Data-driven Regularization via Racecar Training for Generalizing Neural Networks

You Xie  Nils Thuerey
you.xie@tum.de  nils.thuerey@tum.de
Technical University of Munich
https://github.com/tum-pbs/racecar

Abstract

We propose a novel training approach for improving the generalization in neural networks. We show that in contrast to regular constraints for orthogonality, our approach represents a data-dependent orthogonality constraint, and is closely related to singular value decompositions of the weight matrices. We also show how our formulation is easy to realize in practical network architectures via a reverse pass, which aims for reconstructing the full sequence of internal states of the network. Despite being a surprisingly simple change, we demonstrate that this forward-backward training approach, which we refer to as racecar training, leads to significantly more generic features being extracted from a given data set. Networks trained with our approach show more balanced mutual information between input and output throughout all layers, yield improved explainability, and exhibit improved performance for a variety of tasks and task transfers.

1 Introduction

Despite their vast success, training neural networks that generalize well to a wide range of previously unseen tasks remains a fundamental challenge [35, 26, 2, 13]. A variety of techniques have been proposed over the years, ranging from well-established ones like dropout [44], and weight decay [33, 20, 52, 30], to techniques for orthogonalization [37, 25, 2]. While the former ones primarily focus on avoiding over-fitting rather than generality, the latter is closer to the goals of our paper: generalization is often seen in terms of extracting the most basic set of feature templates [45], similar to orthogonal vectors, that span the distribution of a data set under consideration. Here,

![Figure 1: Racecar training yields improved results for a wide range of applications: a): For difficult shape classification tasks, our approach outperforms regular training (StdTS) as well as state-of-the-art regularizers (OrtTS): the RRTS model instead classifies the airplane shape with high confidence. b): Our approach establishes mutual information between input and output distributions, retrieving intuitive and explainable dimensions. c): A racecar model yields an improved reconstruction for stylization tasks. d): For CIFAR 10 classification with a Resnet110, RR_{C10} yields substantial practical improvements over the state-of-the-art Ort_{C10}.]

Preprint. Under review.
especially the dominant features \cite{38} that occur most often in a data set are those that have the highest potential to be applicable in new contexts and yield generalizing capabilities. The challenge of extracting such features is two-fold: finding the dominant features in a large and diverse data-set, and representing them given limited computational resources. Our formulation incurs only a very moderate computational cost, is very easy to integrate and widely applicable, while at the same time outperforming existing methods for extracting dominant features.

Our formulation draws inspiration from the human trait to disassemble and re-assemble objects. These deconstruction tasks serve a wide range of purposes, from a hobby for motorists, to an essential source of learning and exploration for children \cite{19}. We show that dominant, and general features can be learned with a simple, yet powerful and general modification of neural network training in line with these re-assembly tasks: in addition to a regular forward pass, we add a reverse pass that is constrained to reconstruct all in-between results of the forward pass as well as the input. This palindromic structure yields substantial improvements for generalization of the learned representations in a wide range of architectures and significantly improves performance. Our results indicate that the reversible nature of the proposed training setup, which we will subsequently refer to via the palindrome "racecar" due to the backward-forward connectivity of the resulting network, encourages the learning of generic and reusable features that benefit a wide range of objectives.

We will evaluate the generality of the extracted dominant features in the context of transfer learning. For a regular, i.e., a non-transfer task, the goal usually is to train a network that gives the optimal performance for one specific goal. This has been demonstrated in numerous success stories \cite{12} \cite{29} \cite{17} \cite{21}. In a regular training run, the network naturally exploits any observed correlations between input and output distribution. E.g., if the color of an object in any way correlates with its type, the training of a classifier should find and use this information. In recent years, even networks trained only for a very specific task were shown to be surprisingly suitable starting points when training models for different tasks \cite{54} \cite{18} \cite{8}. Often, the original network learned representations that were at least partially applicable to different data domains. An inherent difficulty in this setting is that typically no knowledge about the specifics of the new data and task domains is available at the training time of the source model. While it is common practice to target broad and difficult tasks hoping that this will result in features that are applicable in new domains, we instead leverage the data distribution of the inputs when training the source model.

The structure of our networks is inspired by invertible network architectures that have received significant attention in recent years \cite{16} \cite{24} \cite{56}. However, instead of aiming for a bijective mapping that reproduces inputs, we strive for learning a general representation by constraining the network to represent an as-reversible-as-possible process for all intermediate layer activations. Thus, even for cases where a classifier can, e.g., rely on color for inference of an object type, the model is encouraged to learn a representation that can recover the input. Hence, not only the color of the input should be retrieved, but also, e.g., its shape. In contrast to most structures for invertible networks, racecar training does not impose architectural restrictions. We demonstrate the benefits of racecar training for a variety of architectures, from fully connected layers to convolutional neural networks (CNNs), over networks with and without batch normalization, to GAN architectures.

## 2 Related Work

Several prior methods employed "hard orthogonal constraints" to improve weight orthogonality via singular value decomposition (SVD) at training time \cite{23} \cite{25} \cite{37}. Bansal et al. \cite{2} additionally investigated efficient formulations of the orthogonality constraints. In practice, these constraints are difficult to satisfy, and correspondingly only weakly imposed. Besides, these methods focus on improving performance for a known, given task. This means the training process only extracts features that the network considers useful for improving the performance of the current task, not necessarily improving generalization or transfer performance \cite{49}. While our racecar training shares similarities with SVD-based constraints, it can be realized with a very efficient $L^2$-based formulation, and takes the full input distribution into account, leading to improved generalization.

Recovering all input information from hidden representations of a network is generally very difficult \cite{9} \cite{34}, due to the loss of information throughout the layer transformations. In this context, \cite{48} proposed the information bottleneck principle, which states that for an optimal representation, information unrelated to the current task is omitted. This highlights the common specialization of conventional training approaches. Reversed network architectures were proposed in previous work \cite{1} \cite{24} \cite{15}, but mainly focus on how to make a network fully invertible via augmenting the network
with special structures. As a consequence, the path from input to output is different from the reverse path that translates output to input. Besides, the augmented structures of these approaches can be challenging to apply to general network architectures. In contrast, our approach fully preserves an existing architecture for the backward path, and it does not require any operations that were not part of the source network. As such, it can easily be applied in new settings, e.g., adversarial training [17]. While methods using reverse connections were previously proposed [56, 46], these modules primarily focus on transferring information between layers for a given task, and on auto-encoder structures for domain adaptation, respectively.

Transfer learning with deep neural networks has been very successful for a variety of challenging tasks, such as image classification [11, 31, 60], multi-language text classification [59, 39, 58], and medical imaging problems [40, 54] proposed an approach to obtain task relationship graphs for different tasks. In this context, a central challenge is to set up the training process such that it leads to learning generic features from the data set, which are useful for both source and related tasks rather than specific features for the source task.

3 Method

Regularization via orthogonality has become a popular tool for training deep CNNs and was shown to improve performance [23, 3, 2]. The corresponding constraints can be formulated as:

\[ \mathcal{L}_{\text{ort}} = \sum_{m=1}^{n} \left| \left| M_{m}^{T} M_{m} - I \right| \right|_{F}^{2}, \]  

(1)

i.e., enforcing the transpose of the weight matrix \( M_m \in \mathbb{R}^{r_m \times s_m} \) for all layers \( m \) to yield its inverse when being multiplied with the original matrix. \( I \) denotes the identity matrix with \( I = (e_1^m, \ldots, e_{m}^m) \), \( e_j^m \) denoting the \( j \)-th column unit vector. Minimizing Eq. (1), i.e. \( M_{m}^{T} M_{m} - I = 0 \) is mathematically equivalent to:

\[ M_{m}^{T} M_{m} e_j^m - e_j^m = 0, \quad j = 1, 2, \ldots, s_{m}. \]  

(2)

with \( \text{rank}(M_{m}^{T} M_{m}) = s_{m} \), and \( e_j^m \) as eigenvectors of \( M_{m}^{T} M_{m} \) with eigenvalues of 1. This formulation highlights that Eq. (2) does not depend on the training data, and instead only targets the content of \( M_m \). Our approach reformulates the original orthogonality constraint in a data-driven manner: we take into account the set \( D_m \) of inputs for the current layer (either activation from a previous layer or the training data \( D_1 \)), and instead minimize

\[ \mathcal{L}_{\text{RR}} = \sum_{m=1}^{n} \left( \left| \left| M_{m}^{T} M_{m} d_{m} - d_{m} \right| \right|_{2}^{2} \right) = \sum_{m=1}^{n} \left( (M_{m}^{T} M_{m} - I) d_{m} \right)^{2}, \]  

(3)

where \( d_{m} \in D_m \subset \mathbb{R}^{s_{m}} \). This re-formulation of orthogonality allows for minimizing the loss by extracting the dominant features of the input data instead of only focusing on the content of \( M_m \).

We use \( q \) to denote the number of linearly independent entries in \( D_m \), i.e. its dimension, and \( t \) the size of the training data, i.e. \( |D_m| = t \), usually with \( q < t \). For every single datum \( d_{m}, i = 1, 2, \ldots, t \), Eq. (3) results in

\[ M_{m}^{T} M_{m} d_{m} - d_{m} = 0, \]  

(4)

and hence \( d_{m} \) are eigenvectors of \( M_{m}^{T} M_{m} \) with corresponding eigenvalues being 1. Thus, instead of the generic constraint \( M_{m}^{T} M_{m} = I \) that is completely agnostic to the data at hand, the proposed formulation of Eq. (4) is aware of the training data, which improves the generality of the learned representation, as we will demonstrate in detail below.

As by construction, \( \text{rank}(M_{m}) = r \leq \min(s_{m}, s_{m}^{\text{out}}) \), the SVD of \( M_m \) yields:

\[ M_m = U_m \Sigma_m V_m^{T}, \text{ with } \begin{cases} U_m = (u_1^m, u_2^m, \ldots, u_r^m, u_{r+1}^m, \ldots, u_{s_m}^m) & \in \mathbb{R}^{r_m \times s_m}, \\ V_m = (v_1^m, v_2^m, \ldots, v_r^m, v_{r+1}^m, \ldots, v_{s_m}^m) & \in \mathbb{R}^{s_m \times r_m}. \end{cases} \]  

(5)

with left and right singular vectors in \( U_m \) and \( V_m \), respectively, and \( \Sigma_m \) having square roots of the \( r \) eigenvalues of \( M_{m}^{T} M_{m} \) on its diagonal. \( u_k^m \) and \( v_k^m (k = 1, \ldots, r) \) are the eigenvectors of \( M_m M_m^{T} \) and \( M_m^{T} M_m \), respectively [50]. Here, especially the right singular vectors in \( V_m \) are important, as they determine which structures of the input are processed by the transformation \( M_m \). The original orthogonality constraint with Eq. (2) yields \( r \) unit vectors \( e_j^m \) as the eigenvectors of \( M_{m}^{T} M_{m} \). Hence, the influence of Eq. (2) on \( V_m \) is completely independent of training data and learning objectives.
Next, we show that $\mathcal{L}_{\text{RR}}$ facilitates learning dominant features from a given data set. For this, we consider an arbitrary basis to span the space of inputs $D_m$ for layer $m$. Let $B_m^r : (w_{m1}^1, ..., w_{mq}^r)$ denote a set of $q$ orthonormal basis vectors obtained via a Gram-Schmidt process, with $t \geq q \geq r$, and $D_m$ denoting the matrix of the vectors in $B_m$. As we show in more detail in the appendix, our constraint from Eq. (3) requires eigenvectors of $M^T M$ to be $w_{m}^r$, with $V_m$ containing $r$ orthogonal vectors $(v_{m1}, v_{m2}, ..., v_{mr})$ from $D_m$ and $(s_{m}^r - r)$ vectors from the null space of $M$.

We are especially interested in how $M_m$ changes w.r.t. input in terms of $D_m$, i.e., we express $\mathcal{L}_{\text{RR}}$ in terms of $D_m$. By construction, each input $d_{m}^i$ can be represented as a linear combination via a vector of coefficients $c_{m}^i$ that multiplies $D_m$ so that $d_{m}^i = D_m c_{m}^i$. Since $M_m d_m = U_m s_m v_m d_m$, the loss $\mathcal{L}_{\text{RR}}$ of layer $m$ can be rewritten as

$$\mathcal{L}_{\text{RR},m} = (M_m^T M_m d_m - d_m)^2 = (V_m \Sigma_m^T \Sigma_m V_m^T d_m - d_m)^2 \quad \text{for a successful minimization}, \quad V_m \Sigma_m \Sigma_m^T V_m^T D_m c_m - D_m c_m)^2,$$

where we can assume that the coefficient vector $c_{m}$ is accumulated over the training data set size $t$ via $c_{m} = \sum_{i=1}^{t} c_{m}^i$, since eventually every single datum in $D_m$ will contribute to $\mathcal{L}_{\text{RR},m}$. The central component of Eq. (6) is $V_m^T D_m$. For a successful minimization, $V_m$ needs to retain those $w_{m}^r$ with the largest $c_{m}$ coefficients. As $V_m$ is typically severely limited in terms of its representational capabilities by the number of adjustable weights in a network, it needs to focus on the most important eigenvectors in terms of $c_{m}$ in order to establish a small distance to $D_m c_m$. Thus, features that appear multiple times in the input data with a corresponding factor in $c_{m}$ will more strongly contribute to minimizing $\mathcal{L}_{\text{RR},m}$.

To summarize, $V_m$ is driven towards containing $r$ orthogonal vectors $w_{m}^r$ that represent the most frequent features of the input data, i.e., the dominant features. Additionally, due to the column vectors of $V_m$, being mutually orthogonal, $M_m$ is encouraged to extract different features from the input. By the sake of being distinct and representative for the data set, these features have the potential to be useful for new inference tasks. The feature vectors embedded in $M_m$ can be extracted from the network weights in practical settings, as we will demonstrate below.

**Realization in Neural Networks** Calculating $M_m^T M_m$ is usually very expensive due to the dimensionality of $M_m$. Instead of building it explicitly, we constrain intermediate results to realize Eq. (3) when training. A regular training typically starts with a chosen network structure and trains the model weights for a given task via a suitable loss function. Our approach fully retains this setup and adds a second pass that reverses the initial structure while reusing all weights and biases. E.g., for a typical fully connected layer in the forward pass with $d_{m+1} = M_m d_m + b_m$, the reverse pass operation is given by $d_m = M_m^T (d_{m+1} - b_m)$, where $d_m$ denotes the reconstructed input.

Our goal with the reverse pass is to transpose all operations of the forward pass to obtain identical intermediate activations between the layers with matching dimensionality. We can then constrain the intermediate results of each layer of the forward pass to match the results of the backward pass, as illustrated in Fig. 12. Due to the symmetric structure of the two passes, we can use a simple $L^2$ difference to drive the network towards aligning the results:

$$\mathcal{L}_{\text{racecar}} = \sum_{m=1}^{M} \lambda_m \left\| d_m - d_m' \right\|_F^2.$$

Here $d_m$ denotes the input of layer $m$ in the forward pass and $d_m'$ the output of layer $m$ for the reverse pass. $\lambda_m$ denotes a scaling factor for the loss of layer $m$, which, however, is typically constant in our tests across all layers. Note that with our notation, $d_1$ and $d_1'$ refer to the input data, and the reconstructed input, respectively.

Next, we show how this setup realizes the regularization from Eq. (3). For clarity, we use a fully connected layer with bias. In a neural network with $n$ hidden layers, the forward process for a layer $m$ is given by:

$$d_{m+1} = M_m d_m + b_m.$$
with $d_1$ and $d_{n+1}$ denoting in- and output, respectively. For racecar training, we build a layer-wise reverse pass pass network with transposed operations and intermediate results $d_{m+1}$:

$$d_m' = M^T_m(d_{m+1} - b_m),$$

(9)

which yields $\|d_m - d_m'\|_F^2 = M^T_m M_m d_m - d_m$. When this difference is minimized via Eq. (7), we obtain activated intermediate content during the reverse pass that reconstructs the values computed in the forward pass, i.e. $d_{m+1}' = d_{m+1}$ holds. Replacing $d_{m+1}'$ with $d_{m+1}$ in Eq. (9), yields a full reverse pass from output to input, which we use most racecar training runs below. This version is preferable if a unique path from output to input exists. For architectures where the path is not unique, e.g. in the presence of additive residual connections, we use the layer-wise formulation from Eq. (9).

The full formulation instead gives:

$$d_m' = M^T_m (d_{m+1} - b_m) = M^T_m (d_{m+1} - b_m) = M^T_m M_m d_m,$$

(10)

which is consistent with Eq. (5), and hence satisfies the original constraint $M^T_m M_m d_m - d_m = 0$.

Up to now, the discussion focused on simplified neural networks without activation functions or extensions such as batch normalization (BN). While we leave incorporating such extensions into the derivation for future work, our experiments consistently show that the inherent properties of racecar training remain valid: even with activations and BN, our approach successfully extracts dominant structures and yields improved generalization. In the appendix, we give details on how to ensure that the latent space content for forward and reverse pass is aligned such that differences can be minimized.

To summarize, we realize the loss formulation of Eq. (7) to minimize $\sum_{m=1}^n (M^T_m M_m - I) d_m)^2$ without explicitly having to construct $M^T_m M_m$. We will refer to networks trained with the added reverse structure and the additional loss terms as being trained with racecar training. We consider two variants for the reverse pass: a layer-wise racecar training Eq. (9) using the local datum $d_{m+1}$, and a full version via Eq. (10) which uses $d_{m+1}$.

**Experiments**

We will use the following naming scheme to differentiate variants in the experiments below: Std, RR, Ort for base models trained for a task $T$. Here, Std denotes a regular training run (in orange color in graphs below), while RR denotes models trained with our racecar training (in green). While we typically use all layers of a network in the racecar constraints, a special reduced variant that we compare to below only applies the constraint for the input data, i.e., $m=1$ in our notation. A network trained with this variant, denoted by RR$_1$, is effectively trained to only reconstruct the input. It contains no constraints for the inner activations of the network. Ort additionally denotes models trained with Spectral Restricted Isometry Property (SRIP) orthogonal regularization [2] (in blue). We verify that the column vectors of $V_m$ of models from racecar training contain the dominant features of the input with the help of two classification tests. Both cases employ a single hidden fully connected layer, i.e. $d_2 = M_1 d_1$. In the first MNIST test, the training data consists only of 2 different images. After training, we compute the SVD for $M_1$. Qualitatively, RR shows $v_m^1$ that have obvious similarity with the two inputs, while the vectors of Std and Ort contain no recognizable structures (Fig. 3(a)). To quantify this similarity, we compute an LPIPS distance [57] between $v_m^1$ and the training data (lower values being better). These measurements confirm that the singular values of the RR model with 0.282 ± 0.82 are significantly more similar to the input images that Std and Ort, with 0.411 ± 0.05 and 0.403 ± 0.031, respectively.

For the second classification test, we employ a training data set that is constructed from two dominant classes (a peak in the top left, and bottom right quadrant, respectively), augmented with noise in the form of random scribbles. Based on the analysis above, we expect the racecar training to extract the two dominant peaks during training. Our results confirm this: the peaks are clearly visible for racecar training, an example is shown in Fig. 3(b), while the other models fail to extract structures that resemble the input. The LPIPS distances confirm the visual similarity with 0.217 ± 0.022 for RR, 0.500 ± 0.002 for Std and 0.495 ± 0.006 for Ort. Thus, our training approach yields a way for humans to inspect the structures learned by a neural network.
Figure 4: MI plane comparisons for different models. a) Visual overview of the content. b) Plane of task A. Points on each line correspond to layers of one type of model. All points of RRₐₕ are located in the center of the graph, while Stdₐ and Ortₐ, exhibit large $I(D_m; Y)$, i.e., specialize on the output. c,d): After fine-tuning for A/B. The last layer $D_7$ of RRₐₕ builds the strongest relationship with $Y$. The results above experimentally confirm our derivation of the racecar loss, and its ability to extract dominant and generalizing structures from the training data. We have focused on relatively simple setups here, as this allows us to quantify similarity via the training data. Our results below indicate that racecar training similarly extracts dominant structures from the activations for the deeper layers of a network. However, due to the lack of ground truth data, it is not easy to evaluate these structures. Thus, we turn to measurements in terms of mutual information in the next section.

The constraints of Eq. 7 intentionally only minimize differences in an averaged manner with an $L^2$ norm, as we don’t strive for a perfectly bijective mapping between in- and output domains. Instead, racecar training aims at encouraging the network to extract dominant features that preserve as much information from the input data set as possible, given the representative capabilities of the chosen architecture. Hence, while regular training allows a model to specialize its extracted features to represent its dominant features.

4 Evaluation in Terms of Mutual Information

As our approach hinges on the introduction of the reverse pass, we will show that it succeeds in terms of establishing mutual information (MI) between the input and the constrained intermediates inside a network. More formally, MI $I(X; Y)$ of random variables $X$ and $Y$ measures how different the joint distribution of $X$ and $Y$ is w.r.t. the product of their marginal distributions, i.e., the Kullback-Leibler divergence $I(X; Y) = D_{KL}[P_{X,Y}]||P_X P_Y]$. [48] proposed MI plane to analyze trained models, which show the MI between the input $X$ and activations of a layer $D_m$, i.e., $I(X; D_m)$ and $I(D_m; Y)$, i.e., MI of layer $D_m$ with output $Y$. These two quantities indicate how much information about the input and output distributions are retained at each layer, and we use them to show to which extent racecar training succeeds at incorporating information about the inputs throughout training.

The following tests employ networks with six fully connected layers with the objective to learn the mapping from 12 binary inputs to 2 binary output digits [42], with results accumulated over five runs. We compare the versions Stdₐₕ, Ortₐ, and a variant of the latter: RRₐ¹, i.e. a version where only the input $d₁$ is constrained to be reconstructed. While Fig. 17(a), visually summarizes the content of the MI planes, the graph in (b) highlights that racecar training correlates input and output distributions across all layers: the cluster of green points in the center of the graph shows that all layers contain balanced MI between in- as well as output and the activations of each layer. RRₐ¹ fares slightly worse, while Stdₐ and Ortₐ almost exclusively focus on the output with $I(D_m; Y)$ being close to one. Once we continue fine-tuning these models without regularization, the MI naturally shifts towards the output, as shown in Fig. 17(c). RRₐₕ outperforms others in terms of final performance. Likewise, RRₐₕ performs best for a transfer task B with switched output digits, as shown in graph (d). The final performance for both tasks across all runs is summarized in Fig. 5. It
Transfer-learning Benchmarks  We first evaluate our approach with two state-of-the-art transfer learning benchmark data sets. The first one uses the texture-shape data set from [15], which contains challenging images of various shapes combined with patterns and textures to be classified. The results below are given for 10 runs each. For the stylized data shown in Fig. 7(a), StdTS yields a performance of 44.2%, and OrtTS improves the performance to 47.0%, while RRTS yields a performance of 54.7% (see Fig. 7b). Thus, the accuracy of RRTS is 23.76% higher than StdTS, and 16.38% higher than OrtTS. To assess generality, we also apply the models to new data without retraining, i.e., an edge and a filled data set, also shown in Fig. 7(a). For the edge data set, RRTS outperforms both StdTS and OrtTS by 50% and 16.75%, respectively. It is worth pointing out that the additional constraints of our racecar training lead to increased requirements for memory and additional computations, e.g., 41.86% more time per epoch than regular training for the texture-shape test. On the other hand, it allows us to train smaller models: we can reduce the weight count by 32% for the texture-shape case while still being on-par with OrtTS in terms of classification performance.

As a second test case, we use a CIFAR-based task transfer [41] that measures how well models trained on the original CIFAR 10, generalize to a new data set (CIFAR 10.1) collected according to the same principles as the original one. Here we use a Resnet110 with 110 layers and 1.7 million parameters. In terms of accuracy across 5 runs, OrtC10 outperforms StdC10 by 0.39%, while RR_C10 outperforms OrtC10 by another 0.28% in terms of absolute test accuracy (Fig. 24). This increase for racecar training matches the gains reported for orthogonality in previous work [2], and thus shows that our approach yields substantial practical improvements over the latter. It is especially interesting how well performance for CIFAR 10 translates into transfer performance for CIFAR 10.1. Here, RR_C10 still outperforms OrtC10 and StdC10 by 0.22% and 0.95%, respectively. Hence, the models...
from racecar training very successfully translate gains in performance from the original task to the new one, which indicates that the models have successfully learned a set of more general features. To summarize, both benchmark cases confirm that racecar training benefits generalization. As racecar training consistently outperforms orthogonalization, we focus on comparisons with regular training in the following.

**Generative Adversarial Models** In this section, we employ racecar training in the context of generative models for transferring from synthetic to real-world data from the ScalarFlow data set [12]. As super-resolution task A, we first use a fully-convolutional generator network, adversarially trained with a discriminator network on the synthetic flow data. Fig. 6 demonstrates that the racecar network structure works in conjunction with the GAN training. As shown in the bottom row, the trained generator succeeds in recovering the input via the reverse pass without modifications. A regular model Std, only yields a black image in this case.

We now mirror the generator model from the previous task to create an autoencoder structure that we apply to two different data sets: the synthetic smoke data used for the GAN training (task B1), and a real-world RGB data set of smoke clouds (task B2). Thus both variants represent transfer tasks, the second one being more difficult due to the changed data distribution. The resulting losses, summarized in Fig. 9, show that racecar training performs best for both autoencoder tasks: the $L^2$ loss of RR$_{AB1}$ is 68.88% lower than Std$_{AB1}$ for B1, and 13.3% lower for task B2. The latter is especially encouraging, as it represents a transfer from training with fully synthetic images to real-world images.

**VGG19 Stylization** To provide a qualitative evaluation in a complex visual scenario, we turn to image stylization. We use VGG19 networks [43] with more than 142 million parameters [7], and train models with and without racecar training, i.e., RR and Std. Both networks are then used for stylization tasks [14]. As this task strongly depends on the features of the base network, it visualizes the differences of the learned structures.

Several results of such stylizations are shown in Fig. 11 and Fig. 10. While the former highlights improved reconstruction of the leaf structures of the flower, the latter shows the popular setup of transforming a horse into a zebra. In Fig. 10, the regular model has difficulties distinguishing the background and the horse, and inserts green patches into the body. In contrast, the VGG19 network with racecar training yields a significantly better output. While these images only provide qualitative information about the properties of racecar training due to the lack of a ground truth result, we have found across a wide range of tests that performing stylization with models from racecar training yields more intuitive and robust results. This is most likely caused by the improved representation these networks form of the input data distribution, which allows them to more clearly separate salient objects and visual styles.

### 6 Conclusions

We have proposed a novel training approach for improving neural network generalization by adding a constrained reverse pass. We have shown for a wide range of scenarios, from singular value decompositions, over mutual information, to transfer learning benchmarks, that racecar training yields networks with better generalizing capabilities. Our training approach is very general and imposes no requirements regarding network structure or training methods. As future work, we believe it will be very interesting to evaluate our approach in additional contexts, e.g., for temporal structures [22][6], and for training explainable and interpretable models [55][5][10].
Supplemental Material for *Data-driven Regularization via Racecar Training for Generalizing Neural Networks*

Below, we provide additional details regarding derivation, data sets, network architectures, and experiments mentioned in the main paper. We additionally show results that could not be included in the main document due to page restrictions: an MNIST classification case (Sec. C.1), natural image classification (Sec. C.2), and additional generation and stylization results in Sec. D.2 and Sec. D.4, respectively.

To ensure reproducibility, source code for all tests will be published, in addition to all data sets that are not readily available. The runtimes below were measured on a machine with Nvidia GeForce GTX 1080 Ti GPUs and an Intel Core i7-6850K CPU.

A Method

A.1 Racecar Loss and SVD

In this section we give a more detailed derivation of our racecar loss formulation and extends on the information given in Section 3 of the main paper. As explained there, the racecar loss aims for minimizing

\[
\mathcal{L}_{RR} = \sum_{m=1}^{n} (M_{m}^{T} M_{m} d_{m}^{i} - d_{m}^{i})^2,
\]

(11)

where \( M_{m} \in \mathbb{R}^{s_{\text{out}} m \times s_{\text{in}} m} \) denotes the weight matrix of layer \( m \), and data from the input data set \( D_{m} \) is denoted by \( d_{m}^{i} \subset \mathbb{R}^{s_{\text{in}} m}, i = 1, 2, ..., t \). Here \( t \) denotes the number of samples in the input data set. Minimizing Eq. (11) is mathematically equivalent to

\[
M_{m}^{T} M_{m} d_{m}^{i} - d_{m}^{i} = 0
\]

(12)

for all \( d_{m}^{i} \). Hence, perfectly fulfilling Eq. (11) would require all \( d_{m}^{i} \) to be eigenvectors of \( M_{m}^{T} M_{m} \) with corresponding eigenvalues being 1. As in Sec. 3 of the main paper, we make use of an auxiliary orthonormal basis \( \mathcal{B}_{m} : \{ w_{1}^{i}, ..., w_{q}^{i} \} \), for which \( q \) (with \( q \leq t \)) denotes the number of linearly independent entries in \( D_{m} \). While \( \mathcal{B}_{m} \) never has to be explicitly constructed for our method, it can, e.g., be obtained via Gram-Schmidt. The matrix consisting of the vectors in \( \mathcal{B}_{m} \) is denoted by \( D_{m} \).

![Figure 11: An example of the intuitive analysis of network weights enabled by racecar training: we show the singular vectors for a conventional model Std, a model trained with orthogonal constraints Ort, and a racecar version RR. The singular vectors from racecar training show clear structures that resemble the inputs, while the other variants contain no visible structures. The evaluation via LPIPS of \( d_{1} \) versus all \( v_{1} \) confirms this: it yields 0.316 for RR on average, which is significantly better than Ort with 0.504) and Std with 0.484. Details and further examples are given in Sec. [A.3] below.](image-url)
Since the $w^h_{m}$ ($h = 1, 2, \ldots q$) necessarily can be expressed as linear combinations of $d^i_{m}$, Eq. (11) similarly requires $w^h_{m}$ to be eigenvectors of $M^T_{m}M_{m}$ with corresponding eigenvalues being 1, i.e.:

$$M^T_{m}M_{m}w^h_{m} - w^h_{m} = 0 \quad (13)$$

We denote the vector of coefficients to express $d^i_{m}$ via $D_{m}$ with $c^i_{m}$, i.e. $d^i_{m} = D_{m}c^i_{m}$. Then Eq. (12) can be rewritten as:

$$M^T_{m}M_{m}D_{m}c^i_{m} - D_{m}c^i_{m} = 0 \quad (14)$$

Via an SVD of the matrix $M_{m}$ in Eq. (14) we obtain

$$M^T_{m}M_{m}D_{m}c_{m} - D_{m}c_{m}$$

$$= \sum_{h=1}^{q} M^T_{m}M_{m}w^h_{m}c_{m_{h}} - w^h_{m}c_{m_{h}} \quad (15)$$

$$= \sum_{h=1}^{q} V_{m}\Sigma^T_{m}\Sigma_{m}V^T_{m}w^h_{m}c_{m_{h}} - w^h_{m}c_{m_{h}}$$

where the coefficient vector $c_{m}$ is accumulated over the training data set size $t$ via $c_{m} = \sum_{i=1}^{t} c^i_{m}$. Here we assume that over the course of a typical training run eventually every single datum in $D_{m}$ will contribute to $L_{RR,m}$. This form of the loss highlights that minimizing $L_{RR}$ requires an alignment of $V_{m}\Sigma^T_{m}\Sigma_{m}V^T_{m}w^h_{m}c_{m_{h}}$ and $w^h_{m}c_{m_{h}}$. By construction, $\Sigma_{m}$ contains the square roots of the eigenvalues of $M^T_{m}M_{m}$ as its diagonal entries. The matrix has rank $r = \text{rank}(M^T_{m}M_{m})$, and since all eigenvalues are required to be 1 by Eq. (13), the multiplication with $\Sigma_{m}$ in Eq. (15) effectively performs a selection of $r$ column vectors from $V_{m}$. Hence, we can focus on the interaction between the basis vectors $w_{m}$ and the $r$ active column vectors of $V_{m}$:

$$V_{m}\Sigma^T_{m}\Sigma_{m}V^T_{m}w^h_{m}c_{m_{h}} - w^h_{m}c_{m_{h}}$$

$$= c_{m_{h}}(V_{m}\Sigma^T_{m}\Sigma_{m}V^T_{m}w^h_{m} - w^h_{m})$$

$$= c_{m_{h}}(\sum_{f=1}^{r}(v^f_{m})^T w^h_{m}v^f_{m} - w^h_{m}). \quad (16)$$

As $V_{m}$ is obtained via an SVD it contains $r$ orthogonal eigenvectors of $M^T_{m}M_{m}$. Eq. (13) requires $w^h_{m_{1}}, \ldots, w^h_{m_{r}}$ to be eigenvectors of $M^T_{m}M_{m}$, but since typically the dimension of the input data set is much larger than the dimension of the weight matrix, i.e. $r \leq q$, in practice only $r$ vectors from $B_{m}$ can fulfill Eq. (13). This means the vectors $v^1_{m}, \ldots, v^r_{m}$ in $V_{m}$ are a subset of the orthonormal basis vectors $B_{m} : (w^h_{m_{1}}, \ldots, w^h_{m_{r}})$ with $(w^h_{m})^2 = 1$. Then for any $w^h_{m}$ we have

$$\begin{cases} (v^f_{m})^T w^h_{m} = 1, & \text{if } v^f_{m} = w^h_{m} \\ (v^f_{m})^T w^h_{m} = 0, & \text{otherwise.} \end{cases} \quad (17)$$

Thus if $V_{m}$ contains $w^h_{m}$, we have

$$\sum_{f=1}^{r}(v^f_{m})^T w^h_{m}v^f_{m} = w^h_{m}, \quad (18)$$

and we trivially fulfill the constraint

$$c_{m_{h}}(\sum_{f=1}^{r}(v^f_{m})^T w^h_{m}v^f_{m} - w^h_{m}) = 0. \quad (19)$$

However, due to $r$ being smaller than $q$ in practice, $V_{m}$ typically can not include all vectors from $B_{m}$. Thus, if $V_{m}$ does not contain $w^h_{m}$, we have $(v^f_{m})^T w^h_{m} = 0$ for every vector $v^f_{m}$ in $V_{m}$, which means

$$\sum_{f=1}^{r}(v^f_{m})^T w^h_{m}v^f_{m} = 0. \quad (20)$$
As a consequence, the constraint Eq. (12) is only partially fulfilled:

\[ c_m h \left( \sum_{f=1}^{r} (v_f^m)^T w^h v_f^m - w^h \right) = -c_m h w^h. \]  

(21)

As the \( w^h \) have unit length, the factors \( c_m \) determine the contribution of a datum to the overall loss. A feature \( w^h \) that appears multiple times in the input data will have a correspondingly larger factor in \( c_m \) and hence will more strongly contribute to \( L_{RR} \). The \( L^2 \) formulation of Eq. (11) leads to the largest contributors being minimized most strongly, and hence the repeating features of the data, i.e., dominant features, need to be represented in \( V_m \) to minimize the racecar loss.

In summary, to minimize \( L_{RR} \), \( V_m \) is driven towards containing \( r \) orthogonal vectors \( w^h \) which represent the most frequent features of the input data, i.e., the dominant features. It is worth emphasizing that above \( B_m \) is only an auxiliary basis, i.e., the derivation does not depend on any particular choice of \( B_m \).

A.2 Examples of Network Architectures with Racecar Training

To specify NN architectures, we use the following notation: \( C(k, l, q) \) and \( D(k, l, q) \) denote convolutional and deconvolutional operations, respectively, while fully connected layers are denoted with \( F \), where \( k, l, q \) denote kernel size, output channels and stride size, respectively. The bias of a CNN layer is denoted with \( b \). \( I/O(z) \) denote \( input/output \), their dimensionality is given by \( z \). \( I_r \) denotes the input of the reverse pass network. \( tanh, relu, lrelu \) denote hyperbolic tangent, ReLU, and leaky ReLU activation functions (AF), where we typically use a leaky tangent of 0.2 for the negative half-space. \( UP, MP \) and \( BN \) denote \( 2 \times \) nearest-neighbor up-sampling, max pooling with \( 2 \times 2 \) filters and stride 2, and batch normalization, respectively.

Below we provide additional examples how to realize the racecar loss \( L_{racecar} \) in a neural network architecture. As explained in the main document, the constraint Eq. (11) is formulated via

\[ L_{racecar} = \sum_{m=1}^{n} \lambda_m \left\| d_m - d'_m \right\|_F^2. \]  

(22)

with \( d_m \), and \( \lambda_m \) denoting the vector of activated intermediate data in layer \( m \) from the forward pass, and a scaling factor, respectively. \( d_m \) denotes the activations of layer \( m \) from the reverse pass. E.g., let \( L_m() \) denote the operations of a layer \( m \) in the forward pass, and \( L'_m() \) the corresponding operations for the reverse pass. Then \( d_{m+1} = L_m(d_m) \), and \( d_m = L'_m(d'_{m+1}) \).

When Eq. (22) is minimized, we obtain activated intermediate content during the reverse pass that reconstructs the values computed in the forward pass, i.e. \( d'_{m+1} = d_{m+1} \) holds. Then \( d_m \) can be reconstructed from the incoming activations from the reverse pass, i.e., \( d_{m+1}', \) or from the output of layer \( m \), i.e., \( d_{m+1} \). Using \( d_{m+1}' \) results in a global coupling of input and output throughout all layers, i.e., the full racecar loss variant. On the other hand, \( d_{m+1} \) yields a variant that ensures local reversibility of each layer, and yields a very similar performance, as we will demonstrate below. We employ this layer-wise racecar loss for networks without a unique, i.e., bijective, connection between two layers. Intuitively, when inputs cannot be reliably reconstructed from outputs.

Full Racecar Training: An illustration of a CNN structure with AF and BN and a full racecar loss is shown in Fig. [12] While the construction of the reverse pass is straight-forward for all standard operations, i.e., fully connected layers, convolutions, pooling, etc., slight adjustments are necessary for AF and BN. It is crucial for our formulation that \( d_m \) and \( d_m' \) contain the same latent space content in terms of range and dimensionality, such that they can be compared in the loss. Hence, we use the BN parameters and the AF of layer \( m-1 \) from the forward pass for layer \( m \) in the reverse pass. An example is shown in Fig. [15]

To illustrate this setup, we consider an example network employing convolutions with mixed AFs, BN, and MP. Let the network receives a field of \( 32^2 \) scalar values as input. From this input, 20, 40, and 60 feature maps are extracted in the first three layers. Besides, the kernel sizes are decreased from \( 5 \times 5 \) to \( 3 \times 3 \). To clarify the structure, we use ReLU activation for the first convolution, while the second one uses a hyperbolic tangent, and the third one a sigmoid function. With the notation
Figure 12: Left: An overview of the regular forward pass (blue) and the corresponding reverse pass (yellow). The right side illustrates how parameters are reused for a convolutional layer. conv/deconv denote convolution/deconvolutional operations. $f_m$ and $BN_m$ denote the activation function and batch normalization of layer $m$, respectively. Shared kernel and bias are represented by $M_m$ and $b_m$.

outlined above, the first three layers of the network are

$$I(32, 32, 1) = d_1 \rightarrow C_1(5, 20, 1) + b_1 \rightarrow BN_1 \rightarrow relu$$
$$\rightarrow d_2 \rightarrow MP \rightarrow C_2(4, 40, 1) + b_2 \rightarrow BN_2 \rightarrow tanh$$
$$\rightarrow d_3 \rightarrow MP \rightarrow C_3(3, 60, 1) + b_3 \rightarrow BN_3 \rightarrow sigm$$
$$\rightarrow d_4 \rightarrow ... \tag{23}$$

The reverse pass for evaluating the racecar loss re-uses all weights of the forward pass and ensures that all intermediate vectors of activations, $d_m$ and $d'_m$, have the same size and content in terms of normalization and non-linearity. We always consider states after activation for $L_{racecar}$. Thus, $d_m$ denotes activations before pooling in the forward pass and $d'_m$ contains data after up-sampling in the reverse pass, in order to ensure matching dimensionality. Thus, the last three layers of the reverse network for computing $L_{racecar}$ take the form:

$$... \rightarrow d'_4 \rightarrow -b_3 \rightarrow D_3(3, 40, 1) \rightarrow BN_2 \rightarrow tanh \rightarrow UP$$
$$\rightarrow d'_3 \rightarrow -b_2 \rightarrow D_2(4, 20, 1) \rightarrow BN_1 \rightarrow relu \rightarrow UP$$
$$\rightarrow d'_2 \rightarrow -b_1 \rightarrow D_1(5, 3, 1)$$
$$\rightarrow d'_1 = O(32, 32, 1). \tag{24}$$

Here, the de-convolutions $D_x$ in the reverse network share weights with $C_x$ in the forward network. I.e., the $4 \times 4 \times 20 \times 40$ weight matrix of $C_2$ is reused in its transposed form as a $4 \times 4 \times 40 \times 20$ matrix in $D_2$. Additionally, it becomes apparent that AF and BN of layer 3 from the forward pass do not appear in the listing of the three last layers of the reverse pass. This is caused by the fact that both are required to establish the latent space of the fourth layer. Instead, $d_3$ in our example represents the activations after the second layer (with $BN_2$ and $tanh$), and hence the reverse pass for $d_3$ reuses both functions. This ensures that $d_m$ and $d'_m$ contain the same latent space content in terms of range and dimensionality, and can be compared in Eq. (22).

For the reverse pass, we additionally found it beneficial to employ an AF for the very last layer if the output space has suitable content. E.g., for inputs in the form of RGB data we employ an additional activation with a ReLU function for the output to ensure the network generates only positive values. The architectures in Tab. 11, 13, 15, 19, 22, and 24 use such an activation.

Layer-wise Racecar Training: In the example above, we use a full racecar training with $d'_{m+1}$ to reconstruct the activations $d''_m$. The full racecar structure establishes a slightly stronger relationship among the racecar loss terms of different layers, and allows earlier layers to decrease the accumulated loss of later layers. However, if the architecture of the original network makes use of operations between layers that are not bijective, we instead use the layer-wise racecar loss. E.g., this happens for residual connections with an addition or non-invertible pooling operations such as max-pooling. In the former, we cannot uniquely determine the $b, c$ in $a = b + c$ given only $a$. And unless special care is taken [4], the source neuron of an output is not known for regular max-pooling operations. Note that our racecar loss has no problems with irreversible operations within a layer, e.g., most convolutional or fully-connected layers typically are not fully invertible. In all these cases the loss will drive the network towards a state that is as-invertible-as-possible for the given input data set. However, this requires a reliable vector of target activations in order to apply the constraints. If the
STDEV LPIPS

We, e.g., use this layer-wise loss in the Resnet110 network shown in Tab. 17.

For the MNIST, the examples given above.

Figure 14: Five repeated tests with the peak data shown in Sec. 3 of the main paper. RR\_A robustly extracts dominant features from the data set. The two singular vectors strongly resemble the two peak modes of the training data. This is confirmed by the LPIPS measurements.

connection between layers is not bijective, we cannot reconstruct this target for the constraints, as in the examples given above.

In such cases, we regard every layer as an individual unit to which we apply the racecar loss by building a layer-wise reverse pass. For example, given a simple convolutional architecture with

\[ \mathbf{d}_1 \rightarrow C_1(5, 20, 1) + \mathbf{b}_1 = \mathbf{d}_2 \]  \hspace{1cm} (25)

in the forward pass, we calculate \( \mathbf{d}_1 \) with

\[ (\mathbf{d}_2 - \mathbf{b}_1) \rightarrow D_1(5, 3, 1) = \mathbf{d}_1' \]  \hspace{1cm} (26)

We, e.g., use this layer-wise loss in the Resnet110 network shown in Tab. 17.

A.3 MNIST and Peak tests

Below we give details for the MNIST and peak tests from Sec. 3 of the main paper.

**MNIST Test**: For the MNIST test, additional SVDs of the weight matrices of trained models can be seen in Fig. [11] and Fig. [13]. The LPIPS scores (lower being better) show that features embedded in the weights of RR are significantly closer to the training data set than Std and Ort. All MNIST models are trained for 1000 epochs with a learning rate of 0.0001, and \( \lambda = 1e^{-5} \) for RR\_A. The NN architecture of the tests in this section is given in Tab. 8.

Figure 15: Right singular vectors of \( M_1 \) for peak tests with different network architectures: from a single linear layer to two layers with BN and AF. Across the three architectures, RR\_A successfully extracts dominant features. In contrast to Std and Ort, the singular vectors contain salient structures.
**Peak Test**: For the Peak test we generated a data set of 110 images shown in Fig. 16. 55 images contain a peak located in the upper left corner of the image. The other 55 contain a peak located in the bottom right corner. We added random scribbles in the images to complicate the task. All 110 images were labeled with a one-hot encoding of the two possible positions of the peak. We use 100 images as training data set, and the remaining 10 for testing. All peak models are trained for 5000 epochs with a learning rate of 0.0001, with $\lambda = 1e - 6$ for RR$_A$. To draw reliable conclusions, we show results for five repeated runs here. The neural network in this case contains one fully connected layer, with BN and ReLU activation. The results are shown in Fig. 14 with both peak modes being consistently embedded into the weight matrix of RR$_A$, while regular and orthogonal training show primarily random singular vectors.

We also use different network architectures in Fig. 15 to verify that the dominant features are successfully extracted when using more complex network structures. Even for two layers with BN and ReLU activations, the racecar training clearly extracts the two modes of the training data. The visual resemblance is slightly reduced in this case, as the network has the freedom to embed the features in both layers. Across all three cases (for which we performed 5 runs each), the racecar training clearly outperforms regular training and the orthogonality constraint in terms of extracting and embedding the dominant structures of the training data set in the weight matrix.

Overall, our experiments confirm the theory behind the racecar formulation. They additionally show that racecar training in combination with a subsequent SVD of the network weights yields a simple and convenient method to give humans intuition about the features learned by a network.

**B Evaluation via Mutual Information**

This section gives details of the mutual information and disentangled representation tests from Sec. 4 of the main paper.

**B.1 Mutual Information test**

We now evaluate our approach in terms of mutual information (MI), which measures the dependence of two random variables, i.e., higher MI means that there is more shared information between two parameters. More formally, the mutual information $I(X; Y)$ of random variables $X$ and $Y$ measures how different the joint distribution of $X$ and $Y$ is w.r.t. the product of their marginal distributions, i.e., the Kullback-Leibler divergence $I(X; Y) = \text{KL}[P_{X,Y} || P_X P_Y]$, where KL denotes the Kullback-Leibler divergence. Let $I(X; D_m)$ denote the mutual information between the activations of a layer $D_m$ and input $X$. Similarly $I(D_m; Y)$ denotes the MI between layer $m$ and the output $Y$. We use MI planes in the main paper, which show $I(X; D_m)$ and $I(D_m; Y)$ in a 2D graph for the activations of each layer $D_m$ of a network after training. This visualizes how much information about input and output distribution is retained at each layer, and how these relationships change within the network. For regular training, the information bottleneck principle [48] states that early layers contain more information about the input, i.e., show high values for $I(X; D_m)$ and $I(D_m; Y)$. Hence in the MI plane visualizations, these layers are often visible at the top-right corner. Later layers typically share a large amount of information with the output after training, i.e. show large $I(D_m; Y)$ values, and correlate less with the input (low $I(X; D_m)$). Thus, they typically show up in the top-left corner of the MI plane graphs.

**Training Details**: We use the same numerical studies as in [42] as task $A$, i.e. a regular feed-forward neural network with 6 fully-connected layers. The input variable $X$ contains 12 binary digits that represent 12 uniformly distributed points on a 2D sphere. The learning objective is to discover binary decision rules which are invariant under $O(3)$ rotations of the sphere. $X$ has 4096 different patterns, which are divided into 64 disjoint orbits of the rotation group, forming a minimal sufficient partition for spherically symmetric rules [27]. To generate the input-output distribution $P(X, Y)$, We apply the stochastic rule $p(y = 1|x) = \Psi(f(x) - \theta), (x \in X, y \in Y)$, where $\Psi$ is a standard sigmoidal...
function \( \Psi(u) = 1/(1 + \exp(-\gamma u)) \), following [42]. We then use a spherically symmetric real valued function of the pattern \( f(x) \), evaluated through its spherical harmonics power spectrum [27], and compare with a threshold \( \theta \), which was selected to make \( p(y = 1) = \sum_x p(y = 1|x)p(x) \approx 0.5 \), with uniform \( p(x) \). \( \gamma \) is high enough to keep the mutual information \( I(X; Y) \approx 0.99 \) bits.

For the transfer learning task \( B \), we reverse output labels to check whether the model learned specific or generalizing features. E.g., if the output is \([0,1]\) in the original data set, we swap the entries to \([1,0]\). 80\% of the data (3277 data pairs) are used for training and rests (819 data pairs) are used for testing. The forward and reverse pass architectures of the fully connected neural networks are given in Tab. 9. Hyperparameters used for training are listed in Tab. 10. Models are trained using cross-entropy as the base loss function.

For the MI comparison in Fig. 4 of the main paper, we discuss models before and after fine-tuning separately, in order to illustrate the effects of regularization. We include a regular model Std\( A \), one with orthogonality constraints Ort\( A \), and our regular racecar model RR\( A \), all before fine-tuning. For the regular racecar model RR\( A \) all layers are constrained to be recovered in the backward pass. We additionally include the version RR\( A \), i.e. a model trained with only one racecar loss term \( \lambda_1|\mathbf{d}_1 - \mathbf{d}_1'|\), which means that only the input is constrained to be recovered. Thus, RR\( A \) represents a simplified version of our approach which receives no constraints that intermediate results of the forward and backward pass should match. For Ort\( A \), we used the Spectral Restricted Isometry Property (SRIP) regularization [2].

\[
L_{SRIP} = \beta \sigma(W^TW - I),
\]

(27)

where \( W \) is the kernel, \( I \) denotes an identity matrix, and \( \beta \) represents the regularization coefficient. \( \sigma(W) = \sup_{\mathbf{z} \in \mathbb{R}^n, \mathbf{z} \neq 0} \frac{|\mathbf{W}\mathbf{z}|}{\|\mathbf{z}\|} \) denotes the spectral norm of \( W \).

As explained in the main text, all layers of the first stage, i.e. from RR\( A \), RR\( A \), Ort\( A \) and Std\( A \) are reused for training the fine-tuned models without regularization, i.e. RR\( AA \), RR\( AA \), Ort\( AA \) and Std\( AA \). Likewise, all layers of the transfer task models RR\( AB \), RR\( AB \), Ort\( AB \) and Std\( AB \) are initialized from the models of the first training stage.

**Analysis of Results:** We first compare the version only constraining input and output reconstruction (RR\( A \)) and the full racecar loss version RR\( A \). Fig. 4(b) of the main paper shows that all points of RR\( A \) are located in a central region of the MI place, which means that all layers successfully encode information about the inputs as well as the outputs. This also indicates that every layer contains a similar amount of information about \( X \) and \( Y \), and that the path from input to output is similar to the path from output to input. The points of RR\( A \), on the other hand, form a diagonal line. I.e., this network has different amounts of mutual information across its layers, and potentially a very different path for each direction. This difference in behavior is caused by the difference of the constraints in these two versions: RR\( I \) is only constrained to be able to regenerate its input, while the full racecar loss for RR\( A \) ensures that the network learns features which are beneficial for both directions. This test highlights the importance of the constraints throughout the depth of a network in our formulation. In contrast, the \( I(D; Y) \) values of later layers for Std\( A \) and Ort\( A \) exhibit small values (points near the left side), while \( I(D; Y) \) is high throughout. This indicates that the outputs were successfully encoded and that increasing amounts of information about the inputs are discarded. Hence, more specific features about the given output data-set are learned by Std\( A \) and Ort\( A \). This shows that both models are highly specialized for the given task, and potentially perform worse when applied to new tasks.

During the fine-tuning phase for task \( A \) (i.e. regularizers being disabled), all models focus on the output and maximize \( I(D; Y) \). There are differences in the distributions of the points along the y-axis, i.e., how much MI with the output is retained, as shown in Fig. 4(c) of the main paper. For model RR\( A \), the \( I(D; Y) \) value is higher than for Std\( A \), Ort\( AA \) and RR\( AA \), which means outputs of RR\( AA \) are more closely related to the outputs, i.e., the ground truth labels for task \( A \). Thus, RR\( AA \) outperforms the other variants for the original task.

In the fine-tuning phase for task \( B \), Std\( AB \) stands out with very low accuracy in Fig. 5 of the main paper. This model from a regular training run has large difficulties to adapt to the new task. Model Ort\( AB \) also performs worse than Std\( A \). RR\( AB \) shows the best performance in this setting, demonstrating that our loss formulation helped to learn more generic features from the input data. This improves the performance for related tasks such as the inverted outputs used for \( B \).
We also analyze the two variants of racecar training training, the layer-wise variant \( l_{RR} \) and the full version \( RR_A \) in terms of mutual information. Fig. [7] shows the MI planes for these two models, also showing \( RR_A \) for comparison. Despite the local nature of \( RR_A \) it manages to establish MI for the majority of the layers, as indicated by the cluster of layers in the center of the MI plane. Only the first layer moves towards the top right corner, and the second layer is affected slightly. I.e., these layers exhibit a stronger relationship with the distribution of the outputs. Despite this, the overall performance when fine-tuning or for the task transfer remains largely unaffected, e.g., the \( RR_A \) still clearly outperforms \( RR_A \). This confirms our choice to use the full racecar training when network connectivity permits, and employ the layer-wise version in all other cases.

Numerical accuracies for all models discussed in this section are listed in Tab. 1 and Tab. 2. Mutual information values, i.e., \( I(X, D_m) \) and \( I(D_m, Y) \), for all models and layers are provided in Tab. 26.

### B.2 Disentangled Representations

The InfoGAN approach [5] demonstrated the possibility to control the output of generative models via maximizing mutual information between outputs and structured latent variables. However, mutual information is very hard to estimate in practice [31]. The previous section and Fig. 4(b) of the main paper demonstrated that models from racecar training (both \( RR_A \) and \( RR_A \)) can increase the mutual information between network inputs and outputs. Intuitively, racecar training explicitly constrains the model to recover an input given an output, which directly translates into an increase of mutual information between latent variables and the output, we also focus the racecar training on the corresponding latent variables. I.e., the goal is to maximize their mutual information with the output of the generative model. Hence, we train a model \( RR_1 \) for which only latent dimensions \( c_1, c_2, c_3 \) of the input layer are involved in the racecar loss. We still employ a full reverse pass structure in the neural network architecture. \( c_1 \) is a ten-dimensional category code, which is used for controlling the output digit category, while \( c_2 \) and \( c_3 \) are continuous latent codes, to represent (previously unknown) key properties of the digits, such as orientation or thickness. Building relationship between \( c_2 \) and outputs is more difficult than for \( c_2 \) or \( c_3 \), since the 10 different digit outputs need to be encoded in a single continuous variable \( c_1 \). Thus, for the corresponding racecar loss term for \( c_1 \) we use a slightly larger \( \lambda \) factor (by 33\%) than for \( c_2 \) and \( c_3 \). The forward and reverse pass architectures of the network are given in Tab. 11, and hyperparameters are listed in Tab. 12. Details of our results are shown in Fig. [18]. Models are trained using a GAN loss [17] as the loss function for the outputs.

**Training Details:** As InfoGAN focuses on structuring latent variables and thus only increases the mutual information between latent variables and the output, we also focus the racecar training on the corresponding latent variables. I.e., the goal is to maximize their mutual information with the output of the generative model. Hence, we train a model \( RR_1 \) for which only latent dimensions \( c_1, c_2, c_3 \) of the input layer are involved in the racecar loss. We still employ a full reverse pass structure in the neural network architecture. \( c_1 \) is a ten-dimensional category code, which is used for controlling the output digit category, while \( c_2 \) and \( c_3 \) are continuous latent codes, to represent (previously unknown) key properties of the digits, such as orientation or thickness. Building relationship between \( c_2 \) and outputs is more difficult than for \( c_2 \) or \( c_3 \), since the 10 different digit outputs need to be encoded in a single continuous variable \( c_1 \). Thus, for the corresponding racecar loss term for \( c_1 \) we use a slightly larger \( \lambda \) factor (by 33\%) than for \( c_2 \) and \( c_3 \). The forward and reverse pass architectures of the network are given in Tab. 11, and hyperparameters are listed in Tab. 12. Details of our results are shown in Fig. [18]. Models are trained using a GAN loss [17] as the loss function for the outputs.

**Analysis of Results:** In Fig. [18] we show additional results for the disentangling test case. It is visible that the racecar training with the \( RR_1 \) model yield distinct and meaningful latent space
Figure 18: Additional results for the disentangled representations with the MNIST data: For every row in the figures, we vary the corresponding latent code (left to right), while keeping all other inputs constant. Different rows indicate a different random noise input. For example, in (b): every column contains five results which are generated with 10 different values of $c_1$, which correspond to one digit each for (b). (a): For a regular training (Std), no clear correspondence between $c_1$ and the outputs are apparent (similarly for $c_2, c_3$). (c): Different $c_2$ values result in a tweaked style, while $c_3$ controls the orientation of the digit, as shown in (d). Thus, in contrast to Std, the racecar model learns a meaningful, disentangled representation.

dimensions for $c_{1,2,3}$. While $c_1$ controls the digit, $c_{2,3}$ control the style and orientation of the digits. For comparison, a regular training run with model Std does result in meaningful or visible changes when adjusting the latent space dimensions. This illustrates how strongly racecar training can shape the latent space, and in addition to an intuitive embedding of dominant features, yield a disentangled representation.

C Additional Experiments

In this section we present several additional experiments that were not included in the main document due to space constraints.

C.1 MNIST Classification

The MNIST data set is a commonly used data set for handwritten digit classification [32]. We train three types of models with the regular MNIST data set: one with racecar loss RR, a regular model Std, and one with orthogonal constraints Ort [2] for comparison. Usually, convolutional layers take up most of the model parameters, so we correspondingly compute the racecar loss for the convolutional layers, omitting the fully connected layers that would be required for class label inference. We follow the common practice to first train models with regularization and then fine-tune without [2]. As we discuss model states before fine-tuning in this section, we use the following naming scheme with suffixes mni, MNI and n-MNI to distinguish the different phases. E.g., with racecar training: $RR_{mni}$ for training runs on MNIST with regularization, $RR_{MNI}$ for fine-tuning on MNIST without regularization using $RR_{mni}$ as starting point, and $RR_{n-MNI}$ for fine-tuning on a transfer task without regularization, likewise using $RR_{mni}$ as starting point.

Training Details: The regular MNIST data set [32] contains 55k images for training and 10k images for testing. For the n-MNIST motion blur data set [3], there are 60k images for training and 10k images for testing. All images have a size of 28 × 28. Example data from n-MNIST that illustrates the motion blur is shown in Fig. 20. Details of the network architectures are shown in Tab. 15, with hyperparameters given in Tab. 16. The models are trained using cross-entropy as base loss function, and all three convolutional layers are used for the racecar loss.

Analysis of Results: To highlight the properties of our algorithm, we show comparisons between the original input $d_1$ and the reconstructed inputs $d'_1$ in Fig. 19 for the models $RR_{mni}$, Std$_{mni}$, and Ort$_{mni}$. These models were trained with an enabled regularization, i.e., before fine tuning. For
racecar training, most of the features from the input are recovered, while trying to invert the network in the same way for a regular training run or a training with orthogonal constraints largely fails. In both cases, features are extracted only according to the digit classification task, and both models discard information unrelated to this goal.

For the final MNIST models including fine-tuning, i.e., Ort\_MNI, Std\_MNI, and RR\_MNI, results are shown in Fig. 21 for 5 repeated runs. We can see that RR\_MNI outperforms Ort\_MNI and Std\_MNI, which indicates that racecar training yields generic features that can also improve performance for the original task.

As generic features are typically more robust than specific features [36], we investigate a perturbed data set with motion blur (n-MNIST [3]) for the task transfer. We fine-tune all three previous models for the n-MNIST data set, to obtain the models Ort\_n-MNI, Std\_n-MNI, and RR\_n-MNI. Performance results are likewise given in Fig. 21. Based on the same CNN architecture and parameters, RR\_n-MNI achieves the best performance. This indicates that the RR model learned more generic features via racecar training than Ort and Std. Interestingly, the racecar training not only improves inference accuracy, but also stabilizes training, as indicated by the variance of the results in Fig. 21.

For fine-tuning the models without regularization, i.e. for training RR\_n-MNI/RR\_n-MNI, Std\_n-MNI/Std\_n-MNI and Ort\_n-MNI/Ort\_n-MNI, we load RR\_n-MNI, Std\_n-MNI, and Ort\_n-MNI, and continue training without any orthogonalization or racecar loss terms. Example training processes of the MNIST tests are shown in Fig. 22. We can see that the racecar loss increases the task difficulty, so RR\_n-MNI exhibits a slightly lower performance and longer training time than Std\_n-MNI in the first phase. In the second phase RR\_n-MNI outperforms Std\_n-MNI and Ort\_n-MNI for both tasks. This illustrates that the dominant features extracted by racecar training also improved performance for both task domains. Numeric accuracies of all models are given in Tab. 3 and Tab. 4.

C.2 Natural Image Classification

Natural images arise in many important application scenarios, and in Sec.5 of the main paper we evaluated racecar training for a very deep ResNet110 for CIFAR 10 [28]. This experiment already demonstrated that racecar training can yield substantial gains in SOTA performance compared to orthogonality constraints. These gains are on a level that is similar with gains that orthogonality constraints exhibit over a regular training. Details of the corresponding Resnet110 accuracies, network architecture, and hyperparameters are shown in Tab. 7, 17 and 18, respectively. To show that racecar training also yields gains in shallow networks, we additionally evaluate our approach with the CIFAR data set using a 19-layer network.

Figure 22: Visualizations of training processes for the MNIST cases. Left, constraints enabled: RR\_mni (accuracy: 0.9810, perf.: 5.675 s/epoch), Std\_mni (accuracy: 0.9827, perf.: 3.522 s/epoch) and Ort\_mni (accuracy: 0.9792, perf.: 4.969 s/epoch). Middle, fine tuning: RR\_MNI, Std\_MNI and Ort\_MNI. Right, task transfer: RR\_n-MNI, Std\_n-MNI, Ort\_n-MNI, and Std\_B.

Figure 21: Accuracy comparisons for original task (Acc\_MNI) and transfer learning task (Acc\_n-MNI). The dashed grey line and grey region represent the accuracy of baseline models, Std\_n-MNI-fs trained from scratch for n-MNIST. RR\_MNI and RR\_n-MNI outperform this baseline, and achieve the best performance for the source task and transfer learning task, respectively.
Figure 23: Accuracy comparisons of natural image tasks $A$ and $B$. Grey line and grey region represent accuracy of baseline Std$_{C100-fs}$. RR$_{C10}$ and RR$_{C100}$ got best performance for task $A$ and $B$.

Figure 24: Accuracy comparisons when applying models trained on CIFAR 10 to CIFAR 10.1 data.

Figure 25: Visualizations of training processes. Left: CIFAR 10 with constraints, RR$_{c1}$ (accuracy: 0.5784, cost: 64 seconds/epoch) and Std$_{c1}$ (accuracy: 0.8272, cost: 63 seconds/epoch). Middle: CIFAR 10, fine tuning, RR$_{C10}$ and Std$_{C10}$. Right: Task transfer for CIFAR 100, RR$_{C100}$, Std$_{C100}$ and Std$_{C100-fs}$.

**Training Details:** Both CIFAR 10 and CIFAR 100 data sets consist of 60k $32 \times 32$ images, 50k of them are used for training and 10k of them are used for testing, classes. The forward and reverse pass architectures of the network can be found in Tab. 19 with parameters given in Tab. 20. Models are trained using cross-entropy as loss function, and all 13 convolutional layers are used for the racecar loss. For fine-tuning on CIFAR 100, the last layer is omitted when loading due to the changed size of the output.

**Analysis of Results:** At first, we train two models with the CIFAR 10 data set as classification task $C10$, i.e., RR$_{c1}$ and Std$_{c1}$. Example training processes of the CIFAR tests are shown in Fig. 25. We fine-tune to obtain RR$_{C10}$ and Std$_{C10}$, results for which are shown in Fig. 23. Numerical results are given in Tab. 5. The racecar training also improves performance for this natural image classification task. As transfer learning task we fine-tune RR$_{c1}$ and Std$_{c1}$ for the CIFAR 100 data set classification, yielding models RR$_{C100}$ and Std$_{C100}$. Results are likewise shown in Fig. 23. Training with the same CNN architecture and parameters, Std$_{c1}$ has difficulties adjusting to the new task, while our model from the initial racecar training even slightly outperforms a model trained from scratch for CIFAR 100 (Std$_{C100-fs}$).

As in the main text, we evaluate these models with the CIFAR transfer benchmark from Recht et al. [41]. It measures how well models trained on the original CIFAR data set, i.e. CIFAR 10, generalize to a new data set (CIFAR 10.1) collected according to the same principles as the original CIFAR 10. Here, it is additionally interesting to see how well performance for CIFAR 10 translates into transfer performance for 10.1. As shown in Fig. 24, estimating the slope of this relationship with a linear fit yields 0.9012 for Std$_{c1}$, and 1.5456 for RR$_{C10}$. Thus, while regular training performs worse on the new data when its performance for the original task increases (with a slope of less than one), the models from racecar training very successfully translate gains in performance from the original task to the new one. This indicates that the models have successfully learned a set of generalizing features.
D Experimental Results

Below, we give additional details for the experiments of Sec. 5 of the main paper.

D.1 Texture-shape Benchmark

Training Details: All training data of the texture-shape tests were obtained from [15]. The stylized data set contains 1280 images, 1120 images are used as training data, and 160 as test data. Both edge and filled data sets contain 160 images each, all of which are used for testing only. All three sets (stylized, edge, and filled) contain data for 16 different classes. The forward and reverse pass architectures of the network are given in Tab. 13, with hyperparameters in Tab. 14. Numerical accuracy for RR\textsubscript{TS}, Ort\textsubscript{TS} and Std\textsubscript{TS} is given in Tab. 27, Tab. 28 and Tab. 29, respectively.

Analysis of Results: For a detailed comparison, we list per-class accuracy of stylized data training runs for Ort\textsubscript{TS}, Std\textsubscript{TS} and RR\textsubscript{TS} in Fig. 26. RR\textsubscript{TS} outperforms the other two models for most of the classes. Training times (in seconds) for different models are listed in Tab. 30. RR\textsubscript{TS} requires 41.86% more time for training compared to Std\textsubscript{TS}, but yields a 23.76% higher performance. All models saturated, i.e. training Std\textsubscript{TS} or Ort\textsubscript{TS} longer does not increase classification accuracy any further. We also investigated how much we can reduce model size when using racecar training in comparison to the baselines. A reduced racecar model only has 67.94% of the parameters, while still outperforming Ort\textsubscript{TS}.

D.2 Generative Adversarial Models

Training Details: The data set of smoke simulation was generated with a fluid solver from an open-source library [47]. We generated 20 simulations with 120 frames each, with 10% of the data being used for training. Smoke inflow region, inflow velocity, and buoyancy force were randomized to produce varied data. The low-resolution data were down-sampled from the high-resolution data by a factor of 4. Data augmentation, such as flipping and rotation was used in addition. As outlined in the main text, we consider building an autoencoder model for the synthetic data as task B\textsubscript{1}, and a generating samples from a real-world smoke data set as task B\textsubscript{2}. The smoke capture data set for B\textsubscript{2} contains 2500 smoke images from the ScalarFlow data set [12], and we again used 10% of these images as training data set.

Task A: We use a fully convolutional CNN-based architecture for generator and discriminator networks (cf. Tab. 21, and parameters in Tab. 23.). Note that the inputs of the discriminator contain high resolution data (64, 64, 1), as well as low resolution (16, 16, 1), which is up-sampled to (64, 64, 1) and concatenated with the high resolution data. In line with previous work [53], RR\textsubscript{A} and Std\textsubscript{A} are trained with a non-saturating GAN loss, feature space loss and L2 loss as base loss function. All generator layers are involved in the racecar loss.

Task B\textsubscript{1}: All encoder layers are initialized from RR\textsubscript{A} and Std\textsubscript{A} when training RR\textsubscript{AB\textsubscript{1}} and Std\textsubscript{AB\textsubscript{1}}. It is worth noting that the reverse pass of the generator is also constrained when training RR\textsubscript{A}. So both encoder and decoder are initialized with parameters from RR\textsubscript{A} when training RR\textsubscript{AB\textsubscript{1}}. This is not possible for a regular network like Std\textsubscript{AB\textsubscript{1}}, as the weights obtained with a normal training run are not suitable to be transposed. Hence, the de-convolutions of Std\textsubscript{AB\textsubscript{1}} are initialized randomly.

Task B\textsubscript{2}: As the data set for the task B\textsubscript{2} is substantially different and contains RGB images (instead of single channel gray-scale images), we choose the following setups for the RR\textsubscript{A} and Std\textsubscript{A} models: parameters from all six layers of Std\textsubscript{A} and RR\textsubscript{A} are reused for initializing decoder part of Std\textsubscript{AB\textsubscript{2}} and RR\textsubscript{AB\textsubscript{2}}. Specially, when initializing the last layer of of Std\textsubscript{AB\textsubscript{2}} and RR\textsubscript{AB\textsubscript{2}}, we copy and stack the parameters from the last layer of Std\textsubscript{A} and RR\textsubscript{A} into three channels to fit output data size in task B\textsubscript{2}. Here, the encoder part of RR\textsubscript{AB\textsubscript{2}} is not initialized with RR\textsubscript{A}, due to the significant gap between training data sets of task B\textsubscript{1} and task B\textsubscript{2}. Our experiments show that only initializing the
Figure 27: Example outputs for $\text{RR}_{AB_1}$, $\text{Std}_{AB_1}$, and $\text{Std}_{B_1}$. The reference is shown for comparison. $\text{RR}_{AB_1}$ produces higher quality results than $\text{Std}_{AB_1}$, while a model trained from scratch, $\text{Std}_{B_1}$, fails for this task. It produces a mostly black image.

Figure 28: Mean Absolute Error (MAE) comparisons for smoke task $B_2$ models. $\text{RR}_{AB_2}$ shows the smallest error, and additionally achieves the best visual quality amongst the different models. It even outperforms the baseline model trained from scratch, i.e., $\text{Std}_{B_2}$.

decoder part of $\text{RR}_{AB_2}$ (avg. loss: $1.56e7$, std. dev.: $3.81e5$) outperforms initializing both encoder and decoder (avg. loss: $1.82e7$, std. dev.: $2.07e6$). We believe the reason is that initializing both encoder and decoder part makes it more difficult to adjust the parameters for new data set that is very different from the data set of the source task.

**Analysis of Results:** We first discuss the autoencoder transfer learning task $AB_1$ for synthetic, i.e., simulated fluid data. (architectures for $B_1$ and $B_2$ are given in Tab. 22). Example outputs of $\text{RR}_{AB_1}$, $\text{Std}_{AB_1}$, and $\text{Std}_{B_1}$ are shown in Fig. 27. It is clear that $\text{RR}_{AB_1}$ gives the best performance among these models. We similarly illustrate the behavior of the transfer learning task $AB_2$ for images of real-world fluids. This example likewise uses an autoencoder structure. Visual comparisons are provided in Fig. 28 where $\text{RR}_{AB_2}$ generates results that are closer to the reference. Overall, these results demonstrate the benefits of racecar training for GANs, and indicate its potential to obtain more generic features from synthetic data sets that can be used for tasks involving real-world data.

**D.3 VGG19 Stylization**

The section below gives details of the stylization tests in Sec. 5.4 of the main paper.
VGG19 Training: For the ImageNet data set [7], 1281167 images of 1000 classes are used for training, and 50k images are used for testing. The size of all images is $224 \times 224$. The forward and reverse pass of VGG19 are given in Tab. 24, with hyperparameters shown in Tab. 25. All 16 convolutional layers are used for the racecar loss. To speed up the training process of RR, we first train a model without racecar loss for 6 epochs with batch size 64. We then reuse this model for training RR and Std with a batch size of 24 for 30 epochs.

Stylization Background: Gatys et al. [14] propose to use a loss consisting of a content and a style term, i.e. $L_{\text{total}} = \eta L_{\text{content}} + \delta L_{\text{style}}$, where $\eta$ and $\delta$ denote coefficients. $L_{\text{content}}$ calculates a content difference between source image $p$ and generated image $g$:

$$L_{\text{content}}(p, g, t) = \frac{1}{2} \sum_{m,n} (F_{m,n,t}^p - F_{m,n,t}^g)^2,$$

where $F_{m,n,t}^p$ is the feature representation of $p$ of the $m$th filter at position $n$ in layer $t$. $L_{\text{style}}$ instead computes a style difference between style image $a$ and a generated image $g$:

$$G_{a}^{m,n,t} = \sum_{f} F_{a}^{m,f,t} F_{a}^{n,f,t},$$

$$E_{t} = \frac{\omega_{t}}{N_{t} M_{t}^{2}} \sum_{m,n} (G_{a}^{m,n,t} - G_{a}^{m,n,t})^2,$$

$$L_{\text{style}}(a, g) = \sum_{t=0}^{T} \omega_{t} E_{t},$$

where $\omega_{t}$ are the weighting factors for layer $t$; $G$ is the Gram matrix; $N_{t}$ denotes filter numbers of layer $t$, and $M_{t}$ is the dimension of layer $t$’s filter. $T$ denotes the number of layers included in the style loss.

D.4 Additional Stylization Results

To compare the feature extracting capabilities between the RR and Std runs, we only optimize $L_{\text{style}}$ as a first test. Two geometric shapes are given as source and style images, as shown in Fig. 29. They have the same background and object color. The only difference is the large-scale shape of the object. Thus, as we aim for applying a style that is identical to the source, we can test whether the features can clearly separate and preserve the large scale shape features from the ones for the localized style. Comparing the results of RR and Std, the output of RR is almost identical to the input, while stylization with Std changes the shape of the object introducing undesirable streaks around the outline. Thus, RR performs significantly better.

Fig. 30 shows a painterly style transfer, where both $L_{\text{style}}$ and $L_{\text{content}}$ are optimized. Here, the regular Std model mixes sky and house structures, indicating that the extracted features fail to separate style and content information. A more difficult case with a horse-to-zebra transfer is shown in Fig. 31. After optimizing $L_{\text{content}}$ and $L_{\text{style}}$, our method generates a zebra pattern in the horse’s shadow (zoom-in in the yellow box), but yields improved results compared to a regular model. Those results indicate that RR is able to clearly encode the shape of the object and preserve it during the stylization, while the features of Std fail to clearly separate shape and style.

We additionally confirm the capabilities of the VGG model from racecar training with a low- to high-resolution transfer where the goal is purely to add detail. I.e., we can compute in-place errors w.r.t. a high-resolution reference. Both $L_{\text{content}}$ and $L_{\text{style}}$ are optimized and the result is shown in Fig. 32. The zoomed-in areas in the figure highlight these differences. The VGG model obtained with a regular training run yields significantly higher errors than the racecar model, as shown on the right side of Fig. 32. Here, $E_{\text{Std}}$ contains substantially larger errors w.r.t. the reference than $E_{\text{RR}}$.

E Impact Statement

Racecar training could be applied to a wide range of applications, such as computer vision, computer graphics, classification tasks, generative tasks, and many more. We specifically focused on transfer learning in our paper. For regular training runs, models are trained to establish a direct relationship
between the input and output distribution via a specific loss function. We instead aimed at learning dominant features from the dataset via building a forward and reverse pass network. Motivated from the way humans learn, it improves the transferability of trained models. Generally, if we use the same data set for different purposes, we have to train different models from scratch. With racecar training, we can train a model which contains the dominant features of a data set as a starting point, and then reuse this model for different purposes. This is a more efficient process that likewise could be beneficial for a wide range of applications.

Besides, our research could also be used to analyze which input features are extracted by a layer, and thus improve the interpretability of neural networks in general. Hence, racecar training can be applied when the dominant features or modes of a data set are essential for making decisions about how to employ learned models.

Those properties could also result in potential risks: e.g., for data sets with personal data, i.e., in situations where privacy is critical, a learned representation using racecar training might contain more sensitive personal details than when performing a regular training. Hence, the acquired models will require additional safety measures. Thus, racecar training is, at the same time, a potential tool for analyzing the safety of a learned representation, as well as a potential risk. We encourage researchers to consider these aspects and risks arising from them for future directions of their work.
References

[1] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. arXiv preprint arXiv:1808.04730, 2018.

[2] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? In Advances in Neural Information Processing Systems, pages 4266–4276. Curran Associates Inc., 2018.

[3] Saikat Basu, Manohar Karki, Sangram Ganguly, Robert DiBiano, Supratik Mukhopadhyay, Shreekant Gayaka, Rajgopal Kannan, and Ramakrishna Nemani. Learning sparse feature representations using probabilistic quadtrees and deep belief nets. Neural Processing Letters, 45(3):855–867, 2017.

[4] Joan Bruna, Arthur Szlam, and Yann LeCun. Signal recovery from pooling representations. arXiv preprint arXiv:1311.4025, 2013.

[5] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In Advances in Neural Information Processing Systems, pages 2172–2180, 2016.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[8] Hui Ding, Shaohua Kevin Zhou, and Rama Chellappa. Facenet2expnet: Regularizing a deep face recognition net for expression recognition. In 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), pages 118–126. IEEE, 2017.

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. arXiv preprint arXiv:1605.08803, 2016.

[10] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. arXiv preprint arXiv:1808.00033, 2018.

[11] Lixin Duan, Dong Xu, and Ivor Tsang. Learning with augmented features for heterogeneous domain adaptation. arXiv preprint arXiv:1206.4660, 2012.

[12] Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. ACM Transactions on Graphics (TOG), 38(6):239, 2019.

[13] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635, 2018.

[14] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2414–2423, 2016.

[15] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. arXiv preprint arXiv:1811.12231, 2018.

[16] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In Advances in Neural Information Processing Systems, pages 2214–2224, 2017.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in Neural Information Processing Systems, pages 2672–2680, 2014.

[18] Kasthurirangan Gopalakrishnan, Siddhartha K Khaitan, Alok Choudhary, and Ankit Agrawal. Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. Construction and Building Materials, 157:322–330, 2017.

[19] Alison Gopnik, Andrew N Meltzoff, and Patricia K Kuhl. The scientist in the crib: Minds, brains, and how children learn. William Morrow & Co, 1999.
[20] Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In Advances in Neural Information Processing Systems, pages 177–185, 1989.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[23] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[24] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. arXiv preprint arXiv:1802.07088, 2018.

[25] Kui Jia, Dacheng Tao, Shenghua Gao, and Xiangmin Xu. Improving training of deep neural networks via singular value bounding. In IEEE Conference on Computer Vision and Pattern Recognition, pages 4344–4352, 2017.

[26] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. arXiv preprint arXiv:1710.05468, 2017.

[27] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In Symposium on geometry processing, volume 6, pages 156–164, 2003.

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105, 2012.

[30] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In Advances in Neural Information Processing Systems, pages 950–957, 1992.

[31] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In IEEE conference on computer vision and pattern recognition, pages 1785–1792. IEEE, 2011.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[33] Feng Li, Jacek M Zurada, Yan Liu, and Wei Wu. Input layer regularization of multilayer feedforward neural networks. IEEE Access, 5:10979–10985, 2017.

[34] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. International Journal of Computer Vision, 120(3):233–255, 2016.

[35] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In Advances in Neural Information Processing Systems, pages 5947–5956, 2017.

[36] Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. arXiv preprint arXiv:1802.08760, 2018.

[37] Mete Ozay and Takayuki Okatani. Optimization on submanifolds of convolution kernels in cnns. arXiv preprint arXiv:1610.07008, 2016.

[38] Anh Huy Phan and Andrzej Cichocki. Tensor decompositions for feature extraction and classification of high dimensional datasets. Nonlinear theory and its applications, IEICE, 1(1):37–68, 2010.

[39] Peter Prettenhofer and Benno Stein. Cross-language text classification using structural correspondence learning. In Proceedings of the 48th annual meeting of the association for computational linguistics, pages 1118–1127, 2010.

[40] Hariharan Ravishankar, Prasad Sudhakar, Rahul Venkataramani, Sheshadri Thiruvengadam, Pavan Anningi, Narayanan Babu, and Vivek Vaidya. Understanding the mechanisms of deep
transfer learning for medical images. In Deep Learning and Data Labeling for Medical Applications, pages 188–196. Springer, 2016.

[41] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In International Conference on Machine Learning, 2019.

[42] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810, 2017.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1):1929–1958, 2014.

[45] S-H Tang, Gang Chen, and Y-S Ma. Fundamental concepts of generic features. In Semantic modeling and interoperability in product and process engineering, pages 89–115. Springer, 2013.

[46] Yunfei Teng and Anna Choromanska. Invertible autoencoder for domain adaptation. Computation, 7(2):20, 2019.

[47] Nils Thuerey and Tobias Pfaff. MantaFlow, 2018. http://mantaflow.com.

[48] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In 2015 IEEE Information Theory Workshop (ITW), pages 1–5. IEEE, 2015.

[49] Lisa Torrey and Jude Shavlik. Transfer learning. In Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, pages 242–264. IGI Global, 2010.

[50] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In A practical approach to microarray data analysis, pages 91–109. Springer, 2003.

[51] Janett Walters-Williams and Yan Li. Estimation of mutual information: A survey. In International Conference on Rough Sets and Knowledge Technology, pages 389–396. Springer, 2009.

[52] Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. Generalization by weight-elimination with application to forecasting. In Advances in Neural Information Processing Systems, pages 875–882, 1991.

[53] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. ACM Transactions on Graphics (TOG), 37(4):95, 2018.

[54] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In IEEE Conference on Computer Vision and Pattern Recognition, pages 3712–3722, 2018.

[55] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In European conference on computer vision, pages 818–833. Springer, 2014.

[56] Lijing Zhang, Yao Lu, Ge Song, and Hanfeng Zheng. Re-cnn: Reverse connected convolutional neural network for accurate player detection. In Pacific Rim International Conference on Artificial Intelligence, pages 438–446. Springer, 2018.

[57] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In IEEE Conference on Computer Vision and Pattern Recognition, pages 586–595, 2018.

[58] Joey Tianyi Zhou, Sinno Jianlin Pan, Ivor W Tsang, and Yan Yan. Hybrid heterogeneous transfer learning through deep learning. In Twenty-eighth AAAI conference on artificial intelligence, 2014.

[59] Joey Tianyi Zhou, Ivor W Tsang, Sinno Jianlin Pan, and Mingkui Tan. Heterogeneous domain adaptation for multiple classes. In Artificial Intelligence and Statistics, pages 1095–1103, 2014.

[60] Yin Zhu, Yuqiang Chen, Zhongqi Lu, Sinno Jianlin Pan, Gui-Rong Xue, Yong Yu, and Qiang Yang. Heterogeneous transfer learning for image classification. In Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011.
| Training Runs | StdA | OrtA | RR1A | lRRA | RR_A | StdAA | OrtAA | RR1AA | lRRAA | RRAA |
|---------------|------|------|------|------|------|-------|-------|-------|-------|------|
| 1             | 0.976 | 0.969 | 0.836 | 0.8350 | 0.772 | 0.978 | 0.968 | 0.885 | 0.9770 | 0.992 |
| 2             | 0.975 | 0.921 | 0.68  | 0.8360 | 0.784 | 0.983 | 0.926 | 0.918 | 0.9770 | 0.994 |
| 3             | 0.964 | 0.957 | 0.8    | 0.8140 | 0.725 | 0.963 | 0.958 | 0.993 | 0.9740 | 0.993 |
| 4             | 0.987 | 0.991 | 0.657 | 0.8220 | 0.771 | 0.899 | 0.993 | 0.917 | 0.9980 | 0.989 |
| 5             | 0.957 | 0.979 | 0.825 | 0.8400 | 0.774 | 0.954 | 0.979 | 0.944 | 0.9710 | 0.993 |
| Avg           | 0.972 | 0.963 | 0.760 | 0.8334 | 0.765 | 0.973 | 0.965 | 0.931 | 0.9794 | 0.992 |
| Std. dev.     | 0.012 | 0.027 | 0.085 | 0.0112 | 0.023 | 0.015 | 0.025 | 0.040 | 0.0107 | 0.002 |

| Training Runs | StdAB | OrtAB | StdAP | RR1AP | lRRAP | RRAP | StdNMI | OrtNMI | StdMNI | RRNMNI |
|---------------|-------|-------|-------|-------|-------|------|-------|--------|-------|--------|
| 1             | 0.136 | 0.084 | 0.852 | 0.919 | 0.998 | 0.998 |
| 2             | 0.136 | 0.136 | 0.991 | 0.954 | 0.991 | 0.997 |
| 3             | 0.989 | 0.957 | 0.953 | 0.977 | 0.978 | 0.995 |
| 4             | 0.136 | 0.991 | 0.909 | 0.963 | 0.989 | 0.997 |
| 5             | 0.96  | 0.977 | 0.965 | 0.963 | 0.969 | 1.000 |
| Avg           | 0.459 | 0.376 | 0.055 | 0.022 | 0.0115 | 0.002 |
| Std. dev.     | 0.089 | 0.982 | 0.982 | 0.981 | 0.984 | 0.986 |

| Training Runs | OrtMNI | StdMNI | RRMNI | OrtMNI | StdMNI | RR_MNI |
|---------------|--------|--------|-------|--------|--------|--------|
| 1             | 0.9792 | 0.9827 | 0.981 | 0.9819 | 0.9854 | 0.9856 |
| 2             | 0.8322 | 0.9815 | 0.982 | 0.982 | 0.9837 | 0.9851 |
| 3             | 0.9511 | 0.9815 | 0.982 | 0.9792 | 0.983 | 0.9862 |
| 4             | 0.8841 | 0.9817 | 0.9816 | 0.9814 | 0.9842 | 0.9867 |
| 5             | 0.8266 | 0.9829 | 0.9816 | 0.9816 | 0.9846 | 0.9861 |
| Avg           | 0.895 | 0.982 | 0.982 | 0.981 | 0.984 | 0.986 |
| Std. dev.     | 0.0485 | 0.0007 | 0.0007 | 0.0011 | 0.0009 | 0.0006 |

| Training Runs | OrtN-M | StdN-M | RRN-M | StdN-MI | RRN-MI |
|---------------|--------|--------|-------|---------|-------|
| 1             | 0.949  | 0.956 | 0.9612 | 0.9612 | 0.8714 |
| 2             | 0.8629 | 0.9458 | 0.954 | 0.9458 | 0.9657 |
| 3             | 0.9458 | 0.9543 | 0.9648 | 0.9648 | 0.9659 |
| 4             | 0.9495 | 0.9559 | 0.9604 | 0.9604 | 0.8789 |
| 5             | 0.8564 | 0.9466 | 0.9649 | 0.9649 | 0.9638 |
| Avg           | 0.913  | 0.952 | 0.961 | 0.961 | 0.929 |
| Std. dev.     | 0.0485 | 0.0051 | 0.00044 | 0.0044 | 0.0404 |

| Training Runs | StdC1 | RC1 | StdC10 | RC10 | StdC100 | RC100 | StdC100 |
|---------------|-------|-----|--------|------|---------|-------|---------|
| 1             | 0.8263 | 0.7584 | 0.8351 | 0.869 | 0.2063 | 0.2602 | 0.2581 |
| 2             | 0.7868 | 0.5555 | 0.7936 | 0.8538 | 0.1838 | 0.2687 | 0.2659 |
| 3             | 0.7729 | 0.7686 | 0.7718 | 0.8449 | 0.1795 | 0.2803 | 0.2386 |
| 4             | 0.865 | 0.7382 | 0.8472 | 0.8692 | 0.1738 | 0.2969 | 0.2418 |
| 5             | 0.781 | 0.7775 | 0.7787 | 0.8418 | 0.1704 | 0.2733 | 0.2621 |
| Avg           | 0.806 | 0.664 | 0.805 | 0.856 | 0.183 | 0.276 | 0.253 |
| Std. dev.     | 0.0387 | 0.0945 | 0.0340 | 0.0130 | 0.0141 | 0.0138 | 0.0123 |

| Training Runs | StdAB1 | RRAB1 | StdB1 | RRAB2 | StdB2 |
|---------------|--------|-------|-------|-------|-------|
| 1             | 338.4 | 197.7 | 22951 | 1.93  | 1.62  | 2.84 |
| 2             | 380.2 | 203.6 | 22883 | 1.63  | 1.55  | 2.84 |
| 3             | 2096.2 | 206 | 31911 | 2.40  | 1.56  | 2.81 |
| 4             | 214.3 | 203.1 | 31807 | 1.52  | 1.54  | 2.84 |
| 5             | 219.9 | 200.8 | 34441 | 1.49  | 1.51  | 2.84 |
| Avg           | 649.8 | 202.54 | 28666.6 | 1.80 | 1.56 | 2.84 |
| Std. dev.     | 811.82 | 3.14 | 5653.1 | 0.382 | 0.0381 | 0.012 |
Table 7. Model Accuracy of CIFAR Tests (ResNet 110 network)

| Training Runs | Std.c1 | Ort.c1 | RR.c1 | Std.c16 | Ort.c16 | RR.c10 | Std.c1 | Ort.c10 | RR.c10 |
|---------------|--------|--------|-------|---------|---------|--------|--------|---------|--------|
| CIFAR 10      |        |        |       |         |         |        |        |         |        |
| 1             | 0.9082 | 0.9042 | 0.9076 | 0.9194 | 0.9255 | 0.9277 | 0.8275 | 0.8375 | 0.8435 |
| 2             | 0.909  | 0.9081 | 0.9026 | 0.9228 | 0.9231 | 0.9267 | 0.8315 | 0.8315 | 0.8463 |
| 3             | 0.908  | 0.9085 | 0.9028 | 0.9299 | 0.9327 | 0.9350 | 0.8305 | 0.8415 | 0.8475 |
| 4             | 0.9031 | 0.9068 | 0.9052 | 0.9176 | 0.9257 | 0.9288 | 0.8325 | 0.8395 | 0.8395 |
| 5             | 0.9104 | 0.9106 | 0.9108 | 0.9249 | 0.9259 | 0.9296 | 0.8355 | 0.8435 | 0.8475 |
| Avg.          | 0.9077 | 0.9082 | 0.9058 | 0.9209 | 0.9248 | 0.9276 | 0.8314 | 0.8387 | 0.8409 |
| Std. dev.     | 0.0028 | 0.0025 | 0.0035 | 0.0029 | 0.0012 | 0.0009 | 0.0027 | 0.0046 | 0.0067 |

Table 8. Forward and Reverse Pass Neural Networks for MNIST and peak Tests

Forward pass:
MNIST (Fig. 1): I(784) → FC(10) + b1 → O(10). peak (Fig. 2 & Fig. 4 (a)): I(784) → FC(2) + b1 → O(2).
peak (Fig. 4 (b)): I(784) → relu(BN(FC(2) + b1)) → O(2).
peak (Fig. 4 (c)): I(784) → relu(BN(FC(128) + b1) → relu(BN(FC(2) + b1)) → O(2).

Reverse pass:
MNIST (Fig. 1): I(10) → b1 → FC(784) → O(784) peak (Fig. 2 & Fig. 4 (a)): J(10) → b1 → FC(784) → O(784)
peak (Fig. 4 (b)): J(2) → b1 → FC(784) → O(784).
peak (Fig. 4 (c)): I(2) → b1 → relu(BN(FC(128))) → b1 → FC(784) → O(784).

Table 9. Forward and Reverse Pass Neural Network for MI Tests

Forward pass:
I(12) → tanh(FC(10) + b1) = d1 → tanh(FC(7) + b1) = d2 → tanh(FC(5) + b1) = d3 → tanh(FC(4) + b1) = d4 → tanh(FC(3) + b1) = d5 → tanh(FC(2) + b1) = d6 → O(12).

Reverse pass (RR.c1):
I(2) → b1 → tanh(FC(3)) → b2 → tanh(FC(4)) → b3 → tanh(FC(5)) → b4 → tanh(FC(7)) → b5 → tanh(FC(10)) → b6 → tanh(FC(12)) → O(12).

Reverse pass (RR.c1):
d2 → b1 → tanh(FC(3)) = d2' → d5 → b2 → tanh(FC(4)) = d4' → d4 → b3 → tanh(FC(5)) = d4' → d3 → b4 → tanh(FC(7)) = d5' → d5 → b5 → tanh(FC(10)) = d6' → d6 → b6 → tanh(FC(12)) = d6'.

Table 10. Hyperparameters of MI Tests

| Batch Size | 512 | Learning Rate | 0.0004 |
|------------|-----|---------------|--------|
| λ          |     |               |        |
| RR.c1:     | λ1-4 = 10^-2 |
| RR.c1:     | λ1 = 8 × 10^-3, λ2 = 6 × 10^-2, λ3 = 4 × 10^-2, λ4 = 2 × 10^-2, λ1-2 = 10^-2 |
| Training Epochs | 20000 for RR.c1/a/a | RR.c1/a/a/a, Std.c1/a/a and Std.c1 |

Table 11. Forward and Reverse Pass Neural Network for Digit Generation Tests

Forward pass:
I(74) → relu(BN(FC(1024))) → relu(BN(FC(6272))) → relu(BN(C(4,64,2))) → relu(BN(C(4,1,2))) = l_1.

Reverse pass (RR.c1): l_1 → relu(BN(D(4,64,2))) → relu(BN(D(4,128,2))) → relu(BN(FC(1024))) → relu(BN(Fc(74))) → O(74).

Table 12. Hyperparameters of Digit Generation Tests

| Batch Size | 256 | λ | λ1-2 = 2 × 10^-3, λ1-3 = 1.5 × 10^-3 |
|------------|-----|---|----------------------------------|
| Learning Rate | 0.001 for RR and Std | Training Steps | 150000 steps for RR and Std |

Table 13. Forward and Reverse Pass Network for Texture-shape Tests

Forward pass:
I(224,224,3) → relu(C(4,8,2) + b1) → relu(C(4,8,2) + b1) → relu(C(4,8,2) + b1) → relu(C(4,8,2) + b1) → relu(C(4,8,1) + b1) → l_1 → FC(16) → O(16)

Reverse pass:
l_1 → b1 → relu(D(4,8,1)) → b1 → relu(D(4,8,2)) → b1 → relu(D(4,8,2)) → b1 → relu(D(4,8,2)) → b1 → relu(D(4,8,2)) → O(224,224,3)

Table 14. Hyperparameters of Texture-shape Tests

| Batch Size | 64 | λ | 5 × 10^-1 |
|------------|----|---|-----------|
| Learning Rate | 0.0001 | Training Epochs | 200 |

28
Table 15. Forward and Reverse Pass Neural Network for MNIST Classification Tests

| Forward pass: | \( I(28,28,1) \rightarrow \text{relu}(C(3,64,1) + b_1) \rightarrow MP \rightarrow \text{relu}(C(3,64,1) + b_2) \rightarrow MP \rightarrow \text{relu}(C(3,1,1) + b_3) = I_r \rightarrow FC(10) \rightarrow O(10) \) |

| Reverse pass: | \( I_r - b_3 \rightarrow \text{relu}(D(3,64,1)) \rightarrow UP - b_2 \rightarrow \text{relu}(D(3,64,1)) \rightarrow UP - b_1 \rightarrow \text{relu}(D(3,1,1)) \rightarrow O(28,28,1) \) |

Table 16. Hyperparameters of MNIST Classification Tests

| Batch Size | \( \lambda_{t+3} \) | \( 10^{-5} \) |
|------------|-----------------|------------------|
| Learning Rate | \( 0.001 \) for \( RR_{mn}, Std_{mn} \) and \( Or_{mn} \); \( 0.0001 \) for \( RR_{mn/8\cdots8 \text{MN}}, Std_{mn/8\cdots8 \text{MN}} \) and \( Or_{mn/8\cdots8 \text{MN}} \). |
| Training Epochs | \( 100 \) for \( RR_{mn}, Std_{mn} \) and \( Or_{mn} \); \( 400 \) for \( RR_{mn/8\cdots8 \text{MN}}, Std_{mn/8\cdots8 \text{MN}} \) and \( Or_{mn/8\cdots8 \text{MN}} \). |
| \( 700 \) for \( RR_{mn}, Std_{mn} \) and \( Or_{mn} \). |

Table 17. Forward and Reverse Pass Neural Network for CIFAR Tests (Resnet 110 network)

| Example residual block in forward pass: | \( I(32,32,3) \rightarrow \cdots \rightarrow \text{BN}, \text{relu} \rightarrow d_{m+1m} \rightarrow C(3,64,1) \rightarrow d_{m+1m} \rightarrow \text{BN}, \text{relu} \rightarrow d_{m+1m} \rightarrow C(3,64,1) \rightarrow d_{m+1m} \rightarrow \cdots \) |

| Example block in reverse pass: | \( d_{m+1m} \rightarrow D(3,64,1) \rightarrow d_{m+1m} \rightarrow d_{m+1m} \rightarrow D(3,64,1) \rightarrow d_{m+1m} \rightarrow \cdots \) |

Table 18. Hyperparameters of CIFAR Tests (Resnet 110 network)

| Batch Size | \( \lambda_{t+3} \) | \( 10^{-8} \) | Training Epochs | 200 epochs |
|------------|-----------------|----------------|----------------|-----------|
| Learning Rate | \( RR_{c,1}, Or_{c,1} \) and \( Std_{c,1} = 0.01 \) |
| \( RR_{c,1}, Or_{c,1} \) and \( Std_{c,1} = 0.01 \) | (0-80 epochs), 0.001 (80-120 epochs) |
| \( 0.0001 \) (120-160 epochs), 0.0001 (160-200 epochs) |

Table 19. Forward and Reverse Pass Neural Network for CIFAR Tests (19 layers network)

| Forward pass: | \( I(32,32,3) \rightarrow \text{relu}(BN(C(3,64,1) + b_1)) \rightarrow \text{relu}(BN(C(3,64,1) + b_2)) \rightarrow MP \rightarrow \text{relu}(BN(C(3,128,1) + b_3)) \rightarrow \text{relu}(BN(C(3,256,1) + b_4)) \rightarrow \text{relu}(BN(C(3,512,1) + b_5)) \rightarrow \text{relu}(BN(C(3,1024,1) + b_6)) \rightarrow \text{relu}(BN(C(10,1) + b_7)) \rightarrow \text{relu}(BN(C(10,1) + b_8)) \rightarrow O(10) \) |

| Reverse pass: | \( I_r - b_6 \rightarrow \text{relu}(BN(C(3,512,1))) \rightarrow b_5 \rightarrow \text{relu}(BN(C(3,1024,1))) \rightarrow b_4 \rightarrow \text{relu}(BN(C(3,256,1))) \rightarrow b_3 \rightarrow \text{relu}(BN(C(3,128,1))) \rightarrow b_2 \rightarrow \text{relu}(BN(C(3,64,1))) \rightarrow b_1 \rightarrow \text{relu}(BN(C(3,1,1))) \rightarrow b_0 \rightarrow \text{relu}(BN(C(3,64,1))) \rightarrow b_1 \rightarrow \text{relu}(BN(C(3,1,1))) \rightarrow b_2 \rightarrow \text{relu}(BN(C(3,64,1))) \rightarrow b_3 \rightarrow \text{relu}(BN(C(3,1,1))) \rightarrow b_4 \rightarrow \text{relu}(BN(C(3,64,1))) \rightarrow b_5 \rightarrow \text{relu}(BN(C(3,1,1))) \rightarrow b_6 \rightarrow \text{relu}(BN(C(3,64,1))) \rightarrow O(32,32,3) \) |

Table 20. Hyperparameters of CIFAR Tests (19 layers network)

| Batch Size | \( \lambda_{t+3} \) | \( 10^{-7} \) | Training Epochs |
|------------|-----------------|----------------|----------------|
| Learning Rate | \( 0.1 \) (0 to 80 epochs); \( 0.01 \) (81 to 120 epochs); \( 0.001 \) (after 120 epochs). |
| \( 180 \) for \( RR_{c,1/c,10}, Std_{c,1/c,10} \) |
| \( 50 \) for \( RR_{c,20} \) and \( Std_{c,100} \) |

Table 21. Forward and Reverse Pass Neural Network for Smoke Tests

| Generator forward pass: | \( I(16,16,1) \rightarrow \text{relu}(C(5,64,1) + b_1) \rightarrow UP \rightarrow \text{relu}(C(5,128,1) + b_2) \rightarrow UP \rightarrow \text{relu}(C(5,128,1) + b_3) \rightarrow \text{relu}(C(5,64,1) + b_4) \rightarrow \text{relu}(C(5,32,1) + b_5) \rightarrow \text{relu}(C(5,1,1) + b_6) \rightarrow O(64,64,1) = I_r \) |

| Generator reverse pass: | \( I_r - b_1 \rightarrow \text{relu}(D(5,32,1)) \rightarrow b_2 \rightarrow \text{relu}(D(5,64,1)) \rightarrow b_3 \rightarrow \text{relu}(D(5,128,1)) \rightarrow MP \rightarrow b_4 \rightarrow \text{relu}(D(5,128,1)) \rightarrow MP \rightarrow b_5 \rightarrow \text{relu}(D(5,1,1)) \rightarrow O(16,16,1) \) |

| Discriminator: | \( I(64,64,2) \rightarrow \text{relu}(BN(C(5,32,1) + b_1)) \rightarrow \text{relu}(BN(C(5,64,1) + b_2)) \rightarrow \text{relu}(BN(C(5,128,1) + b_3)) \rightarrow FC(1) + b_3 \rightarrow O(1) \) |
Table 22. Autoencoder Architecture for Smoke Tests

| $Std_{AR}$ | $RR_{AR}$ |
|------------|-----------|
| $I(64,64,1) \rightarrow \text{relu}(C(5,32,1) + b_1) \rightarrow \text{relu}(C(5,64,1) + b_2) \rightarrow \text{relu}(C(5,128,1) + b_3)$ | $I(64,64,1) - b_1 \rightarrow \text{relu}(D(5,32,1) - b_1) \rightarrow \text{relu}(D(5,64,1) - b_2) \rightarrow \text{relu}(D(5,128,1) - b_3)$ |
| $\rightarrow \text{relu}(C(5,128,1) + b_4) \rightarrow MP \rightarrow \text{relu}(C(5,64,1) + b_5) \rightarrow MP \rightarrow \text{relu}(C(5,128,1) + b_6)$ | $\rightarrow \text{relu}(D(5,64,1) + b_7) \rightarrow UP \rightarrow \text{relu}(D(5,128,1) + b_8) \rightarrow UP \rightarrow \text{relu}(D(5,128,1) + b_9)$ |
| $\rightarrow \text{relu}(C(5,64,1) + b_{10}) \rightarrow \text{relu}(C(5,32,1) + b_{11}) \rightarrow \text{relu}(C(5,128,1) + b_{12}) \rightarrow O(64,64,1)$. | $\rightarrow \text{relu}(C(5,64,1) + b_{12}) \rightarrow \text{relu}(C(5,32,1) + b_{13}) \rightarrow \text{relu}(C(5,3,1) + b_{12}) \rightarrow O(64,64,3)$.

$Std_{AB}$ and $RR_{AB}$:

| $I(64,64,1) \rightarrow \text{relu}(D(5,32,1)) - b_1 \rightarrow \text{relu}(D(5,64,1)) - b_2 \rightarrow \text{relu}(D(5,128,1)) - b_3$ | $I(64,64,3) \rightarrow \text{relu}(C(5,32,1) + b_1) \rightarrow \text{relu}(C(5,64,1) + b_2) \rightarrow \text{relu}(C(5,128,1) + b_3)$ |
| $\rightarrow \text{relu}(C(5,128,1) - b_4) \rightarrow MP - b_4 \rightarrow \text{relu}(D(5,64,1)) - b_5 \rightarrow MP - b_5 \rightarrow \text{relu}(D(5,128,1))$ | $\rightarrow \text{relu}(C(5,64,1) + b_6) \rightarrow UP - b_6 \rightarrow \text{relu}(C(5,128,1) + b_7) \rightarrow UP - b_7 \rightarrow \text{relu}(C(5,128,1) + b_8)$ |
| $\rightarrow \text{relu}(C(5,64,1) + b_{10}) \rightarrow \text{relu}(C(5,32,1) + b_{11}) \rightarrow \text{relu}(C(5,128,1) + b_{12}) \rightarrow O(64,64,1)$. | $\rightarrow \text{relu}(C(5,64,1) + b_{12}) \rightarrow \text{relu}(C(5,32,1) + b_{12}) \rightarrow \text{relu}(C(5,3,1) + b_{12}) \rightarrow O(64,64,3)$.

Table 23. Hyperparameters of Smoke Tests

| Batch Size | 64 | Learning Rate | 0.0002 | $\lambda$ | $\lambda_1 = 10; \lambda_{2-6} = 0.1$ |
|------------|----|---------------|--------|----------|-------------------------------|
| Training Epocs | 40000 for $RR_{A}$ and $Std_{A}$; 1000 for $RR_{AB}, RR_{BA}, Std_{AB}$ and $Std_{AB}$. |

Table 24. Forward and Reverse Pass Neural Network for VGG19 Network

Forward pass:

| $I(224,224,3) \rightarrow \text{relu}(C(3,32,1) + b_1) \rightarrow \text{relu}(C(3,64,1) + b_2) \rightarrow MP \rightarrow \text{relu}(C(3,128,1) + b_3)$ | $I(224,224,3) \rightarrow \text{relu}(D(3,32,1) - b_1) \rightarrow \text{relu}(D(3,64,1) - b_2) \rightarrow MP \rightarrow \text{relu}(D(3,128,1) - b_3)$ |
| $\rightarrow \text{relu}(C(3,128,1) + b_4) \rightarrow MP \rightarrow \text{relu}(C(3,256,1) + b_5) \rightarrow \text{relu}(C(3,256,1) + b_{10})$ | $\rightarrow \text{relu}(D(3,256,1) - b_6) \rightarrow MP \rightarrow \text{relu}(D(3,256,1) - b_{10})$ |
| $\rightarrow \text{relu}(C(3,512,1) + b_{11}) \rightarrow \text{relu}(C(3,256,1) + b_{12}) \rightarrow MP \rightarrow \text{relu}(C(3,512,1) + b_{12})$ | $\rightarrow \text{relu}(D(3,128,1) - b_8) \rightarrow \text{relu}(D(3,256,1) - b_{12}) \rightarrow \text{relu}(D(3,512,1) - b_{12})$ |
| $\rightarrow \text{relu}(C(3,512,1) + b_{12}) \rightarrow \text{relu}(C(3,512,1) + b_{12}) = \text{lr} \rightarrow MP \rightarrow \text{relu}(FC(4096) + b_{17}) \rightarrow \text{relu}(FC(4096) + b_{19})$ | $\rightarrow \text{relu}(FC(1000) + b_{19}) \rightarrow O(1000)$ |
| $\rightarrow \text{relu}(FC(1000) + b_{19}) \rightarrow O(1000)$. | $\rightarrow \text{relu}(FC(1000) + b_{19}) \rightarrow O(1000)$. |

Reverse pass:

| $I(224,224,3) \rightarrow \text{relu}(D(3,128,1)) - b_1 \rightarrow \text{relu}(D(3,256,1)) - b_{12} \rightarrow \text{relu}(D(3,512,1)) - b_{12}$ | $I(224,224,3) \rightarrow \text{relu}(C(3,32,1)) - b_1 \rightarrow \text{relu}(C(3,64,1)) - b_2 \rightarrow \text{relu}(C(3,128,1)) - b_3$ |
| $\rightarrow \text{relu}(C(3,256,1) + b_{12}) \rightarrow \text{relu}(C(3,512,1) + b_{12}) \rightarrow \text{relu}(C(3,512,1) + b_{12})$ | $\rightarrow \text{relu}(D(3,128,1)) - b_8 \rightarrow \text{relu}(D(3,256,1) - b_{12}) \rightarrow \text{relu}(D(3,512,1) - b_{12})$ |
| $\rightarrow \text{relu}(D(3,256,1)) - b_6 \rightarrow \text{relu}(D(3,256,1)) - b_{12}$ | $\rightarrow \text{relu}(D(3,128,1)) - b_8 \rightarrow \text{relu}(D(3,256,1)) - b_{12}$ |
| $\rightarrow \text{relu}(D(3,256,1) + b_{12}) \rightarrow \text{relu}(D(3,256,1) + b_{12}) \rightarrow \text{relu}(D(3,512,1) + b_{12}) \rightarrow \text{relu}(D(3,512,1) + b_{12})$ | $\rightarrow \text{relu}(D(3,128,1)) - b_8 \rightarrow \text{relu}(D(3,256,1)) - b_{12}$ |
| $\rightarrow \text{relu}(D(3,512,1)) - b_7 \rightarrow \text{relu}(D(3,128,1)) - b_8$ | $\rightarrow \text{relu}(D(3,128,1)) - b_8 \rightarrow \text{relu}(D(3,256,1)) - b_{12}$ |
| $\rightarrow \text{relu}(D(3,64,1)) - b_{12} \rightarrow \text{relu}(D(3,64,1)) - b_{12}$ | $\rightarrow \text{relu}(D(3,64,1)) - b_{12}$ |
| $\rightarrow \text{relu}(D(3,64,1) + b_{12}) \rightarrow \text{relu}(D(3,64,1) + b_{12}) \rightarrow \text{relu}(D(3,3,1) + b_{12}) \rightarrow \text{relu}(D(3,3,1) + b_{12})$ | $\rightarrow \text{relu}(D(3,64,1) + b_{12}) \rightarrow \text{relu}(D(3,64,1) + b_{12})$ |
| $\rightarrow O(224,224,3)$. | $\rightarrow O(224,224,3)$. |

Table 25. Hyperparameters of VGG19 Training

| $\lambda_1\ldots\lambda_{16}$ | $10^{-10}$ |
|--------------------------|-----------|
| Learning Rate | 0.01 (0 to 7 epochs); 0.001 (7 to 10 epochs); 0.0001 (after 15 epochs). |
| Training Epochs | 36 epochs for $RR_{VGG}$ and $Std_{VGG}$; 22 epochs for $RR$ and $Std$. |
Table 26. \( I(X; D_m) \) and \( I(D_m; Y) \) Values of All MI Models

| Results of 5 Runs | \( I(X; D_h) \) | \( I(X; D_k) \) | \( I(X; D_s) \) | \( I(X; D_h) \) | \( I(D_h; Y) \) | \( I(D_k; Y) \) | \( I(D_s; Y) \) | \( I(D_h; Y) \) | \( I(D_k; Y) \) | \( I(D_s; Y) \) |
|-------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| Std. dev. 0.0011  | 0.0138         | 0.2007         | 1.5367         | 0.8695         | 0.0973         | 0.0000         | 0.0000         | 0.0000         | 0.0012         | 0.0031         |
| Std. dev. 0.0558  | 2.3736         | 0.1872         | 0.2033         | 0.0909         | 0.0070         | 0.0334         | 0.0244         | 0.0233         | 0.0231         | 0.0236         |
| Std. dev. 0.0454  | 0.1569         | 0.3115         | 0.1279         | 0.0783         | 0.0173         | 0.0387         | 0.0337         | 0.0311         | 0.0060         | 0.0115         |
| Std. dev. 0.0051  | 0.0505         | 0.5529         | 0.8500         | 0.4071         | 0.0629         | 0.0000         | 0.0000         | 0.0004         | 0.0128         | 0.0254         |
| Std. dev. 0.0021  | 0.0292         | 0.2305         | 1.2948         | 0.3707         | 0.1700         | 0.0000         | 0.0004         | 0.0009         | 0.0080         | 0.0169         |
| Std. dev. 0.0016  | 0.2075         | 0.9404         | 1.8047         | 1.3972         | 0.4193         | 0.0000         | 0.0118         | 0.0217         | 0.0374         | 0.0420         |
| Std. dev. 0.0527  | 10.2473        | 8.1597         | 6.7212         | 2.4402         | 1.1113         | 0.9992         | 0.9743         | 0.9662         | 0.9917         | 0.9859         |
| Std. dev. 0.0472  | 1.6567         | 2.3965         | 1.8774         | 0.4665         | 0.0512         | 0.0000         | 0.0227         | 0.0124         | 0.0047         | 0.0110         |
| Avg. 11.9200      | 11.5021        | 10.6095        | 7.3398         | 2.2372         | 1.0416         | 0.9992         | 0.9988         | 0.9985         | 0.9975         | 0.9906         |
| Std. dev. 0.0540  | 0.1463         | 0.5485         | 0.8093         | 0.1004         | 0.0505         | 0.0000         | 0.0005         | 0.0007         | 0.0024         | 0.0073         |
| Avg. 11.9970      | 11.9727        | 11.3343        | 6.1690         | 1.9642         | 1.0985         | 0.9992         | 0.9991         | 0.9971         | 0.9832         | 0.9625         |
| Std. dev. 0.0861  | 0.0804         | 0.6796         | 1.2529         | 0.5865         | 0.0891         | 0.0000         | 0.0002         | 0.0032         | 0.0166         | 0.0209         |
| Avg. 11.9897      | 11.9794        | 10.2983        | 3.0652         | 1.2677         | 0.5283         | 0.9992         | 0.9988         | 0.9829         | 0.4041         | 0.3923         |
| Std. dev. 0.0019  | 0.0116         | 1.5095         | 4.0345         | 1.8717         | 0.7445         | 0.0000         | 0.0006         | 0.0918         | 0.5359         | 0.5329         |
| Avg. 11.9905      | 11.3992        | 12.7860        | 1.4953         | 1.2139         | 0.5122         | 0.9992         | 0.9991         | 0.9994         | 0.9861         | 0.9842         |
| Std. dev. 0.0075  | 0.1710         | 1.0207         | 1.4405         | 1.2750         | 0.1537         | 0.0000         | 0.0131         | 0.0310         | 0.0357         | 0.0439         |
| Avg. 11.9847      | 10.8970        | 8.3860         | 5.4257         | 1.7110         | 1.0694         | 0.9989         | 0.9915         | 0.9813         | 0.9929         | 0.9812         |
| Std. dev. 0.0890  | 0.8474         | 1.9089         | 1.1511         | 0.4913         | 0.0529         | 0.0006         | 0.0089         | 0.0145         | 0.0085         | 0.0209         |
| Avg. 11.9093      | 11.5761        | 9.4227         | 5.6190         | 2.2286         | 1.0100         | 0.9992         | 0.9992         | 0.9984         | 0.9898         | 0.9862         |
| Std. dev. 0.0957  | 0.1450         | 1.7109         | 1.4609         | 0.4611         | 0.0091         | 0.0000         | 0.0000         | 0.0014         | 0.0004         | 0.0032         |
| Avg. 11.9946      | 11.9546        | 8.9865         | 3.2802         | 1.4296         | 0.8571         | 0.9992         | 0.9985         | 0.8048         | 0.7851         | 0.7756         |
| Std. dev. 0.0104  | 0.0861         | 3.3571         | 2.7102         | 0.9848         | 0.4858         | 0.0000         | 0.0007         | 0.4279         | 0.4391         | 0.4342         |
| Avg. 12.0000      | 11.9913        | 11.7602        | 8.6184         | 3.2490         | 1.5171         | 0.9992         | 0.9990         | 0.9705         | 0.9880         | 0.9458         |
| Std. dev. 0.0000  | 0.0113         | 0.3400         | 2.0542         | 1.2740         | 0.2025         | 0.0000         | 0.0003         | 0.0007         | 0.0114         | 0.0476         |
Table 30. Performance Comparison of Texture-shape Models

| Model Runs | airplane | bear | bicycle | bird | boat | bottle | car | cat | chair | clock | dog | elephant | keyboard | knife | oven | truck | Acc. | Acc. | Acc. |
|------------|----------|------|---------|------|------|--------|-----|-----|-------|-------|-----|------------|---------|-------|------|------|------|------|
|            |          |      |         |      |      |        |     |     |       |       |     |            |         |       |      |      |      |      |
| Std. dev   | 0.3742   | 0.3408 | 0.2898  | 0.2459 | 0.3438 | 0.4175 | 0.3367 | 0.2741 | 0.4088 | 0.4029 | 0.2378 | 0.2283 | 0.3204 | 0.3143 | 0.3134 | 0.4600 | 0.0552 | 0.0746 | 0.0373 |
| Avg.       | 0.61     | 0.25  | 0.37    | 0.52  | 0.4   | 0.45   | 0.5   | 0.38  | 0.41  | 0.71   | 0.4   | 0.39    | 0.52    | 0.21  | 0.54  | 0.44  | 0.4420 | 0.1575 | 0.3306 |

Table 29. Performance Accuracy of the Proposed Method for Texture-shape Tests

| Model Runs | airplane | bear | bicycle | bird | boat | bottle | car | cat | chair | clock | dog | elephant | keyboard | knife | oven | truck | Acc. | Acc. | Acc. |
|------------|----------|------|---------|------|------|--------|-----|-----|-------|-------|-----|------------|---------|-------|------|------|------|------|
|            |          |      |         |      |      |        |     |     |       |       |     |            |         |       |      |      |      |      |
| Std. dev   | 0.2923   | 0.2767 | 0.4566  | 0.2685 | 0.2981 | 0.5007 | 0.4000 | 0.3801 | 0.3712 | 0.3592 | 0.2173 | 0.4756 | 0.3929 | 0.3204 | 0.3596 | 0.4035 | 0.0604 | 0.0438 | 0.0481 |
| Avg.       | 0.79     | 0.71  | 0.52    | 0.61  | 0.4   | 0.58   | 0.5   | 0.5   | 0.3   | 0.67   | 0.35  | 0.42    | 0.51    | 0.56  | 0.66  | 0.75  | 0.5467 | 0.2369 | 0.4075 |

Table 28. Performance Accuracy of the Proposed Method for Texture-shape Tests

| Model Runs | airplane | bear | bicycle | bird | boat | bottle | car | cat | chair | clock | dog | elephant | keyboard | knife | oven | truck | Acc. | Acc. | Acc. |
|------------|----------|------|---------|------|------|--------|-----|-----|-------|-------|-----|------------|---------|-------|------|------|------|------|
|            |          |      |         |      |      |        |     |     |       |       |     |            |         |       |      |      |      |      |
| Std. dev   | 0.4554   | 0.1438 | 0.2688  | 0.4216 | 0.1438 | 0.2813 | 0.4442 | 0.1188 | 0.2813 | 0.4554 | 0.1438 | 0.2688  | 0.4216  | 0.1438 | 0.2813 | 0.4442 | 0.1188 | 0.2813 | 0.4554 |
| Avg.       | 0.5228   | 0.1688 | 0.3875  | 0.4402 | 0.2125 | 0.3688 | 0.5213 | 0.1813 | 0.3688 | 0.5213 | 0.1813 | 0.3688  | 0.5213  | 0.1813 | 0.3688 | 0.5213 | 0.1813 | 0.3688 | 0.5213 |

Table 27. Performance Accuracy of the Proposed Method for Texture-shape Tests

| Model Runs | airplane | bear | bicycle | bird | boat | bottle | car | cat | chair | clock | dog | elephant | keyboard | knife | oven | truck | Acc. | Acc. | Acc. |
|------------|----------|------|---------|------|------|--------|-----|-----|-------|-------|-----|------------|---------|-------|------|------|------|------|
|            |          |      |         |      |      |        |     |     |       |       |     |            |         |       |      |      |      |      |
| Std. dev   | 0.387    | 0.442 | 0.9     | 0     | 0.7   | 1      | 0   | 0   | 0     | 0.9   | 0   | 0.9       | 0       | 1     | 0    | 0    | 0    | 0    |
| Avg.       | 0.64     | 0.545 | 0.4     | 0.6   | 0.5   | 0.5    | 0   | 0   | 0     | 0.9   | 0   | 0.9      | 0       | 1     | 0    | 0    | 0    | 0    |

Table 26. Performance Accuracy of the Proposed Method for Texture-shape Tests

| Model Runs | airplane | bear | bicycle | bird | boat | bottle | car | cat | chair | clock | dog | elephant | keyboard | knife | oven | truck | Acc. | Acc. | Acc. |
|------------|----------|------|---------|------|------|--------|-----|-----|-------|-------|-----|------------|---------|-------|------|------|------|------|
|            |          |      |         |      |      |        |     |     |       |       |     |            |         |       |      |      |      |      |
| Std. dev   | 0.4442   | 0.1188 | 0.2813  | 0.4216 | 0.1438 | 0.2688 | 0.4442 | 0.1188 | 0.2813 | 0.4554 | 0.1438 | 0.2688  | 0.4216  | 0.1438 | 0.2813 | 0.4442 | 0.1188 | 0.2813 | 0.4554 |
| Avg.       | 0.5228   | 0.1688 | 0.3875  | 0.4402 | 0.2125 | 0.3688 | 0.5213 | 0.1813 | 0.3688 | 0.5213 | 0.1813 | 0.3688  | 0.5213  | 0.1813 | 0.3688 | 0.5213 | 0.1813 | 0.3688 | 0.5213 |