SUBSPACE-CONFIGURABLE NETWORKS

Dong Wang[†,*], Olga Saukh[†◇,*], Xiaoxi He[‡], Lothar Thiele[§]
[†]Graz University of Technology, Austria
[◇]Complexity Science Hub Vienna, Austria
[‡]University of Macao, China
[§]ETH Zurich, Switzerland
{dong.wang,saukh}@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 techniques. 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. Our source code is online.[1]

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 or naturally changing sensing conditions.

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.

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 (Marcus, 2018; Kauderer-Abrams, 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 (Russo et al., 2017; Xu et al., 2018).

Unlike traditional invariant architectures, configurable networks explicitly define invariances by parameterizing desired input data transformations. We introduce subspace-configurable networks (SCNs), which are trained with weights residing in a subspace formed by a few base models. By receiving a parameter vector for an input transformation, SCNs select appropriate high-accuracy model weights from this subspace, effectively isolating the targeted invariances. We evaluate our SCN

[*]Equal contributation
[1]https://github.com/osaukh/subspace-configurable-networks

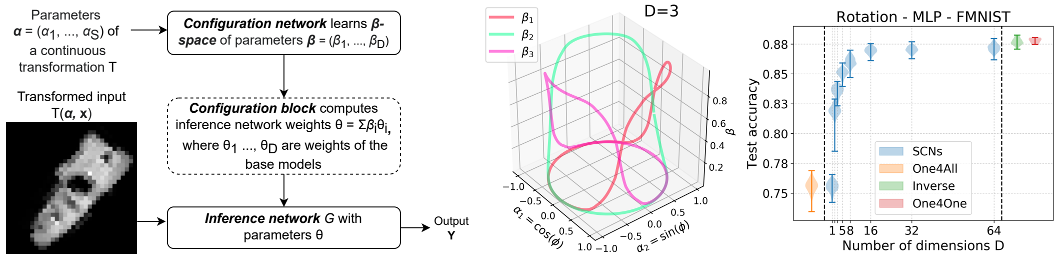


Figure 1: **Training subspace-configurable networks (SCNs)**, where an optimal network for a fixed transformation parameter vector is part of the subspace retained by few configuration parameters. **Left:** Given input transformation parameters α, *e.g.*, a rotation angle for a 2D rotation, we train a *configuration network* which yields a D-dimensional configuration subspace (*β-space*) to construct an efficient inference network with weights $\theta = \sum \beta_i \cdot \theta_i$, where θ_i are the weights of the *base models*, and β is a *configuration vector*. **Middle:** Optimal model parameters in the configuration subspace as functions of the rotation angle α 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 θ_i and three configuration vectors β_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).[2] Each violin shows the performance of a model on all degrees with a discretization step of 1°, expect for One4One where the models are independently trained and evaluated on 0, $\pi/6$, $\pi/4$, $\pi/3$, $\pi/2$ rotated input.

models by studying 2D translation, scaling and translation. In the appendix, we also evaluate a wide range of real-world transformations, including complex irreversible transformations, covering both computer vision and audio signal processing domains, along with dedicated network architectures. To uncover configuration subspaces for a set of input transformations, SCNs leverage a hypernet-inspired architecture (Ha et al., 2016) 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 for 2D rotation is shown in Figure 1 middle). Interestingly, the configuration parameter vectors form structured geometric objects, revealing the optimal parameter subspaces' structure. If the inference network capacity is limited by edge devices' resource constraints, 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. 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. SCNs are evaluated on ten common real-world transformations, using five backbones and five standard benchmark datasets (Section 2-3 and Appendix B- C).
- SCNs take transformation parameters as input, yet these parameters can be estimated from the input data. We provide an algorithm which allows building a transformation-invariant architecture on top of SCNs (Section 2 and Appendix D).
- In practical IoT scenarios the parameter supply can be replaced with a correlated sensor modality. We implemented SCNs on two resource-constrained devices and show their outstanding performance and remarkable efficiency (further details are provided in our arXiv preprint (Saukh et al., 2023)).

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

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 $T : \mathbb{R}^S \times \mathbb{R}^N \to \mathbb{R}^N$ parameterized by the vector $\alpha = (\alpha_1, \cdots, \alpha_S) \in \mathbb{A} \subseteq \mathbb{R}^S$ 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 $T(\alpha, \mathbb{X}) : \mathbb{A} \times \mathbb{R}^{P^2} \to \mathbb{R}^{P^2}$ is modulated by pose parameters α, such as rotation, scaling, translation or cropping. We assume that data transformations $T(\alpha, \mathbb{X})$ preserve the label class of the input and represent a continuous function of $\alpha \in \mathbb{A}$, *i.e.*, for any two transformation parameters α_1 and α_2 there exists a continuous curve in $\mathbb{A}$ that connects two transformation parameters. Note that by changing α we transform all relevant data distributions the same way, *e.g.*, the data used for training and test. The set $\{T(\alpha, x) \,|\, \alpha \in \mathbb{A}\}$ of all possible transformations of input x is called an *orbit* of x. We consider an infinite orbit defined by a continuously changing α.

We consider an inference network to represent a function $g : \mathbb{X} \times \mathbb{R}^L \to \mathbb{Y}$ 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 inference network 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 α, we write $E(\theta, \alpha)$ to make this dependency explicit. Optimal parameters θ^*_α are those that minimize the loss $E(\theta, \alpha)$ for a given transformation vector α.

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 with function $h(\cdot)$ and an inference network with function $g(\cdot)$ connected by a linear transformation of network parameters $\theta = f(\beta)$ computed in the configuration block:

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

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 with the function $h : \mathbb{R}^S \to \mathbb{R}^D$ 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.*, θ_i are the base vectors of the corresponding linear subspace and β the coordinates of θ. The SCN training 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 θ_i. We use the standard categorical cross-entropy loss in all experiments. For a continuous transformation $T(\alpha)$, we provide in our arXiv paper (Saukh et al., 2023) theoretical results that help to understand the structure of the β-space using continuity arguments of the optimal solution space.

SCNs take transformation parameter α as input. However, one can also use a search algorithm to estimate α from the input data, aiming for low-entropy, confident classification results. This approach, inspired by previous techniques (Wortsman et al., 2020; Hendrycks & Gimpel, 2016), utilizes the basin hopping method Iwamatsu & Okabe (2004)targeting hard nonlinear optimization problems with a mix of global and local search phases. To enhance performance, SCN training includes a regularizer to optimize output entropy based on the correct α. Despite the computational demands of the α-search algorithm, it achieves high accuracy, allowing SCNs to serve as transformation-invariant networks. In practice, however, search in the α-space can often be avoided. If the transformation of interest is discrete and limited to a few cases, α-search can be reduced to running inference over a few candidate models. Most importantly, however, the parameter α can be inferred from a correlated sensor modality. We demonstrate this in two edge applications discussed in our arXiv paper (Saukh et al., 2023).

Several peculiarities of SCNs' design make them well-suited for resource-constrained devices. (1) SCNs effectively utilize memory hierarchies. Fast memory, such as SRAM, offers short access

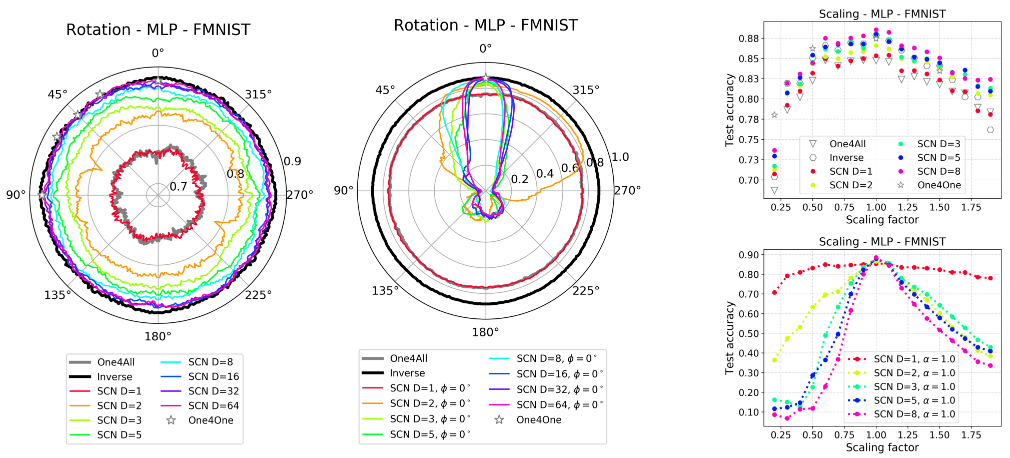


Figure 2: **SCN test accuracy for 2D rotation and scaling transformations. Left and middle:** 2D rotation parameterized by a rotation degree $\phi = 0..2\pi$ input to the configuration network as $\alpha = (\cos(\phi), \sin(\phi))$. For each α, SCN determines a configuration vector β used to build a dedicated model for every angle shown on the right. The right polar plot shows the performance of a single model ($\phi = 0°$) on all angles. The model works best for the input transformed with $T(\phi = 0°)$. 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. As D increases, the model becomes more specialized for the specified input parameters. Inference network is a 5-layer MLP with 32 hidden units in each layer trained on FMNIST. Also see Appendix B and videos showing SCN inference models for each parameter setting.[3]

times, but it usually has limited capacities. In contrast, larger-capacity Flash or EEPROM, despite slower access times, is ideal for storing less time-sensitive SCN reconfiguration data, such as the D base models holding θ_i and the parameters of the configuration network. For rapid inference, the inference network weights θ should better fully fit into RAM. (2) SCNs yield the most benefit if memory and thus the network capacity are limited, contributing to the body of work on optimizing deep models for resource constraints, *e.g.*, Corti et al. (2024). Given unlimited resources, One4All can match the performance of SCNs. Exceptions include corner cases where the transformation parameter α is used to break symmetries. (3) SCNs draw inspiration from the recent linear mode connectivity literature Entezari et al. (2021); Ainsworth et al. (2022), and empirically show that a linear reconfiguration function $f(\beta)$ yields great performance of models for different transformation parameters. This allows efficient SCN reconfiguration memory strategies, like memory page reuse. Furthermore, the computation of equation 1 is efficiently achieved through hardware-friendly linear vector operations, utilizing MLA and FMA instructions, vectorization and pipelining. Beside, integrating SCNs with parameter-efficient fine-tuning methods promises further optimization, a focus for our future research.

3 Experimental Results

In this section, we evaluate the performance of SCNs on 3 transformations (2D rotation, translation and scaling) and 3 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) and ResNet18 (He et al., 2015) on CIFAR10 (Krizhevsky et al., 2009). These transformations are continuous, and their parameterization is straightforward: For example, a rotation angle for a 2D rotation. The main paper assesses SCNs across various complex transformations to demonstrate their effectiveness, with additional results and details on other transformations, training hyper-parameters, architectural choices and dataset information available in Appendix A.

[3]`https://tinyurl.com/2nb8k644`

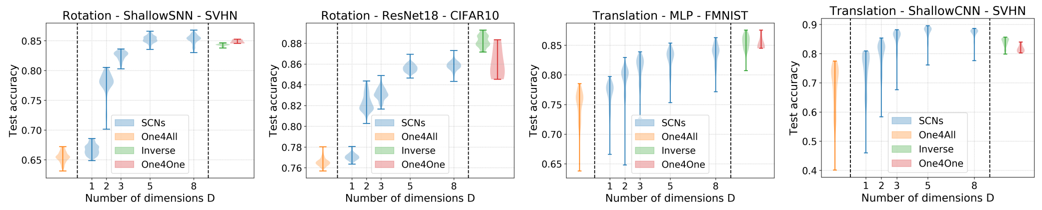


Figure 3: **SCNs achieve high test accuracy already for low D**, outperforming One4All and approaching (and in some cases outperforming) both Inverse and One4One baselines. **2 plots on the left:** 2D rotation on ShallowCNN–SVHN and ResNet18–CIFAR10. **2 plots on the right:** Scaling on MLP–FMNIST and ShallowCNN–SVHN. For translation, the violin for One4One comprises prediction accuracy of independently trained models for (0,0) and (±8,±8) shift parameters.

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. Finally, *One4One* represents a set of networks, each trained and tested on the dataset transformed with a fixed parameter vector α. For a given architecture, dataset, and loss function, a well-trained One4One baseline excels in in-distribution generalization, yet implementing its dynamic configuration on resource-constrained platforms is unfeasible. 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 accuracy, but we also analyze the impact of SCN dimensionality D on its performance, and the structure of the β-space.

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 α. In some cases, SCNs achieve higher accuracy than One4One networks trained and tested only on the transformed data for some fixed value of α, since One4One does not make use of data augmentation but SCN implicitly does due to its structure given in Equation 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°, 0.05 and 1 pixel for 2D rotation, scaling and translation respectively). The only exception here is the One4One baseline, represented by a violin plot for five models, each trained and tested on fixed-parameter transformed inputs. 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 B).

4 Conclusion

This paper presents subspace-configurable networks (SCNs) designed for model reconfiguration and robustness to input transformations under severe resource constraints, achieving high accuracies with low-dimensional configuration subspaces. SCNs facilitate post-deployment adaptation on resource-limited devices, offering a compact solution for robotics, edge computing and embedded systems. Further explorations and more details, refer to the Appendix and our arXiv paper (Saukh et al., 2023).

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.

References

Samuel K Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git Re-Basin: Merging models modulo permutation symmetries. *arXiv preprint*, 2022. URL `https://arxiv.org/abs/2209.04836`.

Aleksander Botev, Matthias Bauer, and Soham De. Regularising for invariance to data augmentation improves supervised learning. *arXiv preprint*, 2022. URL `https://arxiv.org/abs/2203.03304`.

Francesco Corti, Balz Maag, Joachim Schauer, Ulrich Pferschy, and Olga Saukh. Reds: Resource-efficient deep subnetworks for dynamic resource constraints, 2024.

Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. *CoRR*, 2016. URL `http://arxiv.org/abs/1610.00087`.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. *arXiv preprint*, 2017. URL `https://arxiv.org/abs/1712.02779`.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks. *arXiv preprint*, 2021. URL `https://arxiv.org/abs/2110.06296`.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020.

Jonas Geiping, Micah Goldblum, Gowthami Somepalli, Ravid Shwartz-Ziv, Tom Goldstein, and Andrew Gordon 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`.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. *arXiv preprint*, 2016. URL `https://arxiv.org/abs/1609.09106`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint*, 2016. URL `https://arxiv.org/abs/1610.02136`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Masao Iwamatsu and Yutaka Okabe. Basin hopping with occasional jumping. *Chemical Physics Letters*, 399(4-6):396–400, 2004. doi: 10.1016/j.cplett.2004.10.032.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *arXiv preprint*, 2015. URL `https://arxiv.org/abs/1506.02025`.

Sékou-Oumar Kaba, Arnab Kumar Mondal, Yan Zhang, Yoshua Bengio, and Siamak 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`.

Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv preprint*, 2018. URL `https://arxiv.org/abs/1801.01450`.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-100 and CIFAR-10 (Canadian Institute for Advanced Research), 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`. MIT License.

Dmitry Laptev, Nikolay Savinov, Joachim M. Buhmann, and Marc 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.

Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint*, 2018. URL `https://arxiv.org/abs/1801.00631`.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew 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.

Behnam Neyshabur. Towards learning convolutions from scratch. In *Advances in Neural Information Processing Systems*, 2020. URL `https://arxiv.org/abs/2007.13657`.

Paolo Russo, Fabio Maria Carlucci, Tatiana Tommasi, and Barbara Caputo. From source to target and back: symmetric bi-directional adaptive GAN. *CoRR*, 2017. URL `http://arxiv.org/abs/1705.08824`.

Olga Saukh, Dong Wang, Xiaoxi He, and Lothar Thiele. Representing input transformations by low-dimensional parameter subspaces. *arXiv preprint arXiv:2305.13536*, 2023.

M. Savva, F. Yu, Hao Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, Hang 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 (eds.), *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2016. ISBN 978-3-03868-004-8. doi: 10.2312/3dor.20161092.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, 2018. URL `http://arxiv.org/abs/1804.03209`.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *arXiv preprint*, 2020. URL `https://arxiv.org/abs/2006.14769`.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1912–1920, 2015. doi: 10.1109/CVPR.2015.7298801.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.

Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. Unsupervised domain adaptation: An adaptive feature norm approach. *CoRR*, 2018. URL `http://arxiv.org/abs/1811.07456`.

APPENDIX

Table 1: **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.	Normalization	Normalization	Normalization, HorizontalFlip	None	Resample to 16 KHz

A IMPLEMENTATION DETAILS

The source code of all experiments is available online.[4] 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[5] was used to log hyper-parameters and output metrics from runs.

A.1 DATASETS AND NETWORK 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 $64(input\_size + 1) + 65D$ trainable parameters. Note that D is the size of 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 configuration space hypothesis using 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 the 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 & 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.[6] 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:

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

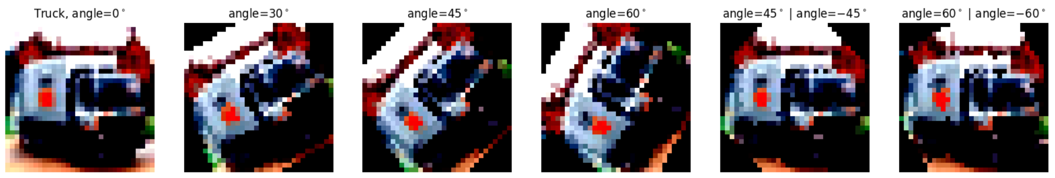
ShallowCNNs on SVHN. SVHN (Netzer et al., 2011) digit classification dataset contains 73'257 digits for training, 26'032 digits for testing, and 531'131 additionally less difficult digits for assisting

[4]Source code: `https://github.com/osaukh/subspace-configurable-networks`
[5]`https://wandb.ai`
[6]`https://github.com/zalandoresearch/fashion-mnist`

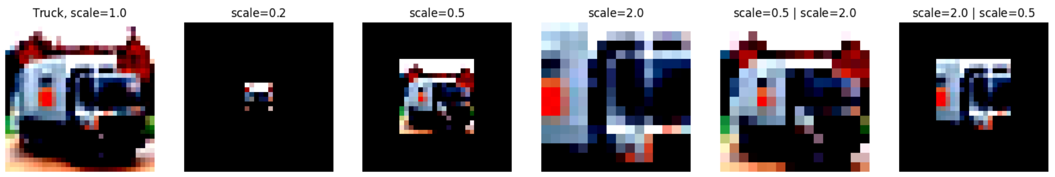
training. No additional images are used. The dataset is available for non-commercial use.[7] Shallow convolutions (ShallowCNNs) were introduced by Neyshabur (2020). We scale the architecture along the width w and depth d dimensions. The number of parameters scales as follows:

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

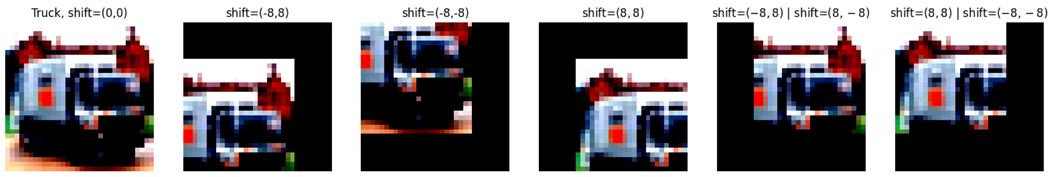
LeNet5 on ModelNet10. ModelNet10 (Wu et al., 2015) is a subset of ModelNet40 comprising a clean collection of 4,899 pre-aligned shapes of 3D CAD models for objects of 10 categories. We use this dataset to evaluate SCN performance on images of 3D rotated objects. We first rotate an object in the 3D space, and 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 α are input to the trained hypernetwork to obtain the parameters in the β-space to construct an optimal inference network. We use LeNet-5 (Lecun et al., 1998) as inference network architecture with 138'562 parameters.



(a) **2D rotation transformation parameterized by an angle ϕ in the range (0–2π)**. 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).

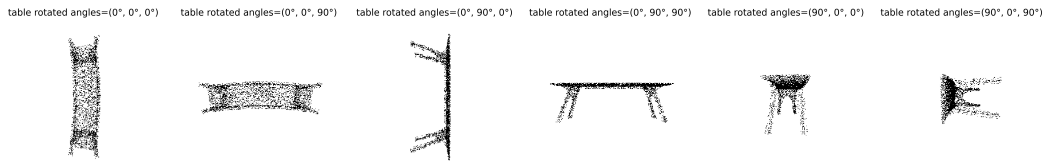


table rotated angles=(0°, 0°, 0°)
table rotated angles=(0°, 0°, 90°)
table rotated angles=(0°, 90°, 0°)
table rotated angles=(0°, 90°, 90°)
table rotated angles=(90°, 0°, 0°)
table rotated angles=(90°, 0°, 90°)

(d) **3D rotation transform**. We rotate an object in 3D along XY, YZ, and XZ planes using 3 angles (ϕ_1, ϕ_2, ϕ_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 4: **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.

[7] http://ufldl.stanford.edu/housenumbers/

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.[8]

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.[9]

A.2 Training hyper-parameters

Table 1 summarizes the set of hyper-parameters used to train different networks throught this work.

A.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 4 additionally illustrates examples of transformations applied to a sample input, showcasing various non-obvious effects that result in a decrease of input quality. Consequently, this decrease in quality adversely affects the accuracy of a trained classifier.

2D rotation. The 2D rotation transformation is parameterized by a single angle ϕ in the range 0–2π. 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.

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 β-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 the 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 (ϕ_1, ϕ_2, ϕ_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 4.

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 present in an image. Lastly, *sharpness* controls the level of detail clarity in an image. We vary the continuously changing α parameter between 0.2 and 2.0 for all considered color transformations. Augmenting the list of standard color transformations, we also experiment with image-based classification tasks under varying natural light conditions on Arduino Nano 33 BLE Sense. An example is shown in our arxiv paper (Saukh et al., 2023).

[8] `https://www.cs.toronto.edu/~kriz/cifar.html`
[9] `https://pytorch.org/audio/stable/datasets.html`

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.

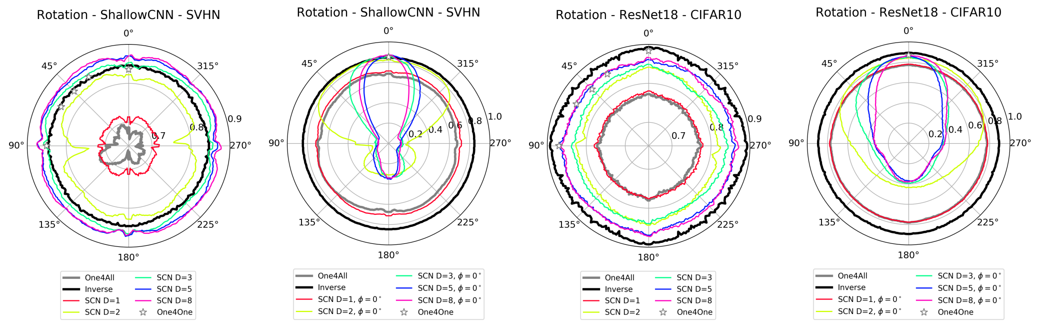


Figure 5: **SCN test accuracy trained on 2D rotations**. **From left to right:** A pair of plots for ShallowCNN–SVHN and ResNet18–CIFAR10. The models in each pair show SCN's performance for changing input $\alpha = (\cos(\phi), \sin(\phi))$ and for the fixed α with $\phi = 0°$.

B Configuration subspaces and SCNs

B.1 SCN performance

SCN performance on geometric transformations. Figure 5 complements Figure 2 in the main paper and presents the performance of SCN for 2D rotation on ShallowCNN–SVHN and ResNet18–CIFAR10. We observe the high efficiency of SCNs compared to the baselines even for very small D. A close inspection of models trained for a fixed input degree ($\phi = 0°$) shows increasingly higher specialization of the trained models to the respective parameter setting.

SCN performance on color transformations. Color transformations are simple. SCNs achieve high performance already for very small D (see Figure 6). 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.

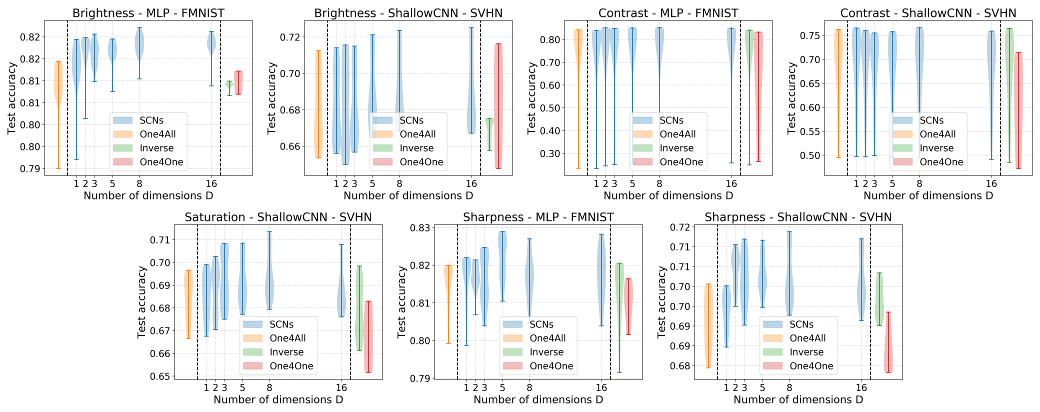


Figure 6: **Summary of SCN performance on color transformations:** brightness, contrast, saturation and sharpness. We present the results for MLP-FMNIST and ShallowCNN-SVHN architecture-dataset pairs. All transformations are simple. SCNs match the baselines for very low D. Note that saturation has no 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 the audio signal transformations. Figure 7 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 as input rather than a spectrogram.

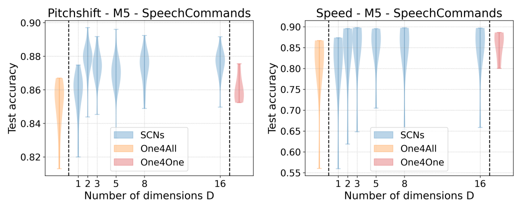


Figure 7: **Summary of SCN performance on audio signal transformations:** pitch shift and speed using M5 as inference architecture. SCNs match the performance of baselines already for small D.

B.2 Configuration β-space visualization

The β-space learned by the configuration network h for different transformations, datasets, and inference network architectures is shown in Figure 8. For 2D rotation, the transformation parameters $\alpha = (\cos(\phi), \sin(\phi))$ are drawn from a circle and result in all β_i being continuous curves arranged around the cylinder in our α-β 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 β_i is high for a certain contiguous range of αs and low outside of this range. For small D, the regions of high βs 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.

We claim that the subspace of optimized configurations for data transformations parameterized by α is *nicely structured*: (i) We achieve good accuracies even for a linear subspace of low degrees 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 our arxiv paper (Saukh et al., 2023). 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.

C 3D rotation transformation

Figure 9 shows all views of the β-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 9 shows the whole β-space for 3D rotation presented as a function of all pairwise combinations of ϕ_i. In Figure 9 middle and bottom, βs show a stable trend along the ϕ_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 9 (top), β-space has cosine-like trend along both ϕ_1 and ϕ_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 β-surfaces are not flat or degenerated. By observing the similarities and changing trends in the learned β-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 (ϕ_1, ϕ_2, ϕ_3). We provide a link[10] to an interactive website visualizing the β-space of sample SCNs, including those trained for 3D rotation.

[10]https://subspace-configurable-networks.pages.dev/

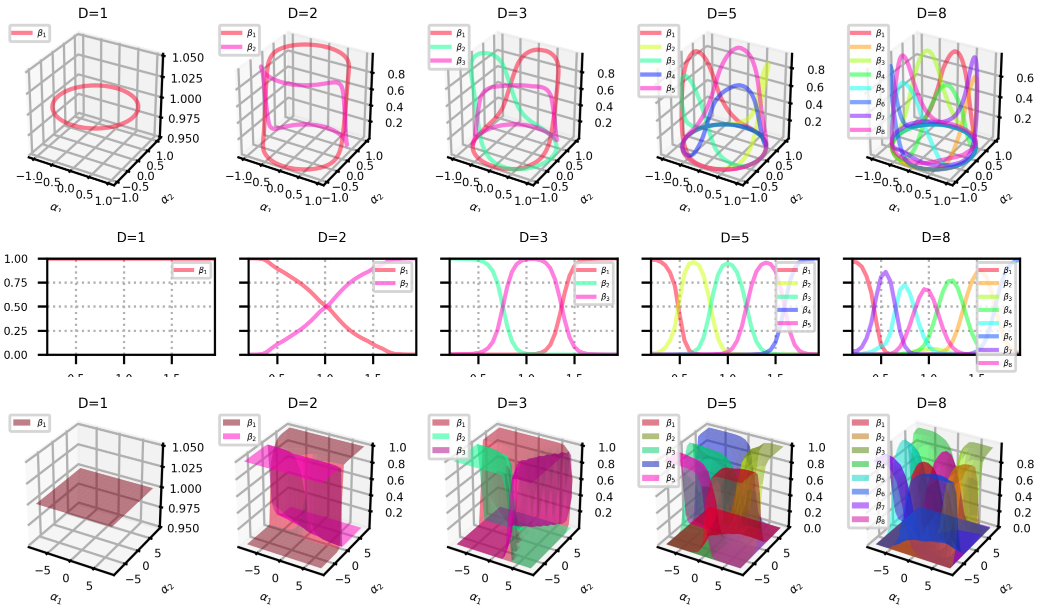


Figure 8: **A typical view of the β-space for 2D rotation, scaling, and translation,** $D = 1..8$. The β-space is nicely shaped, with each β being responsible for a specific range of inputs with smooth transitions. **Top:** SCNs for 2D rotation on ResNet18–CIFAR10. Transformation parameters are a vector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

D SEARCH ALGORITHM IN THE α-SPACE

This section provides details on the performance of the search algorithm which estimates α from a stream of input data. As mentioned in the main paper, we can leverage the fact that the correct input parameters α should produce a confident low-entropy classification result (Wortsman et al., 2020; Hendrycks & Gimpel, 2016). Therefore, our search algorithm estimates α 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 subspace. We use the basinhopping[11] method to find the solution (with default parameters, $iter = 100$, $T = 0.1$, method=BFGS).

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

```
from scipy import optimize

# function to be minimized by the basin hopping algorithm
def f(z, *args):
   alpha = transform_angle(((1+z)*180)%360-180)
   X = args[0]
   logits = model(Tensor(X), hyper_x=Tensor(alpha))
   b = (F.softmax(logits, dim=1)) * (-1 * F.log_softmax(logits, dim=1))
       # entropy
   return b.sum().numpy()

# given a batch of images find the rotation angle alpha using basin
    hopping algorithm
```

[11]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html

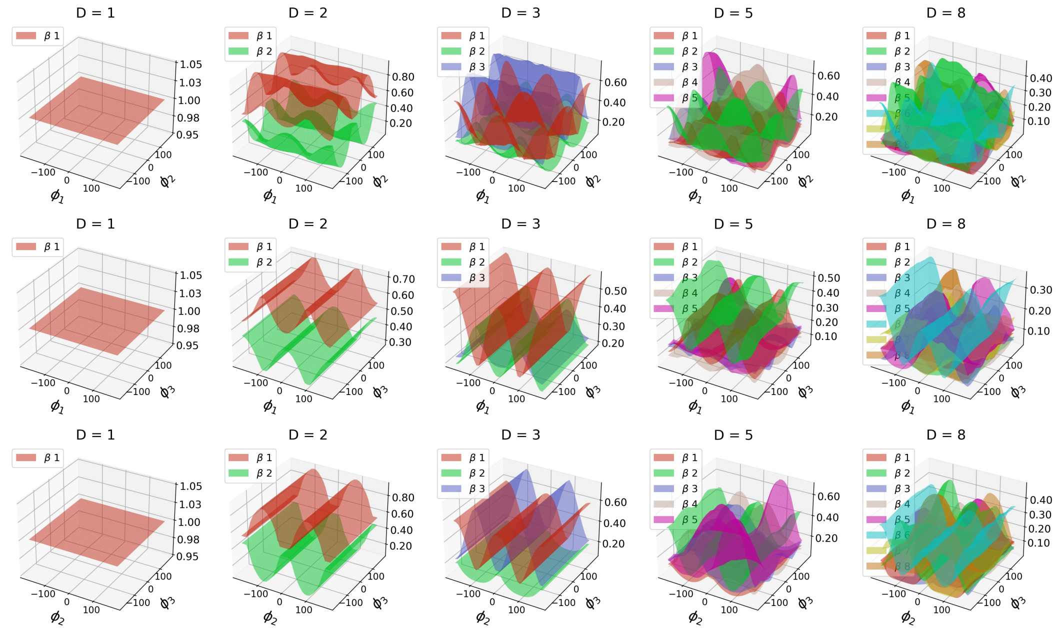


Figure 9: $\alpha - \beta$ **space of SCNs trained for 3D rotation on ModelNet10 with LeNet5 inference architecture for** $D = 1..8$. Transformation parameters α result from applying $\cos(\cdot)$ and $\sin(\cdot)$ functions to the vector of rotation angles (ϕ_1, ϕ_2, ϕ_3), with each ϕ_i in the range $(-\pi, \pi)$. **Top:** Subspace of SCNs when changing (ϕ_1, ϕ_2), and fixing $\phi_3 = -\pi$. **Middle:** Subspace of SCNs when changing (ϕ_1, ϕ_3), and fixing $\phi_2 = -\pi$. **Bottom:** Subspace of SCNs when changing (ϕ_2, ϕ_3), and fixing $\phi_1 = -\pi$.

```
def findalpha(X):
   mkwargs = {"method": "BFGS", "args":X}
   res = optimize.basinhopping(f, 0.0, minimizer_kwargs=mkwargs,
       niter=100, T=0.1)
   alpha = ((1+res.x[0])*180)%360-180
   return alpha

# test search algorithm performance on a test set
result = 0.0
for (X, y) in test_loader:
   angle = random.uniform(-180, 180)
   X = TF.rotate(X, angle)

   alpha = findalpha(X)

   # compute model prediction with the estimated alpha
   logits = model(X, hyper_x=transform_angle(alpha))
   # y is the true label --> calculate accuracy
   correct = (logits.argmax(1) == y).type(torch.float).sum().item() /
       batch_size
   result += correct

result /= len(test_loader.dataset) / batch_size
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 α and maximise it for a randomly sampled α. The train function is sketched in the listing below.

```
loss_fn = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
cos = nn.CosineSimilarity(dim=0, eps=1e-6)

def train(dataloader, model, loss_fn, optimizer):
   for (X, y) in dataloader:
      X, y = X.to(device), y.to(device)
      angle = random.uniform(0, 360)
      X = TF.rotate(X, angle)

      # make prediction and compute the loss
      pred = model(X, hyper_x=transform_angle(angle).to(device))
      loss = loss_fn(pred, y)

      # regularize (cosine similarity squared) in the beta space
      beta1 = model.hyper_stack(transform_angle(angle).to(device))
      angle2 = random.uniform(0, 360)
      beta2 = model.hyper_stack(transform_angle(angle2).to(device))
      loss += pow(cos(beta1, beta2),2)

      # minimize entropy for the correct degree
      b1 = (F.softmax(pred, dim=1)) * (-1 * F.log_softmax(pred, dim=1))
      loss += 0.01*b1.sum()

      # maximize entropy for a wrong / random degree
      logits = model(X, hyper_x=transform_angle(angle2).to(device))
      b2 = (F.softmax(logits, dim=1)) * (-1 * F.log_softmax(logits,
          dim=1))
      loss -= 0.01*b2.sum()

      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
```

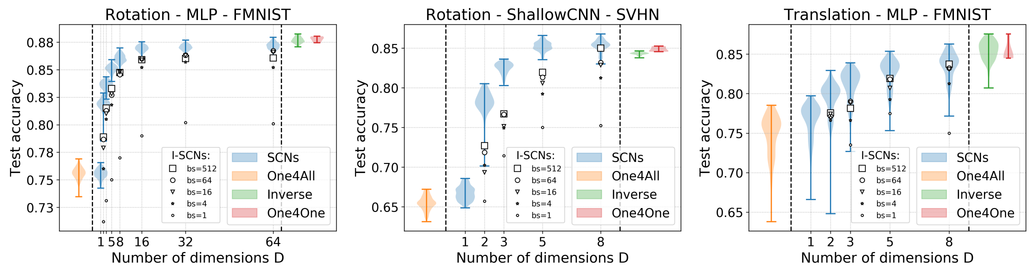


Figure 10: **Performance of the search algorithm in the α-space.** We enhance SCN evaluation plots in Figure 1 right and Figure 3 left and middle right with the performance of the presented search algorithm in the α-space. For higher batch sizes ($\geq$4) the search algorithms performs close to the respective SCNs with known α.

In Figure 10, we enhance three plots from Figure 1 and Figure 3 to show the performance of the search in the α-space. Note that the proposed input-based search algorithm allows constructing invariant SCNs, which we refer to as I-SCNs. We compare to the test accuracy achieved by the respective SCNs with known and correct input α to I-SCNs. The search algorithm operates on batches (`bs` = batch size). Batch size $\geq$4 allows for an accurate estimation of α from the input data and yields high I-SCN performance.

Note that the basin hopping algorithm is computationally expensive. For the 2D rotation transformation on the 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 α-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 α-search shows aims to show that the problem of estimating α and building I-SCNs is solvable in principle.

Table 2: **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

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; Su et al., 2015; Savva et al., 2016). TI-Pooling (called Transformation-Invariant 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 ϕ 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. Tab. 2 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.