Neural Computing and Applications (2023) 35:4587–4619
https://doi.org/10.1007/s00521-022-07892-0

ORIGINAL ARTICLE

Reviving autoencoder pretraining

You Xie[1] · Nils Thuerey[1]

Received: 12 December 2021 / Accepted: 23 September 2022 / Published online: 26 October 2022

Abstract
The pressing need for pretraining algorithms has been diminished by numerous advances in terms of regularization, architectures, and optimizers. Despite this trend, we re-visit the classic idea of unsupervised autoencoder pretraining and propose a modified variant that relies on a full reverse pass trained in conjunction with a given training task. This yields networks that are *as-invertible-as-possible* and share mutual information across all constrained layers. We additionally establish links between singular value decomposition and pretraining and show how it can be leveraged for gaining insights about the learned structures. Most importantly, we demonstrate that our approach yields an improved performance for a wide variety of relevant learning and transfer tasks ranging from fully connected networks over residual neural networks to generative adversarial networks. Our results demonstrate that unsupervised pretraining has not lost its practical relevance in today's deep learning environment.

Keywords Unsupervised pretraining · Greedy layer-wise pretraining · Transfer learning · Orthogonality

1 Introduction

While approaches such as greedy layer-wise autoencoder pretraining [4, 18, 72, 78] paved the way for many fundamental concepts of today's methodologies in deep learning, the pressing need for pretraining neural networks has been diminished in recent years. An inherent problem is the lack of a global view: layer-wise pretraining is limited to adjusting individual layers one at a time. Thus, bottom layers that are optimized first cannot be adjusted to correct errors in higher layers [11, 87]. In addition, numerous advancements in regularization [28, 43, 66, 76], network architectures [30, 63, 71], and improved optimization algorithms [44, 52, 62] have decreased the demand for layer-wise pretraining. Despite these advances, training deep neural networks that generalize well to a wide range of previously unseen tasks remains a fundamental challenge [20, 40, 55, 56] (Fig. 1).

In this paper, we develop an algorithm that reformulates autoencoder pretraining in a global way to arrive at a method that efficiently extracts general, dominant features from datasets. These features in turn improve performance for new tasks. Our approach is also inspired by techniques for orthogonalization [3, 38, 50, 57]. Hence, we propose a modified variant that relies on a full reverse pass trained in conjunction with a given training task. A key insight is that there is no need for "greediness," i.e., layer-wise decompositions of the network structure, and it is additionally beneficial to take into account a specific problem domain at the time of pretraining. We establish links between singular value decomposition (SVD) and pretraining, and show how our approach yields an embedding of problem-aware dominant features in the weight matrices. An SVD can then be leveraged to conveniently gain insights about learned structures. Unlike orthogonalization techniques, we focus on embedding the dominant features of a dataset into the weights of a network. This is achieved via a *reverse pass* network. This reverse pass is generic, simple to construct, and directly relates to model performance, instead of, e.g., constraining the orthogonality of weights. Most importantly, we demonstrate that the proposed pretraining yields an improved performance for a variety of learning and transfer tasks, while incurring only a minor extra computational cost from the reverse pass.

✉ Nils Thuerey
nils.thuerey@tum.de

You Xie
you.xie@tum.de

[1] Department of Informatics, Technical University of Munich, Boltzmannstr. 3, 85748 Garching, Germany

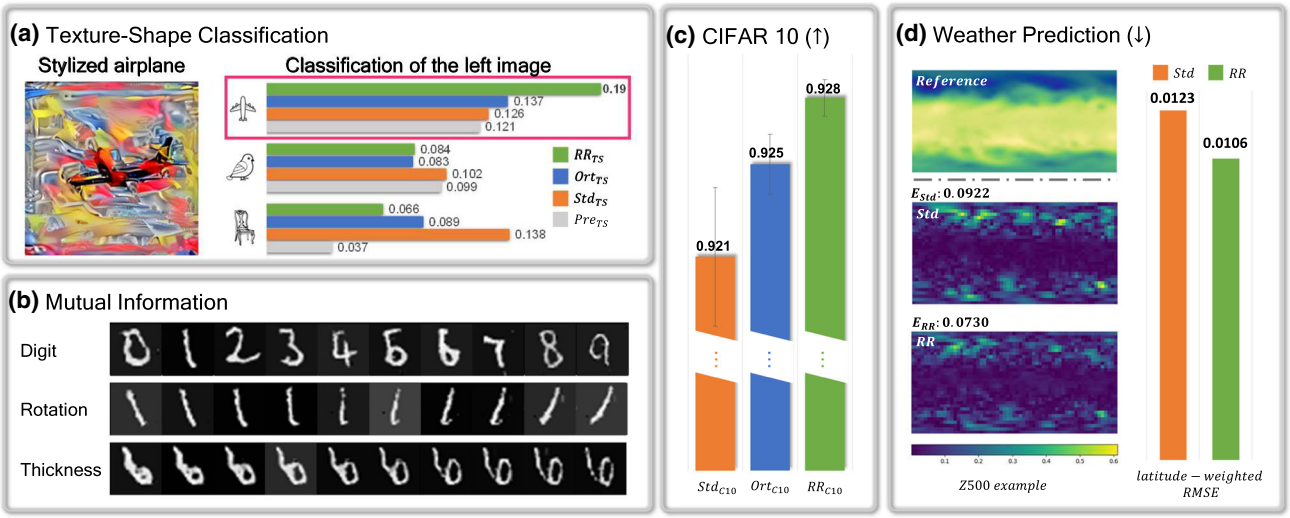

Fig. 1 Our pretraining (denoted as RR) yields improvements for numerous applications: **a** For difficult shape classification tasks, it outperforms existing approaches (Std_{TS}, Ort_{TS}, Pre_{TS}): the RR_{TS} model classifies the airplane shape with significantly higher confidence. **b** Our approach establishes mutual information between input and output distributions. **c** For CIFAR 10 classification with a ResNet 110, RR_{C10} yields substantial practical improvements over the state-of-the-art. **d** Learned weather forecasting likewise benefits from our pretraining, with RR yielding 13.7% improvements in terms of latitude-weighted RMSE for the ERA dataset [31]. Pressure is shown for 2019-08-09, 22:00 UTC, together with Mean Absolute Error (MAE) for Std and RR models

The structure of our networks is influenced by invertible network architectures that have received significant attention in recent years [24, 34, 36, 85]. However, these approaches rely heavily on specific network architectures. 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, so that more dominant features of the input dataset are embedded into the networks. In contrast to most structures for invertible networks, our approach does not impose architectural restrictions. We demonstrate the benefits of our pretraining for a variety of architectures, from fully connected layers to convolutional neural networks (CNNs) [46], over networks with batch normalization or dropout regularization, to generative adversarial networks (GAN) architectures [25].

Below, we will first give an overview of our formulation and its connection to singular values, before evaluating our model 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 optimal performance for one specific goal. During a regular training run, the network naturally exploits any observed correlations between input and output distribution. An inherent difficulty in this setting is that typically no knowledge about the specifics of the new data and task domains is available when training the source model. Hence, it is common practice to target broad and difficult tasks hoping that this will result in features that are applicable in new domains [14, 26, 82]. Motivated by autoencoder pretraining, we instead leverage a pretraining approach that takes into account the data distribution of the inputs and drives the network to extract dominant features from the datasets, which differs from regular training for optimal performance of one specific goal. We demonstrate that our approach boosts the model accuracy for original and new tasks for a wide range of applications, from image classification to data-driven weather forecasting.

2 Related work

Greedy layer-wise pretraining was first proposed by Bengio et al. [4], and influenced a large number of follow-up works, providing a crucial method for feature extraction and enabling stable training runs of deeper networks. A detailed evaluation was performed by Erhan et al. [18], also highlighting cases where it can be detrimental. These problems were later on detailed in other works [1]. Principal component analysis (PCA) [29, 77] is a popular approach for dimensionality reduction and feature extraction, and was proposed to, e.g., handle nonlinear relationships between variables [33, 51], separate interpretable components [5], and improve robustness in

the presence of outliers [80]. However, PCA is computationally intensive in both memory and run time for larger dataset. Clustering is another popular alternative [6, 22, 65, 84, 89]. As these methods rely on data similarities, they yield a high complexity when the dataset size increases [7]. Sharing similarities with our approach, Rasmus et al. [58] combined supervised and unsupervised learning objectives, but focused on denoising autoencoders and a layer-wise approach without weight sharing.

Unsupervised approaches for representation learning [23, 37, 42, 48, 81], especially contrastive learning, such as SimCLR [8], MoCo-v2 [10], ProtoNCE [49], and PaCo [13], similarly aim for learning generic features from a given dataset, but typically necessitate sophisticated training algorithms. We demonstrate the importance of leveraging state-of-the-art methods for training deep networks, i.e., without decomposing or modifying the network structure. This not only improves performance, but also very significantly simplifies the adoption of the pretraining pass in new application settings.

Extending the classic viewpoint of unsupervised autoencoder pretraining, regularization techniques have also been commonly developed to improve the properties of neural networks [45, 47]. Several prior methods employed "hard orthogonal constraints" to improve weight orthogonality via SVD at training time [35, 38, 57]. Bansal et al. [3] additionally investigated efficient formulations of the orthogonality constraints. Orthogonal convolutional neural networks (OCNN) [75] reformulate the orthogonality constraints to be computed efficiently for networks convolutional layers. In practice, these constraints are difficult to satisfy, and correspondingly only weakly imposed. In addition, all of 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 [70]. While our approach 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.

Recovering all input information from hidden representations of a network is generally very difficult [15, 53, 54], due to the loss of information throughout the layer transformations. In this context, [69] 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 [2, 24, 36, 39], 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 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 [25]. While methods using reverse connections were previously proposed [67, 85], these modules primarily focus on transferring information between layers for a given task, and on autoencoder structures for domain adaptation, respectively.

3 Method

With state-of-the-art deep learning methods [27, 88], there is no need for breaking down the training process into single layers. Hence, we consider approaches that target whole networks, and employ orthogonalization regularizers as a starting point [35]. Orthogonality constraints were shown to yield improved training performance in various settings [3], and for an n-layer network, they can be formulated as:

$$\mathcal{L}_{\text{ort}} = \sum_{m=1}^{n} \left\| M_m^T M_m - I \right\|_F^2, \tag{1}$$

i.e., enforcing the transpose of the weight matrix $M_m \in \mathbb{R}^{s_m^{\text{out}} \times s_m^{\text{in}}}$ for all layers m to yield its inverse when being multiplied with the original matrix. I denotes the identity matrix with $I = (\mathbf{e}_m^1, \ldots \mathbf{e}_m^{s_m^{\text{in}}})$, $\mathbf{e}_m^j$ denoting the j_{th} column unit vector. Theoretically, $\mathcal{L}_{\text{ort}} = \mathbf{0}$ cannot be perfectly fulfilled because of the information imbalance between inputs and outputs in most deep learning cases [69]. We will first analyze the influence of the loss function $\mathcal{L}_{\text{ort}}$ assuming that it can be fulfilled, before applying the analysis to our full pretraining method.

Minimizing Eq. (1), i.e., $M_m^T M_m - I = \mathbf{0}$ is mathematically equivalent to:

$$M_m^T M_m \mathbf{e}_m^j - \mathbf{e}_m^j = \mathbf{0}, j = 1, 2, \ldots, s_m^{\text{in}}, m = 1, 2, \ldots, n, \tag{2}$$

with $rank(M_m^T M_m) = s_m^{\text{in}}$, and $\mathbf{e}_m^j$ 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. Instead, we will design a constraint that jointly considers data and the trainable weights, allowing us to learn the dominant features of the training dataset directly. We naturally would like to recover *all* the features of the dataset with a learning task, but finite network capacity makes this infeasible in practice. Instead, we aim for extracting the features that

contribute the most in order to achieve a minimum loss value for our designed constraint. As a result, the features that appear the most, i.e., dominant features, will be extracted. In this section, we will introduce our constraint and analysis how it guides the weights to learn dominant features from the dataset. Then, we will illustrate how we insert our constraint into training with a reversed pass network.

Inspired by the classical unsupervised pretraining, we reformulate the orthogonality constraint in a *data-driven* manner to take into account the set of inputs $\mathcal{D}_m$ for the current layer (either activation from a previous layer or the training data $\mathcal{D}_1$), and instead minimize

$$\begin{aligned}\mathcal{L}_{\text{RR}} &= \sum_{m=1}^{n} \left\| M_m^T M_m \mathbf{d}_m^i - \mathbf{d}_m^i \right\|_2^2 \\ &= \sum_{m=1}^{n} \left\| (M_m^T M_m - I)\mathbf{d}_m^i \right\|_2^2,\end{aligned} \tag{3}$$

where $\mathbf{d}_m^i \in \mathcal{D}_m \subset \mathbb{R}^{s_m^{\text{in}}}$. Due to its reversible nature, we will denote our approach with an RR subscript in the following. In contrast to classical autoencoder pretraining, we are minimizing this loss jointly for all layers of a network, and while orthogonality only focuses on M_m, our formulation allows for minimizing the loss by extracting the dominant features of the input data.

Let q denotes the number of linearly independent entries in $\mathcal{D}_m$, i.e., its dimension, and t the size of the training data, i.e., $\mathcal{D}_m = t$, usually with $q < t$. For every single datum $\mathbf{d}_m^i, i = 1, 2, \ldots, t$, Eq. (3) results in

$$M_m^T M_m \mathbf{d}_m^i - \mathbf{d}_m^i = \mathbf{0}, m = 1, 2, \ldots, n, \tag{4}$$

and hence $\mathbf{d}_m^i$ 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.

The result of applying layer m of a network represents the features extracted this layer via its weight matrix M_m. The singular vectors of the SVD of M_m, can be regarded as input filters, and we can thus analyze the result of M_m by focusing on its singular vectors. We employ SVD to identify what features are extracted by the parameters in M_m. As by construction, $rank(M_m) = r \leqslant min(s_m^{\text{in}}, s_m^{\text{out}})$, the SVD of M_m yields:

$$\begin{aligned} &M_m = U_m \Sigma_m V_m^T, m = 1, 2, \ldots, n, \\ &\text{with} \begin{cases} U_m = (\mathbf{u}_m^1, \mathbf{u}_m^2, \ldots, \mathbf{u}_m^r, \mathbf{u}_m^{r+1}, \ldots, \mathbf{u}_m^{s_m^{\text{out}}}) \in \mathbb{R}^{s_m^{\text{out}} \times s_m^{\text{out}}}, \\ V_m = (\mathbf{v}_m^1, \mathbf{v}_m^2, \ldots, \mathbf{v}_m^r, \mathbf{v}_m^{r+1}, \ldots, \mathbf{v}_m^{s_m^{\text{in}}}) \in \mathbb{R}^{s_m^{\text{in}} \times s_m^{\text{in}}}, \end{cases} \end{aligned} \tag{5}$$

with left and right singular vectors in U_m and V_m, respectively, and Σ_m having square roots of the r eigenvalues of $M_m^T M_m$ on its diagonal. $\mathbf{u}_m^k$ and $\mathbf{v}_m^k (k = 1, \ldots, r)$ are the eigenvectors of $M_m M_m^T$ and $M_m^T M_m$, respectively [73]. Here, especially the right singular vectors in V_m^T 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 $\mathbf{e}_m^j$ 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 dataset. For this, we consider an arbitrary basis for spanning the space of inputs $\mathcal{D}_m$ for layer m. Let $\mathcal{B}_m : \langle \mathbf{w}_m^1, \ldots, \mathbf{w}_m^q \rangle$ denote a set of q orthonormal basis vectors obtained via a Gram–Schmidt process, with $t \geqslant q \geqslant r$, and D_m denoting the matrix of the vectors in $\mathcal{B}_m$. As we show in more detail in Appendix, our constraint from Eq. (4) requires eigenvectors of $M_m^T M_m$ to be $\mathbf{w}_m^i$, with V_m containing r orthogonal vectors $(\mathbf{v}_m^1, \mathbf{v}_m^2, \ldots, \mathbf{v}_m^r)$ from $\mathcal{D}_m$ and $(s_m^{\text{in}} - r)$ 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 $\mathbf{d}_m^i$ can be represented as a linear combination via a vector of coefficients $\mathbf{c}_m^i$ that multiplies D_m so that $\mathbf{d}_m^i = D_m \mathbf{c}_m^i$. Since $M_m \mathbf{d}_m = U_m \Sigma_m V_m^T \mathbf{d}_m$, the loss $\mathcal{L}_{\text{RR}}$ of layer m can be rewritten as

$$\begin{aligned}\mathcal{L}_{\text{RR}_\text{m}} &= \left\| M_m^T M_m \mathbf{d}_m - \mathbf{d}_m \right\|_2^2 \\ &= \left\| V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{d}_m - \mathbf{d}_m \right\|_2^2 \\ &= \left\| V_m \Sigma_m^T \Sigma_m V_m^T D_m \mathbf{c}_m - D_m \mathbf{c}_m \right\|_2^2, m = 1, 2, \ldots, n,\end{aligned} \tag{6}$$

where we can assume that the coefficient vector $\mathbf{c}_m$ is accumulated over the training dataset size t via $\mathbf{c}_m = \sum_{i=1}^{t} \mathbf{c}_m^i$, since eventually every single datum in $\mathcal{D}_m$ will contribute to $\mathcal{L}_{\text{RR}_\text{m}}$. The central component of Eq. (6) is $V_m^T D_m$. For a successful minimization, V_m needs to retain

those $\mathbf{w}_m^i$ with the largest $\mathbf{c}_m$ 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 $\mathbf{c}_m$ in order to establish a small distance to $D_m\mathbf{c}_m$. Thus, features that appear most in the input data with a corresponding factor in $\mathbf{c}_m$ will more strongly contribute to minimizing $\mathcal{L}_{\mathrm{RR_m}}$. Above, D_m is only used implicitly to analyze different approaches, and we do not specify any explicit requirements for D_m. Since a fixed dataset determines the corresponding D_m, different orthogonal decompositions via Gram–Schmidt lead to different orthonormal bases. However, these different orthonormal bases can be aligned via rotation, and all span the same vector space. Thus, regardless of the particular orthonormal basis that is used, our method always focuses on extracting dominant features that appear most frequently in the dataset. This means the components of D_m which contribute most to minimizing the loss will be embedded in the neural network. More in-depths discussions are provided in Appendix A.3.

Comparing our constraint from Eq. (3) with the orthogonal constraint in Eq. (1), we can see that our formulation is actually stricter. As a consequence, our method can retain the advantages of orthogonal constraints while simultaneously embedding dominant features into the weight matrices.

To summarize, V_m is driven toward containing r orthogonal vectors $\mathbf{w}_m^i$ 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. For the sake of being distinct and representative of the dataset, 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.

3.1 Realization in neural networks

Calculating $M_m^T M_m \mathbf{d}_m^i$ in Eq. (3) directly is usually very expensive due to the dimensionality of M_m. Instead, we reuse $M_m\mathbf{d}_m^i$ in the forward pass network and build an extra reverse pass network to calculate $M_m^T M_m \mathbf{d}_m^i$ by reusing parameters from the forward pass network. In the following, we will explain how to constrain the intermediate results of the network to efficiently realize Eq. (3) when training.

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. For instance, for a typical fully connected layer in the forward pass with $\mathbf{d}_{m+1} = M_m\mathbf{d}_m + \mathbf{b}_m$, the reverse pass operation is given by $\mathbf{d}_m^{'} = M_m^T(\mathbf{d}_{m+1} - \mathbf{b}_m)$, where $\mathbf{d}_m^{'}$ 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. 2. While the construction of the reverse pass is straightforward for all standard operations, i.e., fully connected layers, convolutions, pooling, etc., slight adjustments are necessary for nonlinear activation functions (AFs) and batch normalization (BN). It is crucial for our formulation that $\mathbf{d}_m$ and $\mathbf{d}_m^{'}$ 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 activation of layer $m-1$ from the forward pass, i.e., f_{m-1} and BN_{m-1}, for layer m in the reverse pass.

Unlike greedy layer-wise autoencoder pretraining, which trains each layer separately and only constrains $\mathbf{d}_1$ and $\mathbf{d}_1^{'}$, we jointly train all layers and constrain all intermediate results. Due to the symmetric structure of the two passes, we can use a simple L^2 difference to drive the network toward aligning the results:

$$\mathcal{L}_{\mathrm{RR}} = \sum_{m=1}^{n} \lambda_m \left\| \mathbf{d}_m - \mathbf{d}_m^{'} \right\|_2^2. \tag{7}$$

Here, $\mathbf{d}_m$ denotes the input of layer m in the forward pass and $\mathbf{d}_m^{'}$ the output of layer m for the reverse pass. λ_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, $\mathbf{d}_1$ and $\mathbf{d}_1^{'}$ 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 $\mathbf{d}_{m+1} = M_m\mathbf{d}_m + \mathbf{b}_m$, with $\mathbf{d}_1$ and $\mathbf{d}_{n+1}$ denoting input and output, respectively. All neural networks can be classified according to whether the full reverse pass can be built from the output to input, and we also classify our pretraining as full network pretraining and localized pretraining in implementation.

3.1.1 Full network pretraining

For networks where a unique path from output to input exists, we build a reverse pass network with transposed

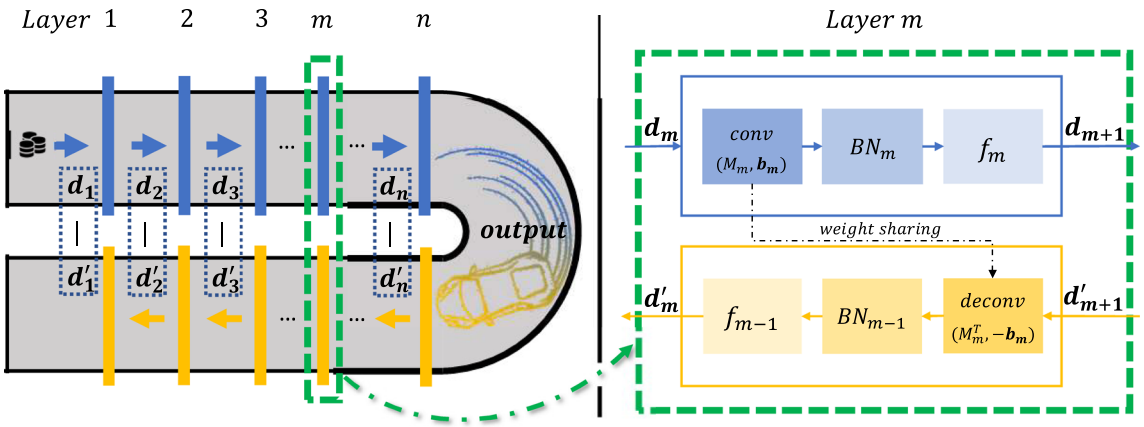

Fig. 2 Left: An overview of the regular forward pass (blue) and the corresponding reverse pass (yellow). The input of layer m is denoted by $\mathbf{d}_m$. The right side illustrates how parameters are reused for a convolutional layer in the reverse pass. The activation function and batch normalization of layer m are denoted by f_m and BN_m, respectively. The shared kernel and bias are represented by M_m and $\mathbf{b}_m$, respectively

operations starting with the final output where $\mathbf{d}_{n+1} = \mathbf{d}'_{n+1}$, and the intermediate results $\mathbf{d}'_{m+1}$:

$$\mathbf{d}'_m = M_m^T(\mathbf{d}'_{m+1} - \mathbf{b}_m), m = 1, 2, \ldots, n, \tag{8}$$

where the reverse pass activation $\mathbf{d}'_m$ depends on $\mathbf{d}_{m+1}{}'$, this formulation yields a full reverse pass from output to input, which we use for most training runs below. Here, we analyze the influence of Eq. (7) during training by assuming $\mathcal{L}_{\text{RR}} = 0$ during the minimization. We then obtain activated intermediate content during the reverse pass that reconstructs the values computed in the forward pass, i.e., $\mathbf{d}'_{m+1} = \mathbf{d}_{m+1}$ holds. In this case

$$\begin{aligned} \mathbf{d}'_m &= M_m^T(\mathbf{d}'_{m+1} - \mathbf{b}_m) \\ &= M_m^T(\mathbf{d}_{m+1} - \mathbf{b}_m) = M_m^T M_m \mathbf{d}_m, m = 1, 2, \ldots, n, \end{aligned} \tag{9}$$

which means that Eq. (7) is consistent with Eq. (3).

3.1.2 Localized pretraining

For architectures that have a reverse path that is not unique, e.g., in the presence of additive residual connections, we cannot uniquely determine the b, c in $a = b + c$ given only a. In such cases, we use a local formulation, and d_{m+1} is used as input of the reverse path of layer m directly. In this case Eq. (8) can be written as:

$$\mathbf{d}'_m = M_m^T(\mathbf{d}_{m+1} - \mathbf{b}_m), m = 1, 2, \ldots, n, \tag{10}$$

which effectively employs $\mathbf{d}_{m+1}$ for jointly constraining all intermediate activations in the reverse pass. Moreover, it is consistent with Eq. (3).

In summary, Eq. (7) will drive the network toward a state that is as-invertible-as-possible for the given input dataset. Comparing the full network pretraining and localized pretraining, the full network pretraining establishes a stronger relationship among the loss terms of different layers, and allows earlier layers to decrease the accumulated loss of later layers. Localized pretraining, on the other hand, is even valid for cases where the reverse path from output to input is not unique.

Up to now, the discussion focused on simplified neural networks with convolutional operations, which are crucial for feature extraction, but without AFs or extensions such as BN, which are applied to increase model nonlinearity. While we leave a more detailed theoretical analysis of these extensions for future work, we apply these nonlinear extensions for all of our tests in Sects. 4 and 5. Thus, our experiments demonstrate that our method works in conjunction with BN and AFs. They show consistently show that the inherent properties of our pretraining remain valid: even in the nonlinear setting our approach successfully extracts dominant structures and yields improved generalization.

In 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, and we give practical examples of full and localized pretraining architectures.

To summarize, we realize the loss formulation of Eq. (7) to minimize $\sum_{m=1}^{n} \left\| (M_m^T M_m - I)\mathbf{d}_m \right\|_2^2$ without explicitly having to construct $M_m^T M_m$. Following the notation above, we will refer to networks trained with the added reverse structure and the additional loss terms as *RR* variants. We consider two variants for the reverse pass: a local pretraining Eq. (10) using the datum $\mathbf{d}_{m+1}$ of a given layer, and a full version via Eq. (8) which uses $\mathbf{d}'_{m+1}$ incoming from the next layer during the reverse pass. It is worth pointing out that our constraint is only used during the pretraining

stage, and pretrained models are used as a starting point for the fine-tuning stage, where the search space for parameters is the same as in standard training, i.e., training is not constrained by our approach.

3.2 Embedding singular values

In the following, we evaluate networks trained with different methodologies. We distinguish our pretraining approach RR(in green), regular autoencoder pretraining Pre (in gray), and orthogonality constraints Ort (in blue). In addition, Std denotes a regular training run (in in graphs below), i.e., models trained without autoencoder pretraining, orthogonality regularization or our proposed method. Besides, a subscript will denote the task variant the model was trained for, such as Std_T for task T. While we typically use all layers of a network in the constraints, a reduced variant that we compare to below only applies the constraint for the input data, i.e., m=1. A network trained with this variant, denoted by RR^1_A, is effectively trained to only reconstruct the input. It contains no constraints for the inner activations and layers of the network. For the Ort models, we use the Spectral Restricted Isometry Property algorithm [3].

We verify that the column vectors of V_m of models from RR training contain the dominant features of the input with the help of a classification test, employing a single fully connected layer, i.e., $\mathbf{d}_2 = M_1\mathbf{d}_1$, with BN and activation. To quantify this similarity, we compute a Learned Perceptual Image Patch Similarity (LPIPS) [86] between v^i_m and the training data (lower values being better).

We employ a training dataset 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, as shown in Fig. 3. Based on the analysis above, we expect the RR training to extract the two dominant peaks during training. The LPIPS measurements confirm our SVD argumentation above, with average scores of 0.217 ± 0.022 for RR, 0.319 ± 0.114 for Pre, 0.495 ± 0.006 for Ort, and 0.500 ± 0.002 for Std, i.e., the RR model fares significantly better than the others. At the same time, the peaks are clearly visible for RR models, while the other models fail to extract structures that resemble the input. Thus, by training with the full network and the original training objective, our pretraining yields structures that are interpretable and be inspected by humans.

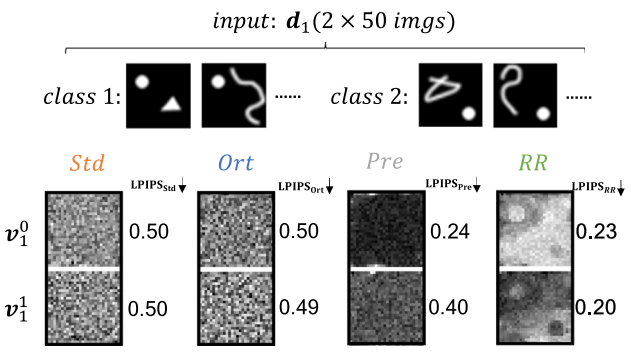

Fig. 3 Column vectors of V_m for different trained models Std, Ort, Pre and RR for peaks. Input features clearly are successfully embedded in the weights of RR, as confirmed by the LPIPS scores

The results above experimentally confirm our formulation of the RR loss and its ability to extract dominant and generalizing structures from the training data. In addition, they give the first indication that this still holds when nonlinear components such as AFs are present. Next, we will focus on quantified metrics and turn to measurements in terms of mutual information to illustrate the behavior of our pretraining for deeper networks.

4 Evaluation in terms of mutual information

Mutual information (MI) measures the dependence of two random variables, i.e., higher MI means that there is more shared information between two parameters. As our approach hinges on the introduction of the reverse pass, we will show that it succeeds in terms of establishing 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 $I(X;Y) = D_{KL}[P_{(X,Y)}||P_X P_Y]$. [69] proposed *MI plane* to analyze trained models, which show the MI between the input X and activations of a layer $\mathcal{D}_m$, i.e., $I(X;\mathcal{D}_m)$ and $I(\mathcal{D}_m;Y)$, i.e., MI of layer $\mathcal{D}_m$ 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 our pretraining succeeds at incorporating information about the inputs throughout training.

The following tests employ networks with six fully connected layers and nonlinear AFs, with the objective to learn the mapping from 12 binary inputs to 2 binary output digits [64], with results accumulated over five runs. Experimental details are illustrated in Appendix. We compare the versions Std_A, Pre_A, Ort_A, and RR_A. We visualize model comparisons with the MI planes, the content of which is visually summarized in Fig. 4a. Horizontal/vertical axis of the MI plane denotes $I(X;\mathcal{D}_m)/I(Y;\mathcal{D}_m)$, which measures the amount of shared information between the m^{th} layer $\mathcal{D}_m$ and X/Y after training. This depicts how much information about input and output distribution is retained at each layer, as well as

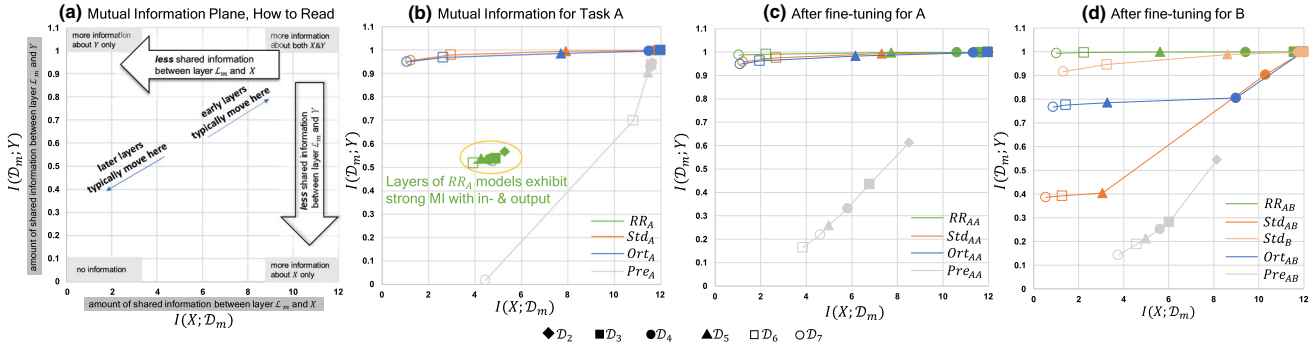

Fig. 4 MI planes for different models: **a** Visual overview of the contents. **b** Plane for task A. Points on each line correspond to layers of one type of model. All points of $\mathrm{RR_A}$, are located in the center of the graph, while $\mathrm{Std_A}$ and $\mathrm{Ort_A}$, exhibit large $I(\mathcal{D}_m;Y)$, i.e., specialize on the output. $\mathrm{Pre_A}$ strongly focuses on reconstructing the input with high $I(X;\mathcal{D}_m)$ for early layers. **c**, **d** After fine-tuning for A/B. The last layer $\mathcal{D}_7$ of $\mathrm{RR_{AA}}$ and $\mathrm{RR_{AB}}$ successfully builds the strongest relationship with Y, yielding the highest accuracy

how these relationships change throughout the network. For regular training, the information bottleneck principle [69] states that early layers contain more information about the input, i.e., show high values for $I(X;\mathcal{D}_m)$ and $I(\mathcal{D}_m;Y)$. As a result, these layers are frequently visible in the top-right corner of MI plane visualizations. After training, later layers typically share a large amount of information with the output, i.e., show high $I(\mathcal{D}_m;Y)$ values, and correlate less with the input (low $I(X;\mathcal{D}_m)$). As a result, they typically appear in the top-left corner of MI plane graphs.

The graph in Fig. 4b highlights that training with the RR loss $\mathrm{RR_A}$ correlates input and output distributions across all layers: the cluster of green points in the center of the graph indicates that all layers contain balanced MI between input as well as output and the activations of each layer. $\mathrm{Std_A}$ and $\mathrm{Ort_A}$ almost exclusively focus on the output with $I(\mathcal{D}_m;Y)$ being close to one and information dropped out layer by layer leads to a low $I(X;\mathcal{D}_7)$ value. $\mathrm{Pre_A}$ instead only focuses on reconstructing inputs. Thus, the early layers cluster in the upper-right corner, while the last layer $I(\mathcal{D}_7;Y)$ fails to align with the outputs. Once we continue to fine-tune these models without regularization, the MI naturally shifts toward the output, as illustrated in Fig. 4c. Here, $\mathrm{RR_{AA}}$ outperforms the other models in terms of the final performance. Furthermore, we design a transfer task B with switched output digits, which means that in task B, the original two binary output digits, e.g., (1, 0), will be switched into (0, 1). This change of the dataset results in significantly different mapping relationships between inputs and outputs compared with original task A. Likewise, $\mathrm{RR_{AB}}$ 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 Table 1. The graph demonstrates that the proposed pretraining succeeds in robustly establishing mutual information between inputs and targets across a full network while extracting reusable features. The nonlinearity of the underlying network architectures does not impede the performance of the RR models. It is worth pointing out that Std and Ort exhibit high performance variance in transfer task B, but not in base task A, because $\mathrm{Std_A}$ and $\mathrm{Ort_A}$ were trained solely to improve task A performance. The extracted features are not guaranteed to be useful for task B in this process. As a result, performance in task B is not consistent across training. On the other hand, $\mathrm{RR_A}$ focuses on extracting dominant features from the dataset, rather than specific tasks, which significantly improves the stability of training across different runs for tasks A and B.

Comparing Fig. 4b and d, we can see that after pretraining via our approach, balanced MI is obtained between input as well as output and the activations of each layer, indicating that our model extracted balanced features from both the input and output. After transfer learning for task B, we can see that all layers are located at the top part of the graph with high $I(\mathcal{D}_m;Y)$ values, indicating that the model aims to improve the performance for a specific task.

We also compare the mutual information of three variants of our pretraining: the local variant $\mathrm{lRR_A}$, the full version

Table 1 Performance of MI source and transfer tasks in Figs. 4 and 5

		Std	Pre	Ort	RR[1]	lRR	RR
Source task A	Avg.	0.973	0.474	0.965	0.931	0.979	0.992
	Std. dev.	0.015	0.107	0.024	0.040	0.011	0.002
Transfer task B	Avg.	0.471	0.561	0.809	0.955	0.985	0.997
	Std. dev.	0.459	0.083	0.376	0.022	0.012	0.002

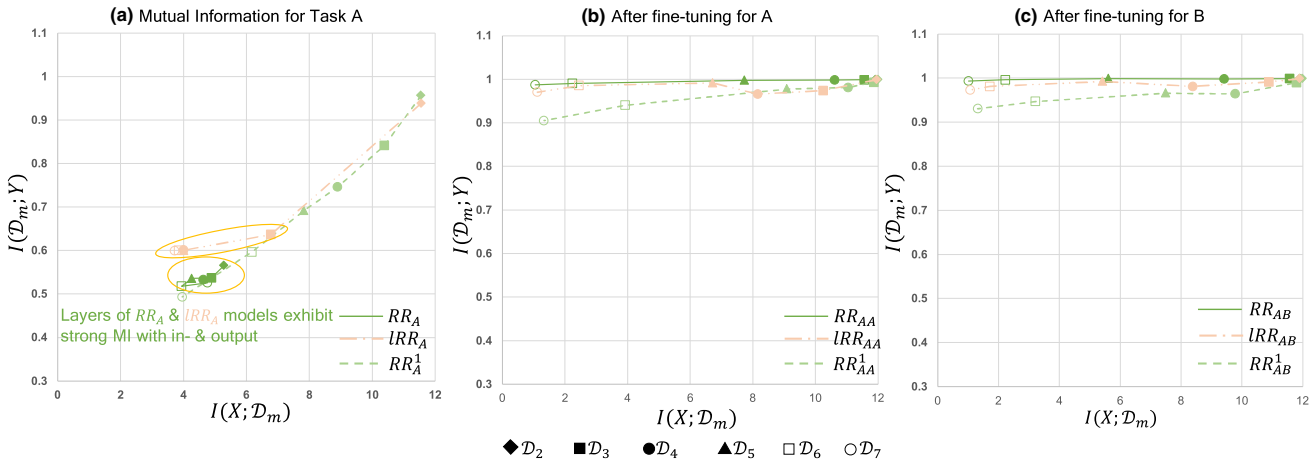

Fig. 5 MI plane comparisons among RR^1_A, local variant lRR_A and the full version RR_A. Points on each line correspond to layers of one type of model. **a** MI Plane for task A. All points of RR_A and the majority of points for lRR_A (five out seven) are located in the center of the graph, i.e., successfully connect input and output distributions. **b**, **c** After fine-tuning for A/B. The last layer $\mathcal{D}_7$ of $\mathrm{RR}_{\mathrm{AA/AB}}$ builds the strongest relationship with Y. $I(\mathcal{D}_7;Y)$ of $\mathrm{lRR}_{\mathrm{AA/AB}}$ is only slightly lower than $\mathrm{RR}_{\mathrm{AA/AB}}$

RR_A, and a variant of the latter: RR^1_A, i.e., a version where only the input $\mathbf{d}_1$ is constrained to be reconstructed. Figure 5 shows the MI planes for these three models. Only one layer is constrained with our formulation in RR^1_A, but we can see that the last two layers of the model are already located in the middle part of the MI plane (Fig. 5a), and the influence is in line with our full version RR_A. Despite the local nature of lRR_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 toward the upper-right corner, and the second layer is affected slightly. In other words, 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 $\mathrm{lRR}_{\mathrm{AA/AB}}$ still clearly outperforms $\mathrm{RR}^1_{\mathrm{AA/AB}}$. This confirms our choice to use the full pretraining when network connectivity permits, and employ the local version in all other cases. Accuracy comparisons among different models are displayed in Table 1. $\mathrm{RR}_{\mathrm{AA/AB}}$ yields the highest performance, while $\mathrm{lRR}_{\mathrm{AA/AB}}$ performs similarly with $\mathrm{RR}_{\mathrm{AA/AB}}$.

In summary, from the MI tests we can conclude that training with our formulation (RR_A and lRR_A) is useful for correlating input and output distributions across all layers. Furthermore, this correlation would be strengthened if more layers were constrained with our formulations, e.g., comparing RR_A with RR^1_A. On the other hand, models pretrained with our formulation, e.g., RR_A and lRR_A, can achieve highest value of $I(\mathcal{D}_7;Y)$ and performance for source task A and transfer task B after fine-tuning.

MI has received attention recently as a learning objective, e.g., in the form of the InfoGAN approach [9] for learning disentangled and interpretable latent representations. While MI is typically challenging to assess and estimate [74], the results above show that our approach provides a straightforward and robust way for including it as a learning objective. In this way, we can easily, e.g., reproduce the disentangling results from [9] without explicitly calculating mutual information, which are shown in Fig. 1c. A generative model with our pretraining extracts intuitive latent dimensions for the different digits, line thickness, and orientation without any additional modifications to the loss function. The joint training of the full network with the proposed reverse structure, including nonlinearities and normalization, yields a natural and intuitive decomposition.

5 Experimental results

We now turn to a broad range of network structures, i.e., CNNs, Autoencoders, and GANs, with a variety of datasets and tasks to show our approach succeeds in improving inference accuracy and generality for modern-day applications and architectures. All tests use nonlinear activations and several of them include BN. Experimental details are provided in Appendix.

5.1 CIFAR-100 classification

We first focus on orthogonalization for a CIFAR-100 classification task with a ResNet 18 network, and compare the performance of RR with the variants Std, Ort, in addition to an OCNN (in light blue) network [75]. The CNN architecture has ca. 11 million trainable parameters in

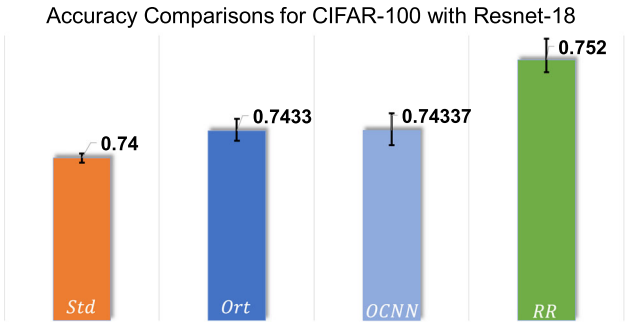

Fig. 6 CIFAR-100 classification performance for RR, Std, Ort and OCNN. RR yields the highest accuracy, and outperforms state-of-the-art methods for orthogonalization (Ort and OCNN)

each case. Pre is not included in this comparison due to its incompatibility with ResNet architectures. The resulting performance for the different variants (evaluated for 3 runs each) is shown in Fig. 6. For CIFAR-100, the orthogonal regularizations (Ort and OCNN) result in noticeable performance gains of 0.33% and 0.337%, but RR clearly outperforms both with an improvements of 1.2%. Despite being different formulations, both Ort and OCNN represent orthogonal regularizers that aim for the same goal of weight orthogonality. Hence, their performance is on-par, and we will focus on the more generic Ort variant for the following evaluations.

5.2 Transfer learning benchmarks

We evaluate our approach with two state-of-the-art benchmarks for transfer learning (Fig. 7). The first one uses the *texture-shape* dataset from [21], 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. 8a, the

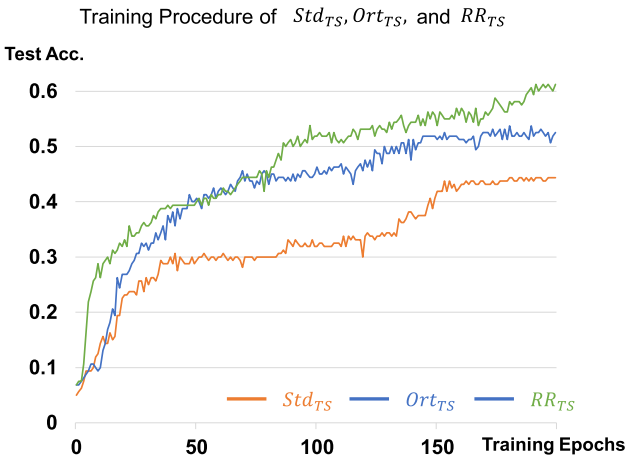

Fig. 7 Test accuracy over training epochs for Std_{TS}, Ort_{TS}, and RR_{TS}. The RR_{TS} model consistently exhibits faster convergence than the other two versions

accuracy of Pre_{TS} is low with 20.8%. This result is in line with observations in previous work and confirms the detrimental effect of classical pretraining. Std_{TS} yields a performance of 44.2%, and Ort_{TS} improves the performance to 47.0%, while RR_{TS} yields a performance of 54.7% (see Fig. 8b). Thus, the accuracy of RR_{TS} is 162.98% higher than Pre_{TS}, 23.76% higher than Std_{TS}, and 16.38% higher than Ort_{TS}. To assess generality, we also apply the models to new data without re-training, i.e., an edge and a filled dataset, also shown in Fig. 8a. For the edge dataset, RR_{TS} outperforms Pre_{TS}, Std_{TS} and Ort_{TS} by 178.82%, 50% and 16.75%, respectively.

Exemplary curves for test accuracy at training time for Std_{TS}, Ort_{TS}, and RR_{TS} are shown in Fig. 7. Pre_{TS} is not included since its layer-wise curriculum precludes a direct comparison. The graph shows that RR_{TS} converges faster than Std_{TS} and Ort_{TS} from the very beginning. It achieves the performance of Std_{TS} and Ort_{TS} with ca. $\frac{1}{3}$ and $\frac{1}{2}$ of number of training epochs, respectively. Achieving comparable performance with less training effort, and a higher final performance support the reasoning given in Sect. 3: RR_{TS} with its reverse pass is more efficient at extracting relevant features from the training data. Over the course of our tests, we observed a similar convergence behavior for a wide range of other runs.

It is worth pointing out that the additional constraints of our training approach lead to moderately increased requirements for memory and computations, e.g., 41.86% more time per epoch than regular training for the texture-shape test. As this test employs a small network with only ca. 1.2×10^4 trainable weights, the computations for our approach still make a noticeable difference in training time. However, as we show below, the difference becomes negligible for larger networks. 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 Ort_{TS} in terms of classification performance. By comparison, regular layer-wise pretraining requires significant overhead and fundamental changes to the training process. Our pretraining fully integrates with existing training methodologies and can easily be deactivated via $\lambda_m = 0$. More details of runtime performance and training behavior are given in Appendix.

As a second test case, we use a CIFAR-based task transfer [61] that measures how well models trained on the original CIFAR 10, generalize to a new dataset (CIFAR 10.1) collected according to the same principles as the original one. Here, we use a ResNet 110 with 110 layers and 1.7 million parameters, Due to the consistently low performance of the Pre models [1], we focus on Std, Ort and RR for this test case. In terms of accuracy across 5 runs, Ort_{C10} outperforms Std_{C10} by 0.39%, while RR_{C10}

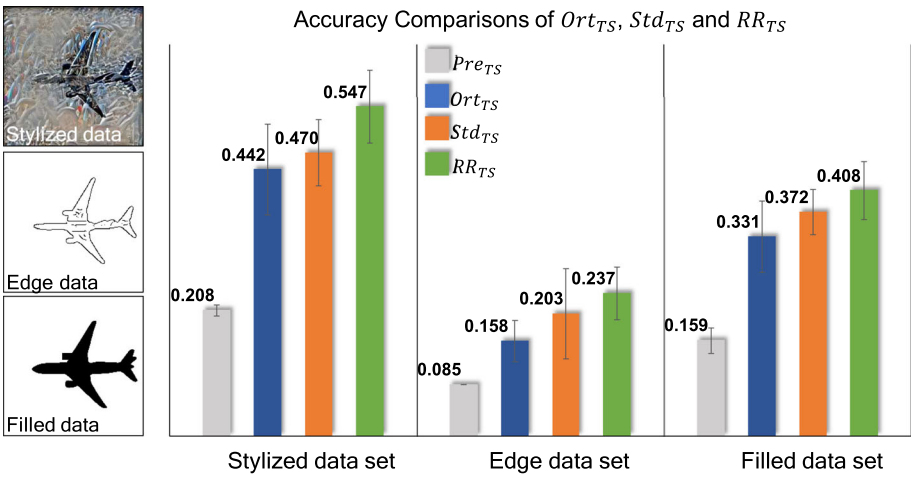

Fig. 8 **a** Examples from texture-shape dataset. **b**, **c**, **d** Texture-shape test accuracy comparisons of Pre_{TS}, Ort_{TS}, Std_{TS} and RR_{TS} for different datasets

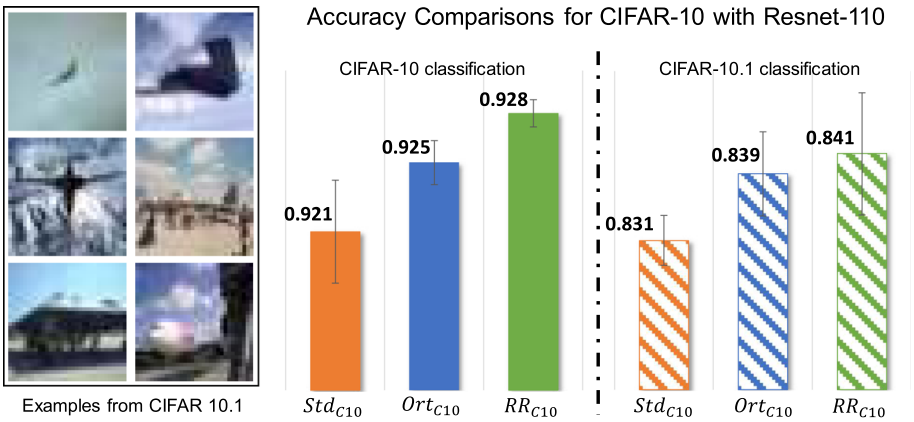

Fig. 9 Left: Examples from CIFAR 10.1 dataset. Right: Accuracy comparisons when applying models trained on CIFAR 10 to CIFAR 10.1 data

outperforms Ort_{C10} by another 0.28% in terms of absolute test accuracy (Fig. 9). This increase for RR training matches the gains reported for orthogonality in previous work [3], thus showing 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 Ort_{C10} and Std_{C10} by 0.22% and 0.95%, respectively. Hence, the models from our pretraining very successfully translate gains in performance from the original task to the new one, indicating that the models have successfully learned a set of more general features. To summarize, both benchmark cases confirm that the proposed pretraining benefits generalization.

5.3 Smoke generation

In this section, we employ our pretraining in the context of generative models for transferring from synthetic to real-world data from the ScalarFlow dataset [17]. As super-resolution task A, we first use a fully convolutional generator network, adversarially trained with a discriminator network on the synthetic flow data. While regular pretraining is more amenable to generative tasks than orthogonal regularization, it cannot be directly combined with adversarial training. Hence, we pretrain a model Pre for a reconstruction task at high-resolution without discriminator instead. Figure 10a demonstrates that our method works well 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_{A}, only yields a black image in this case. For Pre_{A}, the layer-wise nature of the pretraining severely limits its capabilities to learn the correct data distribution [88], leading to low performance.

We now mirror the generator model from the previous task to evaluate an autoencoder structure that we apply to two different datasets: the synthetic smoke data used for the GAN training (task B_1), and a real-world RGB dataset of smoke clouds (task B_2). Thus, both variants represent transfer tasks, the second one being more difficult due to the changed data distribution. The resulting losses, summarized in Fig. 10b, show that RR training performs best for both autoencoder tasks: the L^2 loss of RR_{AB_1} is 68.88% lower than Std_{AB_1}, while it is 13.3% lower for task B_2. The proposed pretraining also clearly outperforms the Pre

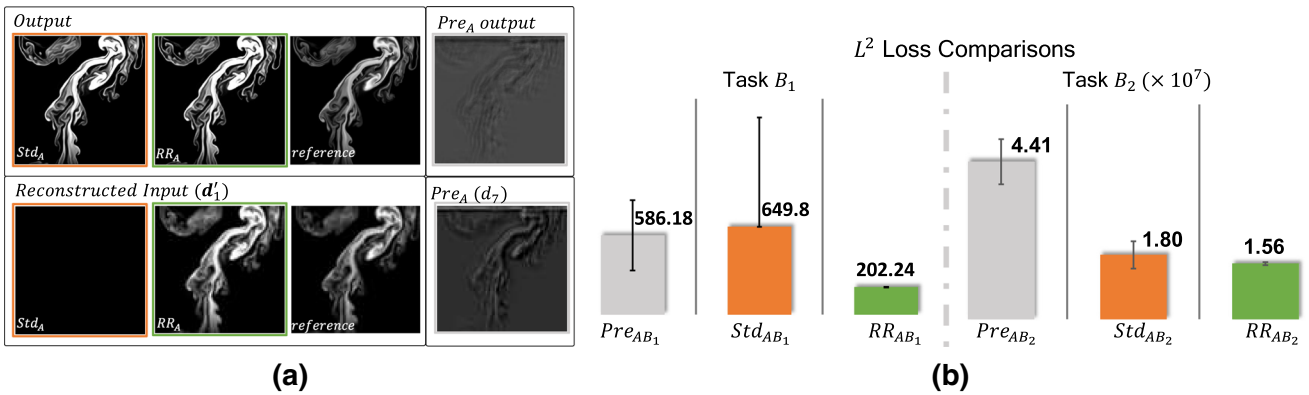

Fig. 10 **a** Example output and reconstructed inputs, with the reference shown right. Only RR_A successfully recovers the input, Std_A produces a black image, while Pre_A fares poorly. **b** Mean squared error L^2 loss comparisons for two different generative transfer learning tasks (averaged across 5 runs each). The RR models show the best performance for both tasks

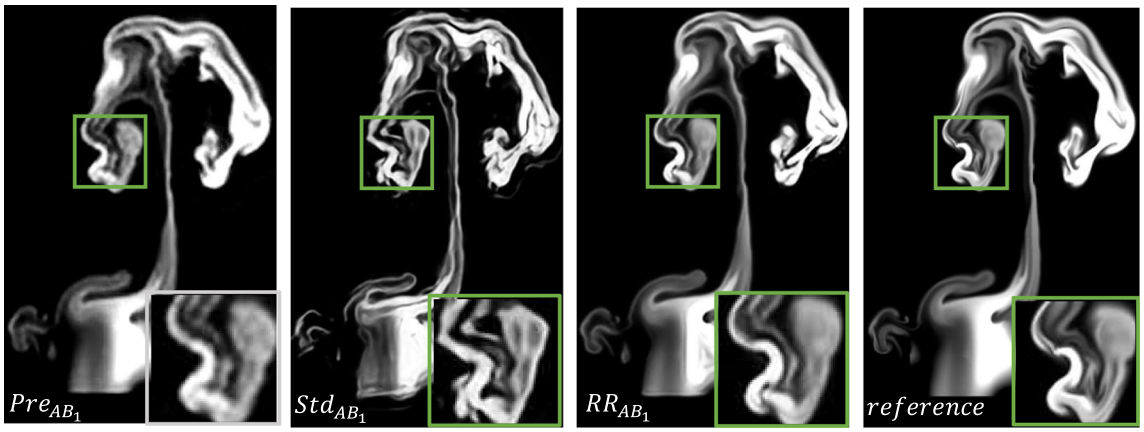

Fig. 11 Example outputs for Pre_{AB_1}, Std_{AB_1}, RR_{AB_1}. The reference is shown for comparison. RR_{AB_1} produces higher quality results than Std_{AB_1} and Pre_{AB_1}

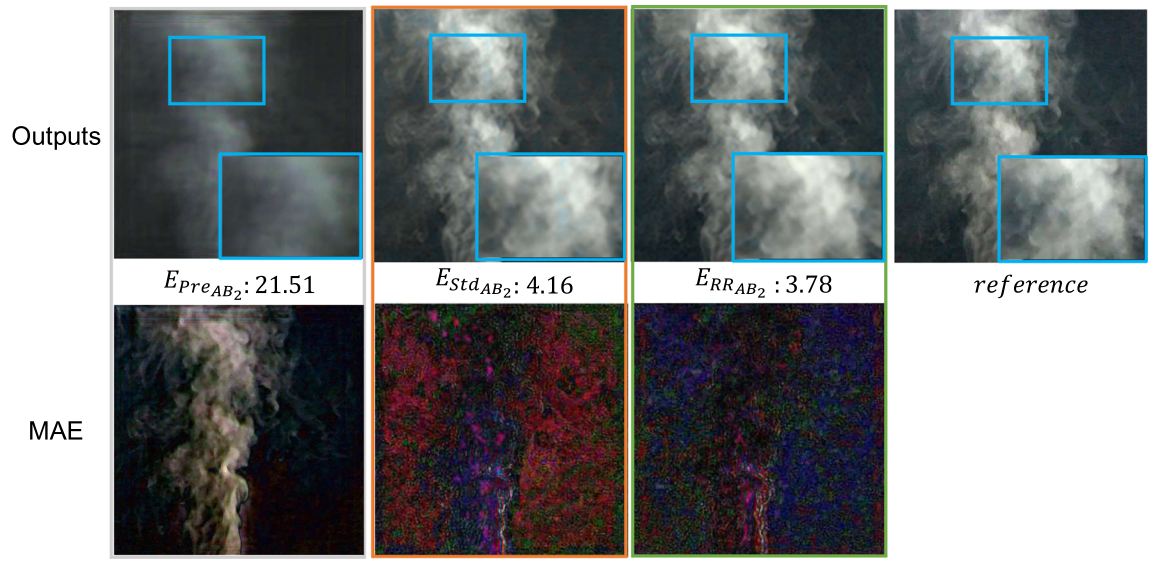

Fig. 12 MAE comparisons for transfer task B_2 models trained with captured smoke dataset reveal that RR_{AB_2} provides the best visual quality while also having the smallest error

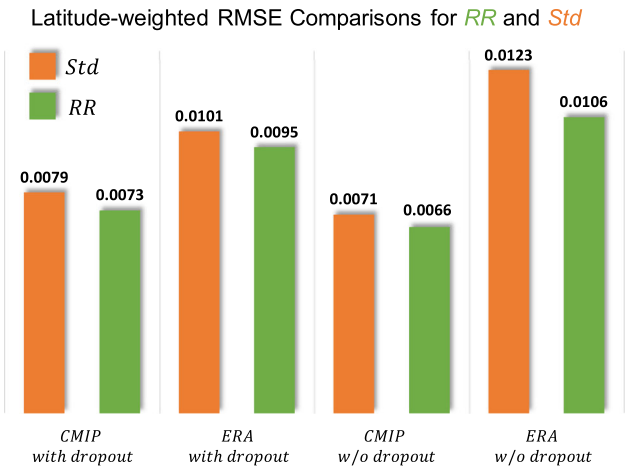

Fig. 13 Latitude-weighted RMSE comparisons between Std and RR for ERA and CMIP datasets. Models trained with RR pretraining significantly outperform state-of-the-art Std for all cases. The minimum performance improvements of RR is 5.7% for the case with ERA dataset and dropout regularization

variants. Example outputs of Pre_{AB_1}, Std_{AB_1} and RR_{AB_1} for transfer task B_1 are shown in Fig. 11. It is apparent that RR_{AB_1} provides the best performance among these models. Figure 12 provides visual comparisons for transfer task B_2, where RR_{AB_2} generates results that are closer to the reference. Within this series of tests, the RR performance for task B_2 is especially encouraging, as this task represents a synthetic to real transfer.

5.4 Weather forecasting

Pretraining is particularly attractive in situations where the amount of data for training is severely limited. Weather forecasting is such a case, as accurate, real-world data for many relevant quantities are only available for approximately 50 years. We use the ERA dataset [31] consisting of assimilated measurements, and additionally evaluate our models with simulated data from the CMIP database [19]. We replicate the architecture and training procedure of the WeatherBench benchmark [59]. Hence, we use prognostic variables at seven vertical levels, together with some surface and constant fields at the current time t as well as $t - 6h$ and $t - 12h$ as input, and target three-day forecasts of 500 hPa geopotential (Z500), 2-meter temperature (T2M), and 850 hPa temperature (T850). For training, we employ a convolutional ResNet architecture with 19 residual blocks and 6.36M trainable parameters, as well as a latitude-weighted root mean squared error (RMSE) as loss functions. For worldwide observations dataset ERA (six-hour intervals with a 5.625° resolution.), we train the models with data from 1979 to 2015 and evaluate performance with RMSE measurements across all data points from the years 2017 and 2018. For the historical simulation dataset CMIP, years 1850 to 2005 are used as training data, while performance is measured with years 2006 to 2014.

We show comparisons between the regular model Std and RR for both ERA and CMIP datasets. As Rasp et al. [59] relied on dropout regularization, we additionally train and evaluate models for both datasets with and without dropout. Following their methodology, L_2 regularization is

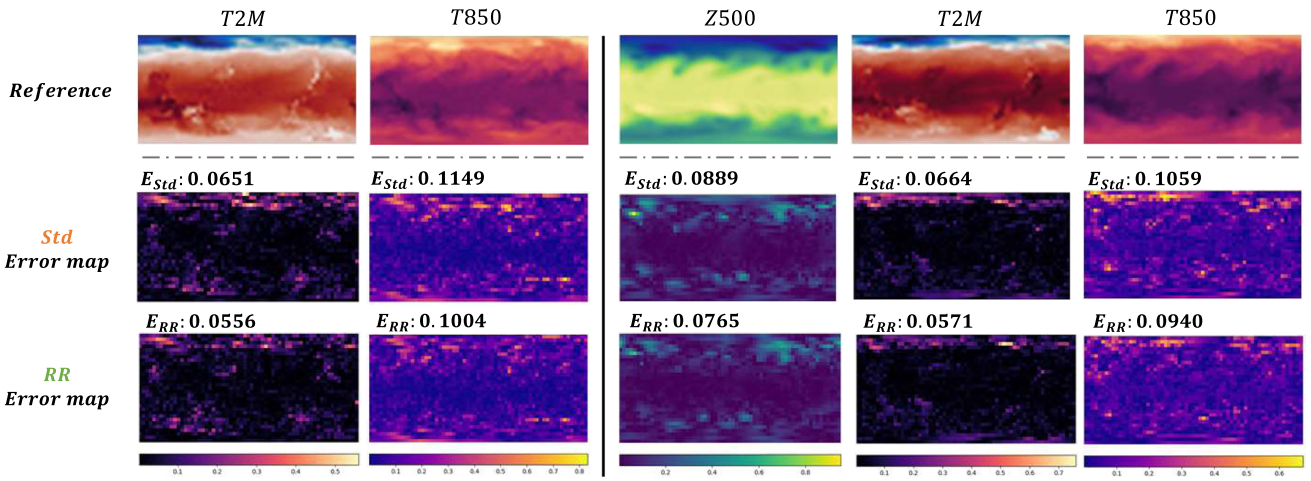

Fig. 14 **a** Comparisons of predictions for T2M and T850 on 9 Aug. 2019, 22:00 for the ERA dataset without dropout regularization. **b** Prediction comparisons of three physical quantities on 26 June 2014, 0:00 for the CMIP dataset without dropout regularization. As confirmed by the quantified results, RR predicts results closer to the reference

applied for all tests. As regular pretraining does not support residual connections, we omit it for the weather forecasting tests.

Performance comparisons are shown in Fig. 13. Across all cases, irrespective of whether observation data or simulation data is used, the RR models clearly outperform the regular models and yield consistent improvements. This also indicates that our approach is compatible with other forms of regularization, such as dropout and L_2 regularization. The RR models yield performance improvements of $6\% \sim 8\%$ for the CMIP cases, and the ERA case with dropout. Here, the re-trained Std version is on-par with the data reported in [59], while our RR model exhibits a performance improvement of 6.3% on average. For the ERA dataset without dropout regularization, the RR model decreases the loss even more strongly by 13.7%.

Visualizations of an inference result for 9 Aug. 2019 22:00 for the ERA dataset without dropout regularization are shown in Figs. 1d and 14a. Predictions of RR yield lower errors, and are closer to the reference. The same conclusions can be drawn from the example at 26 June 2014 0:00 from the CMIP dataset without dropout regularization in Fig. 14b.

6 Conclusions

We have proposed a novel pretraining approach inspired by classic methods for unsupervised autoencoder pretraining and orthogonality constraints. In contrast to the classical methods, we employ a constrained reverse pass for the full nonlinear network structure and include the original learning objective. Weight matrix SVD is applied to visually analyze and interpret that our proposed method is more capable of extracting dominant features from the training dataset. We have shown for a wide range of scenarios, from mutual information, over transfer learning benchmarks to weather forecasting, that the proposed pretraining yields networks with improved performance and better generalizing capabilities. Our training approach is general, easy to integrate, and imposes no requirements regarding network structure or training methods. As a whole, our results show that unsupervised pretraining has not lost its relevance in today's deep learning environment.

As future work, we believe it will be exciting to evaluate our approach in additional contexts, e.g., for temporal predictions [12, 32], and for training explainable and interpretable models [9, 16, 83].

Appendix A Details of the method

A.1 Pretraining and singular value decomposition

In this section, we give a more detailed derivation of our loss formulation, extending Sect. 3 of the main paper. As explained there, our loss formulation aims for minimizing

$$\mathcal{L}_{\mathrm{RR}} = \sum_{m=1}^{n} \left\| M_m^T M_m \mathbf{d}_m^i - \mathbf{d}_m^i \right\|_2^2, \tag{A1}$$

where $M_m \in \mathbb{R}^{s_m^{\mathrm{out}} \times s_m^{\mathrm{in}}}$ denotes the weight matrix of layer m, and data from the input dataset $\mathcal{D}_m$ is denoted by $\mathbf{d}_m^i \subset \mathbb{R}^{s_m^{\mathrm{in}}}, i = 1, 2, \ldots, t$. Here, t denotes the number of samples in the input dataset. Minimizing Eq. (A1) is mathematically equivalent to

$$M_m^T M_m \mathbf{d}_m^i - \mathbf{d}_m^i = \mathbf{0} \tag{A2}$$

for all $\mathbf{d}_m^i$. Hence, perfectly fulfilling Eq. (A1) would require all $\mathbf{d}_m^i$ to be eigenvectors of $M_m^T M_m$ with corresponding eigenvalues being 1. As in Sect. 3 of the main paper, we make use of an auxiliary orthonormal basis $\mathcal{B}_m : \langle \mathbf{w}_m^1, \ldots, \mathbf{w}_m^q \rangle$, for which q (with $q \leq t$) denotes the number of linearly independent entries in $\mathcal{D}_m$. 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.

Since the $\mathbf{w}_m^h (h = 1, 2, \ldots q)$ necessarily can be expressed as linear combinations of $\mathbf{d}_m^i$, Eq. (A1) similarly requires $\mathbf{w}_m^h$ to be eigenvectors of $M_m^T M_m$ with corresponding eigenvalues being 1, i.e.,

$$M_m^T M_m \mathbf{w}_m^h - \mathbf{w}_m^h = \mathbf{0} \tag{A3}$$

We denote the vector of coefficients to express $\mathbf{d}_m^i$ via D_m with $\mathbf{c}_m^i$, i.e., $\mathbf{d}_m^i = D_m \mathbf{c}_m^i$. Then, Eq. (A2) can be rewritten as:

$$M_m^T M_m D_m \mathbf{c}_m^i - D_m \mathbf{c}_m^i = \mathbf{0} \tag{A4}$$

Via an SVD of the matrix M_m in Eq. (A4) we obtain

$$\begin{aligned} & M_m^T M_m D_m \mathbf{c}_m - D_m \mathbf{c}_m \\ &= \sum_{h=1}^{q} M_m^T M_m \mathbf{w}_m^h \mathbf{c}_{m_h} - \mathbf{w}_m^h \mathbf{c}_{m_h} \\ &= \sum_{h=1}^{q} V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h \mathbf{c}_{m_h} - \mathbf{w}_m^h \mathbf{c}_{m_h} \end{aligned} \tag{A5}$$

where the coefficient vector $\mathbf{c}_m$ is accumulated over the training dataset size t via $\mathbf{c}_m = \sum_{i=1}^{t} \mathbf{c}_m^i$. Here, we assume that over the course of a typical training run eventually every single datum in $\mathcal{D}_m$ will contribute to $\mathcal{L}_{\mathrm{RR_m}}$. This form of the loss highlights that minimizing $\mathcal{L}_{\mathrm{RR}}$ requires an alignment of $V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h \mathbf{c}_{m_h}$ and $\mathbf{w}_m^h \mathbf{c}_{m_h}$.

By construction, Σ_m contains the square roots of the eigenvalues of $M_m^T M_m$ as its diagonal entries. The matrix has rank $r = rank(M_m^T M_m)$, and since all eigenvalues are required to be 1 by Eq. (A3), the multiplication with Σ_m in Eq. (A5) effectively performs a selection of r column vectors from V_m. Hence, we can focus on the interaction between the basis vectors $\mathbf{w}_m$ and the r active column vectors of V_m:

$$\begin{aligned} & V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h \mathbf{c}_{m_h} - \mathbf{w}_m^h \mathbf{c}_{m_h} \\ &= \mathbf{c}_{m_h} \left(V_m \Sigma_m^T \Sigma_m V_m^T \mathbf{w}_m^h - \mathbf{w}_m^h \right) \\ &= \mathbf{c}_{m_h} \left(\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f - \mathbf{w}_m^h \right). \end{aligned} \tag{A6}$$

As V_m is obtained via an SVD it contains r orthogonal eigenvectors of $M_m^T M_m$. Eq. (A3) requires $\mathbf{w}_m^1, \ldots, \mathbf{w}_m^q$ to be eigenvectors of $M_m^T M_m$, but since typically the dimension of the input dataset is much larger than the dimension of the weight matrix, i.e., $r \leq q$, in practice only r vectors from $\mathcal{B}_m$ can fulfill Eq. (A3). This means the vectors $\mathbf{v}_m^1, \ldots, \mathbf{v}_m^r$ in V_m are a subset of the orthonormal basis vectors $\mathcal{B}_m : \langle \mathbf{w}_m^1, \ldots, \mathbf{w}_m^q \rangle$ with $\left\| \mathbf{w}_m^h \right\|_2^2 = 1$. Then, for any $\mathbf{w}_m^h$ we have

$$\begin{cases} (\mathbf{v}_m^f)^T \mathbf{w}_m^h = 1, & \text{if } \mathbf{v}_m^f = \mathbf{w}_m^h \\ (\mathbf{v}_m^f)^T \mathbf{w}_m^h = 0, & \text{otherwise.} \end{cases} \tag{A7}$$

Thus, if V_m contains $\mathbf{w}_m^h$, we have

$$\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f = \mathbf{w}_m^h, \tag{A8}$$

and we trivially fulfill the constraint

$$\mathbf{c}_{m_h} \left(\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f - \mathbf{w}_m^h \right) = \mathbf{0}. \tag{A9}$$

However, due to r being smaller than q in practice, V_m typically cannot include all vectors from $\mathcal{B}_m$. Thus, if V_m does not contain $\mathbf{w}_m^h$, we have $(\mathbf{v}_m^f)^T \mathbf{w}_m^h = 0$ for every vector $\mathbf{v}_m^f$ in V_m, which means

$$\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f = \mathbf{0}. \tag{A10}$$

As a consequence, the constraint Eq. (A2) is only partially fulfilled:

$$\mathbf{c}_{m_h} \left(\sum_{f=1}^{r} (\mathbf{v}_m^f)^T \mathbf{w}_m^h \mathbf{v}_m^f - \mathbf{w}_m^h \right) = -\mathbf{c}_{m_h} \mathbf{w}_m^h \,. \tag{A11}$$

As the $\mathbf{w}_m^h$ have unit length, the factors $\mathbf{c}_m$ determine the contribution of a datum to the overall loss. A feature $\mathbf{w}_m^h$ that appears multiple times in the input data will have a correspondingly larger factor in $\mathbf{c}_m$ and hence will more strongly contribute to $\mathcal{L}_{\mathrm{RR}}$. The L^2 formulation of Eq. (A1) 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 loss. Interestingly, this argumentation holds when additional loss terms are present, e.g., a loss term for classification. In such a case, the factors $\mathbf{c}_m$ will be skewed toward those components that fulfill the additional loss terms, i.e., favor basis vectors $\mathbf{w}_m^h$ that contain information for about the loss terms. This, e.g., leads to clear digit structures being embedded in the weight matrices for the MNIST example below.

In summary, to minimize $\mathcal{L}_{\mathrm{RR}}$, V_m is driven toward containing r orthogonal vectors $\mathbf{w}_m^h$ which represent the most frequent features of the input data, i.e., the dominant features. It is worth emphasizing that above $\mathcal{B}_m$ is only an auxiliary basis, i.e., the derivation does not depend on any particular choice of $\mathcal{B}_m$.

A.2 Examples of network architectures with pretraining

While the proposed pretraining is significantly easier to integrate into training pipelines than classic autoencoder pretraining, there are subtleties w.r.t. the order of the operations in the reverse pass that we clarify with examples in the following sections. 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 $F(l)$, 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 (AFs), 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 2×2 filters and stride 2, and batch normalization, respectively.

Below we provide additional examples of how to realize the pretraining loss $\mathcal{L}_{\text{RR}}$ in a neural network architecture. As explained in the main document, the constraint Eq. (A1) is formulated via

$$\mathcal{L}_{\text{RR}} = \sum_{m=1}^{n} \lambda_m \left\| \mathbf{d}_m - \mathbf{d}_m^{'} \right\|_2^2, \tag{A12}$$

with $\mathbf{d}_m$, and λ_m denoting the vector of activated intermediate data in layer m from the forward pass, and a scaling factor, respectively. $\mathbf{d}_m^{'}$ denotes the activations of layer m from the reverse pass. For instance, 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, $\mathbf{d}_{m+1} = L_m(\mathbf{d}_m)$, and $\mathbf{d}_m^{'} = L_m'(\mathbf{d}_{m+1}^{'})$.

When Eq. (A12) is minimized, we obtain activated intermediate content during the reverse pass that reconstructs the values computed in the forward pass, i.e., $\mathbf{d}_{m+1}^{'} = \mathbf{d}_{m+1}$ holds. Then, $\mathbf{d}_m^{'}$ can be reconstructed from the incoming activations from the reverse pass, i.e., $\mathbf{d}_{m+1}^{'}$, or from the output of layer m, i.e., $\mathbf{d}_{m+1}$. Using $\mathbf{d}_{m+1}^{'}$ results in a global coupling of input and output throughout all layers, i.e., the *full* loss variant. On the other hand, $\mathbf{d}_{m+1}$ yields a variant that ensures local reversibility of each layer, and yields a very similar performance, as we will demonstrate below. We employ this *local* loss for networks without a unique, i.e., bijective, connection between two layers. Intuitively, when inputs cannot be reliably reconstructed from outputs.

Full network pretraining An illustration of a CNN structure with AFs and BN and a full loss is shown in Fig. 2 in the main paper. 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 5×5 to 3×3. 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 outlined above, the first three layers of the network are

$$\begin{aligned} I(32,32,1) = \mathbf{d}_1 &\rightarrow C_1(5,20,1) + \mathbf{b}_1 \rightarrow BN_1 \rightarrow \text{relu} \\ &\rightarrow \mathbf{d}_2 \rightarrow MP \rightarrow C_2(4,40,1) + \mathbf{b}_2 \rightarrow BN_2 \rightarrow \tanh \\ &\rightarrow \mathbf{d}_3 \rightarrow MP \rightarrow C_3(3,60,1) + \mathbf{b}_3 \rightarrow BN_3 \rightarrow \text{sigm} \\ &\rightarrow \mathbf{d}_4 \rightarrow \ldots \end{aligned} \tag{A13}$$

The reverse pass for evaluating the loss reuses all weights of the forward pass and ensures that all intermediate vectors of activations, $\mathbf{d}_m$ and $\mathbf{d}_m^{'}$, have the same size and content in terms of normalization and nonlinearity. We always consider states after activation for $\mathcal{L}_{\text{RR}}$. Thus, $\mathbf{d}_m$ denotes activations before pooling in the forward pass and $\mathbf{d}_m^{'}$ 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 $\mathcal{L}_{\text{RR}}$ take the form:

$$\begin{aligned} \ldots \rightarrow \mathbf{d}_4^{'} &\rightarrow -\mathbf{b}_3 \rightarrow D_3(3,40,1) \rightarrow BN_2 \rightarrow \tanh \rightarrow UP \\ &\rightarrow \mathbf{d}_3^{'} \rightarrow -\mathbf{b}_2 \rightarrow D_2(4,20,1) \rightarrow BN_1 \rightarrow \text{relu} \rightarrow UP \\ &\rightarrow \mathbf{d}_2^{'} \rightarrow -\mathbf{b}_1 \rightarrow D_1(5,3,1) \\ &\rightarrow \mathbf{d}_1^{'} = O(32,32,1). \end{aligned} \tag{A14}$$

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 AFs 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, $\mathbf{d}_3$ in our example represents the activations after the second layer (with BN_2 and *tanh*), and hence the reverse pass for $\mathbf{d}_3^{'}$ reuses both functions. This ensures that $\mathbf{d}_m$ and $\mathbf{d}_m^{'}$ contain the same latent space content in terms of range and dimensionality, and can be compared in Eq. (A12).

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. For instance, 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.

Localized pretraining In the example above, we use a full pretraining with $\mathbf{d}_{m+1}^{'}$ to reconstruct the activations $\mathbf{d}_m^{'}$. However, if the architecture of the original network makes use of operations between layers that are not bijective, e.g., residual connections, we instead use the local loss. Note that our loss formulation 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 toward a state that is as-invertible-as-possible for the given input dataset. However, this requires a reliable vector of target activations in order to apply the constraints. If the *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 constraints by building a localized reverse pass. For example, given a simple convolutional architecture with

$$\mathbf{d}_1 \rightarrow C_1(5, 20, 1) + \mathbf{b}_1 = \mathbf{d}_2 \tag{A15}$$

in the forward pass, we calculate $\mathbf{d}_1^{'}$ with

$$(\mathbf{d}_2 - \mathbf{b}_1) \rightarrow D_1(5, 3, 1) = \mathbf{d}_1^{'}. \tag{A16}$$

We, e.g., use this local loss in the ResNet 110 network below. It is important to note that despite being closer to regular autoencoder pretraining, this formulation still incorporates all nonlinearities of the original network structure, and jointly trains full networks while taking into account the original learning objective.

A.3 MNIST and peak tests

Below we give details for the *peak* tests from Sect. 3 of the main paper and show additional tests with the MNIST dataset.

Peak Test For the *Peak* test we generated a dataset of 110 images shown in Fig. 15. 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 the training dataset, 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. 16, with both peak modes being consistently embedded into the weight matrix of RR_A, while regular, autoencoder pretraining and orthogonal training show primarily random singular vectors. For this test, the dataset is constructed such that the two Gaussian peaks are the dominant features of the dataset. No matter what orthonormal basis the network converges to, the two dominant peaks are included in D_m with our approach. Specifically, after training, we can see via SVD that the network consistently learns to encode these two peaks in its parameters, since they contribute most to a reconstruction loss.

We also use different network architectures in Fig. 17 to verify that the dominant features are successfully extracted when using more complex network structures. Even for two layers with BN and ReLU activations, our pretraining 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, our pretraining clearly outperforms regular training and the orthogonality constraint in terms of extracting and embedding the dominant structures of the training dataset in the weight matrix It also yields lower LPIPS evaluations than autoencoder pretraining, which indicates features embedded in RR models represent the training data better.

MNIST Test We additionally verify that the column vectors of V_m of models from RR training contain the dominant features of the input with MNIST tests, which employ a single fully connected layer, i.e., $\mathbf{d}_2 = M_1\mathbf{d}_1$. In the first MNIST test, the training data consists only of 2 different images. All MNIST models are trained for 1000 epochs with a learning rate of 0.0001, and $\lambda = 1e-5$ for RR_A. After training, we compute the SVD for M_1. SVDs of the weight matrices of trained models can be seen in Fig. 18. The LPIPS scores show that features embedded in the weights of RR are consistently closer to the training dataset than all other methods, i.e., regular training Std, classic autoencoder pretraining Pre, and regularization via orthogonalization Ort. While the vectors of Std and Ort contain no recognizable structures.

Overall, our experiments confirm the motivation of our pretraining formulation. They additionally show that employing an SVD of the network weights after our

Fig. 15 Dataset used for the *peak* tests

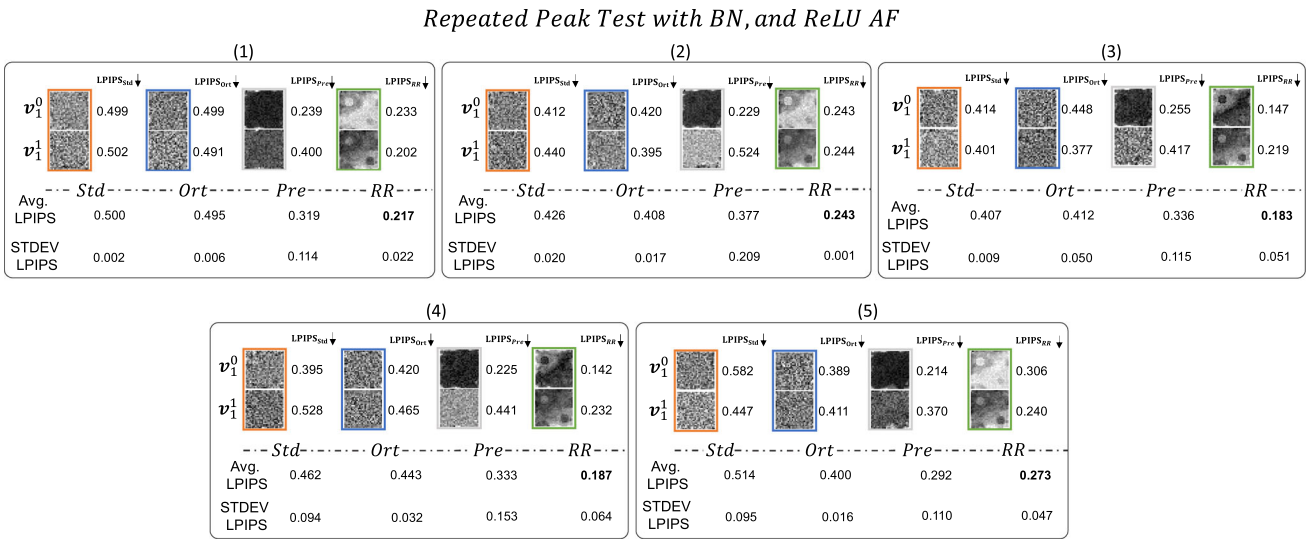

Fig. 16 Five repeated tests with the peak data shown in Sect. 3 of the main paper. RR_A robustly extracts dominant features from the dataset. The two singular vectors strongly resemble the two peak modes of the training data. This is confirmed by the LPIPS measurements

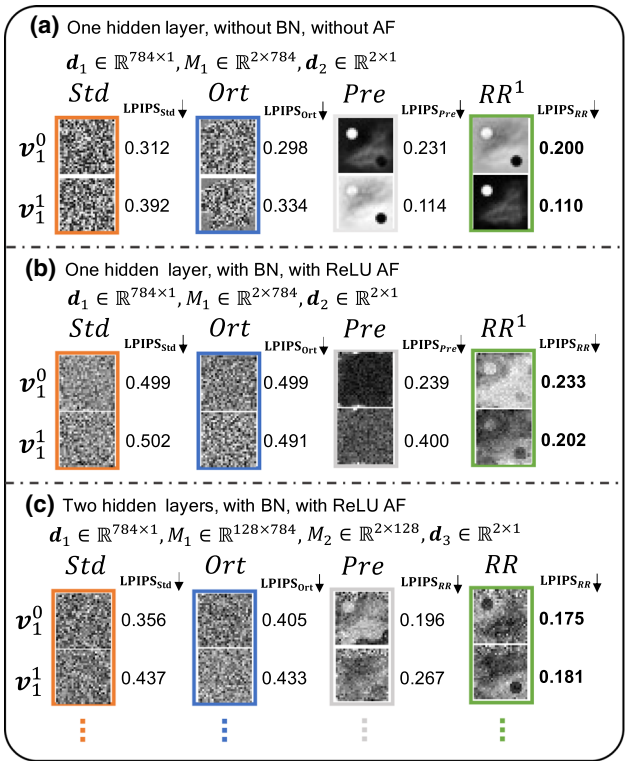

Fig. 17 Right singular vectors of M_1 for Across the three architectures, RR_A successfully extracts dominant and salient features

pretraining yields a simple and convenient method to give humans intuition about the features learned by a network.

Appendix B Mutual information

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

B.1 Mutual information test

Mutual information (MI) 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 $I(X;Y) = \mathrm{KL}[P_{(X,Y)} ||P_X P_Y]$, where KL denotes the Kullback–Leibler divergence. Let $I(X;\mathcal{D}_m)$ denote the mutual information between the activations of a layer $\mathcal{D}_m$ and input X. Similarly $I(\mathcal{D}_m;Y)$ denotes the MI between layer m and the output Y. We use *MI planes* in the main paper, which show $I(X;\mathcal{D}_m)$ and $I(\mathcal{D}_m;Y)$ in a 2D graph for the activations of each layer $\mathcal{D}_m$ of a network after training.

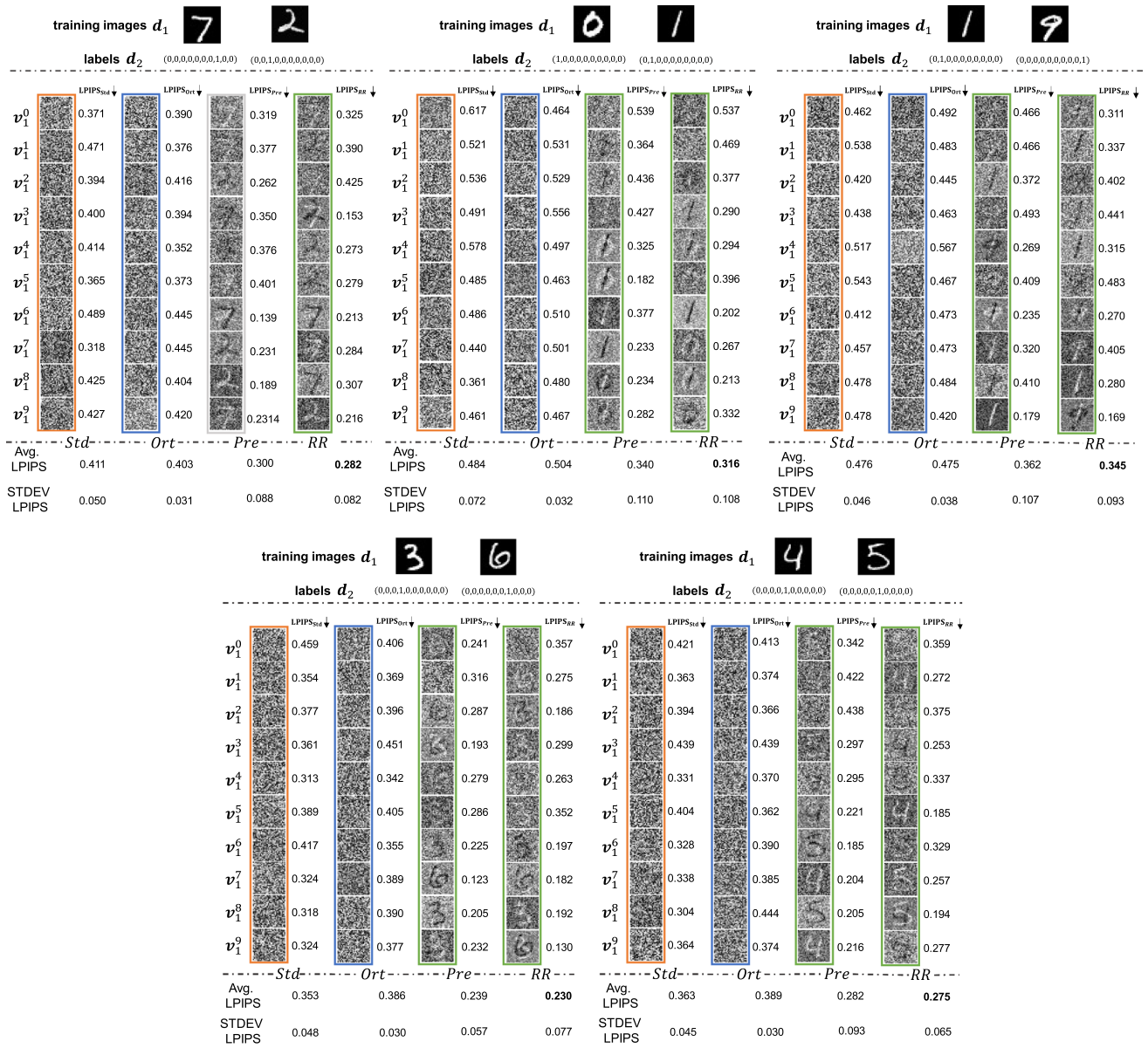

Fig. 18 SVD of the M_1 matrix for five tests with random two digit images as training data. LPIPS distances [86] of RR are consistently lower than Std and Ort

Training details We use the same numerical studies as in [64] 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 [41]. To generate the input–output distribution $P(X, Y)$, we apply the stochastic rule $p(y = 1 \mid x) = \Psi(f(x) - \theta), (x \in X, y \in Y)$, where Ψ is a standard sigmoidal function $\Psi(u) = 1/(1 + exp(-\gamma u))$, following [64]. We then use a spherically symmetric real-valued function of the pattern $f(x)$, evaluated through its spherical harmonics power spectrum [41], and compare with a threshold θ, which was selected to make $p(y = 1) = \sum_x p(y = 1 \mid x)p(x) \approx 0.5$, with uniform $p(x)$. γ 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 dataset, 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.

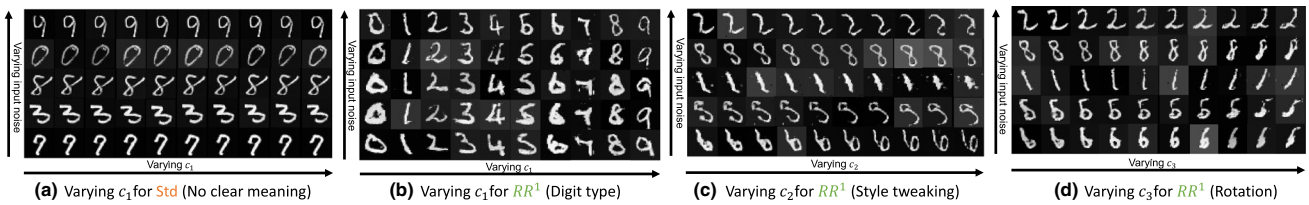

Fig. 19 Additional results for the disentangled representations with the MNIST data: For every row in the figures, we vary the corresponding latent code (left to right), while keeping all other inputs constant. Different rows indicate a different random noise input. For example, in (**b**) every column contains five results which are generated with different noise samples, but the same latent codes $c_{1\sim3}$. In every row, 10 results 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 $c_{2,3}$). **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 pretrained model learns a meaningful, disentangled representation

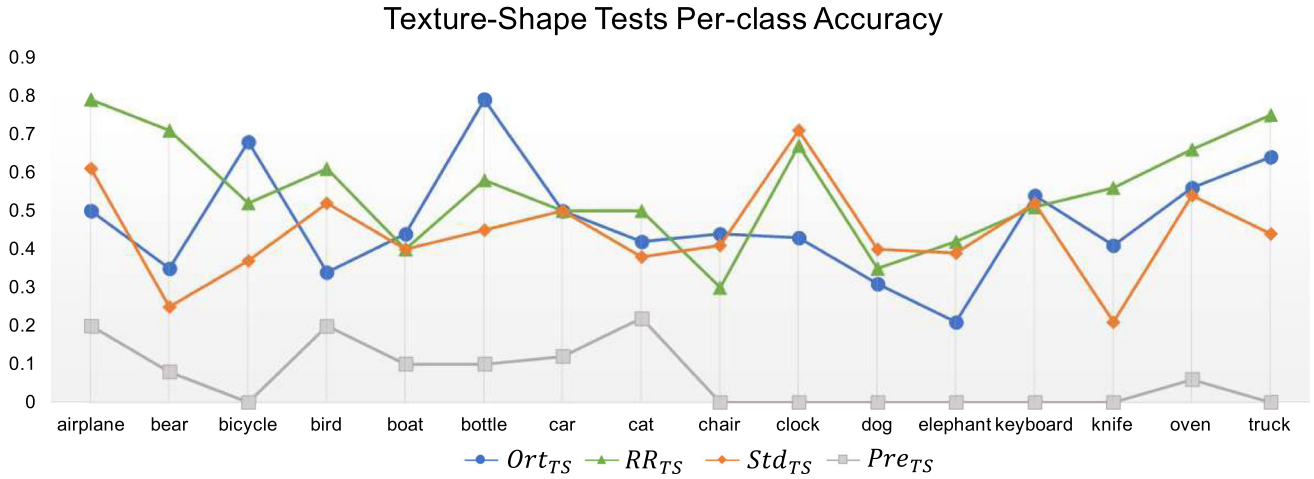

Fig. 20 Separate per-class test accuracies for the four model variants. The RR_{TS} model exhibits a consistently high accuracy across all 16 classes

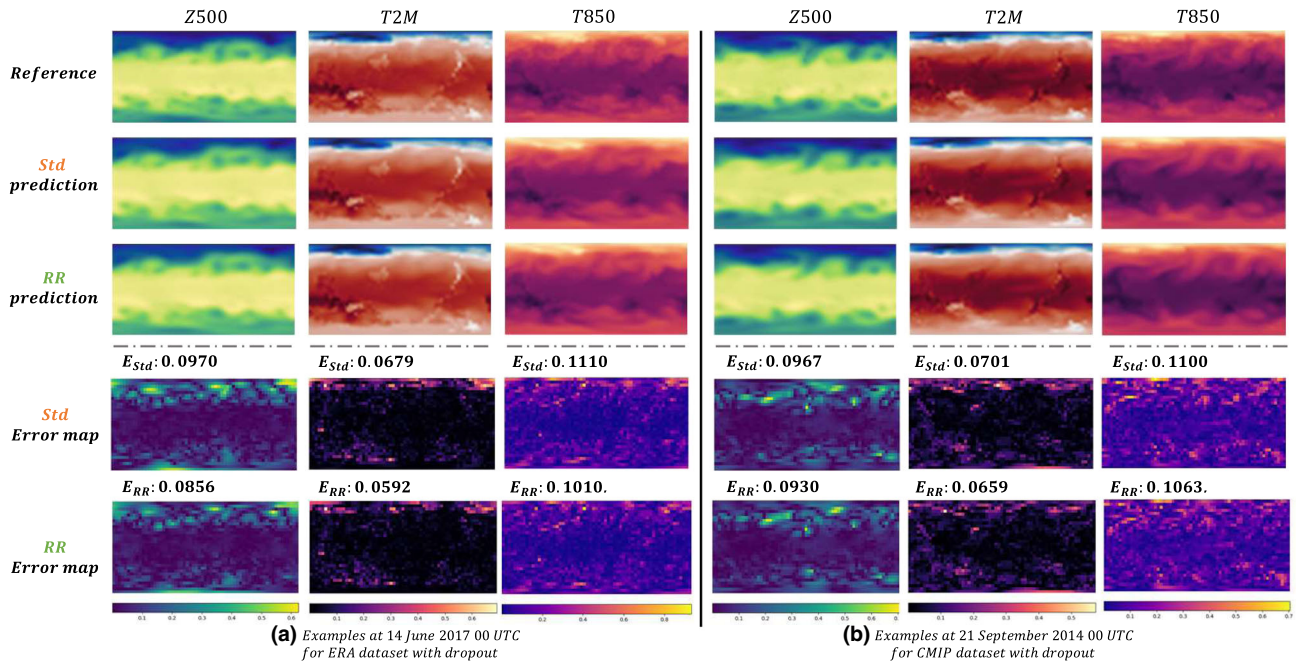

Fig. 21 MAE value comparisons between RR and Std(E_{RR} for RR and E_{Std} for Std). RR consistently yields lower errors than Std. The predictions inferred by the RR model are closer to the observed references

Table 2 Model Accuracy of MI Source Task

Training runs	Std_{A}	Ort_{A}	RR^{1}_{A}	Pre_{A}	lRR_{A}	RR_{A}	Std_{AA}	Ort_{AA}	RR^{1}_{AA}	Pre_{AA}	lRR_{AA}	RR_{AA}
1	0.976	0.969	0.836	0.59	0.8350	0.772	0.978	0.968	0.885	0.638	0.9770	0.992
2	0.975	0.921	0.68	0.548	0.8360	0.784	0.983	0.926	0.918	0.402	0.9770	0.994
3	0.964	0.957	0.8	0.566	0.8140	0.725	0.963	0.958	0.993	0.397	0.9740	0.993
4	0.987	0.991	0.657	0.676	0.8420	0.771	0.989	0.993	0.917	0.404	0.9980	0.989
5	0.957	0.979	0.825	0.533	0.8400	0.774	0.954	0.979	0.944	0.527	0.9710	0.993
Avg.	0.972	0.963	0.760	0.5826	0.8334	0.765	0.973	0.965	0.931	0.4736	0.9794	**0.992**
Std. dev.	0.012	0.027	0.085	0.056	0.0112	0.023	0.015	0.025	0.040	0.1069	0.0107	0.002

Table 3 Model Accuracy of MI Transfer Task

Training runs	Std_{AB}	Ort_{AB}	Std_{B}	RR^{1}_{AB}	Pre_{AB}	lRR_{AB}	RR_{AB}
1	0.136	0.984	0.852	0.919	0.656	0.998	0.998
2	0.136	0.136	0.991	0.954	0.594	0.991	0.997
3	0.989	0.957	0.953	0.977	0.443	0.978	0.995
4	0.136	0.991	0.909	0.963	0.597	0.989	0.997
5	0.96	0.977	0.965	0.963	0.517	0.969	1.000
Avg.	0.471	0.809	0.934	0.955	0.5614	0.9850	**0.997**
Std. dev.	0.459	0.376	0.055	0.022	0.0826	0.0115	0.002

Table 4 Model Accuracy of CIFAR-100 Tests (ResNet 18 network)

Training runs	Std	Ort	OCNN	RR
1	0.7402	0.7421	0.7416	0.7546
2	0.7405	0.7448	0.7455	0.7505
3	0.7394	0.7430	0.7430	0.7509
Avg.	0.7400	0.74330	0.74337	**0.7520**
Std. dev.	3.23×10^{-7}	1.89×10^{-6}	3.90×10^{-6}	5.11×10^{-6}

Table 5 Model Accuracy of CIFAR-10 Tests (ResNet 110 network)

Training runs	Std_{c1}	Ort_{c1}	RR_{c1}	Std_{c10}	Ort_{c10}	RR_{c10}	Std_{c10}	Ort_{c10}	RR_{c10}
	CIFAR 10						*CIFAR* 10.1		
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	0.846
3	0.908	0.9085	0.9028	0.92	0.9239	0.927	0.8305	0.8415	0.834
4	0.9031	0.9098	0.9052	0.9176	0.9257	0.928	0.8325	0.8395	0.8335
5	0.9104	0.9106	0.9108	0.9249	0.9259	0.9286	0.835	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	0.0008	0.0027	0.0046	0.0067

For the MI comparison in Fig. 4, we discuss models before and after fine-tuning separately, in order to illustrate the effects of regularization. We include a model with greedy layer-wise pretraining Pre, a regular model $\mathrm{Std_A}$, one with orthogonality constraints $\mathrm{Ort_A}$, and our regular model $\mathrm{RR_A}$, all before fine-tuning. For the model $\mathrm{RR_A}$ all layers are constrained to be recovered in the backward pass. We additionally include the version $\mathrm{RR^1_A}$, i.e., a model trained with only one loss term $\lambda_1 \left\| \mathbf{d}_1 - \mathbf{d}'_1 \right\|_2^2$, which means that only the input is constrained to be recovered. Thus, $\mathrm{RR^1_A}$ represents a simplified version of our approach which receives no constraints that intermediate results of the forward and backward pass should match. For $\mathrm{Ort_A}$, we

Table 6 Model L^2 Loss of Smoke Tests. Results for B_2: 10^7

Training runs	Pre_{AB_1}	Std_{AB_1}	RR_{AB_1}	Pre_{AB_2}	Std_{AB_2}	RR_{AB_2}
1	1013	338.4	197.7	5.19	1.93	1.62
2	411.4	380.2	203.6	3.63	1.63	1.55
3	621.5	2096.2	206	4.58	2.40	1.56
4	526.5	214.3	203.1	4.74	1.52	1.54
5	358.5	219.9	200.8	3.89	1.49	1.51
Avg.	586.18	649.8	**202.24**	4.41	1.80	**1.56**
Std. dev.	259.50	811.82	3.14	0.64	0.382	0.0381

Table 7 Forward and Reverse Pass Neural Networks for MNIST and *peak* Tests

Forward pass:

MNIST (Fig. 1): $I(784) \rightarrow FC\ (10) + b_1 \rightarrow O(10)$. *peak* (Figs. 2 and 4a): $I(784) \rightarrow FC(2) + b_1 \rightarrow O(2)$

peak (Fig. 4b): $I(784) \rightarrow relu(BN(FC(2) + b_1) \rightarrow O(2)$

peak (Fig. 4c): $I(784) \rightarrow relu(BN(FC(128) + b_1) \rightarrow relu(BN(FC(2) + b_2) \rightarrow O(2)$

Reverse pass:

MNIST (Fig. 1): $I(10) - b_1 \rightarrow FC\ (784) \rightarrow O(784)$ *peak* (Figs. 2 and 4a): $I(10) - b_1 \rightarrow FC\ (784) \rightarrow O(784)$

peak(Fig. 4b): $I(2) - b_1 \rightarrow FC(784) \rightarrow O(784)$

peak (Fig. 4c): $I(2) - b_2 \rightarrow relu(BN(FC(128)) - b_1 \rightarrow FC(784) \rightarrow O(784)$

Table 8 Forward and Reverse Pass Neural Network for MI Tests

Forward pass:

$I(12) \rightarrow tanh(FC(10) + b_1) = d_1 \rightarrow tanh(FC(7) + b_2) = d_2 \rightarrow tanh(FC(5) + b_3) = d_3$
$\rightarrow tanh(FC(4) + b_4) = d_4 \rightarrow tanh(FC(3) + b_5) = d_5 \rightarrow tanh(FC(2) + b_6) = d_6 \rightarrow O(2).$

Reverse pass (RR_A) :

$I(2) - b_6 \rightarrow tanh(FC(3)) - b_5 \rightarrow tanh(FC(4)) - b_4 \rightarrow tanh(FC(5)) - b_3 \rightarrow tanh(FC(7)) - b_2$
$\rightarrow tanh(FC(10)) - b_1 \rightarrow tanh(FC(12)) \rightarrow O(12)$.

Reverse pass (lRR_A) :

$d_6 - b_6 \rightarrow tanh(FC(3)) = d_5', d_5 - b_5 \rightarrow tanh(FC(4)) = d_4', d_4 - b_4 \rightarrow tanh(FC(5)) = d_3',$
$d_3 - b_3 \rightarrow tanh(FC(7)) = d_2', d_2 - b_2 \rightarrow tanh(FC(10)) = d_1', d_1 - b_1 \rightarrow tanh(FC(12)) = d_0'$

Table 9 Hyperparameters of MI Tests

Batch size	512	Learning rate	0.0004
λ	$RR_A : \lambda_{1\sim6} = 10^{-2}$		
	$lRR : \lambda_1 = 8 \times 10^{-2}, \lambda_2 = 6 \times 10^{-2}, \lambda_3 = 4 \times 10^{-2}, \lambda_4 = 2 \times 10^{-2}, \lambda_{1,2} = 10^{-2}$		
Training epochs	20000 for $Pre_{A/AA/AB}$, $RR_{A/AA/AB}$, $lRR_{A/AA/AB}$, Std $_{A/AA/AB}$, and Std_B		

Table 10 Forward and Reverse Pass Neural Network for Digit Generation Tests

Forward pass: $I(74) \rightarrow relu(BN(FC(1024))) \rightarrow relu(BN(FC(6272)) \rightarrow relu(BN(C(4,64,2))) \rightarrow relu(BN(C(4,1,2))) = I_r$
Reverse pass $\left(RR_{A/AA/AB}\right)$:
$I_r \rightarrow relu(BN(D(4,64,2))) \rightarrow relu(BN(D(4,128,2))) \rightarrow relu(BN(FC(1024))) \rightarrow relu(FC(74)) \rightarrow O(74)$.

Table 11 Hyperparameters of Digit Generation Tests

Batch size	256	λ	$\lambda_{c_1} = 2 \times 10^{-3}; \lambda_{c_{2,3}} = 1.5 \times 10^{-3}$
Learning rate	0.001 for RR^1 and *Std*	Training steps	150000 steps for RR^1 and *Std*

Table 12 Forward and Reverse Pass Neural Network for CIFAR-100 Tests (ResNet 18 network)

Example convolutional operation in forward pass:
$I(32,32,3) \rightarrow \cdots \rightarrow BN, relu \rightarrow d_{m_{in}} \rightarrow C(3,64,1) \rightarrow d_{m_{out}} \rightarrow BN \rightarrow d_{m+1_{in}} \rightarrow C(3,64,1) \rightarrow d_{m+1_{out}} \rightarrow \oplus \rightarrow \cdots$ (BN)
Example convolutional operation in reverse pass:
$d_{m+1_{out}} \rightarrow D(3,64,1) \rightarrow d'_{m+1_{in}}, d_{m_{out}} \rightarrow D(3,64,1) \rightarrow d'_{m_{in}}, \ldots\ldots$

Table 13 Hyperparameters of CIFAR-100 Tests (ResNet 18 network)

Batch size	128	Training epochs	400 epochs
Learning rate	0–60 epochs: 0.1; 60–120 epochs: 0.02		
	120–180 epochs: 0.004; 180–40 epochs: 0.0008		
λ	0–60 epochs: 10^{-4}; 60–120 epochs: 10^{-6}		
	120–180 epochs:10^{-3}; 180–40 epochs: 10^{-3}		

used the Spectral Restricted Isometry Property (SRIP) regularization [3],

$$\mathcal{L}_{\text{SRIP}} = \beta\sigma(W^T W - I), \tag{B17}$$

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

As explained in the main text, all layers of the first stage, i.e., from RR_A, RR^1_A, Ort_A , Pre_A and Std_A are reused for training the fine-tuned models without regularization, i.e., RR_AA, RR^1_AA, Ort_AA , Pre_AA and Std_AA. Likewise, all layers of the transfer task models RR_AB, RR^1_AB, Ort_AB , Pre_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^1_A) and the full loss version RR_A. Fig. 4b 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^1_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^1_A is only constrained to be able to regenerate its input, while the

Table 14 Forward and Reverse Pass Network for Texture-shape Tests

Forward pass: $I(224,224,3) \rightarrow relu(C(4,8,2)+b_1) \rightarrow relu(C(4,8,2)+b_2) \rightarrow relu(C(4,8,2)+b_3) \rightarrow relu(C(4,8,2)+b_4)$ $\rightarrow relu(C(4,8,2)+b_5) \rightarrow relu(C(4,8,1)+b_6) = I_r \rightarrow FC(16) \rightarrow O(16)$
Reverse pass: $I_r - b_6 \rightarrow relu(D(4,8,1)) - b_5 \rightarrow relu(D(4,8,2)) - b_4 \rightarrow relu(D(4,8,2)) - b_3 \rightarrow relu(D(4,8,2)) - b_2$ $\rightarrow relu(D(4,8,2)) - b_1 \rightarrow relu(D(4,3,2)) \rightarrow O(224,224,3)$

Table 15 Hyperparameters of Texture-shape Tests

Batch size	64	$\lambda_{1\sim6}$	5×10^{-6}
Learning rate	0.0001	Training epochs	200

Table 16 Forward and Reverse Pass Neural Network for CIFAR-10 Tests (ResNet 110 network)

Example residual block in forward pass: $I(32,32,3) \rightarrow \cdots \rightarrow BN, relu \rightarrow d_{m_{in}} \rightarrow C(3,64,1) \rightarrow d_{m_{out}} \rightarrow BN, relu \rightarrow d_{m+1_{in}} \rightarrow C(3,64,1) \rightarrow d_{m+1_{out}} \rightarrow \oplus \rightarrow \cdots$
Example residual block in reverse pass: $d_{m+1_{out}} \rightarrow D(3,64,1) \rightarrow d'_{m+1_{in}}, d_{m_{out}} \rightarrow D(3,64,1) \rightarrow d'_{m_{in}}, \ldots\ldots$

Table 17 Hyperparameters of CIFAR-10 Tests (ResNet 110 network)

Batch size	256	λ	10^{-8}	Training epochs	200 epochs
Learning rate	RR_{c1}, Ort_{c1} and Std_{c1} : 0.01				
	RR_{c10}, Ort_{c10} and Std_{c10} : 0.01 (0-80 epochs), 0.001 (80-120 epochs)				
			0.0001 (120-160 epochs), 0.00001 (160-200 epochs)		

full 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 $I(X;\mathcal{D})$ values of later layers for Std_A and Ort_A exhibit small values (points near the left side), while $I(\mathcal{D};Y)$ 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 dataset 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. Pre_A only focuses on decreasing the reconstruction loss, which results in high $I(X;\mathcal{D})$ values for early layers, and low $I(\mathcal{D};Y)$ values for later layers.

During the fine-tuning phase for task A (i.e., regularizers being disabled), all models focus on the output and maximize $I(\mathcal{D};Y)$. 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. 4c of the main paper. For model RR_{AA}, the $I(\mathcal{D};Y)$ value is higher than for Std_{AA}, Ort_{AA}, Pre_{AA} and RR^1_{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 Table 1 of the main paper. This model from a regular training run has large difficulties to adapt to the new task. Pre_A aims at extracting features from inputs and reconstructing them. Pre_{AB} outperforms Std_{AB}, which means features helpful for task B are extracted by Pre_A, however, it's hard to guide the feature extracting process. Model Ort_{AB} also performs worse than Std_B. RR_{AB} shows the best performance in this setting, demonstrating that our loss formulation yielded more generic features, improving the performance for related tasks such as the inverted outputs for B.

Table 18 Forward and Reverse Pass Neural Network for Smoke Tests

Generator forward pass: $I(16,16,1) \rightarrow relu(C(5,64,1)+b_1) \rightarrow UP \rightarrow relu(C(5,128,1)+b_2) \rightarrow UP \rightarrow relu(C(5,128,1)+b_3)$ $\rightarrow relu(C(5,64,1)+b_4) \rightarrow relu(C(5,32,1)+b_5) \rightarrow relu(C(5,1,1)+b_6) \rightarrow O(64,64,1) = I_r$.
Generator reverse pass: $I_r - b_6 \rightarrow relu(D(5,32,1)) - b_5 \rightarrow relu(D(5,64,1)) - b_4 \rightarrow relu(D(5,128,1)) - b_3 \rightarrow relu(D(5,128,1))$ $\rightarrow MP - b_2 \rightarrow relu(D(5,64,1)) \rightarrow MP - b_1 \rightarrow relu(D(5,1,1)) \rightarrow O(16,16,1)$.
Discriminator: $I(64,64,2) \rightarrow lrelu(BN(C(5,32,1)+b_1)) \rightarrow lrelu(BN(C(5,64,1)+b_2)) \rightarrow lrelu(BN(C(5,128,1)+b_3))$ $\rightarrow lrelu(BN(C(5,256,1)+b_4)) \rightarrow FC(1)+b_5 \rightarrow O(1)$.

Table 19 Autoencoder Architecture for Smoke Tests

Std_{AB_1} :
$I(64,64,1) \rightarrow relu(C(5,32,1)+b_1) \rightarrow relu(C(5,64,1)+b_2) \rightarrow relu(C(5,128,1)+b_3)$ $\rightarrow relu(C(5,128,1)+b_4) \rightarrow MP \rightarrow relu(C(5,64,1)+b_5) \rightarrow MP \rightarrow relu(C(5,1,1)+b_6)$ $\rightarrow relu(C(5,64,1)+b_7) \rightarrow UP \rightarrow relu(C(5,128,1)+b_8) \rightarrow UP \rightarrow relu(C(5,128,1)+b_9)$ $\rightarrow relu(C(5,64,1)+b_{10}) \rightarrow relu(C(5,32,1)+b_{11}) \rightarrow relu(C(5,1,1)+b_{12}) \rightarrow O(64,64,1)$
Pre_{AB_1} :
$I(64,64,1) \rightarrow relu(C(5,64,1)+b_1) \rightarrow MP \rightarrow relu(C(5,128,1)+b_2) \rightarrow MP \rightarrow relu(C(5,128,1)+b_3)$ $\rightarrow relu(C(5,64,1)+b_4) \rightarrow relu(C(5,32,1)+b_5) \rightarrow relu(C(5,1,1)+b_6)$ $\rightarrow relu(D(5,32,1)+b_7) \rightarrow relu(C(5,64,1)+b_8) \rightarrow relu(C(5,128,1)+b_9) \rightarrow relu(C(5,128,1)+b_{10})$ $\rightarrow UP \rightarrow relu(C(5,64,1)+b_{11}) \rightarrow UP \rightarrow relu(C(5,1,1)+b_{12}) \rightarrow O(64,64,1)$
RR_{AB_1} :
$I(64,64,1) - b_1 \rightarrow relu(D(5,32,1)) - b_2 \rightarrow relu(D(5,64,1)) - b_3 \rightarrow relu(D(5,128,1)) - b_4$ $\rightarrow relu(D(5,128,1)) \rightarrow MP - b_5 \rightarrow relu(D(5,64,1)) \rightarrow MP - b_6 \rightarrow relu(D(5,1,1))$ $\rightarrow relu(C(5,64,1)+b_7) \rightarrow UP \rightarrow relu(C(5,128,1)+b_8) \rightarrow UP \rightarrow relu(C(5,128,1)+b_9)$ $\rightarrow relu(C(5,64,1)+b_{10}) \rightarrow relu(C(5,32,1)+b_{11}) \rightarrow relu(C(5,1,1)+b_{12}) \rightarrow O(64,64,1)$
Pre_{AB_2}, Std_{AB_2} and RR_{AB_2}
$I(64,64,3) \rightarrow relu(C(5,32,1)+b_1) \rightarrow relu(C(5,64,1)+b_2) \rightarrow relu(C(5,128,1)+b_3)$ $\rightarrow relu(C(5,128,1)+b_4) \rightarrow MP \rightarrow relu(C(5,64,1)+b_5) \rightarrow MP \rightarrow relu(C(5,1,1)+b_6)$ $\rightarrow relu(C(5,64,1)+b_7) \rightarrow UP \rightarrow relu(C(5,128,1)+b_8) \rightarrow UP \rightarrow relu(C(5,128,1)+b_9)$ $\rightarrow relu(C(5,64,1)+b_{10}) \rightarrow relu(C(5,32,1)+b_{11}) \rightarrow relu(C(5,3,1)+b_{12}) \rightarrow O(64,64,3)$

Table 20 Hyperparameters of Smoke Tests

Batch size	64	Learning rate	0.0002	λ	$\lambda_1 = 10; \lambda_{2\sim 6} = 0.1$
Training epochs	40000 for Pre_A, RR_A and Std_A; 1000 for Pre_{AB_1}, $Pre_{AB_2}, RR_{AB_1}, RR_{AB_2}$, Std A_{AB_1} and Std_{AB_2}				

Unlike regular training, where MI consistently decreases from the first to the last layer, the MI of layers produced by our formulation can be higher than the MI of preceding layers. During our pretraining stage, information can be transported from the first layer to the last layer as in a regular training process. However, it can also be transported from the last layer to the previous layers via the reverse pass network. This allows previous layers to be adjusted via later layers, resulting in an increased MI. Compared to regular training, our pretraining achieves a stronger correlation between the input and output distribution across all layers. The fine-tuning stage afterward aims to increase $I(\mathcal{D}_7; Y)$ for a higher accuracy. As a result of the strong correlation between all layers, increasing $I(\mathcal{D}_7; Y)$ leads to inner layers exhibiting an increase in MI.

Table 21 Hyperparameters of Weather Forecasting Tests

Batch size	32
λ	10^{-12} for ERA dataset test and CMIP dataset with dropout 10^{-10} for CMIP dataset without dropout
Learning rate	*Std*: begin with 5^{-5} for all tests *RR* : begin with 1^{-4} for ERA dataset with dropout; begin with 5^{-4} for CMIP dataset with dropout; begin with 5^{-4} for ERA dataset without dropout; begin with 5^{-5} for CMIP dataset without dropout.

B.2 Disentangled representations

The InfoGAN approach [9] 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 [74]. The previous section and Fig. 4b of the main paper demonstrated that models from our pretraining (both RR^1_A and RR_A) can increase the mutual information between network inputs and outputs. Intuitively, the pretraining explicitly constrains the model to recover an input given an output, which directly translates into an increase in mutual information between input and output distributions compared to regular training runs. For highlighting how our pretraining can yield disentangled representations (as discussed in the later paragraphs of Sect. 4 of the main text), we follow the experimental setup of InfoGAN [9]: the input dimension of our network is 74, containing 1 ten-dimensional category code c_1, 2 continuous latent codes $c_2, c_3 \sim \mathcal{U}(-1, 1)$ and 62 noise variables. Here, $\mathcal{U}$ denotes a uniform distribution.

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 pretraining 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 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_1 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 loss term for c_1 we use a slightly larger λ factor (by 33%) than for c_2 and c_3. Details of our results are shown in Fig. 19. Models are trained using a GAN loss [25] as the loss function for the outputs.

Analysis of results In Fig. 19, we show additional results for the disentangling test case. It is visible that our pretraining of the RR^1 model yields distinct and meaningful latent space 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 the pretraining can shape the latent space, and in addition to an intuitive embedding of dominant features, yield a disentangled representation.

Appendix C Details of experimental results

To ensure reproducibility, source code and data for all tests will be published. Runtimes were measured on a machine with Nvidia GeForce GTX 1080 Ti GPUs and an Intel Core i7-6850K CPU.

C.1 Texture-shape benchmark

Training details

All training data of the texture-shape tests were obtained from [21]. The stylized dataset contains 1280 images, 1120 images are used as training data, and 160 as test data. Both edge and filled datasets 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.

Analysis of results For a detailed comparison, we list per-class accuracy of stylized data training runs for Ort_TS, Std_TS, Pre_TS and RR_TS in Fig. 20. RR_TS outperforms the other three models for most of the classes. RR_TS requires an additional 41.86% for training compared to Std_TS, but yields a 23.76% higher performance. (Training times for these models are given in the supplementary document.)

Table 22 $I(X;D_m)$ and $I(D_m;Y)$ Values of All MI Models

	Results of 5 runs	$I(X;D_2)$	$I(X;D_3)$	$I(X;D_4)$	$I(X;D_5)$	$I(X;D_6)$	$I(X;D_7)$	$I(D_2;Y)$	$I(D_3;Y)$	$I(D_4;Y)$	$I(D_5;Y)$	$I(D_6;Y)$	$I(D_7;Y)$
Std_A	Avg.	11.9992	11.9879	11.7279	7.9088	2.9612	1.2367	0.9992	0.9992	0.9973	0.9947	0.9786	0.9554
	Std. dev.	0.0011	0.0138	0.2007	1.5376	0.8695	0.0973	0.0000	0.0000	0.0012	0.0031	0.0112	0.0233
Pre_A	Avg.	11.6338	11.5846	11.5518	11.4495	10.8126	4.4459	0.9468	0.9371	0.9289	0.9037	0.6987	0.0188
	Std. dev.	0.2079	0.1342	0.1192	0.0732	0.2083	0.4858	0.0295	0.0149	0.0108	0.0140	0.0732	0.0122
RR_A^1	Avg.	11.5487	10.3822	8.8948	7.8285	6.1742	3.9528	0.9564	0.8418	0.7463	0.6921	0.5969	0.4937
	Std. dev.	0.9988	1.1225	0.5715	0.7409	0.7583	0.2322	0.0940	0.0785	0.0440	0.0131	0.0652	0.0736
lRR_A	Avg.	11.5521	6.7737	4.0047	4.0087	3.8456	3.7120	0.9388	0.6372	0.6013	0.6002	0.6008	0.5997
	Std. dev.	0.0558	2.3736	0.1872	0.2787	0.2033	0.0909	0.0070	0.0334	0.0244	0.0233	0.0231	0.0236
RR_A	Avg.	5.2845	4.8838	4.2668	3.9356	4.7603	4.6238	0.5654	0.5369	0.5358	0.5182	0.5261	0.5336
	Std. dev.	0.4545	0.1569	0.3115	0.1279	0.0783	0.0173	0.0337	0.0039	0.0111	0.0060	0.0115	0.0031
Ort_A	Avg.	11.9976	11.9702	11.4784	7.7023	2.6314	1.0755	0.9992	0.9992	0.9955	0.9849	0.9681	0.9493
	Std. dev.	0.0051	0.0505	0.5529	0.8500	0.4071	0.0629	0.0000	0.0000	0.0054	0.0128	0.0254	0.0325
Std_{AA}	Avg.	11.9984	11.9770	11.7029	7.3237	2.6905	1.2204	0.9992	0.9990	0.9973	0.9932	0.9752	0.9583
	Std. dev.	0.0021	0.0292	0.2305	1.2948	0.3707	0.1700	0.0000	0.0004	0.0009	0.0080	0.0169	0.0242
Pre_{AA}	Avg.	8.5178	6.7796	5.8199	4.9982	4.6117	3.8572	0.6125	0.4349	0.3312	0.2596	0.2187	0.1649
	Std. dev.	1.2999	0.8952	0.4626	0.3502	0.2979	0.3172	0.1350	0.1015	0.0423	0.0346	0.0272	0.0310
RR_{AA}^1	Avg.	11.9989	11.8699	11.0450	9.0914	3.9232	1.3239	0.9992	0.9934	0.9813	0.9778	0.9404	0.9054
	Std. dev.	0.0016	0.2075	0.9404	1.8047	1.3972	0.4193	0.0000	0.0118	0.0217	0.0374	0.0420	0.0528
lRR_{AA}	Avg.	11.9527	10.2473	8.1597	6.7212	2.4402	1.1113	0.9992	0.9743	0.9662	0.9917	0.9859	0.9706
	Std. dev.	0.0472	1.6567	2.3965	1.8774	0.4665	0.0512	0.0000	0.0227	0.0124	0.0047	0.0110	0.0197
RR_{AA}	Avg.	11.9200	11.5621	10.6095	7.7398	2.2372	1.0416	0.9992	0.9988	0.9985	0.9975	0.9906	0.9875
	Std. dev.	0.0540	0.1463	0.5465	0.8093	1.0104	0.0505	0.0000	0.0005	0.0007	0.0024	0.0073	0.0046
Ort_{AA}	Avg.	11.9970	11.9727	11.3343	6.1690	1.9642	1.0985	0.9992	0.9991	0.9971	0.9832	0.9625	0.9477
	Std. dev.	0.0061	0.0504	0.6796	1.2529	0.5865	0.0891	0.0000	0.0002	0.0032	0.0166	0.0269	0.0320
Std_{AB}	Avg.	11.9989	11.9794	10.2985	3.0652	1.2677	0.5283	0.9992	0.9988	0.9029	0.4041	0.3923	0.3860
	Std. dev.	0.0019	0.0116	1.5095	4.0345	1.8717	0.7445	0.0000	0.0006	0.0918	0.5359	0.5329	0.5288
Pre_{AB}	Avg.	8.1364	6.0174	5.6115	4.9866	4.5760	3.7471	0.5445	0.2819	0.2525	0.2119	0.1885	0.1429
	Std. dev.	1.2778	0.5196	0.3912	0.2972	0.2805	0.2942	0.1498	0.0481	0.0276	0.0165	0.0158	0.0133
RR_{AB}^1	Avg.	11.9895	11.7982	9.7800	7.4955	3.2139	1.3122	0.9992	0.9901	0.9644	0.9661	0.9472	0.9308
	Std. dev.	0.0075	0.1710	1.0207	1.4405	1.2750	0.1537	0.0000	0.0131	0.0310	0.0357	0.0349	0.0344
lRR_{AB}	Avg.	11.9047	10.8970	8.3860	5.4257	1.7110	1.0694	0.9989	0.9915	0.9813	0.9929	0.9812	0.9735
	Std. dev.	0.0890	0.8474	1.9089	1.1511	0.4913	0.0529	0.0006	0.0089	0.0145	0.0085	0.0209	0.0235
RR_{AB}	Avg.	11.9093	11.5761	9.4227	5.6190	2.2286	1.0100	0.9992	0.9992	0.9984	0.9989	0.9962	0.9935
	Std. dev.	0.0957	0.1450	1.7109	1.4609	0.4611	0.0091	0.0000	0.0000	0.0014	0.0004	0.0032	0.0035
Ort_{AB}	Avg.	11.9946	11.9546	8.9865	3.2802	1.4296	0.8571	0.9992	0.9985	0.8048	0.7851	0.7756	0.7675
	Std. dev.	0.0104	0.0561	3.3571	2.7102	0.9848	0.4858	0.0000	0.0007	0.4279	0.4391	0.4342	0.4296
Std_B	Avg.	12.0000	11.9913	11.7602	8.6184	3.2490	1.3171	0.9992	0.9990	0.9975	0.9880	0.9458	0.9160
	Std. dev.	0.0000	0.0113	0.3400	2.0542	1.2740	0.2025	0.0000	0.0003	0.0007	0.0114	0.0476	0.0565

All models saturated, i.e., training Std_{TS} or Ort_{TS} longer does not increase classification accuracy any further. We also investigated how much we can reduce model size when using our pretraining in comparison to the baselines. A reduced model only uses 67.94% of the parameters, while still outperforming Ort_{TS}.

C.2 Smoke generation

Training details The dataset of the smoke simulation was generated with a Navier–Stokes solver from an open-source library [68]. We generated 20 randomized simulations with 120 frames each, with 10% of the data being

Table 23 Per Category Accuracy of RR_{TS} for Texture-shape Tests

Datasets		Stylized data																	Edge data	Filled data
Model	Runs	Airplane	Bear	Bicycle	Bird	Boat	Bottle	Car	Cat	Chair	Clock	Dog	Elephant	Keyboard	Knife	Oven	Truck	Acc.	Acc.	Acc.
RR_{TS}	0	0.9	0	0.7	0.6	0	0.9	0.6	0.8	0.2	0.7	0	1	0.6	0.5	0	0.8	0.5214	0.3063	0.4500
	1	0.9	0.6	0	0.4	0.7	1	0	0.9	0.5	0	0.4	0	0.7	0.5	0.9	1	0.5145	0.2813	0.4375
	2	1	0.9	1	0.6	0.8	0	0.7	0	0.7	0.9	0.5	0.8	0	0	1	0.9	0.6043	0.1938	0.4500
	3	0	0.8	1	0.9	0.5	0	1	0	0	0	0.3	0	0.9	0.7	0.9	0	0.4402	0.2125	0.3313
	4	0.8	0.9	0	0.7	0.5	1	0	0.5	0	0.9	0	1	0.4	0.7	0.8	1	0.5746	0.2125	0.3625
	5	0.9	0.7	0.8	0	0	1	1	0	1	0.8	0.6	0	1	0.9	0.8	1	0.6471	0.2500	0.4563
	6	0.9	0.9	0.9	0.9	0.4	0.9	0.8	0.4	0.6	0.8	0.3	0	0	0	0	1	0.5465	0.2563	0.4375
	7	1	0.9	0.8	0.7	0.6	0	0.3	0.7	0	0.9	0.6	0.4	0	0.8	0.8	1	0.5897	0.2813	0.4313
	8	0.8	0.6	0	0.5	0.5	0	0.6	0.8	0	0.8	0.3	0	0.6	0.8	0.7	0.8	0.4850	0.1813	0.3563
	9	0.7	0.8	0	0.8	0	1	0	0.9	0	0.9	0.5	1	0.9	0.7	0.7	0	0.5440	0.1938	0.3625
	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
	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

Table 24 Per Category Accuracy of Ort_{TS} for Texture-shape Tests

Datasets		Stylized data																	Edge data	Filled data
Model	Runs	Airplane	Bear	Bicycle	Bird	Boat	Bottle	Car	Cat	Chair	Clock	Dog	Elephant	Keyboard	Knife	Oven	Truck	Acc.	Acc.	Acc.
Ort_{TS}	0	0.7	0	0.8	0.3	0	1	0.7	0.8	0.5	0	0.4	0.5	0.7	0.5	0.9	0	0.4987	0.1688	0.4188
	1	0.6	0.3	0.6	0.1	0.4	1	0.4	0.5	0.5	1	0.4	0.1	0.6	0	0.6	0.8	0.4877	0.2688	0.3313
	2	0.7	0.4	0.7	0.1	0.8	0	0.8	0	0	0.6	0	0.6	0	0.5	0.6	1	0.42	0.0875	0.3313
	3	1	0.9	0.4	0.2	0.7	1	0	0.3	0.8	0.7	0.2	0.4	0.7	0	0.6	1	0.543	0.2188	0.4
	4	0	0.2	0	0.7	0.9	1	0.9	0.6	1	0	0	0.3	0.8	0.4	0.7	1	0.521	0.1813	0.3688
	5	0.5	0.2	0.7	0.4	0.1	0.9	0.3	0.4	0	0.4	0.1	0.2	0.3	0.5	0.6	0.6	0.3864	0.3438	0.3375
	6	0.9	0	0.9	0.5	0	1	0.9	0.3	0	0.7	0.4	0	0	0.9	0	1	0.4683	0.1375	0.3938
	7	0	0	0.9	0	0.6	0	0.4	0.7	0.9	0	0.3	0	0.8	0	0.8	1	0.3952	0.2375	0.3313
	8	0	0.7	0.9	0.7	0.7	1	0	0.6	0	0.9	0.7	0	0.8	0.6	0.8	0	0.525	0.2438	0.3813
	9	0.6	0.8	0.9	0.4	0.2	1	0.6	0	0.7	0	0.6	0	0.7	0.7	0	0	0.4554	0.1438	0.4250
	Avg.	0.5	0.35	0.68	0.34	0.44	0.79	0.5	0.42	0.44	0.43	0.31	0.21	0.54	0.41	0.56	0.64	0.47007	0.2031	0.3719
	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

Table 25 Per Category Accuracy of Std_{TS} for Texture-shape Tests

Datasets		Stylized data																Edge data	Filled data
Model	Runs	Airplane	Bear	Bicycle	Bird	Boat	Bottle	Car	Cat	Chair	Clock	Dog	Elephant	Keyboard	Knife	Oven	Truck	Acc.	Acc.
Std_{TS}	0	0.8	0.4	0	0.8	0.6	1	0.9	0.3	0.3	0	0.8	0	0	0	0	0	0.3700	0.1250	0.325
	1	0.9	0.1	0.7	0.5	0.7	0.9	0.3	0.1	0.5	0.9	0.6	0.7	0	0	0.8	0.8	0.5228	0.1688	0.3875
	2	1	0	0	0.8	0.8	0	0.6	0	0.4	0.9	0	0.7	0.8	0	0	0	0.3679	0.1625	0.2625
	3	0	0.2	1	0.2	0.1	0	0	0.8	0	0.9	0.4	0	0.9	0.4	0.9	1	0.4216	0.1438	0.2688
	4	0.8	0	0	0	0.1	1	0.7	0	0	0	0.1	0.9	0.7	0.7	0.4	0	0.3460	0.1438	0.2813
	5	0.6	0.6	0	0.7	0	0	0	0.8	0.8	0.9	0.2	0	0.8	0.7	0.8	1	0.4937	0.1813	0.3438
	6	0.8	0.4	0.6	0.5	0.7	0	0.8	0.8	0.6	1	0.8	0	0.6	0	0.3	0.8	0.5409	0.1250	0.3750
	7	0	0	0.8	0.6	0	1	0	0	0.7	0.9	0.5	0	0.9	0	0.7	0	0.3812	0.1750	0.3375
	8	0.5	0.8	0.6	0.4	0.6	0.6	1	0.4	0.3	0.7	0.3	0.7	0.5	0.3	0.9	0	0.5313	0.2313	0.4438
	9	0.7	0	0	0.7	0.4	0	0.7	0.6	0.5	0.9	0.3	0.9	0	0	0.6	0.8	0.4442	0.1188	0.2813
	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
	Std. dev.	0.3510	0.2877	0.4057	0.2616	0.3197	0.4882	0.3916	0.3490	0.2685	0.3814	0.2749	0.4175	0.3795	0.2961	0.3471	0.4695	0.0753	0.0341	0.0593

used for training. 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_1, and generating samples from a real-world smoke dataset as task B_2. The smoke capture dataset for B_2 contains 2500 smoke images from the ScalarFlow dataset [17], and we again used 10% of these images as training dataset.

Task A We use a fully convolutional CNN-based architecture for generator and discriminator networks. 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 [79], $\mathrm{RR_A}$ and $\mathrm{Std_A}$ are trained with a non-saturating GAN loss, feature space loss and L^2 loss as base loss function. All generator layers are involved in the pretraining loss. As greedy layer-wise autoencoder pretraining is not compatible with adversarial training, we pretrain $\mathrm{Pre_A}$ for reconstructing the high-resolution data instead.

Task $\mathrm{B_1}$ All encoder layers are initialized from $\mathrm{RR_A}$ and $\mathrm{Std_A}$ when training RR_{AB_1} and Std_{AB_1}. It is worth noting that the reverse pass of the generator is also constrained when training $\mathrm{Pre_A}$ and $\mathrm{RR_A}$. Therefore, both encoder and decoder are initialized with parameters from $\mathrm{Pre_A}$ and $\mathrm{RR_A}$ when training Pre_{AB_1} and RR_{AB_1}, respectively. This is not possible for a regular network like Std_{AB_1}, as the weights obtained with a normal training run are not suitable to be transposed. Hence, the de-convolutions of Std_{AB_1} are initialized randomly.

Task B_2 As the dataset for the task B_2 is substantially different and contains RBG images (instead of single channel gray-scale images), we choose the following set-ups for the $\mathrm{RR_A}$, $\mathrm{Pre_A}$ and $\mathrm{Std_A}$ models: parameters from all six layers of $\mathrm{Std_A}$ and $\mathrm{RR_A}$ are reused for initializing decoder part of Std_{AB_2} and RR_{AB_2}, parameters from all six layers of $\mathrm{Pre_A}$ are reused for initializing the encoder part of Pre_{AB_2}. Specially, when initializing the last layer of Pre_{AB_2}, Std_{AB_2} and RR_{AB_2}, we copy and stack the parameters from the last layer of $\mathrm{Pre_A}$, $\mathrm{Std_A}$ and $\mathrm{RR_A}$, respectively, into three channels to match the dimensions of the outputs for task B_2. Here, the encoder part of RR_{AB_2} and the decoder of Pre_{AB_2} are not initialized with $\mathrm{RR_A}$ and $\mathrm{Pre_A}$, due to the significant gap between training datasets of task B_1 and task B_2. Our experiments show that only initializing the decoder part of RR_{AB_2} (avg. loss:$1.56e7$, std. dev.:$3.81e5$) outperforms initializing both encoder and decoder (avg. loss:$1.82e7 \pm 2.07e6$), and only initializing the encoder part of Pre_{AB_2} (avg. loss:$4.41e7 \pm 6.36e6$) outperforms initializing both encoder and decoder (avg.

Table 26 Per Category Accuracy of Pre_{TS} for Texture-shape Tests

Datasets		Stylized data															Edge data	Filled data
Model	Runs	Airplane	Bear	Bicycle	Bird	Boat	Bottle	Car	Cat	Chair	Clock	Dog	Elephant	Keyboard	Knife	Oven	Truck	Acc.	Acc.	Acc.
Pre_{TS}	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0.0625	0.0625	0.0875
	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0.0625	0.0625	0.075
	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0.0625	0.0625	0.06875
	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0.0625	0.0625	0.03125
	4	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0.0875	0.0625	0.06875
	5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0.0625	0.0625	0.0625
	6	1	0	0	0	0	0	0.1	0.2	0	0	0	0	0	0	0	0	0.08125	0.0625	0.1125
	7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0625	0.0625	0.0875
	8	0	0	0	1	0	0	0.1	0	0	0	0	0	0	0	0	0	0.06875	0.0625	0.08125
	9	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.0625	0.0625	0.0625
Avg.		0.2	0.08	0	0.2	0.1	0.1	0.12	0.22	0	0	0	0	0	0	0.06	0	0.0675	0.0625	0.07375
Std. dev.		0.4216	0.2530	0.0000	0.4216	0.3162	0.3162	0.3120	0.4158	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.1897	0.0000	0.0092	0.0000	0.0212

loss:$9.42e7 \pm 6.11e7$). We believe the reason is that initializing both the encoder and decoder parts makes it more difficult to adjust the parameters for the new dataset that is very different from the dataset of the source task.

Analysis of results Example outputs of Pre_{AB_1}, Std_{AB_1} and RR_{AB_1} are displayed in Fig. 11. We can observe that the results of Pre_{AB_1} are blurry, indicating that features learned from task A with greedy layer-wise pretraining are not successfully transferred to task B_1. Likewise, Std_{AB_1} cannot provide the smoke frame with correct details, while RR_{AB_1} produces the closest results to the reference. We similarly illustrate the behavior of the transfer learning task B_2 for images of real-world fluids. This example likewise uses an autoencoder structure. Visual comparisons are provided in Fig. 12 in the main paper. Similar to task B_1, Pre_{AB_2} and Std_{AB_2} cannot recover the smoke details properly, e.g., there are noisy colors in the results of Std_{AB_2}. On the other hand, the results of RR_{AB_2} are closer to the reference. Overall, these findings demonstrate the benefits of our pretraining for GANs, as well as its potential to obtain more generic features from synthetic datasets that can be used for tasks involving real-world data.

C.3 Weather forecasting

Training details The weather forecasting scenario discussed in the main text follows the methodology of the *WeatherBench* benchmark [60]. This benchmark contains 40 years of data from the ERA reanalysis project [31] which was re-sampled to a 5.625° resolution, yielding 32×64 grid points in ca. two-hour intervals. Data from the year of 1979 to 2015 (i.e., 324192 samples) are used for training. The benchmark also contains 165 years of historical simulation data from [19], and data from the year 1850 to 2005 (i.e., 224672 samples) are used for training. All RMSE measurements are latitude-weighted to account for area distortions from the spherical projection.

The neural networks for the forecasting tasks employ a ResNet architecture with 19 layers, all of which contain 128 features with 3×3 kernels (apart from 7×7 in the first layer). All layers use batch normalization, leaky ReLU activation (tangent 0.3), and dropout with strength 0.1. As inputs, the model receives feature-wise concatenated data from the WeatherBench data for 3 consecutive time steps, i.e., t, $t - 6h$, and $t - 12h$, yielding 117 channels in total. The last convolution jointly generates all three output fields, i.e., pressure at 500 hPa (Z500), temperature at 850 hPa (T850), and the 2-meter temperature (T2M). Following [59], the learning rate was decreased by a factor of 5 when the loss did not decrease for two epochs, and the training is

Table 27 Performance Comparison of Texture-shape Models

Models	Parameters	Cost (min/epoch)	Acc.
Std_{TS}	11840	0.387	0.442
Ort_{TS}	11840	0.407	0.470
RR_{TS}	11840	0.627	0.547
$RR_{TS} - reduced$	8044	0.552	0.483

terminated after 5 epochs without improvements. It is worth pointing out that for networks with large sizes, such as this weather forecasting test with 6.36M trainable parameters, the training time difference between RR and Std is negligible, with about 68.01 and 68.44 min/epoch correspondingly.

Analysis of results In addition to the quantitative results given in the main text, Fig. 21 contains additional example visualizations from the test dataset. A visualization of the spatial error distribution w.r.t. ground truth results is also shown. It becomes apparent that our pretraining achieves reduced errors across the whole range of samples. Both temperature targets contain a larger number of smaller-scale features than the pressure fields. The improvements of MAE from our pretraining approach are significant (c.a. 3% $\sim$10% across all cases), which represents a substantial improvement. The learning objective is highly non-trivial, and the improvements were achieved with the same limited set of training data. Being very easy to integrate into existing training pipelines, these results indicate that the proposed pretraining methodology has the potential to yield improved learning results for a wide range of problem settings.

The following document contains supplementary tables for detailing network architectures, training hyper-parameters, and numeric results of experiments discussed in the main document, i.e., Tables 2, 3, 8, 9, and 22 for MI tests, Tables 4, 5, 12, 13, 16, and 17 for CIFAR tests, Tables 6, 18, 19, and 20 for smoke tests, Table 7 for MNIST and peak tests, Tables 10 and 11 for digit generation, Tables 14, 15, 23, 24, 25, 26, and 27 for texture-shape tests, Table 21 for weather forecasting test.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was funded by the ERC-2019-COG-863850 SpaTe project.

Data availability All data generated or analyzed during this study are included in this published article and its supplementary material.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Alberti M, Seuret M, Ingold R, et al (2017) A pitfall of unsupervised pre-training. arXiv preprint arXiv:1703.04332
2. Ardizzone L, Kruse J, Wirkert S, et al (2018) Analyzing inverse problems with invertible neural networks. arXiv preprint arXiv:1808.04730
3. Bansal N, Chen X, Wang Z (2018) Can we gain more from orthogonality regularizations in training deep cnns? In: Advances in Neural Information Processing Systems, Curran Associates Inc., pp 4266–4276
4. Bengio Y, Lamblin P, Popovici D, et al (2007) Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems, pp 153–160
5. Cai TT, Ma Z, Wu Y (2013) Sparse pca: optimal rates and adaptive estimation. Ann Stat 41(6):3074–3110
6. Caron M, Bojanowski P, Joulin A, et al (2018) Deep clustering for unsupervised learning of visual features. In: Proceedings of the European Conference on Computer Vision (ECCV), pp 132–149
7. Caron M, Bojanowski P, Mairal J, et al (2019) Unsupervised pre-training of image features on non-curated data. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 2959–2968
8. Chen T, Kornblith S, Norouzi M, et al (2020a) A simple framework for contrastive learning of visual representations. In: International Conference on Machine Learning, PMLR, pp 1597–1607
9. Chen X, Duan Y, Houthooft R, et al (2016) Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in Neural Information Processing Systems, pp 2172–2180
10. Chen X, Fan H, Girshick R, et al (2020b) Improved baselines with momentum contrastive learning. arXiv preprint arXiv:2003.04297
11. Chen Y, Li J, Jiang H, et al (2022) Metalr: Layer-wise learning rate based on meta-learning for adaptively fine-tuning medical pre-trained models. arXiv preprint arXiv:2206.01408
12. Cho K, Van Merriënboer B, Gulcehre C, et al (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078
13. Cui J, Zhong Z, Liu S, et al (2021) Parametric contrastive learning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 715–724
14. Ding H, Zhou SK, Chellappa R (2017) Facenet2expnet: Regularizing a deep face recognition net for expression recognition. In: 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), IEEE, pp 118–126
15. Dinh L, Sohl-Dickstein J, Bengio S (2016) Density estimation using real nvp. arXiv preprint arXiv:1605.08803

16. Du M, Liu N, Hu X (2018) Techniques for interpretable machine learning. arXiv preprint arXiv:1808.00033
17. Eckert ML, Um K, Thuerey N (2019) Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. ACM Trans Graph TOG 38(6):239
18. Erhan D, Courville A, Bengio Y, et al (2010) Why does unsupervised pre-training help deep learning? In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp 201–208
19. Eyring V, Bony S, Meehl GA et al (2016) Overview of the coupled model intercomparison project phase 6 (cmip6) experimental design and organization. Geosci Model Dev 9(5):1937–1958
20. Frankle J, Carbin M (2018) The lottery ticket hypothesis: finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635
21. Geirhos R, Rubisch P, Michaelis C, et al (2018) Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. arXiv preprint arXiv:1811.12231
22. Ghazal TM, Hussain MZ, Said RA, et al (2021) Performances of k-means clustering algorithm with different distance metrics. Intell Autom Soft Comput
23. Gidaris S, Singh P, Komodakis N (2018) Unsupervised representation learning by predicting image rotations. arXiv preprint arXiv:1803.07728
24. Gomez AN, Ren M, Urtasun R, et al (2017) The reversible residual network: Backpropagation without storing activations. In: Advances in Neural Information Processing Systems, pp 2214–2224
25. Goodfellow I, Pouget-Abadie J, Mirza M, et al (2014) Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp 2672–2680
26. Gopalakrishnan K, Khaitan SK, Choudhary A et al (2017) Deep convolutional neural networks with transfer learning for computer vision-based data-driven pavement distress detection. Constr Build Mater 157:322–330
27. Hanafy YA, Mashaly M, Abd El Ghany MA (2021) An efficient hardware design for a low-latency traffic flow prediction system using an online neural network. Electronics 10(16):1875
28. Hanson SJ, Pratt LY (1989) Comparing biases for minimal network construction with back-propagation. In: Advances in Neural Information Processing Systems, pp 177–185
29. Hasan BMS, Abdulazeez AM (2021) A review of principal component analysis algorithm for dimensionality reduction. J Soft Comput Data Min 2(1):20–30
30. He K, Zhang X, Ren S, et al (2016) Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, pp 770–778
31. Hersbach H, Bell B, Berrisford P et al (2020) The era5 global reanalysis. Q J R Meteorol Soc 146(730):1999–2049
32. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
33. Hoffmann H (2007) Kernel pca for novelty detection. Pattern Recognit 40(3):863–874
34. Huang JJ, Dragotti PL (2022) Winnet: wavelet-inspired invertible network for image denoising. IEEE Trans Image Process
35. Huang L, Liu X, Lang B, et al (2018) Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In: Thirty-Second AAAI Conference on Artificial Intelligence
36. Jacobsen JH, Smeulders A, Oyallon E (2018) i-revnet: deep invertible networks. arXiv preprint arXiv:1802.07088
37. Jean N, Wang S, Samar A, et al (2019) Tile2vec: Unsupervised representation learning for spatially distributed data. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 3967–3974
38. Jia K, Tao D, Gao S, et al (2017) Improving training of deep neural networks via singular value bounding. In: IEEE Conference on Computer Vision and Pattern Recognition, pp 4344–4352
39. Jing J, Deng X, Xu M, et al (2021) Hinet: deep image hiding by invertible network. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 4733–4742
40. Kawaguchi K, Kaelbling LP, Bengio Y (2017) Generalization in deep learning. arXiv preprint arXiv:1710.05468
41. Kazhdan M, Funkhouser T, Rusinkiewicz S (2003) Rotation invariant spherical harmonic representation of 3 d shape descriptors. In: Symposium on Geometry Processing, pp 156–164
42. Kim D, Choi J (2022) Unsupervised representation learning for binary networks by joint classifier learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 9747–9756
43. Kim T, Yun SY (2022) Revisiting orthogonality regularization: a study for convolutional neural networks in image classification. IEEE Access
44. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
45. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. Tech. rep, Citeseer
46. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp 1097–1105
47. Kulkarni P, Zepeda J, Jurie F, et al (2015) Learning the structure of deep architectures using l1 regularization. In: British Machine Vision Conference, 2015
48. Lee HY, Huang JB, Singh M, et al (2017) Unsupervised representation learning by sorting sequences. In: Proceedings of the IEEE International Conference on Computer Vision, pp 667–676
49. Li J, Zhou P, Xiong C, et al (2020) Prototypical contrastive learning of unsupervised representations. arXiv preprint arXiv:2005.04966
50. Li M, Wang Y, Lin Z (2022) Cerdeq: Certifiable deep equilibrium model. In: Int Conf Mach Learn PMLR, pp 12,998–13,013
51. Linting M, Meulman JJ, Groenen PJ et al (2007) Nonlinear principal components analysis: introduction and application. Psychol Methods 12(3):336
52. Loshchilov I, Hutter F (2017) Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101
53. Madono K, Tanaka M, Onishi M et al (2021) Sia-gan: scrambling inversion attack using generative adversarial network. IEEE Access 9:129385–129393
54. Mahendran A, Vedaldi A (2016) Visualizing deep convolutional neural networks using natural pre-images. Int J Comput Vis 120(3):233–255
55. Momeny M, Neshat AA, Hussain MA et al (2021) Learning-to-augment strategy using noisy and denoised data: improving generalizability of deep cnn for the detection of covid-19 in x-ray images. Comput Biol Med 136(104):704
56. Neyshabur B, Bhojanapalli S, McAllester D, et al (2017) Exploring generalization in deep learning. In: Advances in Neural Information Processing Systems, pp 5947–5956
57. Ozay M, Okatani T (2016) Optimization on submanifolds of convolution kernels in cnns. arXiv preprint arXiv:1610.07008
58. Rasmus A, Berglund M, Honkala M, et al (2015) Semi-supervised learning with ladder networks. In: Advances in Neural Information Processing Systems, pp 3546–3554
59. Rasp S, Thuerey N (2021) Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: a new model for weatherbench. J Adv Model Earth Syst, p e2020MS002405

60. Rasp S, Dueben PD, Scher S, et al (2020) Weatherbench: a benchmark dataset for data-driven weather forecasting. arXiv preprint arXiv:2002.00469
61. Recht B, Roelofs R, Schmidt L, et al (2019) Do imagenet classifiers generalize to imagenet? In: International Conference on Machine Learning
62. Reddi SJ, Kale S, Kumar S (2019) On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237
63. Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, pp 234–241
64. Shwartz-Ziv R, Tishby N (2017) Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810
65. Sinaga KP, Yang MS (2020) Unsupervised k-means clustering algorithm. IEEE Access 8:80716–80727
66. Srivastava N, Hinton G, Krizhevsky A et al (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
67. Teng Y, Choromanska A (2019) Invertible autoencoder for domain adaptation. Computation 7(2):20
68. Thuerey N, Pfaff T (2018) MantaFlow. *http://mantaflow.com*
69. Tishby N, Zaslavsky N (2015) Deep learning and the information bottleneck principle. In: 2015 IEEE Information Theory Workshop (ITW), IEEE, pp 1–5
70. Torrey L, Shavlik J (2010) Transfer learning. In: Handbook of research on machine learning applications and trends: algorithms, methods, and techniques. IGI Global, pp 242–264
71. Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. In: Advances in Neural Information Processing Systems, pp 5998–6008
72. Vincent P, Larochelle H, Lajoie I, et al (2010) Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 11(12)
73. Wall ME, Rechtsteiner A, Rocha LM (2003) Singular value decomposition and principal component analysis. In: A Practical Approach to Microarray Data Analysis. Springer, pp 91–109
74. Walters-Williams J, Li Y (2009) Estimation of mutual information: A survey. In: International Conference on Rough Sets and Knowledge Technology, Springer, pp 389–396
75. Wang J, Chen Y, Chakraborty R, et al (2020) Orthogonal convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 11,505–11,515
76. Weigend AS, Rumelhart DE, Huberman BA (1991) Generalization by weight-elimination with application to forecasting. In: Advances in Neural Information Processing Systems, pp 875–882
77. Wold S, Esbensen K, Geladi P (1987) Principal component analysis. Chemom Intell Lab Syst 2(1–3):37–52
78. Wu Z, Wang X, Zhou P, et al (2021) Transmission line fault location based on the stacked sparse auto-encoder deep neural network. In: 2021 IEEE 5th Conference on Energy Internet and Energy System Integration (EI2), IEEE, pp 3201–3206
79. Xie Y, Franz E, Chu M et al (2018) tempogan: a temporally coherent, volumetric gan for super-resolution fluid flow. ACM Trans Graph TOG 37(4):95
80. Xu H, Caramanis C, Sanghavi S (2010) Robust pca via outlier pursuit. arXiv preprint arXiv:1010.4237
81. Yu Y, Odobez JM (2020) Unsupervised representation learning for gaze estimation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 7314–7324
82. Zamir AR, Sax A, Shen W, et al (2018) Taskonomy: Disentangling task transfer learning. In: IEEE Conference on Computer Vision and Pattern Recognition, pp 3712–3722
83. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: European Conference on Computer Vision, Springer, pp 818–833
84. Zhan X, Xie J, Liu Z, et al (2020) Online deep clustering for unsupervised representation learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 6688–6697
85. Zhang L, Lu Y, Song G, et al (2018a) Rc-cnn: Reverse connected convolutional neural network for accurate player detection. In: Pacific Rim International Conference on Artificial Intelligence, Springer, pp 438–446
86. Zhang R, Isola P, Efros AA, et al (2018b) The unreasonable effectiveness of deep features as a perceptual metric. In: IEEE Conference on Computer Vision and Pattern Recognition, pp 586–595
87. Zhou Y, Govindaraju V (2014) Learning deep autoencoders without layer-wise training. stat 1050:14
88. Zhou Y, Arpit D, Nwogu I, et al (2014) Is joint training better for deep auto-encoders? arXiv preprint arXiv:1405.1380
89. Zhuang Y, Rui Y, Huang TS, et al (1998) Adaptive key frame extraction using unsupervised clustering. In: Proceedings 1998 International Conference on Image Processing. icip98 (cat. no. 98cb36269), IEEE, pp 866–870