq t \leq \hat{t}$  in the parameter space of network weights and a relation  $(s, t) \in R$  such that

- the domain and range of  $R$  are the intervals  $s \in [0, \hat{s}]$  and  $t \in [0, \hat{t}]$ , respectively, and
- the relation  $R$  is monotone, i.e., if  $(s_1, t_1), (s_2, t_2) \in R$  then  $(s_1 \geq s_2) \Rightarrow (t_1 \geq t_2)$ , and
- for every  $(s, t) \in R$  the network parameter vector  $\theta(t)$  minimizes the loss function  $E(\theta, \alpha)$  for the data transformation parameter  $\alpha(s)$ .

We are also interested in the relation between  $\alpha$  and corresponding optimal vectors  $\beta$  that define optimal locations on the linear subspace of admissible network parameters as defined by (1). To simplify the formulation of the further theoretical result and proof, we suppose that  $\beta \in \mathbb{B}$  where  $\mathbb{B}$  is a bounded subspace of  $\mathbb{R}^D$ , and all basis vectors  $\theta_j$  that define  $f(\beta)$  in (1) have bounded elements. Under these assumptions, we can derive a corollary from Theorem 2.1.

**Corollary 2.2.** *Suppose that the loss function  $E(\theta, \alpha)$  satisfies (2), and for any  $\alpha \in \mathbb{A}$  all local minima of  $E(f(\beta), \alpha)$  w.r.t.  $\beta$  are global. Then the following holds: For any continuous curve  $\alpha(s) \in \mathbb{A}$  with  $0 \leq s \leq \hat{s}$  in the parameter space of data transformations there exists a corresponding curve  $\beta(t) \in \mathbb{B}$  with  $0 \leq t \leq \hat{t}$  on the linear network parameter subspace according to (1) and a relation  $(s, t) \in R$  such that*

- the domain and range of  $R$  are the intervals  $s \in [0, \hat{s}]$  and  $t \in [0, \hat{t}]$ , respectively, and
- the relation  $R$  is monotone, i.e., if  $(s_1, t_1), (s_2, t_2) \in R$  then  $(s_1 \geq s_2) \Rightarrow (t_1 \geq t_2)$ , and
- for every  $(s, t) \in R$  the network parameter vector  $\beta(t)$  minimizes the loss function  $E(f(\beta), \alpha)$  for the data transformation parameter  $\alpha(s)$ .

The proof of the corollary is in Appendix A.1. The above corollary provides a theoretical argument for the continuous curves in Figure 1 (middle), i.e., the curves explicitly show the relation  $R$ . The existence of such a relation  $R$  is also apparent for all the dataset-architecture pairs used to empirically evaluate SCNs in Section 3.

Appendix A.2 contains a further result: Small changes to transform parameters  $\alpha$  result in small changes of optimal configuration vectors  $\beta_\alpha^*$  for suitable loss functions  $E(f(\beta), \alpha)$ . In other words, the relation  $R$  can be represented as a continuous function  $r$  with  $t = r(s)$ , i.e., the parameter vector  $\beta(t)$  that minimizes the loss function can be determined as a function of the data transformation  $\alpha(s)$ .

## 2.4 Search in the $\alpha$ -space

SCNs can be used to build invariant network architectures on top of them by replacing the supply of  $\alpha$  parameter vector with a search algorithm operating in the  $\alpha$ -space. The algorithm is used to estimate the transformation parameter  $\alpha$  solely from a batch of input data. We leverage the fact that the correct input parameters  $\alpha$  should produce a confident low-entropy classification result. The method works similarly to the techniques proposed in (Hendrycks and Gimpel, 2016; Wortsman et al., 2020). Our search algorithm estimates  $\alpha$  from a batch of input data by minimizing the entropy of the model output on this batch. We use the basin hopping method (Iwamatsu and Okabe, 2004) to find a solution. The algorithm is designed for hard nonlinear optimization problems. It runs in two phases including a global stepping algorithm and a sequence of local searches at each step. To improve the  $\alpha$ -search performance, SCN training is enhanced with an additional regularizer to minimize the model output entropy for the correct  $\alpha$  and maximise it for a randomly sampled  $\alpha$ . The algorithm is described in detail in Appendix G. Although  $\alpha$ -search algorithm using basin hopping is computationally expensive, it yields excellent accuracy and allows SCNs acting as transformation-invariant networks, *i.e.*, the search algorithm can run on top of a trained SCN. We evaluate the performance of  $\alpha$ -search in Section 3.5 and show that SCNs can compare well to invariant architectures. In practice, however, search in the  $\alpha$ -space can often be avoided. In the following, we discuss the practical aspects of the SCN design for resource-constrained environments.

## 2.5 Practical value of SCNs

Several peculiarities of SCNs’ design make them well suited for resource-constrained embedded devices, as discussed below.

**Taking advantage of memory hierarchy.** Storage capacity and access times play an important role on embedded devices. Fast memory, such as SRAM, offers access times in the range of nanoseconds, allowing rapid data retrieval. However, it is usually available in smaller capacities, often in the range of kB to a few MB, due to cost and size constraints typical for embedded systems. In contrast, slow memory like flash or EEPROM provides larger storage capacities, typically from several MB to GB, but with slower access times, often in the range of microseconds to milliseconds, making them ideal for storing less time-sensitive SCN reconfiguration data. SCNs naturally make use of the presence of slow and fast memory. SCN reconfiguration may be rare or once-in-a-lifetime task. For example, reconfiguration is required if the camera mounting point or view has changed. Therefore, the  $D$  base models holding  $\theta_i$  and the parameters of the reconfiguration network  $\mathcal{H}$  can be stored in slow memory and pulled on demand. At the same time, the inference network weights  $\theta$  should better completely fit into RAM to support fast inference.

**Network capacity and limited SRAM.** SCNs yield the most benefit if memory and thus the network capacity are limited. Given unlimited resources, One4All can match the performance of SCNs. Exceptions include corner cases where the transformation parameter  $\alpha$  is used to break symmetries, *e.g.*, in MNIST the classes 6 and 9 may be confusing if no 2D rotation information is available. In the context of the fruit classification application (discussed in Section 4), the knowledge of light intensity helps to reduce color ambiguity for more accurate prediction.

**Linearity of the reconfiguration function  $f$ .** SCNs draw inspiration from the recent linear mode connectivity literature (Ainsworth et al., 2022; Entezari et al., 2021), and empirically show that a linear reconfiguration function  $f(\beta)$  yields great performance of models for different transformation parameters. This decision allows implementing efficient memory access strategies for SCN reconfiguration, such as reuse of memory pages. Also computation of Eq. (1) can be efficiently realized since linear vector operations are hardware-friendly, *i.e.*, with help of MLA and FMA instructions, vectorization and pipelining.

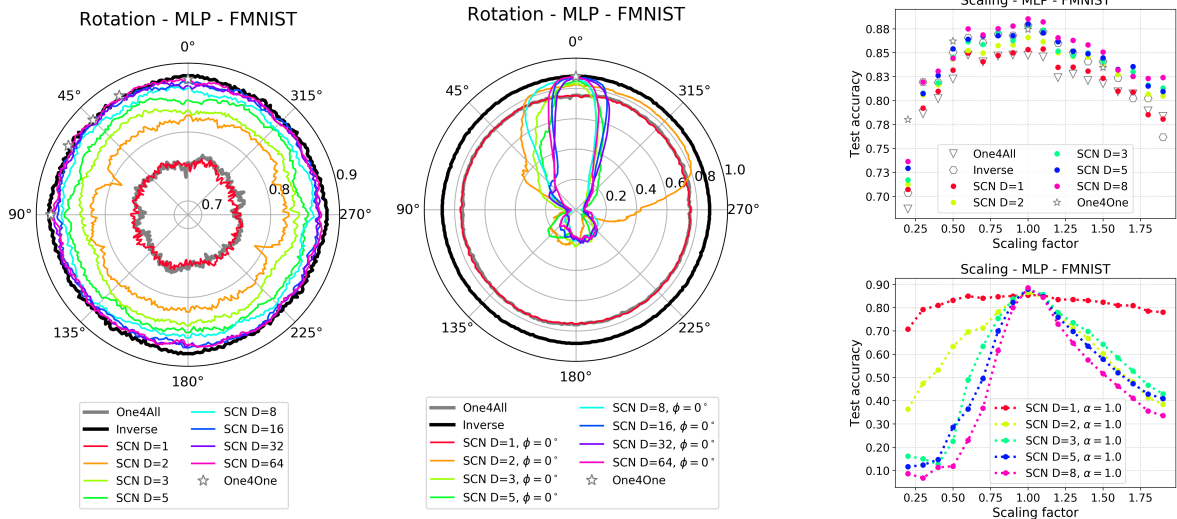


Figure 2: **SCN test accuracy for 2D rotation and scaling transformations.** **Left and middle:** 2D rotation parameterized by a rotation degree  $\phi = 0..2\pi$  input to the configuration network as  $\alpha = (\cos(\phi), \sin(\phi))$ . For each  $\alpha$ , SCN determines a configuration vector  $\beta$  used to build a dedicated model for every angle shown on the right. The left polar plot shows the performance of a single model ( $\phi = 0^\circ$ ) on all angles. The model works best for the input transformed with  $T(\phi = 0^\circ)$ . Inference network architecture is a 1-layer MLP with 32 hidden units trained on FMNIST. The models constructed by SCN outperform One4All approaching Inverse and One4One accuracy already for small  $D$ . **Right top:** Scaling transformation parameterized by the scaling factor  $\alpha = 0.2..2.0$ . **Right bottom:** SCN performance of a single model ( $\alpha = 1.0$ ) on all inputs. The dedicated model gets increasingly specialized for the target input parameters with higher  $D$ . Inference network is a 5-layer MLP with 32 hidden units in each layer trained on FMNIST. Also see Appendix D.1 and videos showing SCN inference models for each parameter setting.\*

**Low-cost sensors replace expensive search.**  $\alpha$ -search is computationally expensive. However, in practice the complexity can be significantly reduced. If the transformation of interest is discrete and limited to a few cases, the search algorithm can be reduced to running inference over a few candidate models. Most importantly, however, the parameter  $\alpha$  can be inferred from a correlated sensor modality. We demonstrate this in two IoT contexts in Section 4.

### 3 Experimental Results

In this section, we evaluate the performance of SCNs on 10 popular transformations (2D rotation, scaling, translation, brightness, contrast, saturation, sharpness, 3D rotation-and-projection, pitch shift and audio speed change) and five dataset-architecture pairs from computer vision and audio signal processing domains (MLPs on FMNIST (Xiao et al., 2017), ShallowCNNs (Neyshabur, 2020) on SVHN (Netzer et al., 2011), LeNet-5 (Lecun et al., 1998) on ModelNet10 (Wu et al., 2015), ResNet18 (He et al., 2015) on CIFAR10 (Krizhevsky et al., 2009), and M5 (Dai et al., 2016) on Google Speech Commands Dataset (Warden, 2018)). All considered transformations are continuous, and their parameterization is straightforward: For example, a rotation angle for a 2D rotation, and 3 rotation angles for a 3D rotation of a point cloud. The main paper evaluates SCNs on several non-trivial transformations to highlight the performance and the value of the proposed method, while further results, also for other transformations, can be found in the appendix. Training hyper-parameters, architectural choices, dataset description and samples of the transformed images are listed in Appendix B.

\*<https://github.com/osaukh/subspace-configurable-networks/tree/main/videos>

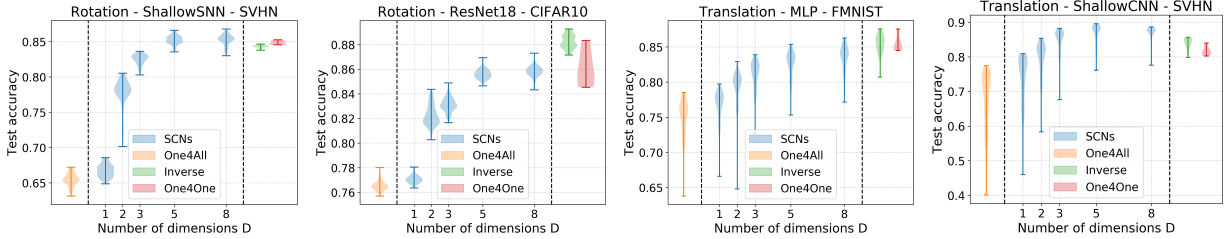


Figure 3: **SCNs achieve high test accuracy already for low  $D$** , outperforming One4All and approaching (and in some cases outperforming) both Inverse and One4One baselines. **2 plots on the left:** 2D rotation on ShallowCNN–SVHN and ResNet18–CIFAR10. **2 plots on the right:** Scaling on MLP–FMNIST and ShallowCNN–SVHN. The plots are complementary to Figure 2 evaluating the performance of SCN on different transformations and dataset-architecture pairs. For translation, the violin for One4One comprises prediction accuracy of independently trained models for  $(0,0)$  and  $(\pm 8, \pm 8)$  shift parameters. A detailed evaluation of SCNs for translation is in Appendix E.

We compare SCNs to the following baselines. *One4All* represents a network trained with data augmentation obtained by transforming the input by randomly chosen parameters  $\alpha \in \mathbb{A}$ . *Inverse* classifier is trained on a canonicalized data representation achieved by first applying the inverse transformation to the transformed input. Note that 2D rotation is a fully invertible transformation in theory, yet introduces small distortions in practice due to rounding effects. Translation is fully invertible if the relevant part of the input stays within the input boundaries. Scaling and 3D rotation bring significant distortion to the input, and inversion leads to a considerable loss of input quality. Finally, *One4One* represents a set of networks, each trained and tested on the dataset transformed with a fixed parameter vector  $\alpha$ . Note that for a fixed architecture, dataset, and loss function, a well-trained One4One baseline achieves the best in-distribution generalization, but dynamic configuration based on One4One is not implementable on a resource-constrained platform. In this sense, it upper bounds the performance which can be achieved by any domain adaptation method using the same data. When comparing model performance throughout this work, all baselines feature the same model architecture and have the same capacity as the SCN inference network.

We use a 1-layer MLP with 64 hidden units as the configuration network architecture to learn the configuration subspace  $\beta = h(\alpha)$ . Our main evaluation metric is the test set accuracy, but we also analyze the impact of SCN dimensionality  $D$  on its performance, and the structure of the  $\beta$ -space.

### 3.1 SCN test set accuracy

Figure 2 and Figure 3 present different views on the SCN test accuracy as a function of the number of dimensions  $D$  when the concept is applied to different transformations, datasets and architectures. Figure 2 (left) shows the performance of SCNs on  $0 - 2\pi$  rotation angles. The test accuracy for  $D = 1$  matches One4All but quickly approaches Inverse and One4One baselines for higher  $D$ . For the scaling transformation shown in Figure 2 (top right), SCNs for  $D > 1$  easily outperform One4All. They also outperform Inverse for  $\alpha < 0.3$  and  $\alpha > 1.2$  already for small  $D$ . Non-invertible transformations introduce significant distortion to the input data complicating feature re-use across inputs for different  $\alpha$ . A large performance gap between One4One and Inverse for  $\alpha = 0.2$  suggests that at small scales different features in the input become relevant than in the original dataset. In some cases, SCNs achieve higher accuracy than One4One networks trained and tested only on the transformed data for some fixed value of  $\alpha$ , since One4One does not make use of data augmentation but SCN implicitly does due to its structure given in Eq. (1).

Figure 3 presents an aggregated view on the SCN test accuracy for 2D rotations on ShallowCNN–SVHN and ResNet18–CIFAR10, and also for translation on MLP–FMNIST and ShallowCNN–SVHN. Each violin comprises accuracies achieved by models tested on all parameter settings traversed with a very small discretization step (with a granularity of  $1^\circ$ , 0.05 and 1 pixel for 2D rotation, scaling and translation



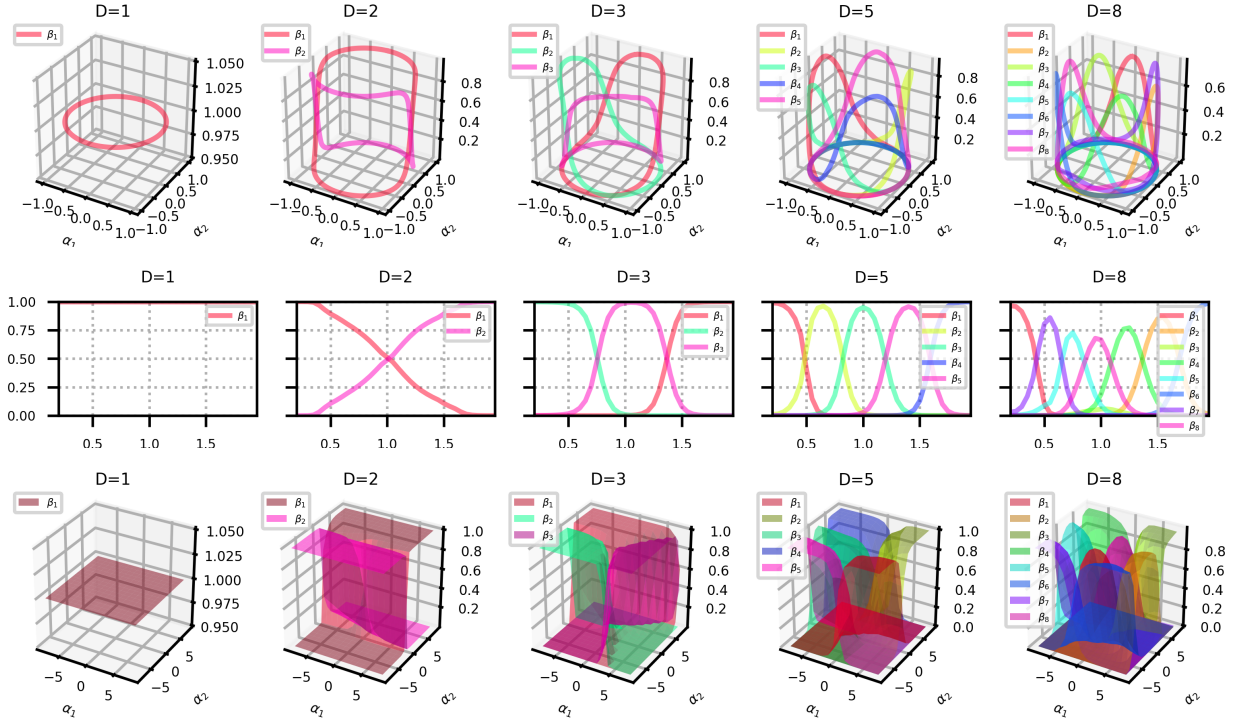


Figure 4: **A typical view of the  $\beta$ -space for 2D rotation, scaling and translation,  $D = 1..8$ .** The  $\beta$ -space is nicely shaped, with each  $\beta$  being responsible for a specific range of inputs with smooth transitions. **Top:** SCNs for 2D rotation on ResNet18–CIFAR10. Transformation parameters are a vector  $\alpha = (\alpha_1, \alpha_2) = (\cos(\phi), \sin(\phi))$ , with  $\phi$  being a rotation angle. **Middle:** SCNs for scaling on ShallowCNN–SVHN, with a scaling factor  $\alpha$  between 0.2 and 2.0. **Bottom:** SCNs for translation on MLP–FMNIST. A shift is specified by two parameters  $(\alpha_x, \alpha_y)$  varying in the range  $(-8, 8)$  along  $x$  and  $y$  axes. Visualization for other dataset-architecture pairs is in Appendix D.2.

respectively). The only exception here is the One4One baseline, where a violin comprises the performance of five models independently trained and tested on the transformed inputs for a fixed parameter setting. These fixed settings are listed in the captions of the respective figures and shown with grey stars in Figure 2 (left) and (right top). The fixed parameters are chosen to cover  $\mathbb{A}$  from the most beneficial (*e.g.*,  $\alpha = (0, 0)$  for translation) to the most suboptimal ( $\alpha = (\pm 8, \pm 8)$  for translation) setting. This is why the violins for scaling and translation transformations have a long tail of low accuracies. The performance of SCNs is consistent across dataset-architecture pairs, matching the best performing baselines already for a small number of dimensions  $D$  (also see Appendix D.1).

SCN-composed models get increasingly specialized with growing  $D$ . Figure 2 (middle) and (bottom right) show the performance of SCN for a fixed  $\alpha$ : While the model accuracy for a target  $\alpha$  improves with higher  $D$ , the performance of the model on other degrees declines increasingly fast.

### 3.2 Structure of the configuration subspace

The  $\beta$ -space learned by the configuration network  $h$  for different transformations, datasets and inference network architectures is shown in Figure 4 and Appendix D.2. For 2D rotation, the transformation parameters  $\alpha = (\cos(\phi), \sin(\phi))$  are drawn from a circle and result in all  $\beta_i$  being continuous curves arranged around the cylinder in our  $\alpha$ - $\beta$  visualization. For all transformations, if  $D = 1$ , the SCN training yields  $\beta_1 = 1$  due to the use of softmax in the last layer of the configuration network and a single base model. For  $D \neq 1$ , each  $\beta_i$

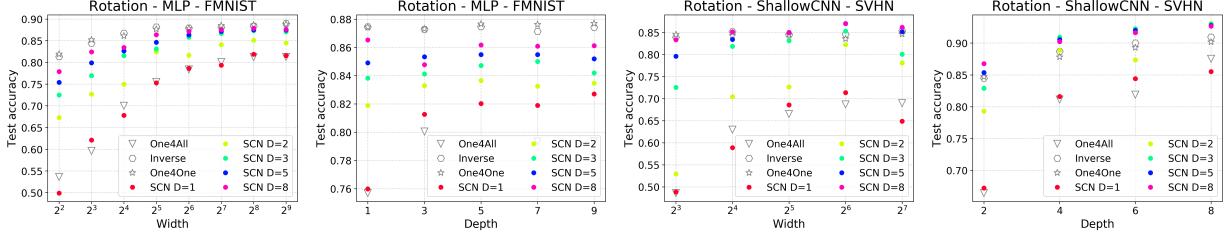


Figure 5: **Effect of network capacity on SCN test accuracy for 2D rotation.** We vary inference network width and depth to obtain the models of different capacity. **2 plots on the left:** Effect of width and depth for MLPs on FMNIST. In the width experiments, all networks are 1-layer MLPs. In the depth experiments, network width is fixed to 32 hidden units. **2 plots on the right:** Effect of width and depth for ShallowCNNs on SVHN. In the width experiments, the depth is fixed to two layers scaled together. In the depth experiments, the width of the hidden layers is fixed to 32 channels.

is high for a certain contiguous range of  $\alpha$  and low outside of this range. For small  $D$ , the regions of high  $\beta$  are largely disjoint, yet overlap as  $D$  is scaled up. Interestingly, the shape of the learned transformation is preserved across datasets and inference network architectures, although minor differences do exist, see Appendix D.2.

We claim that the subspace of optimized configurations for data transformations parameterized by  $\alpha$  is *nice* structured: (i) We achieve good accuracies even for a linear subspace of low degree of freedom  $D$ . (ii) We observe a nice structure of optimal solutions in the space, as represented by the function  $\beta = h(\alpha)$  and supported by our theoretical results in Appendix A. This finding is related to the recent literature on linear mode connectivity of independently trained solutions (Entezari et al., 2021), the solutions that share a part of the training trajectory (Frankle et al., 2020), and those trained on data splits (Ainsworth et al., 2022). SCNs establish linear mode connectivity between models trained for different transformation parameters, enhancing the existing literature.

Although the inference network architecture seems to have little impact on the shape of the learned  $\beta$ -space, there are interesting exceptions. How does configuration subspace look like if the inference network architecture is invariant to the applied transformation? We trained a SCN for translation with a translation-invariant CNN as inference network architecture. The learned configuration space, in this case, appears degenerated with only one translation-invariant model being learned, regardless of  $D$ , *i.e.*, all but one  $\beta_i$  are zero for all transformation parameters  $\alpha$  (see Appendix E).

### 3.3 SCN dimensionality & capacity constraints

SCNs yield high performance already for low  $D$ , and we observe diminishing returns from adding dimensions for all tested transformations and dataset-architecture pairs, see Figure 3. SCN dimensionality  $D$  impacts the overhead of training SCN, including the weights of the configuration network to compute  $\beta$ s and the weights  $\theta_i$  of the base models. It also affects the overhead of computing a new inference model  $\mathcal{G}$  if the transformation parameters  $\alpha$  change, *e.g.*, to adapt an object detection model to a new position of a camera. Our empirical evaluation suggests that small  $D$  is sufficient to achieve high SCN performance.

The optimal  $D$  depends on the inference network architecture and capacity. These trade-offs are explored when scaling inference network architectures along the width and depth dimensions in Figure 5. We present the capacity scaling results for the 2D rotation transformation for MLPs on FMNIST, and for ShallowCNNs on the SVHN dataset. Both architectures incorporate BatchNorm (Ioffe and Szegedy, 2015) layers in their deeper versions to facilitate training. Although increasing width proves to be more effective for all baselines for MLPs on FMNIST, higher depth leads to better test accuracy for ShallowCNNs on SVHN. Even for small values of  $D$ , SCNs consistently deliver performance improvements that are on par with the high accuracy achieved by models trained with specific parameter settings. We note that as capacity constraints get increasingly relaxed, One4All models can quickly improve performance approaching One4One and Inverse.

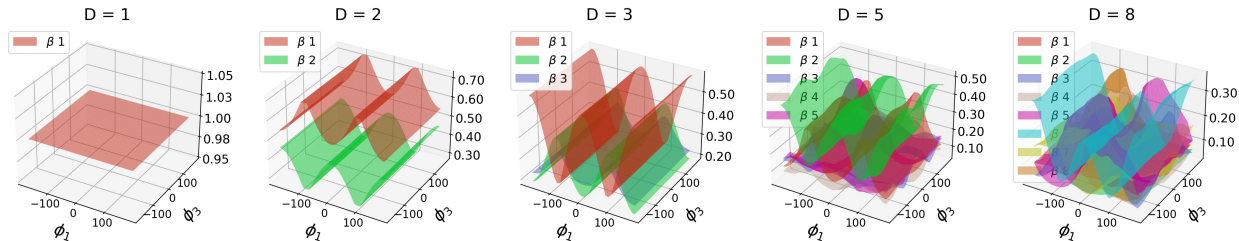


Figure 6: **A typical view of the SCN  $\beta$ -space for 3D rotation on LeNet5-ModelNet10.** Transformation parameters are a vector of ordered Euler angles  $(\phi_1, \phi_2, \phi_3)$ , each taking values from  $(-\pi, \pi)$ . We show the learned  $\beta$ -space for  $\phi_2 = -\pi$  with  $D = 1..8$ . Further views can be found in Appendix F. An interactive visualisation is available\*. The structure follows typical sine and cosine curves along multiple dimensions.

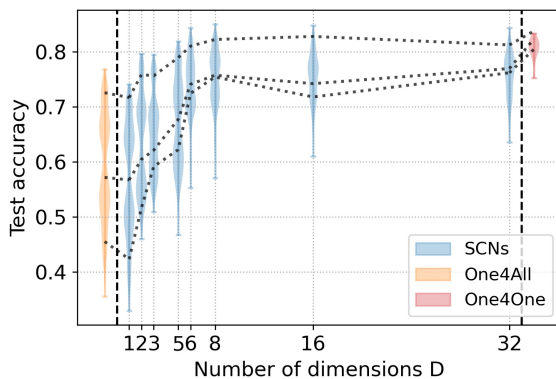


Figure 7: **Impact of  $D$  on the SCN’s test accuracy for 3D rotation.** Each line is associated with a specific test angle and connects accuracies tested on the same rotation  $(\phi_1, \phi_2, \phi_3)$ . Some rotations of a 3D object lead to a suboptimal view of the object and may significantly hurt classification accuracy. With increasing  $D$ , SCN outperforms One4All and approaches the One4One baseline.

### 3.4 3D rotation-and-projection transformation

We evaluate SCNs on 3D rotations that present complex transformations with multiple suboptimal views of the object that hurt classification accuracy. We sample a point cloud from a 3D object representation part of the MobileNet10 dataset, rotate it using a vector of Euler angles  $(\phi_1, \phi_2, \phi_3)$ , and then project the point cloud to a 2D plane. The projection is then used as input to a classifier. We use LeNet5 as a backbone for the inference network to train SCNs. Figure 6 presents the view of the  $\beta$ -space as a function of two rotation angles  $\phi_1$  and  $\phi_3$ , while  $\phi_2$  is fixed at  $-\pi$ . The configuration space nicely reflects the characteristics of the input  $\alpha$ , provided as  $\sin(\cdot)$  and  $\cos(\cdot)$  of the input angles.

One can observe that  $\beta$ s are insensitive to changes of  $\phi_3$ . Here  $\phi_3$  correspond to object rotations in the plane that does not change the object’s visibility and thus leads to stable classification predictions, similarly to the 2D rotation transformation of a flat image. The effect is best visible for low  $D$  and can be verified using the interactive visualization we provide and by inspecting further graphics in Appendix F.

Figure 7 compares SCN to One4All and One4One baselines. Inverse is not feasible due to the projection of the point cloud on the 2D plane. Each violin comprises the model test accuracy evaluated on 30 randomly chosen angles. By comparing the accuracy for the same rotation angle (dotted lines in the plot), we observe a positive correlation between  $D$  and SCN test accuracy. The result is similar to the SCN performance on 2D transformations.

\* <https://subspace-configurable-networks.pages.dev/>

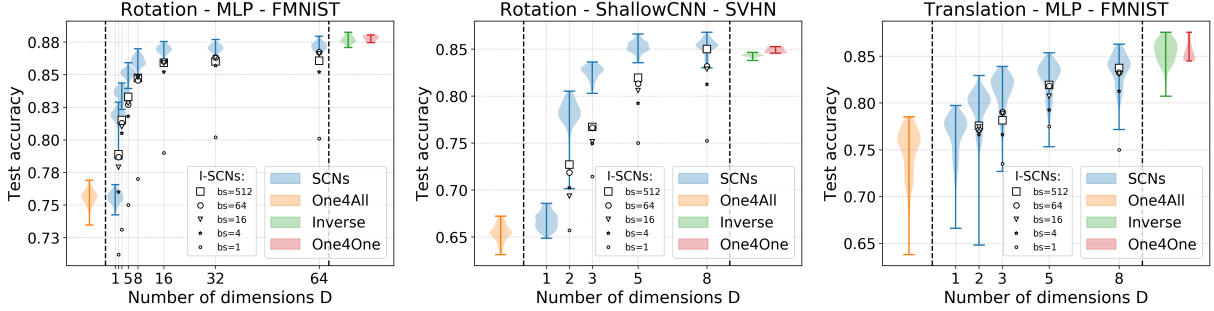


Figure 8: **Performance of the search algorithm in the  $\alpha$ -space.** We enhance SCN evaluation plots in Figure 1 (right) and Figure 3 (left) and (middle right) with the performance of the presented search algorithm in the  $\alpha$ -space. For higher batch sizes ( $\geq 4$ ) the search algorithm performs close to the respective SCNs with known  $\alpha$ .

Table 1: **Comparison of SCN to rotation-invariant TI-Pooling network.** With  $D=16$  SCN is more parameter-efficient and yields higher accuracy than the baseline.

Model	Test accuracy [%]	#parameters
TI-Pooling	88.03	13'308'170
SCN( $D=4$ ) [ours]	87.37	374'582
SCN( $D=8$ ) [ours]	88.04	674'146
SCN( $D=16$ ) [ours]	88.42	1'273'274

### 3.5 Search in the $\alpha$ -space & I-SCNs

In Figure 8, we enhance three plots from Figure 1 and Figure 3 to show the performance of the search in the  $\alpha$ -space. Note that the proposed input-based search algorithm allows constructing invariant SCNs, which we refer to as I-SCNs. We compare the performance of I-SCNs to the test accuracy achieved by the respective SCNs with known and correct input  $\alpha$ . The search algorithm operates on batches ( $bs =$  batch size). Batch size  $\geq 4$  allows for an accurate estimation of  $\alpha$  from the input data and yields high I-SCN performance.

Note that the search algorithm using basin hopping is computationally expensive, computing several hundreds of model evaluations to find an accurate estimate of  $\alpha$  for a given batch. A cheaper option might be to test models for some feasible  $\alpha$  and pick the one with the lowest entropy. Further options to minimize the computational overhead of  $\alpha$ -search by using additional sensor modalities are discussed in Section 2.5.

Network architectures can be designed to be invariant to transformations. For example, to achieve rotation invariance in 2D and 3D, an element-wise maxpooling layer can be utilized (Laptev et al., 2016; Savva et al., 2016; Su et al., 2015). Transformation-Invariant Pooling (TI-Pooling) model (Laptev et al., 2016) employs parallel Siamese network layers with shared weights and different transformed inputs. We compare SCN and TI-Pooling models trained on 2D rotations with  $\phi$  in the range  $(0, \pi)$  on the FMNIST dataset. For SCNs, the inference network architecture is a 3-layer MLP with 64 hidden units in each layer. Table 1 shows the average classification accuracy and the number of parameters. SCN with  $D=16$  dimensions demonstrates greater parameter efficiency compared to TI-Pooling, while also achieving higher accuracy than the baseline model.

## 4 SCNs on Low-resource Devices

Embedded devices, often battery-powered and with limited runtime memory, are frequently used in IoT applications on the edge. These devices feature severe resource constraints when it comes to running machine learning models. Furthermore, these models are expected to be robust to typical input transformations,

such as rotation, due to a camera’s imperfect placement, or naturally varying lighting conditions. Below, we showcase and measure the SCN’s performance on two embedded off-the-shelf platforms within the context of two example IoT applications: fruit classification using RGB sensors and traffic sign classification from camera inputs. For all tasks, additional sensor data is used to derive the input  $\alpha$ , elevating the need to perform  $\alpha$ -search on the device.

We evaluate the models’ test accuracy on the held-out datasets. On IoT devices, we compare SCN to the original One4All baseline, as well as to the wider and deeper One4All variants. To evaluate the time-efficiency of SCN, we measure the latency of three phases: executing the configuration network to obtain the vector  $\beta$  ("Hypernet Inference"), computing  $\theta$  from base models  $\theta_i$  ("Configuration"), and executing the inference network  $\mathcal{G}$  ("Inference"). All reported times present averages over 100 measurements. The running time of the One4All model is solely determined by its inference time. We measure the required storage capacity by separately quantifying the flash and RAM usage of the SCN models, original One4All models, and modified One4All models. Our embedded experiments utilize two MCUs: Tensilica Xtensa 32-bit LX7 dual-core and nRF52840. One copy of the initialized model weights is stored in flash and loaded into RAM upon program start. The  $D$  base models of SCNs are stored exclusively in flash, thereby conserving valuable RAM resources.

## 4.1 Fruit Classification with RGB sensor

We use Arduino Nano 33 BLE Sense featuring nRF52840 SoC with 256 KB SRAM and 1MB flash (Arduino, 2023) to evaluate SCN’s performance on the fruit classification task. We use Arduino equipped with the APDS-9960 sensor to gather RGB and light intensity data, targeting a variety of fruits—bananas, apples, oranges, lemons, and kiwis—under fluctuating natural light conditions. This process involves taking consistent measurements from various positions on the fruit surfaces and ensuring a large variability of light conditions and times throughout the day. Each fruit undergoes the same data collection process, yielding 1024 data samples, which are then employed to train SCN. Light intensity is used as input to the SCN’s configuration model to derive the model weights for varying light conditions.

Figure 9 (left) showcases the performance of SCN( $D = 3$ ) and SCN( $D = 5$ ) on classifying fruits using RGB data parameterized by the light intensity transformation. The One4All model used in this experiment has the same architecture as the SCN base model and features a LeNet5-like layout, which comprises a 2D convolutional layer (with 4 channels, 5 kernels, and a stride of 1) followed by ReLU and MaxPool2D (with a stride of 2), and two fully-connected layers with 32 and 43 neurons, respectively. To build a 2xDeeper network, the number of layers is doubled. The width of each hidden layer is doubled to construct a 2xWider One4All network. SCN and One4All feature the same inference time if no reconfiguration is required. SCN reconfiguration overhead includes computation of parameters  $\beta$  ("Hypernet Inference") and re-computation of inference model parameters ("Configuration"). In addition, more flash is used to store the base model weights  $\theta_i$ . Notice, that the wider and deeper One4All variants can easily generate higher resource consumption than SCNs, yet they perform considerably worse (achieve 10% lower test set accuracy) on the fruit classification task due to color ambiguity under varying light conditions.

## 4.2 Traffic Sign Classification

The German Traffic Sign Benchmark contains 39,209 images of 43 traffic signs captured on German roads (Houben et al., 2013). During training, we rotate each traffic sign image at an arbitrary angle, supplied as  $\alpha$  to the SCN’s configuration network. We evaluate SCN’s performance on 12,630 self-gathered traffic sign images, each fixed at a randomly chosen rotation angle. These images were collected using the onboard camera of the ESP32S3-EYE development board featuring Tensilica Xtensa 32-bit LX7 dual-core processor with 8 MB Octal PSRAM and 8 MB flash (Espressif, 2023). The data measured with the on-board IMU sensor of the ESP32S3-EYE development board is used to calculate the rotation angle during testing.

Figure 9 (right) presents the performance of SCN( $D = 3$ ) and SCN( $D = 5$ ) on classifying traffic signs from 2D-rotated images. The One4All model used in this experiment shares the same architecture as SCN: It features a sequence of three fully connected layers, with 12, 8, and 5 neurons respectively, each followed by a ReLU activation function. For the modified One4All models, including the  $N$ xDeeper and  $N$ xWider

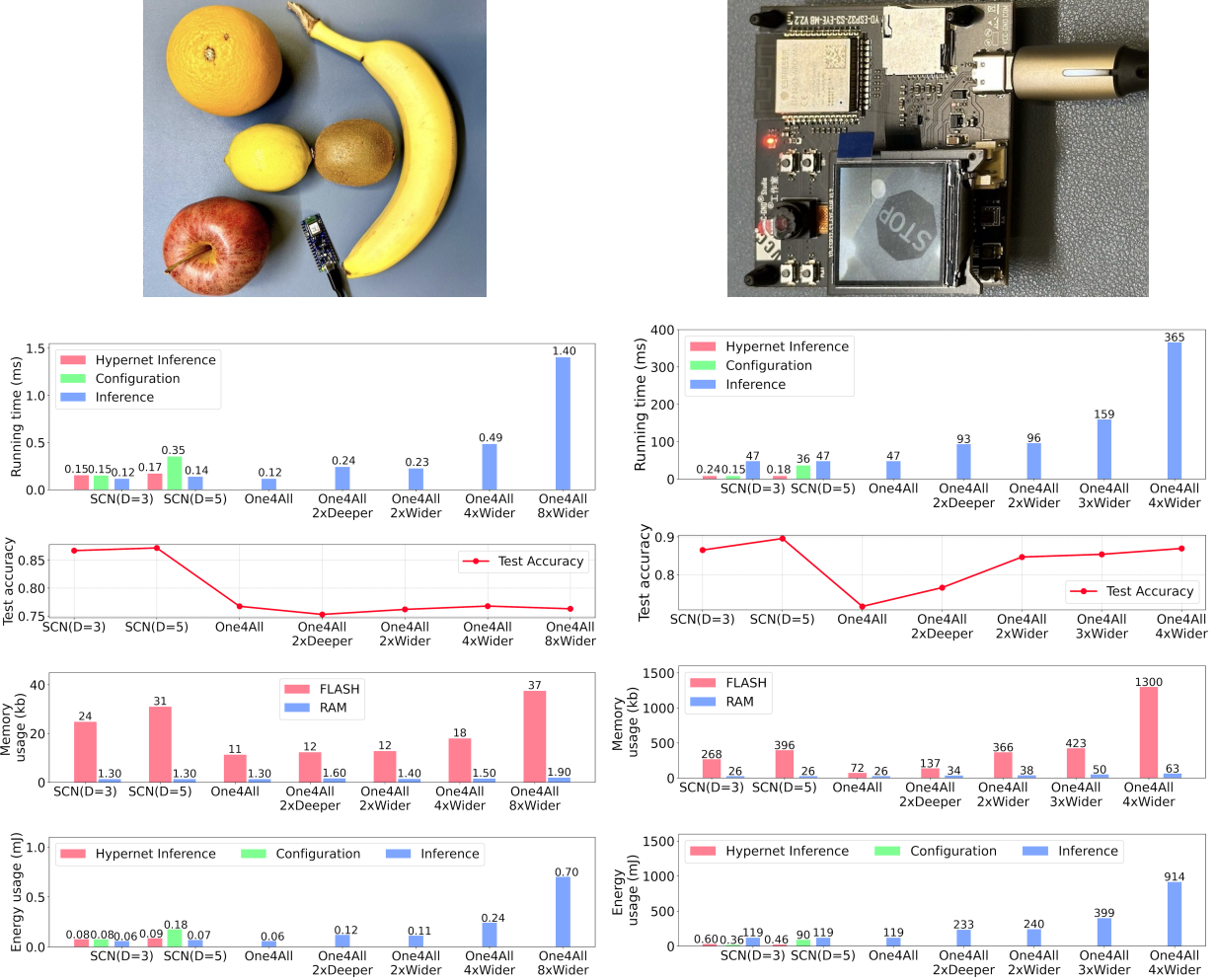


Figure 9: SCN performance on the fruit classification task on Arduino Nano 33 BLE Sense (left), and on the traffic sign classification task on ESP32-S3-EYE (right), along with the visualization of both setups (top). From top to bottom the plots show: (1) Inference time in milliseconds, showcasing the efficiency of SCN, where the deeper and wider One4All variants lead to increased inference times. For SCNs, we also measure the execution latencies of the configuration network used to obtain vector  $\beta$  ("Hypernet Inference"), and the computation time for generating  $\theta$  from base models  $\theta_i$  ("Configuration"). These latencies are only incurred when the deployment environment changes. (2) Test accuracy across various architectures, highlighting SCN's highly competitive performance. (3) RAM and flash memory usage in kB, indicating the increased resource consumption as the One4All model expands. (4) Energy consumption in mJ.

modifications, the number of layers or the number of hidden units per layer are increased. Figure 9 (right) shows that the One4All architecture, when made four times wider, matches the accuracy of SCN( $D = 3$ ). However, the resource consumption of SCN( $D = 3$ ) is notably lower: inference time and energy usage are reduced by the factor of 7.6, flash usage by the factor of 4.9, and RAM usage by the factor of 2.4. The overhead of reconfiguring the SCN model constitutes only a tiny fraction of the SCN inference time and is almost negligible. Despite sharing the same network architecture, One4All may lack sufficient capacity to store all the necessary augmentations for the desired input transformations. This could result in a significant drop in accuracy, which becomes more pronounced under tighter resource constraints.

The data gathered as part of our IoT experiments is available online.\*

## 5 Conclusion, Limitations, and Future Work

This paper addresses the problem of model reconfiguration and robustness to input transformations under severe resource constraints. We design subspace-configurable networks (SCNs) that learn low-dimensional configuration subspaces and draw optimal model weights from this structure. We achieve surprisingly high accuracies even for low number of configuration dimensions and observe a simple and intuitive structure of the subspace of optimal solutions for all investigated input transformations. Our findings open up practical applications of SCNs summarized below.

**Post-deployment adaptation.** SCNs can be used for the post-deployment model adaptation on resource-constrained devices as an alternative to costly backpropagation. SCN-configured inference networks are compact and can easily be deployed on devices with limited memory and processing power, *e.g.*, in robotics applications, edge computing, or classical embedded systems.

**SCNs as invariant architectures.** SCNs can be used to build invariant network architectures on top of them by replacing the supply of  $\alpha$  parameter vector with a search algorithm operating in the  $\alpha$ -space. We can leverage the fact that the correct input parameters  $\alpha$  should produce a confident low-entropy classification result (Hendrycks and Gimpel, 2016; Wortsman et al., 2020). The modified SCN training procedure is detailed in Section 2.4. Note that SCNs without search are of interest in their own right, since various sensor modalities can serve as input parameter  $\alpha$ , *e.g.*, using IMU (Inertial Measurement Unit) sensor measurements to determine the current 2D rotation parameter  $\alpha$ , as we show in Section 4.

**Limitations and future work.** One of the major difficulties of this work is to effectively train SCNs for high  $D$ . The effects of the learning rate and training schedule are significant. With carefully chosen hyperparameters, we were able to train SCNs with  $D = 32$  for 3D rotation using LeNet5 without any degenerated dimension. Although the current work mainly explores continuous transformations, the extension of SCNs to discrete transformations is imaginable, yet requires rethinking of the theoretical arguments to provide a better understanding of the SCN design choices, obtained performance and limitations. Finally, SCNs require the knowledge of correct  $\alpha$  and may present an additional vector of input manipulation and adversarial attacks. This direction requires future research and should be carefully considered before SCNs can be safely deployed.

## Acknowledgements

We thank Mitchell Wortsman and Rahim Entezari for their insightful comments on the early draft of the manuscript. This work was partly funded by the Austrian Research Promotion Agency (FFG) and Pro2Future (STRATP II 4.1.4 E-MINDS strategic project). The results presented in this paper were computed using computational resources of ETH Zurich and the HLR resources of the Zentralen Informatikdienstes of Graz University of Technology.

---

\*<https://github.com/osaukh/subspace-configurable-networks/tree/main/IoT/>

## References

- S. K. Ainsworth, J. Hayase, and S. Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries. *arXiv preprint*, 2022. URL <https://arxiv.org/abs/2209.04836>.
- Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.
- Arduino. Nano 33 ble sense, 2023. <https://docs.arduino.cc/hardware/nano-33-ble-sense> [Accessed: (2023-10-12)].
- V. Biscione and J. S. Bowers. Convolutional neural networks are not invariant to translation, but they can learn to be. *J. Mach. Learn. Res.*, 22(1), jul 2022. ISSN 1532-4435.
- R. Blything, V. Biscione, I. I. Vankov, C. J. H. Ludwig, and J. S. Bowers. The human visual system and cnns can both support robust online translation tolerance following extreme displacements. *arXiv preprint*, 2020. URL <https://arxiv.org/abs/2009.12855>.
- A. Botev, M. Bauer, and S. De. Regularising for invariance to data augmentation improves supervised learning. *arXiv preprint*, 2022. URL <https://arxiv.org/abs/2203.03304>.
- M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint*, 2021. URL <https://arxiv.org/abs/2104.13478>.
- T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint*, 2020. URL <https://arxiv.org/abs/2002.05709>.
- T. S. Cohen and M. Welling. Group equivariant convolutional networks. *arXiv preprint*, 2016. URL <https://arxiv.org/abs/1602.07576>.
- W. Dai, C. Dai, S. Qu, J. Li, and S. Das. Very deep convolutional neural networks for raw waveforms. *CoRR*, 2016. URL <http://arxiv.org/abs/1610.00087>.
- T. Dao, A. Gu, M. Eichhorn, A. Rudra, and C. Ré. Learning fast algorithms for linear transforms using butterfly factorizations. *arXiv preprint*, 2019. URL <https://arxiv.org/abs/1903.05895>.
- L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry. Exploring the landscape of spatial robustness. *arXiv preprint*, 2017. URL <https://arxiv.org/abs/1712.02779>.
- R. Entezari, H. Sedghi, O. Saukh, and B. Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. *arXiv preprint*, 2021. URL <https://arxiv.org/abs/2110.06296>.
- R. Entezari, M. Wortsman, O. Saukh, M. M. Shariatnia, H. Sedghi, and L. Schmidt. The role of pre-training data in transfer learning. 2023. URL <https://arxiv.org/abs/2302.13602>.
- Espressif. Esp32-s3-eye get started, 2023. [https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE\\_Getting\\_Started\\_Guide.md](https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP32-S3-EYE_Getting_Started_Guide.md) [Accessed: (2023-10-12)].
- C. Esteves, C. Allen-Blanchette, X. Zhou, and K. Daniilidis. Polar transformer networks. *arXiv preprint*, 2017. URL <https://arxiv.org/abs/1709.01889>.
- S. Fort, H. Hu, and B. Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint*, 2019. URL <https://arxiv.org/abs/1912.02757>.
- J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.
- F. Frasca, B. Bevilacqua, M. M. Bronstein, and H. Maron. Understanding and extending subgraph GNNs by rethinking their symmetries. *arXiv preprint*, 2022. URL <https://arxiv.org/abs/2206.11140>.



- K. V. Gandikota, J. Geiping, Z. Löhner, A. Czaplinski, and M. Möller. Training or architecture? how to incorporate invariance in neural networks. *CoRR*, abs/2106.10044, 2021.
- M. Geiger, S. Spigler, S. d’Ascoli, L. Sagun, M. Baity-Jesi, G. Biroli, and M. Wyart. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. *Physical Review E*, 100(1):012115, 2019.
- J. Geiping, M. Goldblum, G. Somepalli, R. Shwartz-Ziv, T. Goldstein, and A. G. Wilson. How much data are augmentations worth? An investigation into scaling laws, invariance, and implicit regularization. *arXiv preprint*, 2022. URL <https://arxiv.org/abs/2210.06441>.
- Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. *arXiv preprint*, 2014. URL <https://arxiv.org/abs/1403.1840>.
- D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint*, 2016. URL <https://arxiv.org/abs/1609.09106>.
- F. He and D. Tao. Recent advances in deep learning theory. *arXiv preprint*, 2020. URL <https://arxiv.org/abs/2012.10931>.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>.
- D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint*, 2016. URL <https://arxiv.org/abs/1610.02136>.
- J. F. Henriques and A. Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. *arXiv preprint*, 2016. URL <https://arxiv.org/abs/1609.04382>.
- S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.
- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
- M. Iwamatsu and Y. Okabe. Basin hopping with occasional jumping. *Chemical Physics Letters*, 399(4-6): 396–400, 2004. doi: 10.1016/j.cplett.2004.10.032.
- M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *arXiv preprint*, 2015. URL <https://arxiv.org/abs/1506.02025>.
- K. Jafari-Khouzani and H. Soltanian-Zadeh. Radon transform orientation estimation for rotation invariant texture analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):1004–1008, 2005. doi: 10.1109/TPAMI.2005.126.
- X. Jiaolong, X. Liang, and A. M. López. Self-supervised domain adaptation for computer vision tasks. *IEEE Access*, 7:156694–156706, 2019.
- K. Jordan, H. Sedghi, O. Saukh, R. Entezari, and B. Neyshabur. REPAIR: Renormalizing permuted activations for interpolation repair. *arXiv preprint*, 2022. URL <https://arxiv.org/abs/2211.08403>.
- J. Juneja, R. Bansal, K. Cho, J. Sedoc, and N. Saphra. Linear connectivity reveals generalization strategies. *arXiv preprint*, 2022. URL <https://arxiv.org/abs/2205.12411>.
- S.-O. Kaba, A. K. Mondal, Y. Zhang, Y. Bengio, and S. Ravanbakhsh. Equivariance with learned canonicalization functions. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022. URL <https://openreview.net/forum?id=pVD1k8ge25a>.

- E. Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv preprint*, 2018. URL <https://arxiv.org/abs/1801.01450>.
- K. Kawaguchi and Q. Sun. A recipe for global convergence guarantee in deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 8074–8082, 2021.
- K. Kawaguchi, J. Huang, and L. P. Kaelbling. Effect of depth and width on local minima in deep learning. *Neural computation*, 31(7):1462–1498, 2019.
- P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *arXiv preprint*, 2020. URL <https://arxiv.org/abs/2004.11362>.
- A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-100 and CIFAR-10 (Canadian Institute for Advanced Research), 2009. URL <http://www.cs.toronto.edu/~kriz/cifar.html>. MIT License.
- D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. TI-POOLING: transformation-invariant pooling for feature learning in convolutional neural networks. *CoRR*, abs/1604.06318, 2016. URL <http://arxiv.org/abs/1604.06318>.
- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.
- H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint*, 2017. URL <https://arxiv.org/abs/1712.09913>.
- L. D. Libera, V. Golkov, Y. Zhu, A. Mielke, and D. Cremers. Deep learning for 2D and 3D rotatable data: An overview of methods. *CoRR*, 2019. URL <http://arxiv.org/abs/1910.14594>.
- B. Liu. Spurious local minima are common for deep neural networks with piecewise linear activations. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
- M. R. Loghmani, L. Robbiano, M. Planamente, K. Park, B. Caputo, and M. Vincze. Unsupervised domain adaptation through inter-modal rotation for RGB-D object recognition. *IEEE Robotics and Automation Letters*, abs/2004.10016, 2020. URL <https://arxiv.org/abs/2004.10016>.
- R. Manthalkar, P. K. Biswas, and B. N. Chatterji. Rotation and scale invariant texture features using discrete wavelet packet transform. *Pattern Recogn. Lett.*, 24(14):2455–2462, oct 2003. ISSN 0167-8655. doi: 10.1016/S0167-8655(03)00090-4.
- D. Marcos, M. Volpi, N. Komodakis, and D. Tuia. Rotation equivariant vector field networks. *CoRR*, 2016. URL <http://arxiv.org/abs/1612.09346>.
- G. Marcus. Deep learning: A critical appraisal. *arXiv preprint*, 2018. URL <https://arxiv.org/abs/1801.00631>.
- S. Mei, A. Montanari, and P.-M. Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.
- V. Nagarajan and Z. Kolter. Deterministic PAC-Bayesian generalization bounds for deep networks via generalizing noise-resilience. In *Proceedings of the International Conference on Learning Representations*, 2019.
- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL <http://ufldl.stanford.edu/housenumbers>. License: CC0: Public Domain.
- B. Neyshabur. Towards learning convolutions from scratch. In *Advances in Neural Information Processing Systems*, 2020. URL <https://arxiv.org/abs/2007.13657>.

- Q. Nguyen, M. C. Mukkamala, and M. Hein. On the loss landscape of a class of deep neural networks with no bad local valleys. *arXiv preprint*, 2018. URL <https://arxiv.org/abs/1809.10749>.
- C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv preprint*, 2016. URL <https://arxiv.org/abs/1612.00593>.
- C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint*, 2017. URL <https://arxiv.org/abs/1706.02413>.
- F. D. S. Ribeiro, K. Duarte, M. Everett, G. Leontidis, and M. Shah. Learning with capsules: A survey, 2022. URL <https://arxiv.org/abs/2206.02664>.
- P. Russo, F. M. Carlucci, T. Tommasi, and B. Caputo. From source to target and back: symmetric bi-directional adaptive GAN. *CoRR*, 2017. URL <http://arxiv.org/abs/1705.08824>.
- M. Savva, F. Yu, H. Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, H. Su, S. Bai, X. Bai, N. Fish, J. Han, E. Kalogerakis, E. G. Learned-Miller, Y. Li, M. Liao, S. Maji, A. Tatsuma, Y. Wang, N. Zhang, and Z. Zhou. Large-Scale 3D Shape Retrieval from ShapeNet Core55. In A. Ferreira, A. Giachetti, and D. Giorgi, editors, *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2016. ISBN 978-3-03868-004-8. doi: 10.2312/3dor.20161092.
- A. Schneuing, Y. Du, C. Harris, A. Jamasb, I. Igashov, W. Du, T. Blundell, P. Lió, C. Gomes, M. Welling, M. Bronstein, and B. Correia. Structure-based drug design with equivariant diffusion models, 2022. URL <https://arxiv.org/abs/2210.13695>.
- M. M. Shariatnia, R. Entezari, M. Wortsman, O. Saukh, and L. Schmidt. How well do contrastively trained models transfer? In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022. URL <https://openreview.net/forum?id=wS7LJi8NZg>.
- P. Simard, B. Victorri, Y. LeCun, and J. Denker. Tangent prop - a formalism for specifying selected invariances in an adaptive network. In J. Moody, S. Hanson, and R. Lippmann, editors, *NeurIPS*, volume 4. Morgan-Kaufmann, 1991.
- B. Şimşek, F. Ged, A. Jacot, F. Spadaro, C. Hongler, W. Gerstner, and J. Brea. Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. *arXiv preprint arXiv:2105.12221*, 2021.
- H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.
- K. S. Tai, P. Bailis, and G. Valiant. Equivariant transformer networks. *arXiv preprint*, 2019. URL <https://arxiv.org/abs/1901.11399>.
- N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds. *arXiv preprint*, 2018. URL <https://arxiv.org/abs/1802.08219>.
- B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, and M. Welling. Rotation equivariant CNNs for digital pathology. *arXiv preprint*, 2018. URL <https://arxiv.org/abs/1806.03962>.
- D. Vucetic, M. Tayaranian, M. Ziaefard, J. Clark, B. Meyer, and W. Gross. Efficient fine-tuning of BERT models on the edge. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, may 2022. URL <https://doi.org/10.1109%2Fiscas48785.2022.9937567>.
- R. Wang, R. Walters, and R. Yu. Approximately equivariant networks for imperfectly symmetric dynamics. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 23078–23091. PMLR, 2022a. URL <https://proceedings.mlr.press/v162/wang22aa.html>.

- Y. Wang, Y. Zhao, S. Ying, S. Du, and Y. Gao. Rotation-invariant point cloud representation for 3-D model recognition. *IEEE Transactions on Cybernetics*, 52(10):10948–10956, 2022b. doi: 10.1109/TCYB.2022.3157593.
- P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, 2018. URL <http://arxiv.org/abs/1804.03209>.
- M. Weiler and G. Cesa. General  $e(2)$ -equivariant steerable CNNs. *arXiv preprint*, 2019. URL <https://arxiv.org/abs/1911.08251>.
- M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen. 3D Steerable CNNs: Learning rotationally equivariant features in volumetric data. *CoRR*, abs/1807.02547, 2018.
- D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. *arXiv preprint*, 2016. URL <https://arxiv.org/abs/1612.04642>.
- M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi. Supermasks in superposition. *arXiv preprint*, 2020. URL <https://arxiv.org/abs/2006.14769>.
- M. Wortsman, M. C. Horton, C. Guestrin, A. Farhadi, and M. Rastegari. Learning neural network subspaces. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11217–11227. PMLR, 18–24 Jul 2021. URL <https://proceedings.mlr.press/v139/wortsman21a.html>.
- Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. doi: 10.1109/CVPR.2015.7298801.
- H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.
- R. Xu, G. Li, J. Yang, and L. Lin. Unsupervised domain adaptation: An adaptive feature norm approach. *CoRR*, 2018. URL <http://arxiv.org/abs/1811.07456>.
- F. Yang, Z. Wang, and C. Heinze-Deml. Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. In *ICML 2019 Workshop on Identifying and Understanding Deep Learning Phenomena*, 2019. URL <https://openreview.net/forum?id=B1e6oy39aE>.
- P.-T. Yap, X. Jiang, and A. Chichung Kot. Two-dimensional polar harmonic transforms for invariant image representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1259–1270, 2010. doi: 10.1109/TPAMI.2009.119.
- C. Ye, X. Zhou, T. McKinney, Y. Liu, Q. Zhou, and F. Zhdanov. Exploiting invariance in training deep neural networks. *arXiv preprint*, 2021. URL <https://arxiv.org/abs/2103.16634>.
- R. Zhang. Making convolutional networks shift-invariant again. *arXiv preprint*, 2019. URL <http://arxiv.org/abs/1904.11486>.
- Y. Zhang and M. Rabbat. A Graph-CNN for 3D point cloud classification. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018. URL <https://arxiv.org/abs/1812.01711>.

# Appendix

## A Theoretical Results

For a continuous transformation  $T(\alpha)$ , we provide theoretical results that help to understand the structure of the  $\beta$ -space using continuity arguments of the optimal solution space.

### A.1 Continuity of $\alpha$ -to- $\beta$ subspace mapping

To simplify the formulation of the theorems and proofs, we suppose that the set of admissible parameters  $\theta \in \mathbb{T}$  is a bounded subspace of  $\mathbb{R}^L$  and all optimal parameter vectors  $\theta_\alpha^*$  are in the interior of  $\mathbb{T}$ . To support theoretical analysis of the continuity of configuration space  $\beta$ , given continuous transformation parameters  $\alpha$ , we first introduce the necessary definitions and assumptions.

**Parameterized curves.** We define parameterized curves  $c : I \rightarrow \mathbb{R}^n$  in  $n$ -dimensional Euclidean space that are at least one time continuously differentiable, where  $I$  is a non-empty interval of real numbers.  $t \in I$  is parameterizing the curve where each  $c(t)$  is a point on  $c$ . In the following, we suppose that  $c(t)$  is a natural parameterization of curve  $c$ , *i.e.*, the arc length on the curve between  $c(a)$  and  $c(b)$  is  $b - a$ :

$$b - a = \int_a^b \|c'(x)\|_2 dx.$$

As the shortest curve between two points is a line, we also find that  $b - a \geq \|c(b) - c(a)\|_2$ . For example, let us define a curve  $\alpha(s) = (1 - \frac{s}{d})\alpha^0 + \frac{s}{d}\alpha^1$  where  $d = \|\alpha^1 - \alpha^0\|_2$ . Then we have

$$\int_a^b \|c'(s)\|_2 ds = \int_a^b \frac{1}{d} \|\alpha^1 - \alpha^0\|_2 ds = b - a.$$

**Assumptions.** At first, we are interested in the relation between  $\alpha$  and corresponding optimal parameter vectors  $\theta_\alpha^*$ , *i.e.*, parameter vectors that minimize  $E(\theta, \alpha)$ . In order to simplify the forthcoming discussions, we suppose that the set of admissible parameters  $\theta \in \mathbb{T}$  is a bounded subspace of  $\mathbb{R}^L$  and all optimal parameter vectors  $\theta_\alpha^*$  are in the interior of  $\mathbb{T}$ . We assume that the loss function  $E(\theta, \alpha)$  is differentiable w.r.t. to  $\theta$  and  $\alpha$ , and that it satisfies the Lipschitz condition

$$|E(\theta, \alpha_2) - E(\theta, \alpha_1)| \leq K_\alpha \|\alpha_2 - \alpha_1\|_2 \quad (3)$$

for some finite constant  $K_\alpha$ , and for all  $\alpha_1, \alpha_2 \in \mathbb{A}$  and  $\theta \in \mathbb{T}$ .

We are given data transformation parameters  $\alpha \in \mathbb{A}$  and a network parameter vector  $\theta_\alpha^*$ . A point  $\theta(0) = \theta_\alpha^*$  is a local minimum of the loss function if there is no curve segment  $\theta(t)$  with  $t \in [0, \hat{t}]$  for some  $\hat{t} > 0$  where  $E(\theta(t), \alpha) \leq E(\theta(0), \alpha)$  and  $E(\theta(\hat{t}), \alpha) < E(\theta(0), \alpha)$ . All curves with  $t \in [0, \hat{t}]$  for some  $\hat{t}$  where  $E(\theta(t), \alpha) = E(\theta(0), \alpha)$  define a maximal connected subset of locally optimal parameter values. In principle, the loss landscape for a given  $\alpha$  may contain many disconnected subsets with local minima, *i.e.*, there is no path with a constant loss function between the locally minimal subsets.

The analysis of the loss-landscape of (over-parameterized networks) is still an active area of research, see for example (He and Tao, 2020; Kawaguchi et al., 2019; Liu, 2022). It turns out that in the case of over-parameterized networks, typical optimization methods like SGD do not get stuck in local minima when they exist, see for example (Allen-Zhu et al., 2019; Kawaguchi and Sun, 2021). Therefore, it is reasonable to assume that all local minima found by the optimizer are also global, *i.e.*, for any given  $\alpha \in \mathbb{A}$  the values of the loss functions  $E(\theta_\alpha^*, \alpha)$  for all local minima  $\theta_\alpha^*$  are equal.

**Relation between data and network transformation.** Loosely speaking, the following theorem shows that for any continuous curve that connects two transformation parameters in  $\mathbb{A}$  there exists a corresponding continuous curve in the network parameter space  $\mathbb{T}$ . These two curves completely map onto each other where the network parameters are optimal for the corresponding data transformations. In particular, the curve in the network parameter space  $\mathbb{T}$  has no jumps as is continuous.

**Theorem A.1.** *Suppose that the loss function  $E(\theta, \alpha)$  satisfies (3), then the following holds: For any continuous curve  $\alpha(s) \in \mathbb{A}$  with  $0 \leq s \leq \hat{s}$  in the parameter space of data transformations there exists a corresponding curve  $\theta(t) \in \mathbb{T}$  with  $0 \leq t \leq \hat{t}$  in the parameter space of network parameters and a relation  $(s, t) \in R$  such that*

- the domain and range of  $R$  are the intervals  $s \in [0, \hat{s}]$  and  $t \in [0, \hat{t}]$ , respectively, and
- the relation  $R$  is monotone, i.e., if  $(s_1, t_1), (s_2, t_2) \in R$  then  $(s_1 \geq s_2) \Rightarrow (t_1 \geq t_2)$ , and
- for every  $(s, t) \in R$  the network parameter vector  $\theta(t)$  minimizes the loss function  $E(\theta, \alpha)$  for the data transformation parameter  $\alpha(s)$ .

*Proof.* At first let us define a connected region  $\mathbb{B}(\theta_\alpha^*, \delta, \alpha)$  of  $\delta$ -minimal loss functions values for given transformation parameters  $\alpha$  and corresponding locally optimal network parameters  $\theta_\alpha^*$ , where  $\theta_\alpha^* \in \mathbb{B}(\theta_\alpha^*, \delta, \alpha)$  and

$$\mathbb{B}(\theta_\alpha^*, \delta, \alpha) = \{\theta \mid E(\theta, \alpha) - E(\theta_\alpha^*, \alpha) \leq \delta\}. \quad (4)$$

In other words, within the connected region  $\mathbb{B}(\theta_\alpha^*, \delta, \alpha)$  that contains  $\theta_\alpha^*$ , the loss function is at most  $\delta$  larger than the optimal loss  $E(\theta_\alpha^*, \alpha)$ . Note that  $\{\theta \mid E(\theta, \alpha) - E(\theta_\alpha^*, \alpha) \leq \delta\}$  may be the union of many connected regions, but  $\mathbb{B}(\theta_\alpha^*, \delta, \alpha)$  is the unique connected region that contains  $\theta_\alpha^*$ .

In order to prove the theorem, we show first that a small change in the data transformation from  $\alpha(s)$  to  $\alpha(s + \epsilon)$  leads to a new optimal network parameter vector  $\theta_{\alpha(s+\epsilon)}^*$  that is within  $\mathbb{B}(\theta_\alpha^*, \delta, \alpha)$ , and  $\delta$  decreases with the amount of change in  $\alpha$ . More precisely, we show the following statement: Given transformation parameters  $\alpha(s)$ , corresponding optimal network parameters  $\theta_{\alpha(s)}^*$ , and neighboring transformation parameters  $\alpha(s + \epsilon)$  with a distance  $\epsilon$  on the curve. Then the new optimal network parameter vector  $\theta_{\alpha(s+\epsilon)}^*$  corresponding to  $\alpha(s + \epsilon)$  is within the  $\delta$ -minimal region of  $\theta_{\alpha(s)}^*$ , namely

$$\theta_{\alpha(s+\epsilon)}^* \in \mathbb{B}(\theta_{\alpha(s)}^*, \delta, \alpha(s)) \quad (5)$$

if  $\delta > 2\epsilon K_\alpha$ . Therefore, for an infinitesimally distance  $\epsilon$  on the  $\alpha$ -curve, the new optimal network parameter vector  $\theta_{\alpha(s+\epsilon)}^*$  is within the  $\delta$ -minimal region around  $\theta_{\alpha(s)}^*$ . Furthermore, there exists a curve segment between  $\theta_{\alpha(s)}^*$  and  $\theta_{\alpha(s+\epsilon)}^*$  where every point  $\theta$  on this curve satisfies  $E(\theta, \alpha(s)) - E(\theta_{\alpha(s)}^*, \alpha(s)) \leq \delta$  according to (4), i.e., its loss for  $\alpha(s)$  is at most  $\delta$  higher than the loss at the beginning of the curve segment. Such a curve segment always exists as  $\mathbb{B}$  is a connected region and the curve segment can completely be within  $\mathbb{B}$ . The change in loss  $\delta$  for  $\alpha(s)$  on the curve segment decreases with  $\epsilon$  and is infinitesimally small.

We now prove the above statement. From (3) we find

$$|E(\theta, \alpha(s + \epsilon)) - E(\theta, \alpha(s))| \leq K_\alpha \|\alpha(s + \epsilon) - \alpha(s)\|_2 \leq \epsilon K_\alpha. \quad (6)$$

First, we show that the minimum for  $\alpha(s + \epsilon)$  is within the region  $\mathbb{B}(\theta_{\alpha(s)}^*, \delta, \alpha(s))$ . At the border of the region we find  $E(\theta, \alpha(s + \epsilon)) \geq E(\theta_{\alpha(s)}^*, \alpha(s)) + \delta$  due to (4). Combining this with (6) yields  $E(\theta, \alpha(s + \epsilon)) \geq E(\theta_{\alpha(s)}^*, \alpha(s)) + \delta - \epsilon K_\alpha$ . In the interior of the region we find as the best bound  $E(\theta, \alpha(s + \epsilon)) \leq E(\theta_{\alpha(s)}^*, \alpha(s)) + \epsilon K_\alpha$  using (6). If the loss for  $\alpha(s + \epsilon)$  is larger at the border of the region than in its interior, we know that a locally minimal loss is within the region, i.e., (5) holds. Therefore, we require  $E(\theta_{\alpha(s)}^*, \alpha(s)) + \epsilon K_\alpha < E(\theta_{\alpha(s)}^*, \alpha(s)) + \delta - \epsilon K_\alpha$  and therefore,  $\delta > 2\epsilon K_\alpha$ .

Using the above statement, see (5), we start from some curve  $\alpha(s)$ ,  $0 \leq s \leq \hat{s}$  and construct a corresponding optimal curve in the network parameter space  $\theta(t)$  for  $0 \leq t \leq \hat{t}$ . We begin with some  $\alpha(s)$  and an optimal network  $\theta_{\alpha(s)}^* \in \arg \min E(\theta, \alpha(s))$ . We know that the optimal parameter vector  $\theta_{\alpha(s+\epsilon)}^*$  for infinitesimally

close transformation parameters  $\alpha(s + \epsilon)$  on the curve  $\alpha(s)$  is within the  $\delta$ -minimal region around  $\theta_{\alpha(s)}^*$ . Therefore, to a small segment from  $\alpha(s)$  to  $\alpha(s + \epsilon)$  we assign a finite segment from  $\theta_{\alpha(s)}^*$  to  $\theta_{\alpha(s+\epsilon)}^*$  completely within the  $\delta$ -minimal region around  $\theta_{\alpha(s)}^*$ . Every point on this curve segment corresponds to a network whose loss is either infinitesimally close to the optimal values  $E(\theta_{\alpha(s)}^*, \alpha(s))$  or  $E(\theta_{\alpha(s+\epsilon)}^*, \alpha(s + \epsilon))$ . In other words, the curve segment starts from optimal network parameters  $\theta_{\alpha(s)}^*$ , ends at optimal network parameters  $\theta_{\alpha(s+\epsilon)}^*$ , and in between traverses the region with loss values that are infinitesimally close to either of these optimal loss values. This process is repeated, starting from  $\alpha(s + \epsilon)$  and  $\theta_{\alpha(s+\epsilon)}^*$ . As a result, the two curves  $\alpha(s)$  and  $\theta(t)$  are connected by a relation  $(s, t) \in R$  such that the domains are the intervals of the curve parameters  $[0, \hat{s}]$  and  $[0, \hat{t}]$ . If  $(s, t) \in R$  then  $\theta(t)$  is optimal for  $\alpha(s)$ . No points on the curves are missing, *i.e.*, without a relation to the other curve. Moreover, the relation  $R$  is monotone: If  $(s_1, t_1), (s_2, t_2) \in R$  then  $(s_1 \geq s_2) \Rightarrow (t_1 \geq t_2)$ .  $\square$

Note that the assumption that local minima are also global minima is crucial. For example, suppose that for a given  $\alpha$  there are two local minima that are separated by a single barrier. Suppose further that by changing  $\alpha$ , just the height of the barrier reduces until it vanishes completely. At this value of  $\alpha$ , a small change, *i.e.*, a short distance on the curve  $\alpha(s)$ , leads to a large change in the optimal  $\theta$ . In other words, given a curve  $\alpha(s)$  there may be no corresponding continuous curve  $\theta(t)$  that satisfies the properties of the above theorem.

We are also interested in the relation between  $\alpha$  and corresponding optimal vectors  $\beta_\alpha^*$  that define optimal locations on the linear subspace of admissible network parameters as defined by (1). To simplify the discussion, we suppose that  $\beta \in \mathbb{B}$  are in a bounded subspace of  $\mathbb{R}^D$ , and all basis vectors  $\theta_j$  that define  $f(\beta)$  in (1) have bounded elements. Under these assumptions, we can derive a corollary from Theorem A.1.

**Corollary A.2.** *Suppose that the loss function  $E(\theta, \alpha)$  satisfies (3), and for any  $\alpha \in \mathbb{A}$  all local minima of  $E(f(\beta), \alpha)$  w.r.t.  $\beta$  are global. Then the following holds: For any continuous curve  $\alpha(s) \in \mathbb{A}$  with  $0 \leq s \leq \hat{s}$  in the parameter space of data transformations there exists a corresponding curve  $\beta(t) \in \mathbb{B}$  with  $0 \leq t \leq \hat{t}$  on the linear network parameter subspace according to (1) and a relation  $(s, t) \in R$  such that*

- *the domain and range of  $R$  are the intervals  $s \in [0, \hat{s}]$  and  $t \in [0, \hat{t}]$ , respectively, and*
- *the relation  $R$  is monotone, *i.e.*, if  $(s_1, t_1), (s_2, t_2) \in R$  then  $(s_1 \geq s_2) \Rightarrow (t_1 \geq t_2)$ , and*
- *for every  $(s, t) \in R$  the network parameter vector  $\beta(t)$  minimizes the loss function  $E(f(\beta), \alpha)$  for the data transformation parameter  $\alpha(s)$ .*

*Proof.* Sketch: The relation between the parameterization  $\beta$  of the linear subspace spanned by  $\theta_i$  and the resulting network parameters  $\theta = f(\beta)$  is given by (1). As  $\beta \in \mathbb{B}$  is bounded and the basis vectors  $\theta_j$  are finite,  $f(\theta)$  is differentiable and Lipschitz constrained. Therefore, the proof as provided for Theorem A.1 holds as well by just replacing  $\theta = f(\beta)$ . Therefore, the results of Theorem A.1 hold for  $\beta$  as well if for any  $\alpha \in \mathbb{A}$  all local minima of  $E(f(\beta), \alpha)$  w.r.t.  $\beta$  are global.  $\square$

## A.2 Small parameter changes

This section shows that small changes to transform parameters  $\alpha$  result in small changes of optimal configuration  $\beta_\alpha^*$  for suitable loss functions  $E(f(\beta), \alpha)$ . For the forthcoming analysis, we suppose that  $E(f(\beta), \alpha)$  is at least twice differentiable w.r.t.  $\alpha$  and  $\beta$ .

**Theorem A.3.** *Suppose that  $\beta_0^* \in \mathbb{B}$  locally minimizes  $E(f(\beta), \alpha)$  for  $\alpha_0 \in \mathbb{A}$ . Moreover, the Hessian  $\nabla_{(\beta, \alpha)}^2 E(f(\beta), \alpha)$  of the loss function at  $\alpha = \alpha_0$  and  $\beta = \beta_0^*$  exists, and its submatrix  $\nabla_\beta^2 E(f(\beta), \alpha)$  is non-singular.*

*Then, if  $\|\alpha_1 - \alpha_0\| \leq \epsilon$  for some small  $\epsilon$ , then there exists a locally optimal  $\beta_1^*$  for  $\alpha_1$  such that*

$$\|\beta_1^* - \beta_0^*\| \leq \|(\nabla_\beta^2 E(f(\beta), \alpha))^{-1}\| \cdot \|\nabla_\alpha(\nabla_\beta E(f(\beta), \alpha))\| \cdot \epsilon.$$

*Proof.* For a network trained to a minimum in  $E(f(\beta), \alpha)$  for a given  $\alpha$ , its first derivative over weights equals to 0, *i.e.*,  $\nabla_{\beta} E(f(\beta), \alpha) = 0$  for an optimal vector  $\beta_{\alpha}^*$ . We assume that this derivative exists and is abbreviated by a function  $F(\beta, \alpha) = \nabla_{\beta} E(f(\beta), \alpha)$ .

Let  $F(\beta, \alpha)$  at  $\beta = \beta_{\alpha}^*$  be a differentiable function of  $\beta$  and  $\alpha$ . We apply the first-order Taylor expansion of  $F$  at a point  $(\beta_0^*, \alpha_0)$ , *i.e.*,  $\beta_0^*$  is optimal for  $\alpha_0$ :

$$F(\beta, \alpha) = F(\beta_0^*, \alpha_0) + \nabla_{\beta^*} F \Big|_{(\beta_0^*, \alpha_0)} \delta\beta^* + \nabla_{\alpha} F \Big|_{(\beta_0^*, \alpha_0)} \delta\alpha, \quad (7)$$

where  $\delta\beta^* = \beta_1^* - \beta_0^*$  and  $\delta\alpha = \alpha_1 - \alpha_0$ . We have  $F(\beta^*, \alpha) = 0$  and  $F(\beta_0^*, \alpha_0) = 0$  due to the optimality of the loss function.

We abbreviate the evaluated partial derivatives  $\nabla_{\beta^*} F \Big|_{(\beta_0^*, \alpha_0)} := P$  and  $\nabla_{\alpha} F \Big|_{(\beta_0^*, \alpha_0)} := Q$ . Since  $\beta^*$  and  $\alpha$  are vectors, we find that  $P \in \mathbb{R}^{D \times D}$  and  $Q \in \mathbb{R}^{S \times D}$  are matrices. Thus,

$$P \cdot \delta\beta^* + Q \cdot \delta\alpha = 0 \quad (8)$$

and therefore,

$$\delta\beta^* = -P^{-1}Q\delta\alpha.$$

Using basic results from linear algebra we find

$$\|\delta\beta^*\| \leq \|P^{-1}\| \cdot \|Q\| \cdot \|\delta\alpha\|$$

and therefore

$$\|\beta_1^* - \beta_0^*\| \leq \|P^{-1}\| \cdot \|Q\| \cdot \|\alpha_1 - \alpha_0\| \leq \|P^{-1}\| \cdot \|Q\| \cdot \epsilon.$$

From the last equation it follows that small changes of transform parameters  $\delta\alpha$  result in small changes  $\delta\beta^*$  of the optimal solution  $\beta_0^*$  if  $P$  is invertible. Note that  $P = \nabla_{\beta}^2 E(f(\beta), \alpha)$  and  $Q = \nabla_{\alpha}(\nabla_{\beta} E(f(\beta), \alpha))$  for  $\alpha = \alpha_0$  and  $\beta = \beta_0^*$ .  $\square$

Note that the condition of the theorem is crucial, *i.e.*, the Hessian of the loss function with respect to the parameters  $\beta$  of the linear subspace at the optimal solution is invertible. This excludes cases with saddle points, where there is no optimal point vector in the neighborhood after a small change in  $\alpha$ . Moreover, we can only make statements about local minima of the loss function due to the use of the Taylor expansion.

## B Implementation details

The source code of all experiments is available online.\* We trained over 1'000 models on a workstation featuring two NVIDIA GeForce RTX 2080 Ti GPUs to evaluate the performance of SCNs presented in this work. Training a model takes up to several hours and depends on the SCN dimensionality and model complexity. WandB\* was used to log hyper-parameters and output metrics from runs.

### B.1 Datasets and architectures

**Configuration network.** Throughout all experiments we used the configuration network architecture featuring one fully-connected layer comprising 64 neurons. Depending on the input (whether 2 values for 2D rotation and translation, 6 values for 3D rotation, and 1 value for all other considered transformations), the configuration network contains  $64 \cdot (\text{input\_size} + 1) + 65 \cdot D$  trainable parameters. Note that  $D$  is the size the the configuration network's output. The architecture includes the bias term. For example, for 2D rotation with 3 outputs, the configuration network has 387 parameters.

We test the performance of SCNs on five dataset-architecture pairs described below. For MLPs and ShallowCNNs we vary architectures' width and depth to understand the impact of network capacity on

\*<https://github.com/osaukh/subspace-configurable-networks.git>

\*<https://wandb.ai>



efficiency of SCNs for different  $D$ . To scale up along the width dimension, we double the number of neurons in each hidden layer. When increasing depth, we increase the number of layers of the same width. To improve training efficiency for deeper networks (deeper than 3 layers), we use BatchNorm layers (Ioffe and Szegedy, 2015) when scaling up MLPs and ShallowCNNs along the depth dimension. The number of parameters for the network architectures specified below (excluding BatchNorm parameters) is only for a single inference network  $\mathcal{G}$ .  $D$  base models of this size are learned when training a SCN.

**MLPs on FMNIST.** FMNIST (Xiao et al., 2017) is the simplest dataset considered in this work. The dataset includes 60’000 images for training and 10’000 images for testing. The dataset is available under the MIT License.\* We use MLPs of varying width  $w$  and depth  $l$  to evaluate the impact of the dense network capacity on SCNs. The number of parameters of the MLP inference network for 10 output classes scales as follows:

$$(32^2 + 1) \cdot w + (l - 1) \cdot (w^2 + w) + 10 \cdot (w + 1).$$

**ShallowCNNs on SVHN.** SVHN (Netzer et al., 2011) digit classification dataset contains 73’257 digits for training, 26’032 digits for testing, and 531’131 additionally less difficult digits for assisting training. No additional images are used. The dataset is available for non-commercial use.\* Shallow convolutions (ShallowCNNs) were introduced by Neyshabur (2020). We scale the architecture along the width  $w$  and depth  $d$  dimensions. The number of parameters scales as follows:

$$(9 \times 9 \times 3 + 1) \cdot w + (l - 1) \cdot (13 \times 13 \times w + 1) \cdot w + 10 \cdot (w + 1).$$

**LeNet5 on ModelNet10.** ModelNet10 (Wu et al., 2015) is a subset of ModelNet40 comprising a clean collection of 4,899 pre-aligned shapes of 3D CAD models for objects of 10 categories. We use this dataset to evaluate SCN performance on images of 3D rotated objects. We first rotate an object in the 3D space, subsample a point cloud from the rotated object, which is then projected to a fixed plane. The projection is then used as input to the inference network. Rotation parameters  $\alpha$  are input to the trained hypernetwork to obtain the parameters in the  $\beta$ -space to construct an optimal inference network. We use LeNet-5 (Lecun et al., 1998) as inference network architecture with 138’562 parameters.

**ResNet18 on CIFAR10.** This work adopts the ResNet18 implementation by He et al. (2015) with around 11 million trainable parameters. We use ResNet18 on CIFAR10 (Krizhevsky et al., 2009), one of the most widely used datasets in machine learning research. The dataset comprises 60’000 color images from 10 classes and is publicly available.\*

**M5 on Google Speech Commands.** We explore the performance of SCNs in the audio signal processing domain by adopting M5 (Dai et al., 2016) convolutional architecture to classify keywords in the Google Speech Commands dataset (Warden, 2018). M5 networks are trained on the time domain waveform inputs. The dataset consists of over 105,000 WAV audio files of various speakers saying 35 different words and is available under the Creative Commons BY 4.0 license. It is part of the Pytorch common datasets.\*

## B.2 Training hyper-parameters

Table 2 summarizes the set of hyper-parameters used to train different networks throughout this work.

---

\* <https://github.com/zalandoresearch/fashion-mnist>

\* <http://ufldl.stanford.edu/housenumbers/>

\* <https://www.cs.toronto.edu/~kriz/cifar.html>

\* <https://pytorch.org/audio/stable/datasets.html>

Table 2: **Training hyper-parameters** for all architecture-dataset pairs.

Hyper-param.	MLP-FMNIST	ShallowCNN-SVHN	ResNet18-CIFAR10	LeNet5-ModelNet10	M5-SpeechCmds
Optimizer	Adam	Adam	Adam	Adam	Adam
LR	0.001	0.001	0.001	0.006	0.01
Weight decay			0.0001		
LR schedule	CosineLR	CosineLR	CosineAnnealing WarmRestarts, $T_0 = 25$ $T_{mult} = 25$	CosineAnnealing $T_{max} = 6'000$ , $\eta_{min} = 5 \cdot 10^{-6}$	CosineAnnealing $T_{max} = 100$ , $\eta_{min} = 0$
Batch size	64	256	512	256	256
Epochs	500	500	1'000	6'000	100
Data augment.	Normali- zation	Normali- zation	Normalization, HorizontalFlip	None	Resample to 16 KHz

### B.3 Transformations

This paper evaluates SCNs on the following computer vision and audio signal transformations: 2D rotation, scaling, translation, 3D rotation-and-projection, brightness, contrast, saturation, sharpness, pitch shift and speed change described below. Figure 10 illustrates examples of transformations applied to a sample input, showcasing various non-obvious effects that result in a decrease in input quality. Consequently, this decrease in quality adversely affects the accuracy of a trained classifier.

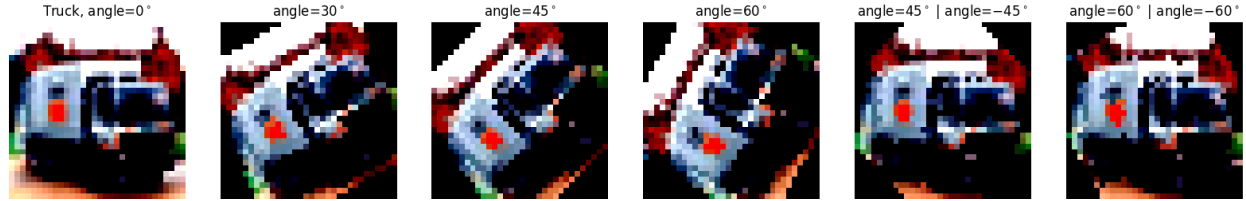
**2D rotation.** The rotation transformation is parameterized by a single angle  $\phi$  in the range  $0-2\pi$ . We use  $\alpha = (\cos(\phi), \sin(\phi))$  as input to the configuration network when learning SCNs for 2D rotations. The transformation preserves distances and angles, yet may lead to information loss due to cropped image corners and rounding effects. It can be inverted with little loss of image quality, as can be observed in Figure 10.

**Scaling.** The transformation is parameterized by the scaling factor in the range  $0.2-2.0$ , which is input to the hypernetwork to learn the configuration  $\beta$ -space for this transformation. Scaling transformation leads to a considerable loss of image quality. When inverted, the image appears highly pixelated or cropped.

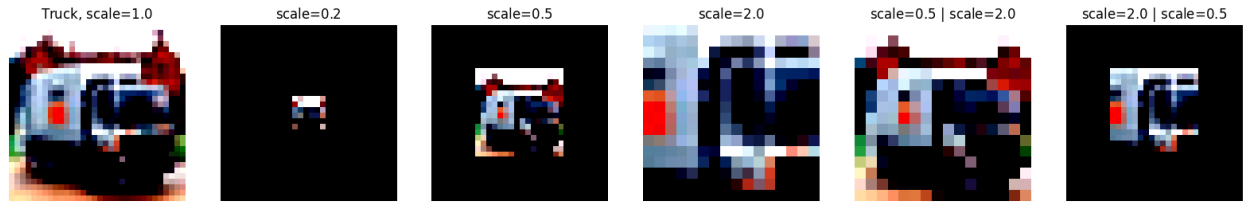
**Translation.** We consider image shifts within the bounds  $(-8,8)$  and  $(8,8)$  pixels. A shift is represented by two parameters  $\alpha = (\alpha_x, \alpha_y)$  reflecting the shift along the  $x$  and  $y$  axes. Note that an image gets cropped when translation is undone. In the FMNIST dataset the feature objects are positioned at the center of the image, which mitigates the negative impact of translations compared to other datasets like SVHN and CIFAR10.

**3D rotation.** The 3D rotation transformation is parameterized by the three Euler angles that vary in the range  $(-\pi, \pi)$ . We use  $\alpha = (\cos(\phi_1), \sin(\phi_1), \cos(\phi_2), \sin(\phi_2), \cos(\phi_3), \sin(\phi_3))$  as the input to the hypernetwork for learning SCNs on 3D rotations. Note that a different order of the same combination of these three angles may produce a different transformation output. We apply a fixed order  $(\phi_1, \phi_2, \phi_3)$  in all 3D rotation experiments. After rotation the 3D point cloud is projected on a 2D plane. When applying 3D rotations, it is possible to lose pixels in cases where the rotation axis is parallel to the projection plane. An example is shown in Figure 10.

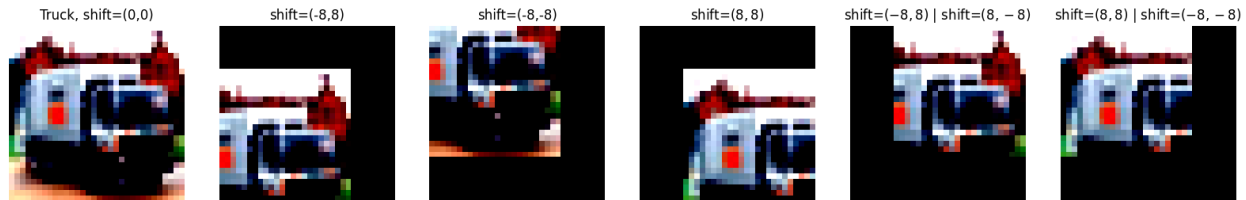
**Color transformations.** We explore SCN performance on four common color transformations: brightness, contrast, saturation and sharpness. The *brightness* parameter governs the amount of brightness jitter applied to an image and is determined by a continuously varying brightness factor. The *contrast* parameter influences the distinction between light and dark colors in the image. *Saturation* determines the intensity of colors



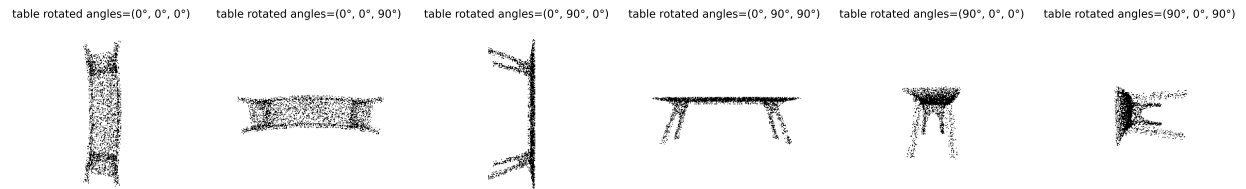
(a) **2D rotation transformation** parameterized by an angle  $\phi$  in the range  $(0-2\pi)$ . The transformation preserves angles and distances and can be undone with little loss of image quality (the edges of the input image may get cropped, rounding effects may occur).



(b) **Scaling transformation** parameterized by a scaling factor in the range  $(0.2-2.0)$ . Preserves only angles, not fully invertible, reduces input quality, large portions of the input image may get cropped.



(c) **Translation transform with a shift in  $(-8,-8)-(8,8)$** . Fully invertible only for the part of the input image inside the middle square  $(8,8)$  to  $(24,24)$ .



(d) **3D rotation transform**. We rotate an object in 3D along XY, YZ, and XZ planes using 3 angles  $(\phi_1, \phi_2, \phi_3)$ ,  $\phi_i \in (-\pi, \pi)$  and sample a point cloud of 4'096 points. Rotations in XZ (e.g., angles= $(0, \frac{\pi}{2}, 0)$ ) and YZ (e.g., angles= $(\frac{\pi}{2}, 0, 0)$ ) planes can block some pixels (e.g., the table surface, which is not visible in the picture).

Figure 10: **Geometric transformations** used in this work applied to a sample input. Notice how the images get impacted when inverse transformation is applied, showing a loss of input quality due to rounding, re-scaling and cropping.

present in an image. Lastly, *sharpness* controls the level of detail clarity in an image. We vary the continuously changing  $\alpha$  parameter between 0.2 and 2.0 for all considered color transformations.

**Audio signal transformations.** We use SCNs with pitch shift and speed adjustment transformations. Pitch shift modifies the pitch of an audio frame by a specified number of steps, with the parameter adjusted within the range of -10 to +10. Similarly, speed adjustment alters the playback speed by applying an adjustment factor, with speed changes applied within the range of 0.1 to 1.0.

## C Related work

Networks trained on extensive datasets lack robustness to common transformations of the input, such as rotation (Gandikota et al., 2021), scaling (Ye et al., 2021), translation (Biscione and Bowers, 2022) or small deformations (Engstrom et al., 2017). For example, Gong et al. (2014) showed that CNNs achieve neither rotation nor scale invariance, and their translation invariance to identify an object at multiple locations after seeing it at one location is limited (Biscione and Bowers, 2022; Blything et al., 2020; Kauderer-Abrams, 2018). Moreover, deep networks remain susceptible to adversarial attacks with respect to these transformations, and small perturbations can cause significant changes in the network predictions (Gandikota et al., 2021). There are three major directions of research to address the problem in the model design phase: Modifying the training procedure, the network architecture, or the data representation. Alternatively, the problem can be treated as a domain adaptation challenge and solved in the post-deployment phase. Below we summarize related literature on these topics.

**Modifying the training scheme.** The methods that modify the training scheme replace the loss function  $\mathcal{L}$  with a function that considers all parameters of transformations  $T$  in a range where the solution is expected to be invariant. Common choices are minimizing the mean loss of all predictions  $\{G(u(x_i), \theta) | u \in T\}$  resulting in training with data augmentation (Botev et al., 2022), or minimizing the maximum loss among all predictions leading to adversarial training (Engstrom et al., 2017). Both training schemes do not yield an invariant solution with respect to transformations such as rotation, as discussed in (Gandikota et al., 2021). The use of regularization can also improve robustness, yet provides no guarantees (Simard et al., 1991; Yang et al., 2019). Pre-training on large-scale datasets also improves model robustness (Entezari et al., 2023; Shariatnia et al., 2022). Overall, modifications of the training procedure are popular in practical applications, since they do not require characterization of the transformations applied to the input data, that are often unknown and may include a mix of complex effects.

**Designing invariant network architectures.** Network architectures can be designed to be invariant to structured transformations based on a symmetric group action that preserves class labels. For example, it is commonly believed that convolutional neural networks are architecturally invariant to translation due to the design characteristics of their convolution and max pooling layers (Kauderer-Abrams, 2018; Marcus, 2018). However, multiple studies argue that translation invariance of CNNs is rather limited (Biscione and Bowers, 2022; Blything et al., 2020). Invariant CNNs also fail to learn the spatial relation between higher-level features due to max pooling layers, leading to the Picasso Problem (Ribeiro et al., 2022). Nevertheless, designing invariant architectures to a particular transformation is the subject of many recent works (Bronstein et al., 2021; Frasca et al., 2022; Libera et al., 2019; Weiler and Cesa, 2019) due to the desirable robustness properties they offer in practice (Gandikota et al., 2021; Schneuing et al., 2022).

2D and 3D rotation invariant architectures play an important role in computer vision tasks. For instance, for a successful object classification, orientation of the coordinate system should not affect the meaning of the data. Therefore, a broad research literature is devoted to designing rotation invariant and equivariant architectures. Cohen and Welling (2016); Marcos et al. (2016); Veeling et al. (2018) use rotated filters to achieve layer-wise equivariance to discrete rotation angles. For continuous rotations, Worrall et al. (2016) proposed circular harmonic filters at each layer. These approaches were consolidated in (Weiler and Cesa, 2019). To achieve robustness to image rotation and in contrast to layer-wise schemes, Henriques and Vedaldi (2016) include multiple rotated copies of images to extract rotation invariant features. Jaderberg et al. (2015); Jafari-Khouzani and Soltanian-Zadeh (2005); Tai et al. (2019) align transformed images using different methods, *e.g.*, using principal component analysis (PCA). Real world rarely conforms to strict mathematical symmetry either due to noisy or incomplete data. Wang et al. (2022a) explore approximately equivariant networks which relax symmetry preserving constraints.

Weiler et al. (2018) and Thomas et al. (2018) propose 3D rotation equivariant kernels for convolutions. Esteves et al. (2017) propose a polar transformer network by learning a transformation in a polar space in which rotations become translations and so convolutional neural networks become effective to achieve

rotation invariance. When 3D objects are presented as point clouds, this solves problems that arise due to object discretization, but leads to a loss of information about the neighbor relationship between individual points. Zhang and Rabbat (2018) add graph connections to compensate for this information loss and use graph convolutions to process the cloud points. Qi et al. (2016, 2017) additionally include hierarchical and neighborhood information. A number of works build 3D rotation-invariant architectures, *e.g.*, Wang et al. (2022b) introduce a relation of equivalence under the action of rotation group.

**Canonicalization of data representation.** Input canonicalization is the process of converting the data into a specific form to simplify the task to be solved by a deep model. For example, by learning to map all self-augmentations of an image to similar representations is the main idea behind contrastive learning methods such as SimCLR (Chen et al., 2020) and Supervised Contrastive Learning (Khosla et al., 2020). Canonicalization can also be achieved by learning to undo the applied transformation or learning a canonical representation of the data (Kaba et al., 2022). For example, Jaderberg et al. (2015) propose a Spatial Transformer layer to transform inputs to a canonical pose to simplify recognition in the following layers. BFT layers (Dao et al., 2019) can be used to learn linear maps that invert the applied transform. Earlier works on the topic considered manual extraction of features in the input that are robust, or invariant, to the desired transformation (Manthalkar et al., 2003; Yap et al., 2010).

**Domain adaptation.** Robustness to input transformations can be framed as a domain shift problem (Lohmani et al., 2020). Domain adaptation methods described in the related literature follow different strategies as to how they align the source and the target. For example, Xu et al. (2018) define a distance measure between source and target data in the feature space and minimize this during training the network to reduce domain shift. Russo et al. (2017) train a generator that makes the source and target data indistinguishable for the domain discriminator. Another group of methods uses self-supervised learning to reduce domain shift (Jiaolong et al., 2019). In many real world scenarios, the data from the target domain are available only in the post-deployment phase. Therefore, domain adaptation methods often face memory and computing resource constraints making the use of backpropagation too costly. Partial model updates, especially those executed sequentially, may reduce model quality (Vucetic et al., 2022).

**Linear mode connectivity, generalization and robustness.** In this work we show that optimal model weights that correspond to parameterized continuous transformations of the input reside in a low-dimensional linear subspace. This finding, supported by our theoretical insights, connects this work to recent research on the properties of the loss landscape and its relationship with generalization and optimization (Entezari et al., 2021; Fort et al., 2019; Geiger et al., 2019; Jordan et al., 2022; Juneja et al., 2022; Li et al., 2017; Mei et al., 2018; Nguyen et al., 2018; Şimşek et al., 2021). In particular, the existence of linear paths between solutions trained from independent initializations (Entezari et al., 2021), those that share a part of their learning trajectories (Frankle et al., 2020), or trained on data splits (Ainsworth et al., 2022). Wortsman et al. (2021) learn neural network subspaces containing diverse and at the same time linear mode connected (Frankle et al., 2020; Nagarajan and Kolter, 2019) solutions that can be effectively weight-space ensembled. This work builds upon and extends these works to linear mode connectivity between optimal models trained for different input transformation parameters.

## D Configuration subspaces and SCN efficiency

### D.1 SCN performance

**SCN performance on geometric transformations.** Figure 11 complements Figure 2 in the main paper and presents the performance of SCNs for 2D rotation on ShallowCNN-SVHN and ResNet18-CIFAR10. We observe high efficiency of SCNs compared to the baselines even for very small  $D$ . A close inspection of models trained for a fixed input degree ( $\phi = 0^\circ$ ) shows increasingly higher specialization of the trained models to the respective transformation parameter setting.

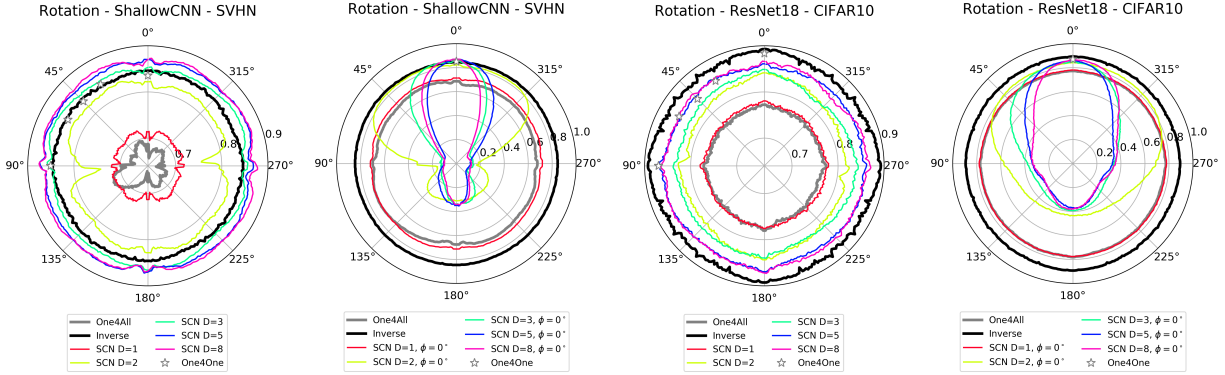


Figure 11: **SCN test set accuracy on 2D rotations.** From left to right: A pair of plots for ShallowCNN–SVHN and ResNet18–CIFAR10. The models in each pair show SCN’s performance for changing input  $\alpha = (\cos(\phi), \sin(\phi))$  and for a fixed  $\alpha$  with  $\phi = 0^\circ$ .

**SCN performance on color transformations.** Color transformations are simple. SCNs achieve high performance already for very small  $D$  (see Figure 12). There is little performance difference between our baselines One4All, One4One and Inverse despite the small inference network architectures (1-layer MLP with 32 hidden units for FMNIST and 2-layer ShallowCNN with 32 channels for SVHN) used in the experiments. Nevertheless, SCNs can often outperform all baselines.

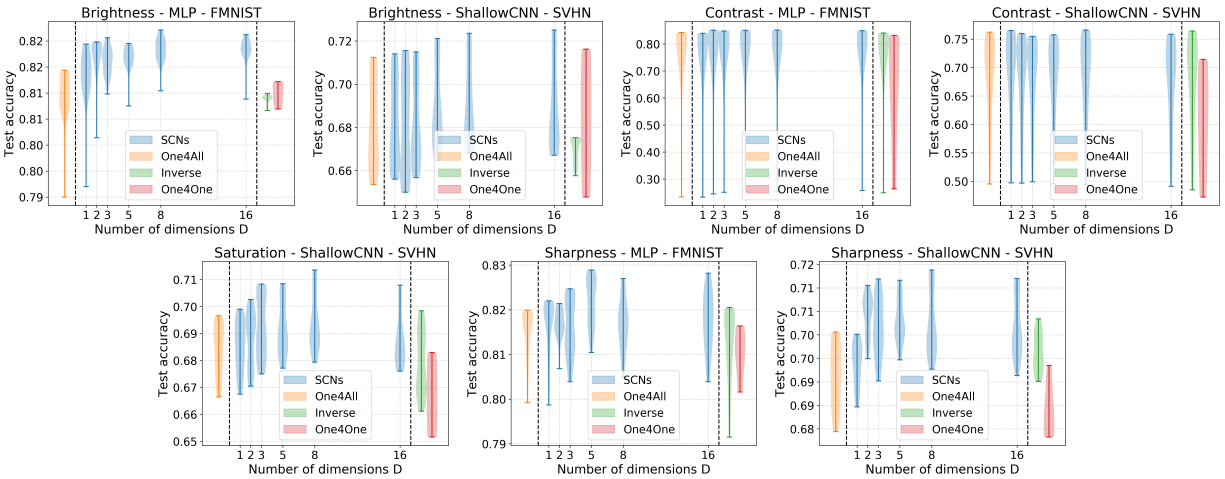


Figure 12: **Summary of SCN performance on color transformations:** brightness, contrast, saturation and sharpness. We present the results for MLP-FMNIST and ShallowCNN-SVHN architecture-dataset pairs. All transformations are simple. SCNs match the baselines for very low  $D$ . Note that saturation has no effect on black-white images. Therefore, for MLP-FMNIST the difference in model performance is the same up to the choice of a random seed.

**SCN performance on audio signal transformations.** Figure 13 presents the performance of SCNs on two audio signal transformations: pitch shift and speed change. For both transformations a small  $D$  is sufficient for SCN to match or outperform the baselines. Note that the M5 architecture takes raw waveform in the time domain as input rather than a spectrogram.

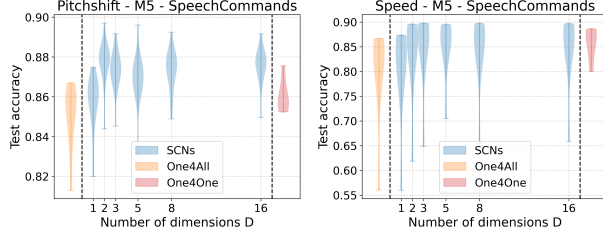


Figure 13: **Summary of SCN performance on audio signal transformations:** pitch shift and speed using M5 as inference architecture. SCNs match the performance of baselines already for small  $D$ .

## D.2 Configuration $\beta$ -space visualization

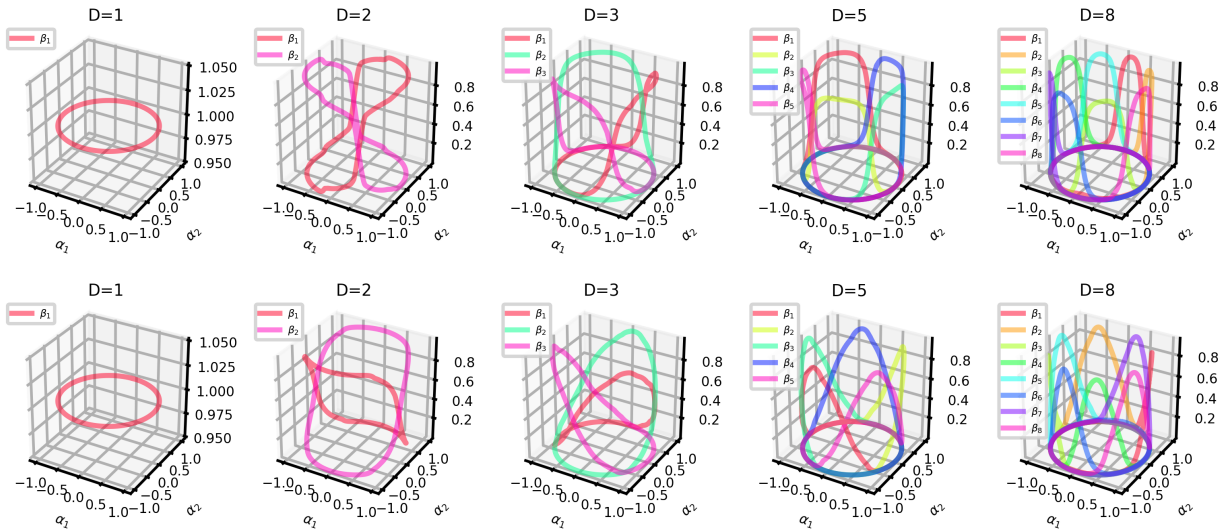


Figure 14: **Configuration  $\beta$ -space of SCNs trained for 2D rotation** on further datasets and inference network architectures, complementing Figure 4. **Top:** 1-layer MLP with 32 hidden units on FMNIST. **Bottom:** 2-layer ShallowCNNs with 32 filters in the hidden layers on SVHN. Transform parameters are  $\alpha = (\alpha_1, \alpha_2) = (\cos(\phi), \sin(\phi))$ , with  $\phi$  being a rotation angle.

Different dataset-architecture pairs exhibit a similar structure in the  $\beta$ -space. Figure 14 and Figure 15 present the learned configuration parameters  $\beta$  as a function of the transformation parameters  $\alpha$  for 2D rotation and scaling, respectively, complementing the findings in Figure 4 of the main paper. It is worth noting the slight variations in the shape of the learned curves, which are specific to the architecture-dataset pairs used to train SCNs. Based on the consistent  $\beta$ -space across different dataset-architecture pairs, we infer that the configuration space primarily relies on the transformation and the characteristics of its parameter vector  $\alpha$ .

## E Translation transformation

**SCNs on architectures NOT invariant to translation.** Figure 16 shows SCN performance on translation transformation for 1-layer MLP with 32 hidden units as inference network trained on the FMNIST dataset. SCN’s test accuracy increases with higher  $D$  matching the accuracy of the Inverse baseline. The visualization

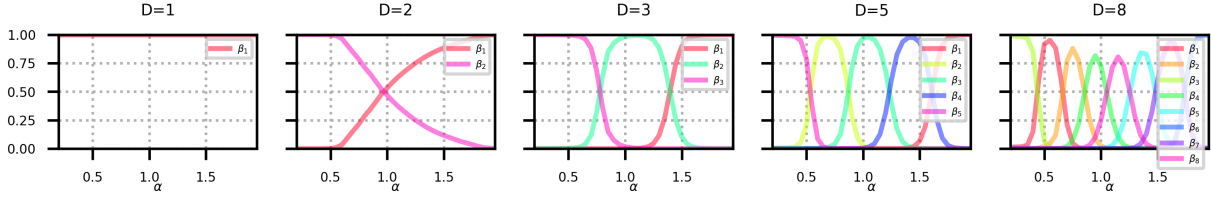


Figure 15: **Configuration  $\beta$ -space of SCNs trained for scaling using MLP on FMNIST.** Scaling factor  $\alpha$  varies in the range 0.2–2.0. The  $\beta$ -space looks similarly as the one presented in the main paper for a different dataset-architecture pair.

allows identifying high accuracy areas of each base model. With higher  $D$ , the area of a dedicated model for a specific parameter setting decreases, whereas its test accuracy increases.

**SCNs on translation-invariant architectures.** Using translation-invariant architecture as inference network part of SCN trained for translation results in a degenerated  $\beta$ -space with only one base model. This special case is exemplified in Figure 17. On the FMNIST dataset side, we scale the input images down by 50% and apply padding of 8 to ensure that shifting the image within  $(-8,8)$  along horizontal and vertical axes leads to a pure translation of the object in the image without information loss. As translation-invariant network architecture, we use a 2-layer CNN built only of convolutional and max pooling layers with kernel size of 4 and 16 channels. For any  $D$ , SCNs learn a single model with only one  $\beta_i=1$  and other  $\beta_j, j \neq i$  being zero. The checkered structure of the test accuracy plot reflects the size of the filters. A detailed explanation of its origin and its relation to the Nyquist–Shannon sampling theorem is given in (Zhang, 2019).

## F 3D rotation transformation

Figure 18 shows all views of the  $\beta$ -space of SCN for 3D rotation as a function of input parameters  $\alpha = (\cos(\phi_1), \sin(\phi_1), \cos(\phi_2), \sin(\phi_2), \cos(\phi_3), \sin(\phi_3))$ , where  $\phi_i, i = 1..3$  is a rotation angle in the YZ, XZ and XY planes, respectively. Figure 18 shows the whole  $\beta$ -space for 3D rotation presented as a function of all pairwise combinations of  $\phi_i$ . In Figure 18 middle and bottom,  $\beta$ s show a stable trend along the  $\phi_3$ -axis, indicating that the 3D rotation in the XY plane keeps all object pixels (and is basically the same as 2D rotation in this case). In Figure 18 (top),  $\beta$ -space has cosine-like trend along both  $\phi_1$  and  $\phi_2$  axes, reflecting the 3D rotations in YZ and XZ planes. These transformations lead to information loss as some parts of an object rotate out of the view and get blocked. In all plots  $\beta$ -surfaces are not flat or degenerated. By observing the similarities and changing trends in the learned  $\beta$ -space for 3D rotation, it can be inferred that the shape of this configuration space primarily relies on the transformation itself and its associated parameters, namely  $(\phi_1, \phi_2, \phi_3)$ . We provide a link\* to an interactive website visualizing the  $\beta$ -space of sample SCNs, including those trained for 3D rotation.

## G Search algorithm in the $\alpha$ -space

This section provides details on the performance of the search algorithm which estimates  $\alpha$  from a stream of input data. As mentioned in the main paper, we can leverage the fact that the correct input parameters  $\alpha$  should produce a confident low-entropy classification result (Hendrycks and Gimpel, 2016; Wortsman et al., 2020). Therefore, our search algorithm estimates  $\alpha$  from a batch of input data by minimizing the entropy of the model output on this batch by exploring the output of optimal models in the learned low-dimensional

\*<https://subspace-configurable-networks.pages.dev/>



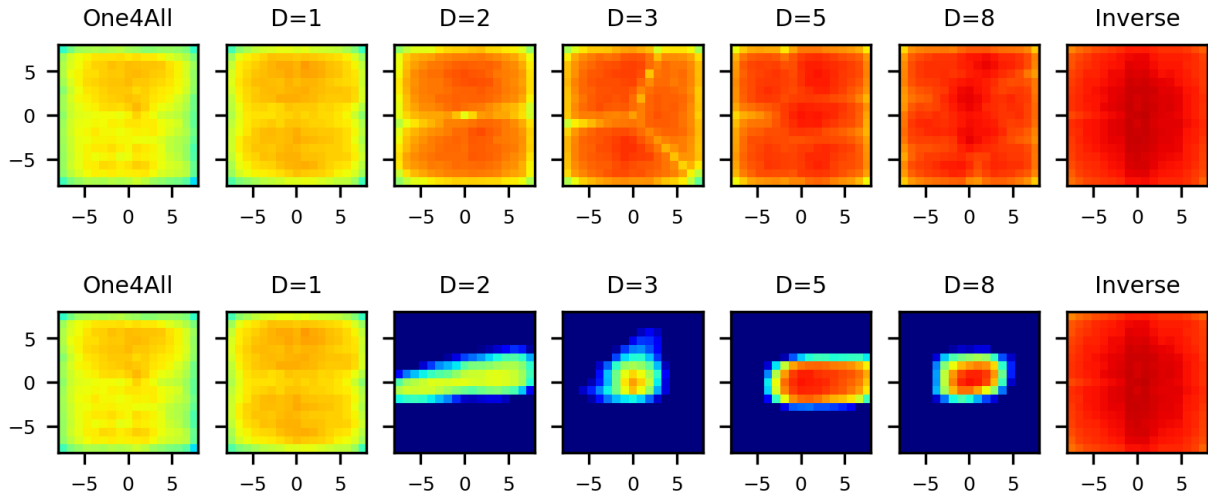


Figure 16: **SCN performance for translation trained with MLP inference network on FMNIST.** In this example, applying translation to an input image leads to information loss, since the part of the image shifted outside the image boundary gets cut. We use 1-layer MLPs with 32 hidden units and a bias term. This architecture is not translation-invariant. In all plots the color map is "rainbow" ranging uniformly between 0.5 or below (dark blue) to 0.9 (dark red). X and Y axes are horizontal and vertical shift parameters ( $\alpha_x, \alpha_y$ ) applied to the input. **Top:** Test accuracy of SCNs for  $D = 1..8$  for every  $(\alpha_x, \alpha_y)$  combination, compared to One4All and Inverse baselines. **Bottom:** Test accuracy of SCNs for the dedicated fixed (0,0) shift evaluated on shifted inputs. The area of high accuracy decreases with higher  $D$ , leading to higher degree of model specialization, higher accuracy of the dedicated model for each setting, and a better overall performance of SCNs.

subspace. We use the basin hopping\* method to find the solution (with default parameters, iter=100, T=0.1, method=BFGS).

The following code snippet runs the search in the  $\alpha$ -space to estimate the best rotation angle  $\alpha$  from a batch of data  $X$  by minimizing the function  $f(\cdot)$ . The angle transformation function converts an input angle in degrees to the corresponding  $(\cos, \sin)$  pair.

```

1 from scipy import optimize
2
3 # function to be minimized by the basin hopping algorithm
4 def f(z, *args):
5     alpha = transform_angle(((1+z)*180)%360-180)
6     X = args[0]
7     logits = model(Tensor(X), hyper_x=Tensor(alpha))
8     b = (F.softmax(logits, dim=1)) * (-1 * F.log_softmax(logits, dim=1)) # entropy
9     return b.sum().numpy()
10
11 # given a batch of images find the rotation angle alpha using basin hopping algorithm
12 def findalpha(X):
13     mkwargs = {"method": "BFGS", "args":X}
14     res = optimize.basinhopping(f, 0.0, minimizer_kwargs=mkwargs, niter=100, T=0.1)
15     alpha = ((1+res.x[0])*180)%360-180
16     return alpha

```

\*<https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html>

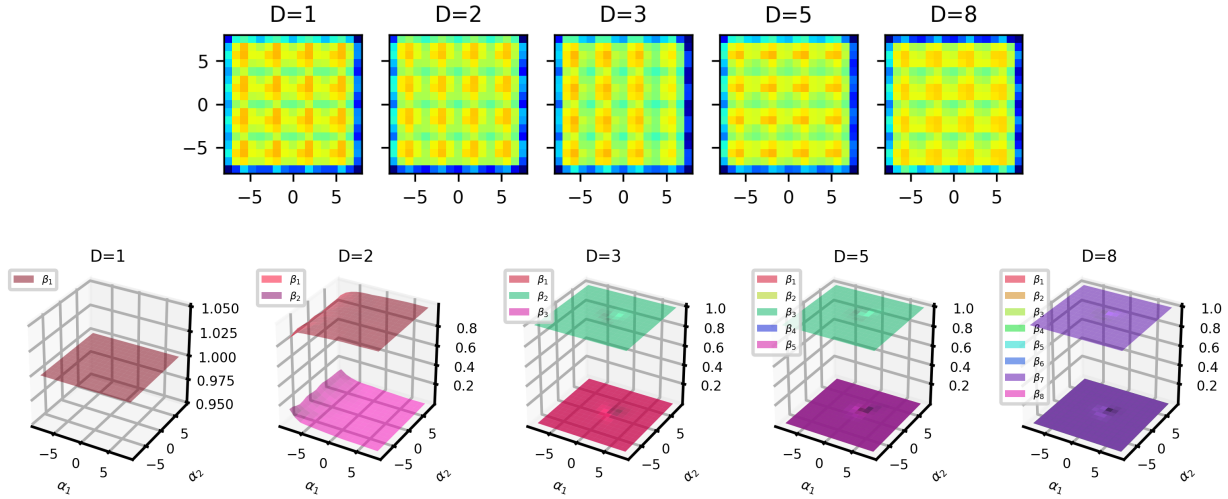


Figure 17: **SCN performance for translation trained on translation-invariant CNN architecture on FMNIST.** SCNs for all  $D$  learn a degenerated  $\beta$ -space with only one active model (only one  $\beta_i=1$ ) for all inputs. **Top:** Independently trained SCNs for different  $D$  yield very similar accuracy of 0.85. The checkered structure of the plots reflects the size of the filters, which is  $4 \times 4$ . **Bottom:** Configuration  $\beta$ -space showing that only one  $\beta_i$  equals 1.0 for all input parameters  $\alpha$ .

```

17
18 # test search algorithm performance on a test set
19 result = 0.0
20 for (X, y) in test_loader:
21     angle = random.uniform(-180, 180)
22     X = TF.rotate(X, angle)
23
24     alpha = findalpha(X)
25
26     # compute model prediction with the estimated alpha
27     logits = model(X, hyper_x=transform_angle(alpha))
28     # y is the true label --> calculate accuracy
29     correct = (logits.argmax(1) == y).type(torch.float).sum().item() / batch_size
30     result += correct
31
32 result /= len(test_loader.dataset) / batch_size
33 print(f"Test accuracy: {(100*result):>0.1f}%")

```

To improve the accuracy of the search, SCN training is enhanced with an additional regularizer to minimize the model output entropy value for the correct  $\alpha$  and maximise it for a randomly sampled  $\alpha$ . The train function is sketched in the listing below.

```

1 loss_fn = nn.CrossEntropyLoss()
2 optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
3 cos = nn.CosineSimilarity(dim=0, eps=1e-6)
4
5 def train(data_loader, model, loss_fn, optimizer):
6     for (X, y) in data_loader:
7         X, y = X.to(device), y.to(device)
8         angle = random.uniform(0, 360)

```

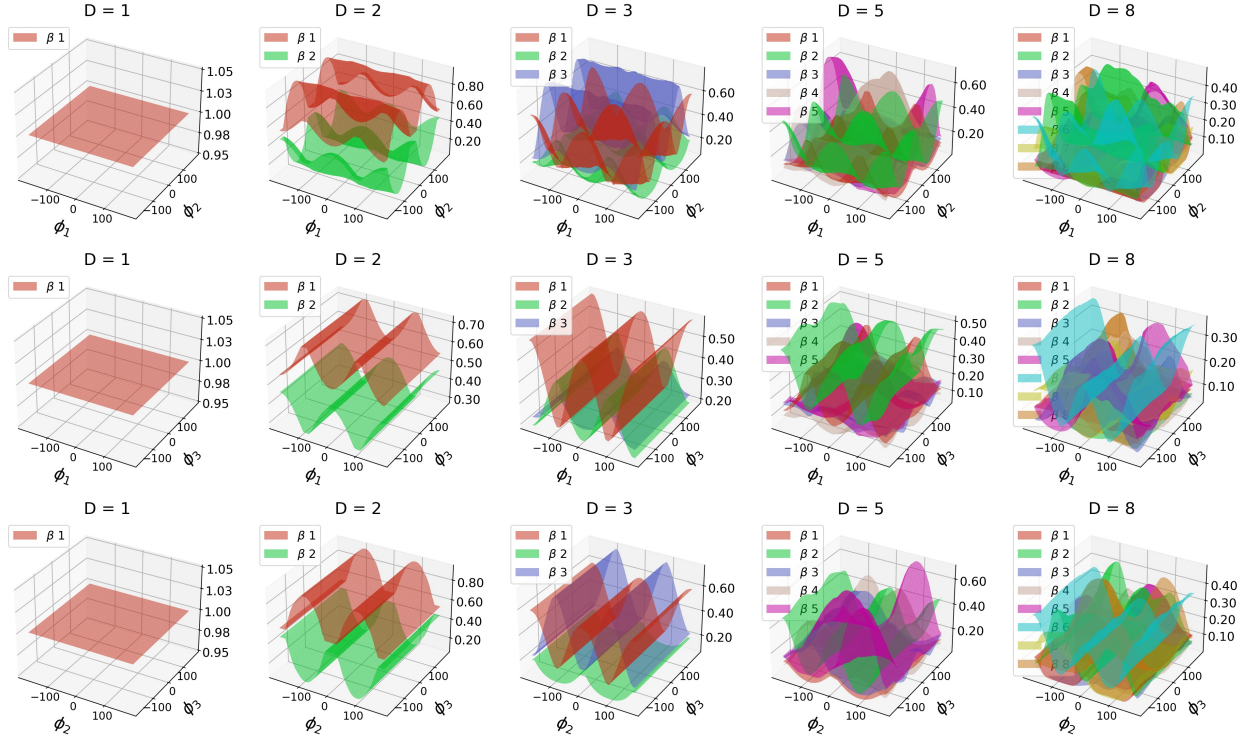


Figure 18:  $\alpha - \beta$  space of SCNs trained for 3D rotation on ModelNet10 with LeNet5 inference architecture for  $D = 1..8$ . Transformation parameters  $\alpha$  result from applying  $\cos(\cdot)$  and  $\sin(\cdot)$  functions to the vector of rotation angles  $(\phi_1, \phi_2, \phi_3)$ , with each  $\phi_i$  in the range  $(-\pi, \pi)$ . **Top:** Subspace of SCNs when changing  $(\phi_1, \phi_2)$ , and fixing  $\phi_3 = -\pi$ . **Middle:** Subspace of SCNs when changing  $(\phi_1, \phi_3)$ , and fixing  $\phi_2 = -\pi$ . **Bottom:** Subspace of SCNs when changing  $(\phi_2, \phi_3)$ , and fixing  $\phi_1 = -\pi$ .

```

9       X = TF.rotate(X, angle)
10
11      # make prediction and compute the loss
12      pred = model(X, hyper_x=transform_angle(angle).to(device))
13      loss = loss_fn(pred, y)
14
15      # regularize (cosine similarity squared) in the beta space
16      beta1 = model.hyper_stack(transform_angle(angle).to(device))
17      angle2 = random.uniform(0, 360)
18      beta2 = model.hyper_stack(transform_angle(angle2).to(device))
19      loss += pow(cos(beta1, beta2), 2)
20
21      # minimize entropy for the correct degree
22      b1 = (F.softmax(pred, dim=1)) * (-1 * F.log_softmax(pred, dim=1))
23      loss += 0.01*b1.sum()
24
25      # maximize entropy for a wrong / random degree
26      logits = model(X, hyper_x=transform_angle(angle2).to(device))
27      b2 = (F.softmax(logits, dim=1)) * (-1 * F.log_softmax(logits, dim=1))
28      loss -= 0.01*b2.sum()
29
30      optimizer.zero_grad()

```

```
31     loss.backward()
32     optimizer.step()
```

---

The interested reader can check the source code for further details.\*

Note that the basin hopping algorithm is computationally expensive. For the 2D rotation transformation on FMNIST dataset, the method may run several hundreds of model inferences to find a good solution. Optimizing the running time of the method is beyond the scope of this paper, because in practice  $\alpha$ -search can be avoided, *e.g.*, by using an additional sensor modality as input or by discretizing the search space to a manageable number of models. The expensive  $\alpha$ -search aims to show that the problem of estimating  $\alpha$  and building I-SCNs is solvable in principle.

---

\*<https://github.com/osaukh/subspace-configurable-networks.git>