A Disentangling Invertible Interpretation Network for Explaining Latent Representations

Patrick Esser∗ Robin Rombach∗ Björn Ommer
Heidelberg Collaboratory for Image Processing
IWR, Heidelberg University, Germany

Abstract

Neural networks have greatly boosted performance in computer vision by learning powerful representations of input data. The drawback of end-to-end training for maximal overall performance are black-box models whose hidden representations are lacking interpretability: Since distributed coding is optimal for latent layers to improve their robustness, attributing meaning to parts of a hidden feature vector or to individual neurons is hindered. We formulate interpretation as a translation of hidden representations onto semantic concepts that are comprehensible to the user. The mapping between both domains has to be bijective so that semantic modifications in the target domain correctly alter the original representation. The proposed invertible interpretation network can be transparently applied on top of existing architectures with no need to modify or retrain them. Consequently, we translate an original representation to an equivalent yet interpretable one and backwards without affecting the expressiveness and performance of the original. The invertible interpretation network disentangles the hidden representation into separate, semantically meaningful concepts. Moreover, we present an efficient approach to define semantic concepts by only sketching two images and also an unsupervised strategy. Experimental evaluation demonstrates the wide applicability to interpretation of existing classification and image generation networks as well as to semantically guided image manipulation.

1. Introduction

Deep neural networks have achieved unprecedented performance in various computer vision tasks [51, 15] by learning task-specific hidden representations rather than relying on predefined hand-crafted image features. However, the significant performance boost due to end-to-end learning comes at the cost of now having black-box models that are lacking interpretability: A deep network may have found a solution to a task, but human users cannot understand the causes for the predictions that the model makes [36]. Conversely, users must also be able to understand what the hidden representations have not learned and on which data the overall model will, consequently, fail. Interpretability is therefore a prerequisite for safeguarding AI, making its decisions transparent to the users it affects, and understanding its applicability, limitations, and the most promising options for its future improvement.

A key challenge is that learned latent representations typically do not correspond to semantic concepts that are comprehensible to human users. Hidden layer neurons are trained to help in solving an overall task in the output layer of the network. Therefore, the output neurons correspond to human-interpretable concepts such as object classes in semantic image segmentation [3] or bounding boxes in object detection [48]. In contrast, the hidden layer representa-

*Both authors contributed equally to this work.
tion of semantic concepts is a distributed pattern [9]. This distributed coding is crucial for the robustness and generalization performance of neural networks despite noisy inputs, large intra-class variability, and the stochastic nature of the learning algorithm [13]. However, as a downside of semantics being distributed over numerous neurons it is impossible to attribute semantic meaning to only an individual neuron despite attempts to backpropagate [37] or synthesize [47, 52] their associated semantic concepts. One solution has been to modify and constrain the network so that abstract concepts can be localized in the hidden representation [55]. However, this alters the network architecture and typically deteriorates overall performance [45].

**Objective:** Therefore, our goal needs to be an approach that can be transparently applied on top of arbitrary existing networks and their already learned representations without altering them. We seek a *translation* between these hidden representations and human-comprehensible semantic concepts—a non-linear mapping between the two domains. This translation needs to be invertible, i.e. an invertible neural network (INN) [5, 6, 19, 22], so that modifications in the domain of semantic concepts correctly alter the original representation.

To interpret a representation, we need to attribute meaning to parts of the feature encoding. That is, we have to disentangle the high-dimensional feature vector into multiple multi-dimensional factors so that each is mapped to a separate semantic concept that is comprehensible to the user. As discussed above, this disentangled mapping should be bijective so that modifications of the disentangled semantic factors correctly translate back to the original representation. We can now, without any supervision, disentangle the representation into independent concepts so that a user can post-hoc identify their meaning. Moreover, we present an efficient strategy for defining semantic concepts. It only requires two sketches that exhibit a change in a concept of interest rather than large annotated training sets for each concept. Given this input, we derive the invariance properties that characterize a concept and generate synthesized training data to train our invertible interpretation network. This network then acts as a translator that disentangles the original representation into multiple factors that correspond to the semantic concepts.

Besides interpreting a network representation, we can also interpret the structure that is hidden in a dataset and explain it to the user. Applying the original representation and then translating onto the disentangled semantic factors allows seeing which concepts explain the data and its variability. Finally, the invertible translation supports semantically meaningful modifications of input images: Given an autoencoder representation, its representation is mapped onto interpretable factors, these can be modified and inverse translation allows to apply the decoder to project back into the image domain. In contrast to existing disentangled image synthesis [34, 8, 32, 24, 7], our invertible approach can be applied on top of existing autoencoder representations, which therefore do not have to be altered or retrained to handle different semantic concepts. Moreover, for other architectures such as classification networks, interpretability helps to analyze their invariance and robustness.

To summarize, (i) we present a new approach to the interpretability of neural networks, which can be applied to arbitrary existing models without affecting their performance; (ii) we obtain an invertible translation from hidden representations to disentangled representations of semantic concepts; (iii) we propose a method that allows users to efficiently define semantic concepts to be used for our interpretable representation; (iv) we investigate the interpretation of hidden representations, of the original data, and demonstrate semantic image modifications enabled by the invertibility of the translation network.

### 2. Interpretability

An interpretation is a translation between two domains such that concepts of the first domain can be understood in terms of concepts of the second domain. Here, we are interested in interpretations of internal representations of a neural network in terms of human-understandable representations. Examples for the latter are given by textual descriptions, visual attributes or images.

To interpret neural networks, some approaches modify network architectures or losses used for training to obtain inherently more interpretable networks. [55] relies on a global average pooling layer to obtain class activation maps, i.e. heatmaps showing which regions of an input are most relevant for the prediction of a certain object class. [54] learn part specific convolutional filters by restricting filter activations to localized regions. Invertible neural networks [5, 6, 19, 22] have been used to get a better understanding of adversarial attacks [18]. Instead of replacing existing architectures with invertible ones, we propose to augment them with invertible transformations. Using the invertibility, we can always map back and forth between original representations and interpretable ones without loss of information. Thus, our approach can be applied to arbitrary existing architectures without affecting their performance, whereas approaches modifying architectures always involve a trade-off between interpretability and performance.

Most works on interpretability of existing networks focus on visualizations. [53] reconstruct images which activated a specific feature layer of a network. [47] uses gradient ascent to synthesize images which maximize class probabilities for different object classes. [52] generalizes this to arbitrary neurons within a network. Instead of directly optimizing over pixel values, [38] optimize over input codes of a generator network which was trained to reconstruct im-
images from hidden layers. [55] avoid synthesizing images from scratch and look for regions within a given image that activate certain neurons. For a specific class of functions, [1] decompose the function into relevance scores which can be visualized pixel-wise. Layer-wise relevance propagation [37] is a more general approach to propagate relevance scores through a network based on rules to distribute the relevance among input neurons. [41] shows how saliency maps representing the importance of pixels for a classifier’s decision can be obtained without access to the classifiers gradients. All these approaches assume that a fixed set of neurons is given and should be interpreted in terms of inputs which activate them. However, [2], [9] demonstrated that networks use distributed representations. In particular, semantic concepts are encoded by activation patterns of multiple neurons and single neurons are not concept specific but involved in the representation of different concepts. We directly address this finding by learning a non-linear transformation from a distributed representation to an interpretable representation with concept specific factors.

While [9] shows that for general networks we must expect internal representations to be distributed, there are situations where representations can be expected to have a simpler structure: Generative models are trained with the explicit goal to produce images from samples of a simple distribution, e.g. a Gaussian distribution. Most approaches are based either on Variational Autoencoders [23, 44], which try to reconstruct images from a representation whose marginal distribution is matched to a standard normal distribution, or on Generative Adversarial Networks [11, 14, 39], which directly map samples from a standard normal distribution to realistic looking images as judged by a discriminator network. The convexity of the Gaussian density makes linear operations between representations meaningful. Linear interpolations between representations enable walks along nonlinear data manifolds [42]. [26] finds visual attribute vectors which can be used to interpolate between binary attributes. To this end, two sets of images containing examples with or without an attribute are encoded to their representations, and the direction between their means is the visual attribute vector. Such attribute vectors have also been found for classifier networks [49], but because their representations have no linear structure, the approach is limited to aligned images. [42, 43] demonstrated that vector arithmetic also enables analogy making. [46] interprets the latent space of a GAN by finding attribute vectors as the normal direction of the decision boundary of an attribute classifier. [10] uses a similar approach to find attribute vectors associated with cognitive properties such as memorability, aesthetics and emotional valence. While these approaches provide enhanced interpretability through modification of attributes they are limited to representations with a linear structure. In contrast, we provide an approach to map arbitrary representations into a space of interpretable representations. This space consists of factors representing semantic attributes and admits linear operations. Thus, we can perform semantic modifications in our interpretable space and, due to the invertibility of our transformation, map the modified representation back to the original space.

3. Approach

3.1. Interpreting Hidden Representations

Invertible Transformation of Hidden Representations: Let $f$ be a given neural network to be interpreted. We place no restrictions on the network $f$. For example, $f$ could be an object classifier, a segmentation network or an autoencoder. $f$ maps an image $x \in \mathbb{R}^{H \times W \times 3}$ through a sequence of hidden layers to a final output $f(x)$. Intermediate activations $E(x) \in \mathbb{R}^{H \times W \times C}$ of a hidden layer are a task-specific representation of the image $x$. Such hidden representations convey no meaning to a human and we must transform them into meaningful representations. We introduce the notation $z = E(x) \in \mathbb{R}^{H \cdot W \cdot C}$, i.e. $z$ is the $N = H \cdot W \cdot C$ dimensional, flattened version of the hidden representation to be interpreted. $E$ is the sub-network of $f$ consisting of all layers of $f$ up to and including the hidden layer that produces $z$, and the sub-network after this layer will be denoted by $G$, such that $f(x) = G \circ E(x)$ as illustrated in Fig. 1.

To turn $z$ into an interpretable representation, we aim to translate the distributed representation $z$ to a factorized representation $\tilde{z} = (\tilde{z}_k)_{k=0}^K \in \mathbb{R}^N$ where each of the $K + 1$ factors $\tilde{z}_k \in \mathbb{R}^{N_k}$, with $\sum_{k=0}^K N_k = N$, represents an interpretable concept. The goal of this translation is twofold: On the one hand, it should enable an analysis of the relationship between data and internal representations of $f$ in terms of interpretable concepts; this requires a forward map $T$ from $z$ to $T(z) = \tilde{z}$. On the other hand, it should enable semantic modifications on internal representations of $f$; this requires the inverse of $T$. With this inverse map, $T^{-1}$, an internal representation $z$ can be mapped to $\tilde{z}$, modified in semantically meaningful ways to obtain $\tilde{z}^\ast$ (e.g. changing a single interpretable concept), and mapped back to an internal representation of $f$. This way, semantic modifications, $\tilde{z} \mapsto \tilde{z}^\ast$, which were previously only defined on $z$ can be applied to internal representations via $z \mapsto z^\ast := T^{-1}(T(z)^\ast)$. See Fig. 2 for an example, where $z$ is modified by replacing one of its semantic factors $\tilde{z}_k$ with that of another image.

Disentangling Interpretable Concepts: For meaningful analysis and modification, each factor $\tilde{z}_k$ must represent a specific interpretable concept and taken together, $\tilde{z}$ should support a wide range of modifications. Most importantly, it must be possible to analyze and modify different factors $\tilde{z}_k$ independently of each other. This implies a factorization of their joint density $p(\tilde{z}) = \prod_{k=0}^K p(\tilde{z}_k)$. To explore different
To fit this model to data, we utilize the invertibility of $T$ to directly compute and maximize the likelihood of pairs $(z^a, z^b) = (E(x^a), E(x^b))$. We compute the likelihood with the absolute value of the Jacobian determinant of $T$, denoted $|T'(\cdot)|$, as

$$p(z^a, z^b|F) = p(z^a) p(z^b|z^a, F)$$
$$= |T'(z^a)| p(T(z^a)) p(T(z^b)|T(z^a), F)$$

To be able to compute the Jacobian determinant efficiently, we follow previous works [22] and build $T$ based on ActNorm, AffineCoupling and Shuffling layers as described in more detail in Sec. A.1 of the supplementary. For training we use the negative log-likelihood as our loss function. Substituting Eq. (1) into Eq. (5), Eq. (2) and (3) into Eq. (6), leads to the per-example loss $\ell(z^a, z^b|F)$,

$$\ell(z^a, z^b|F) = \sum_{k=0}^{K} \|T(z^a)_k^2 - \log |T'(z^a)| \|_1 + \|T(z^b)_k^2 - \log |T'(z^b)| \|_1 + \|T(z^b)_F - \sigma_{ab} T(z^a)_F \|_2^2$$

which is optimized over training pairs $(x^a, x^b)$ for all semantic concepts $F \in \{1, \ldots, K\}$:

$$\mathcal{L} = \sum_{F=1}^{K} \mathbb{E}_{(x^a, x^b) \sim p(x^a, x^b|F)} \ell(E(x^a), E(x^b)|F)$$

Note that we have described the case where image pairs share at least one semantic concept, which includes the case where they share more than one semantic concept. Moreover, our approach is readily applicable in the case where image pairs differ in a semantic concept. In this case, Eq. (2) holds for all factors $z^b_k, k \in \{0, \ldots, K\} \setminus \{F\}$ and Eq. (3) holds for factor $z^b_F$. This case will also be used in the next section, where we discuss the dimensionality and origin of semantic concepts.

### 3.2. Obtaining Semantic Concepts

#### Estimating Dimensionality of Factors:

Semantic concepts differ in complexity and thus also in dimensionality. Given image pairs $(x^a, x^b) \sim p(x^a, x^b|F)$ that define the $F$-th semantic concept, we must estimate the dimensionality of factor $\tilde{z}_F$ that represents this concept. Due to the invertibility of $T$, the sum of dimensions of all these factors equals the dimensionality of the original representation. Thus, semantic concepts captured by the network $E$ require a larger share of the overall dimensionality than those $E$ is invariant to.
The similarity of $x^a, x^b$ in the $F$-th semantic concept implies a positive mutual information between them, which will only be preserved in the latent representations $E(x^a), E(x^b)$ if the $F$-th semantic concept is captured by $E$. Thus, based on the simplifying assumption that components of hidden representations $E(x^a), E(x^b)$ are jointly Gaussian distributed, we approximate their mutual information with their correlation for each component $i$. Summing over all components $i$ yields a relative score $s_F$ that serves as proxy for the dimensionality of $z_F$ in case of training images $(x^a, x^b) \sim p(x^a, x^b|F)$ for concept $F$,

$$s_F = \sum_i \frac{\text{Cov}(E(x^a)_i, E(x^b)_i)}{\sqrt{\text{Var}(E(x^a)_i) \text{Var}(E(x^b)_i)}}$$

(11)

Since correlation is in $[-1, 1]$, scores $s_F$ are in $[-N, N]$ for $N$-dimensional latent representations of $E$. Using the maximum score $N$ for the residual factor $z_0$ ensures that all factors have equal dimensionality if all semantic concepts are captured by $E$. The dimensionality $N_F$ of $z_F$ is then $N_F = \left\lfloor \frac{\exp(s_F)}{\sum_k \exp(s_k)} \right\rfloor$. Tab. 1 demonstrates the feasibility of predicting dimensionalities with this approximation.

**Table 1:** Estimated dimensionalities of interpretable factors $z_F$ representing different semantic concepts. Remaining dimensions are assigned to the residual factor $z_0$. Compared to an autoencoder, the color factor is smaller in case of a color-invariant classifier.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Latent $z$ Dim.</th>
<th>Interpretable $z'$ Dim.</th>
<th>Factor $z_F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color-MNIST</td>
<td>AE</td>
<td>64</td>
<td>12</td>
<td>Digit</td>
</tr>
<tr>
<td></td>
<td>Classifier</td>
<td>64</td>
<td>22</td>
<td>Digit</td>
</tr>
</tbody>
</table>

**Figure 3:** Efficient generation of training examples for semantic concepts: A user must only provide two sketches (first row) for a change of a semantic concept, here: *roundness*. We then synthesize training images to reflect this semantic change.

**Figure 4:** The inverse of our interpretation network $T$ maps linear walks in the interpretable domain back to nonlinear walks on the data manifold in the encoder space, which get decoded to meaningful images (bottom right). In contrast, decoded images of linear walks in the encoder space contain ghosting artifacts (bottom left).

**Sketch-Based Description of Semantic Concepts:** Training requires the availability of image pairs that depict changes in a semantic concept. Most often, a sufficiently large number of such examples is not easy to obtain. The following describes an approach to help a user specify semantic concepts effortlessly.

Two sketches are worth a thousand labels: Instead of labeling thousands of images with semantic concepts, a user only has to provide two sketches, $y^a$ and $y^b$ which demonstrate a change in a concept. For example, one sketch may serve as examples for a change in the concept of interest.

Unsupervised Interpretations: Even without examples for changes in semantic factors, our approach can still produce disentangled factors. In this case, we minimize the negative log-likelihood of the marginal distribution of hidden representations $z = E(x)$:

$$\mathcal{L}_{\text{unsup}} = -\mathbb{E}_x [T(E(x))]^2 - \log|T'(E(x))|$$

(12)

As this leads to independent components in the transformed representation, it allows users to attribute meaning to this representation after training. Mapping a linear interpolation in our disentangled space back to $E$’s representation space
leads to a nonlinear interpolation on the data manifold embedded by \( E \) (see Fig. 4). This linear structure allows to explore the representations using vector arithmetics [49]. For example, based on a few examples of images with a change in a semantic concept, we can find a vector representing this concept as the mean direction between these images (see Eq. (14)). In contrast to previous works, we do not rely on disentangled latent representations but learn to translate arbitrary given representations into disentangled ones.

4. Experiments

The subsequent experiments use the following datasets: AnimalFaces [28], DeepFashion [29, 31], CelebA [30], MNIST [27], Cifar10 [25], and FashionMNIST [50]. Moreover, we augment MNIST by randomly coloring its images to provide a benchmark for disentangling experiments (denoted ColorMNIST).

4.1. Interpretation of Autoencoder-Frameworks

Autoencoders learn to reconstruct images from a low-dimensional latent representation \( z = E(x) \). Subsequently, we map \( z \) onto interpretable factors to perform semantic image modification. Note that \( z \) is only obtained using a given network; our invertible interpretation network has never seen an image itself.

Disentangling Latent Codes of Autoencoders: Now we alter the \( \tilde{z}_k \) which should in turn modify the \( z \) in a semantically meaningful manner. This tests two aspects of our translation onto mutually disentangled, interpretable representations: First, if its factors have been successfully disentangled, swapping factors from different images should still yield valid representations. Second, if the factors represent their semantic concepts faithfully, modifying a factor should alter its corresponding semantic concept.

To evaluate these aspects, we trained an autoencoder on the AnimalFaces dataset. As semantic concepts we utilize the animal category and a residual factor. Fig. 5 shows the results of combining the residual factor of the image on the left with the animal-class-factor of the image at the top. After decoding, the results depict animals from the class of the image at the top. However, their gaze direction corresponds to the image on the left. This demonstrates a successful disentangling of semantic concepts in our interpreted space.

The previous case has confirmed the applicability of our approach to roughly aligned images. We now test it on unaligned images of articulated persons on DeepFashion. Fig. 6 presents results for attribute swapping as in the previous experiment. Evidently, our approach can handle articulated objects and enables pose guided human synthesis.

Finally, we conduct this swapping experiment on ColorMNIST to investigate simultaneous disentangling of multiple factors. Fig. 2 shows a swapping using an interpretation of three factors: digit type, color, and residual.

Evaluating the Unsupervised Case: To investigate our approach in case of no supervision regarding semantic concepts, we analyze its capability to turn simple autoencoders into generative models. Because our interpretations yield normally distributed representations, we can sample them, translate them back onto the latent space of the autoencoder, and finally decode them to images,

\[
\tilde{z} \sim \mathcal{N}(\tilde{z}|0, 1), \quad x = G(T^{-1}(\tilde{z})).
\]
These sets are only required after training, i.e., X−tribute in X−tain semantic axes substitutes a lossless encoder/decoder framework. We observe the representation space, since the invertible network constitutes stable training and avoids the mode-collapse problem that relies on an autoencoder and a reconstruction loss. This enables approaches to GANs, which explains our improvement in FID.

Our approach can utilize a learned similarity metric similar to GANs, which enables them to produce high-quality images. In contrast to approaches based on GANs, we can achieve results competitive with approaches based on AE-based and GAN models as reported in [4]. Tab. 2 presents mean and std of FID scores over three trials with 10K generated images. We significantly improve over state-of-the-art FID reported in [4].

We employ the standard evaluation protocol for generative modeling using two Variational Autoencoders and measure image quality with Fréchet Inception Distance (FID scores). [4] presented an approach to generative modeling using two Variational Autoencoders and achieved results competitive with approaches based on GANs. We follow [4] and use an autoencoder architecture based on [33] and train on the same datasets with losses as in [26].

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST</th>
<th>FashionMNIST</th>
<th>CIFAR-10</th>
<th>CelebA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwoStageVAE</td>
<td>12.6 ± 1.5</td>
<td>29.3 ± 1.0</td>
<td>72.9 ± 0.9</td>
<td>44.4 ± 0.7</td>
</tr>
<tr>
<td>WGAN GP</td>
<td>20.3 ± 5.0</td>
<td>24.5 ± 2.1</td>
<td>55.8 ± 0.9</td>
<td>30.3 ± 1.0</td>
</tr>
<tr>
<td>WGAN</td>
<td>6.7 ± 0.4</td>
<td>21.5 ± 1.6</td>
<td>55.2 ± 2.3</td>
<td>41.3 ± 2.0</td>
</tr>
<tr>
<td>DRAGAN</td>
<td>7.6 ± 0.4</td>
<td>27.7 ± 1.2</td>
<td>69.8 ± 2.0</td>
<td>42.3 ± 3.0</td>
</tr>
<tr>
<td>BEGAN</td>
<td>13.1 ± 1.0</td>
<td>22.9 ± 0.9</td>
<td>71.9 ± 1.6</td>
<td>38.9 ± 0.9</td>
</tr>
<tr>
<td>Ours</td>
<td>6.4 ± 0.1</td>
<td>16.0 ± 0.1</td>
<td>45.7 ± 0.3</td>
<td>20.2 ± 0.5</td>
</tr>
</tbody>
</table>

Table 2: FID scores of various AE-based and GAN models as reported in [4].

Such vector arithmetic depends on a meaningful linear structure of our interpreted representation space. We illustrate this structure in Fig. 4. Linear walks in our interpretable space always result in meaningful decoded images, indicating that the backtransformed representations lie on the data manifold. In contrast, decoded images of linear walks in the encoder’s hidden representation space contain ghosting artifacts. Consequently, our model can transform nonlinear hidden representations to an interpretable space with linear structure. Fig. 7 visualizes a 2D submanifold on CelebA.

Fig. 8 provides an example for an interpolation as described in Eq. 14 between attributes on the CelebA dataset. We linearly walk along the beardiness and smiling attributes, increasing the former and decreasing the latter.


Figure 7: CelebA: Four randomly drawn samples (corners) and corresponding interpolations obtained with unsupervised training, see Sec. 4.1.

![Figure 7: CelebA](image)

Figure 8: Interpolating along semantic directions in disentangled representation space: First four rows show interpolations along beardiness, while the last four depict interpolations along smiling attribute. Note the change of gender in rows 1, 2, 4, reflecting the strong correlation of beard and gender in the original data.

![Figure 8: Interpolating](image)
4.2. Interpretation of Classifiers

After interpreting autoencoder architectures we now analyze classification networks: (i) A digit classifier on ColorMNIST (accuracy \( \approx 97\% \)). To interpret this network, we extract hidden representations \( z \in \mathbb{R}^{64} \) just before the classification head. (ii) A ResNet-50 classifier [12] trained on classes of AnimalFaces. Hidden representations \( z \in \mathbb{R}^{2048} \) are extracted after the fully convolutional layers.

Network Response Analysis: We now analyze how class output probabilities change under manipulations in the interpretation space: First, we train the translator \( T \) to disentangle \( K \) (plus a residual) distinct factors \( \tilde{z}_k \). For evaluation we modify a single factor \( \tilde{z}_k \) while keeping all others fixed. More precisely, we modify \( \tilde{z}_k \) by replacing it with samples drawn from a random walk in a harmonic potential (an Ornstein-Uhlenbeck process, see Sec. B of the supplementary), starting at \( \tilde{z}_k \). This yields a sequence of modified factors \((\tilde{z}_k^{(1)}, \tilde{z}_k^{(2)}, \ldots, \tilde{z}_k^{(n)})\) when performing \( n \) modification steps. We invert every element in this sequence back to its hidden representation and apply the classifier. We analyze the response of the network to each modified factor \( k \) through the distribution of the logits and class predictions.

Figure 9: Left: Output variance per class of a digit classifier on ColorMNIST, assessed via distribution of log-softmaxed logits and class predictions. \( T \) disentangles \( \tilde{z}_0 \) (residual), \( \tilde{z}_1 \) (digit) and \( \tilde{z}_2 \) (color). Right: 1d disentangled UMAP embeddings of \( \tilde{z}_1 \) and \( \tilde{z}_2 \). See Sec. 4.2.

Interpreting Classifiers to Estimate their Invariance:

Network interpretation also identifies the invariance properties of a learned representation. Here we evaluate invariances of a digit classifier to color. We learn a translation \( T \) to disentangle digit \( \tilde{z}_1 \), color \( \tilde{z}_2 \), and a residual \( \tilde{z}_0 \). Fig. 9 shows the network response analysis. The distribution of log softmax-values and predicted classes is indeed not sensitive to variations in the factor color, but turns out to be quite responsive when altering the digit representation. We additionally show a UMAP [35] of the reversed factor manipulations in Fig. 10 (in black). Since the entire modification occurs within one cluster, this underlines that \( T \) found a disentangled representation and that the classifier is almost invariant to color. Additionally, we employ another 1D-UMAP dimensionality reduction to each factor separately and then plot their pair-wise correlation in Fig. 9.

Next, we trained a transformer \( T \) to evaluate interpretability in case of the popular ResNet-50. The analysis of three factors, grayscale value \( \tilde{z}_1 \), roundness \( \tilde{z}_2 \), and a residual \( \tilde{z}_0 \) reveals an invariance of the classifier towards grayness but not roundness. More details can be found in Sec. B of the supplementary.

5. Conclusion

We have shown that latent representations of black boxes can be translated to interpretable representations where disentangled factors represent semantic concepts. We presented an approach to perform this translation without loss of information. For arbitrary models, we provide the ability to work with interpretable representations which are equivalent to the ones used internally by the model. We have shown how this provides a better understanding of models and data as seen by a model. Invertibility of our approach enables semantic modifications and we showed how it can be used to obtain state-of-the-art autoencoder-based generative models.

This work has been supported in part by the German federal ministry BMWi within the project "KI Absicherung", the German Research Foundation (DFG) projects 371923335 and 421703927, and a hardware donation from NVIDIA corporation.
References


A Disentangling Invertible Interpretation Network for Explaining Latent Representations

Supplementary Materials

A. Implementation Details

A.1. Invertible Interpretation Network

As described in Sec. 3.1, we require $T$ to be an invertible transformation. To achieve this, we use an invertible neural network. In our implementation, this network is built from three invertible layers: coupling blocks [6], actnorm layers [22] and shuffling layers. A sequence of these three layers builds one invertible block, c.f. Fig. 12. After passing the input $z$ through multiple blocks, c.f. Fig. 11, we split the output $\tilde{z}$ into factors $(\tilde{z}_k)_{k=0}^K$.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig11.png}
\caption{Overview of our invertible interpretation network consisting of multiple blocks, see Fig. 12.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig12.png}
\caption{A single block of our interpretation network built from invertible layers described in Sec. A.1.}
\end{figure}

Each shuffling layer uses a fixed, randomly initialized permutation to shuffle the channels of its input. Actnorm consists of learnable shift and scale parameters for each channel, which are initialized to provide activations with zero mean and unit variance. Coupling blocks equally split their input $h = (h_1, h_2)$ along its channel dimension and compute

\begin{align}
\tilde{h}_1 &= h_1 \cdot s_1(h_2) + t_1(h_2) \\
\tilde{h}_2 &= h_2 \cdot s_2(\tilde{h}_1) + t_2(\tilde{h}_1)
\end{align}

where $s_i, t_i$ are fully connected networks.

**Definition of notation for network architectures** To describe the architecture we use in our experiments, we introduce the following notation:

- Conv2d($c_{in}, c_{out}, k, s, p$): A two-dimensional convolution operation, working on $c_{in}$ input channels and producing $c_{out}$ output channels. We use square kernels of size $k$. $s$ denotes the stride, $p$ the amount of padding used.
- ConvT2d($c_{in}, c_{out}, k, s, p$): A two-dimensional transposed convolution operation, working on $c_{in}$ input channels and producing $c_{out}$ output channels. We use square kernels of size $k$. $s$ denotes the stride, $p$ the amount of padding used.
- Linear($c_{in}, c_{out}$): Maps a vector $z_1 \in \mathbb{R}^{c_{in}}$ onto a vector $z_2 \in \mathbb{R}^{c_{out}}$.

A.2. Autoencoder Architecture

The architecture used for our autoencoder experiments is based on [33]. We replace batch normalization [16] by act normalization [22] to avoid issues caused by differences in batch statistics during training and testing. Details regarding the architecture can be found in Tab. 3. Furthermore, we remove the highest fully connected layer which results in a more lightweight model with less parameters, see Tab. 4 for a comparison. To implement the adversarial loss, we use the discriminator from [17], operating on patches of input images with a receptive field of 70 pixels.
Table 3: Architecture of our autoencoder-model. We assume quadratic images, i.e. $h = w$ for an image of size $c \times h \times w$ with $c$ channels.

<table>
<thead>
<tr>
<th>input size</th>
<th>encoder</th>
<th>decoder</th>
<th>total</th>
<th>encoder</th>
<th>decoder</th>
<th>total</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>$32 \times 32, z \in \mathbb{R}^{64}$</td>
<td>3.0</td>
<td>2.9</td>
<td>5.9</td>
<td>8.7</td>
<td>8.6</td>
<td>17.3</td>
<td>0.34</td>
</tr>
<tr>
<td>$64 \times 64, z \in \mathbb{R}^{64}$</td>
<td>3.8</td>
<td>3.3</td>
<td>7.1</td>
<td>33.9</td>
<td>33.8</td>
<td>67.7</td>
<td>0.10</td>
</tr>
<tr>
<td>$128 \times 128, z \in \mathbb{R}^{256}$</td>
<td>19.5</td>
<td>11.2</td>
<td>30.7</td>
<td>135.1</td>
<td>134.9</td>
<td>270.0</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Table 4: Comparison of the total number of parameters (in millions) used in the described autoencoding architectures. Note that the numbers in the last row for [4] are calculated based on their architecture, but never used in their experiments.

A.3. Classifier Architecture

We use two different classifier architectures in our experiments. The first one uses the same encoder as in the autoencoder, followed by a fully connected layer, c.f. Tab. 5. The second one uses the ResNet-50 [12] architecture. We use the network up to the last convolutional layer for $E$ and the remaining network producing the class scores for $G$.

Table 5: Architecture of our basic classification model.

A.4. Details on Sketch Based Description of Semantic Concepts

Efficient generation of training pairs for semantic concepts involves a style transfer algorithm as described in Sec. 3.2. For this, we use the approach of [40], with the following set of hyperparameters:

- content loss: $\lambda_c = 1.0$,
- style loss: $\lambda_s = 5.0$,
- identity losses: $\lambda_{id,1} = 50.0$, $\lambda_{id,2} = 1.0$.

In addition, we use the same discriminator architecture as in our autoencoder experiments to distinguish real from stylized images. We denote this loss the realism prior and weight it by $\lambda_g = 1.0$.

Using this setting, the model is trained on AnimalFaces (content) and the Wikiart [20] (style) datasets.

A.5. Training Hyperparameters

We train all models using a batch size of 25 and a learning rate of $10^{-4}$ for the Adam optimizer [21]. The translation network $T$ is trained by optimizing Eq. 10, where we fix $\sigma_{ab} = 0.9$ for all experiments. Note that $\sigma_{ab} = 0.0$ corresponds to uncorrelated examples $a$ and $b$, whereas $\sigma_{ab} = 1.0$ denotes perfect correlation. Hence, we allow for a small amount of stochasticity when providing pairs $(a, b)$ to account for low-quality pairs within the training set. The hyperparameters for the transformer $T$ are:
• $n_{\text{flow}}$: Number of flow blocks (c.f. Fig. 12) used to build $T$.

• $H$: Dimensionality of hidden layers in subnetworks $s_i$ and $t_i$.

• $D$: Depth of subnetworks $s_i$ and $t_i$.

We list all configurations of these hyperparameters used in our experiments in Tab. 6.

<table>
<thead>
<tr>
<th>input size $z \in \mathbb{R}^n$</th>
<th>$n_{\text{flow}}$</th>
<th>$H$</th>
<th>$D$</th>
<th>$\sigma_{ab}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}^{64}$</td>
<td>12</td>
<td>512</td>
<td>2</td>
<td>0.9</td>
</tr>
<tr>
<td>$\mathbb{R}^{256}$</td>
<td>12</td>
<td>512</td>
<td>2</td>
<td>0.9</td>
</tr>
<tr>
<td>$\mathbb{R}^{2048}$</td>
<td>6</td>
<td>2048</td>
<td>2</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 6: Hyperparameters used for training the transformer $T$. See Sec. A.5 for a definition of the notation used.
B. Additional Results

**Semantic Image Manipulations and Semantic Embeddings** In the case of autoencoders, our invertible interpretation network enables semantic image modifications. By transforming the latent representation \( z \) of an image \( x \) to \( \tilde{z} \), we can modify semantic concepts of the image: We modify the factor \( \tilde{z}_k \) corresponding to the semantic concept, invert the modified representation back to the latent space of the autoencoder and finally decode it to the semantically modified image.

In Fig. 13 and 14 we manipulate individual semantic factors by interpolation on the ColorMNIST and CelebA datasets, respectively. In both cases, colors of the embeddings represent the semantics of \( \tilde{z}_1 \), in particular for ColorMNIST, the colors represent the digit class, and for CelebA, the colors represent the gender. The top shows the interpolation status for each of the semantic concepts. Next, to the left we display a two-dimensional embedding of the residual space which illustrates the Gaussian structure of our prior and its independence with respect to \( \tilde{z}_1 \). To the right we plot a one-dimensional embedding of \( \tilde{z}_1 \) against a one-dimensional embedding of \( \tilde{z}_2 \), providing a semantically meaningful two-dimensional embedding. For ColorMNIST, the observed product structure of this embedding shows the independence of \( \tilde{z}_1 \) and \( \tilde{z}_2 \). On the other hand, on CelebA we observe missing data points in the top-left quadrant, demonstrating a lack of training examples showing women with beards.

![Figure 13: Semantic image modifications and embeddings](image)

We then invert the modified representation back to the latent space of the autoencoder and visualize the resulting representation in a two-dimensional embedding of the latent space (bottom left). In the animated versions, which can be found at [https://compvis.github.io/iin/](https://compvis.github.io/iin/), we can see that semantic modifications, which have a simple linear structure in \( \tilde{z} \), get mapped to complex paths in \( z \) due to the entangled structure of the latent space. Finally, the bottom right shows the semantically modified image \( G(T^{-1}(\tilde{z})) \).
Partial Sampling of Factors   Instead of swapping disentangled factors of provided pairs, we can sample a factor $\tilde{z}_k$ while keeping other factors $\tilde{z}_i$ fixed. See Fig. 15 for an application on the DeepFashion dataset.

Manipulating a Network’s Decision by Meaningful Variation of Disentangled Factors   As described in the main text, we modify a factor $\tilde{z}_k$ while keeping all other factors fixed and analyze the response of the classifier by inverting and decoding the code $\tilde{z}$. More precisely, we change each factor by performing a random walk in transformed space, keeping a relation to the input example. To regularize the walk to stay within proximity of the input example, we use a Ornstein-Uhlenbeck process to simulate the walk, cf. Fig. 16. This process can be expressed as a simplified discretized version of a stochastic differential equation:

$$\tilde{z}_{k,t+1} = -\gamma \tilde{z}_{k,t} + \sigma W_t,$$

(17)

where $t$ indexes the random sequence, starting at $\tilde{z}_k \equiv \tilde{z}_{k,0}$, $W_t \in \mathcal{N}(0,1)$ and $\gamma$ and $\sigma$ scalar parameters. We repeat the analysis done in the main text for a classifier trained on ColorMNIST, see Fig. 17. Again, changing factor color has no effect on the prediction of the classifier (hidden representations stay within the same UMAP cluster), whereas changes in factor digit cause variations in the classifier’s prediction.

ResNet-50 Classifier Response Analysis   To analyze the expressiveness of our disentangling interpretation approach, we train an invertible transformation on a ResNet-50 classifier trained to perform class prediction on AnimalFaces. To this end, we interpret the effect of three factors: greyscale, roundness (i.e. softness of contours) and a residual factor. As can be concluded from the response analysis (c.f. Sec. 4.2 and Fig. 18), the classifier is not sensitive to changes in factor greyscale, but does often change its class prediction when altering the factor roundness (right-hand side of Fig. 18, where log-probabilities are plotted). This suggests that the classifier is (to some degree) relying on the shape of the contours when
Figure 15: Sampling on DeepFashion: Instead of swapping factors between images, we use the ability of our interpretation network to explore a factor’s variability by directly sampling it. In each row, we combine a randomly sampled appearance factor $\tilde{z}_1$ with the residual factor $\tilde{z}_0$ of the topmost image.

Figure 16: Brownian motion in flat space (red) and in a harmonic potential (blue). The latter is known as Ornstein-Uhlenbeck process and used to explore a factor in proximity to a given reference point.

predicting the respective class. This behavior is further confirmed by visualizing the variation within clusters of the classifier’s hidden code by a 2D UMAP embedding in Fig. 19.

**Interpretable Representations $\tilde{z}$ Improve Sampling-Based Image Synthesis w.r.t. Latents $z$.** Fig. 20, 21 and 22 provide insight into how our translation network $T$ can map a complex, hidden representation of a given network onto a interpretable and accessible representation. Here, we compare decoded samples $x$ when drawing (i) from the prior of the autoencoding network, i.e. $x = G(z)$ for $z \sim \mathcal{N}(0, 1)$ and (ii) from the prior of the transformer network, i.e. $x = G(T^{-1}(\tilde{z}))$ for $\tilde{z} \sim \mathcal{N}(0, 1)$. Samples obtained from $\tilde{z}$-space yield structured and more coherent images than samples from the latent space $z$. These figures also provide a qualitative examples for the quantitative results in Tab. 2 on unconditional image synthesis.
Figure 17: Same as Fig. 10 for a different input example. Left: Changing factor digit in ColorMNIST classifier’s hidden representation by a random walk in a harmonic potential. Right: Changing factor color in ColorMNIST classifier’s hidden representation by a walk in a harmonic potential.

Figure 18: Output variance per class of a ResNet-50 classifier trained on AnimalFaces class identity, assessed via distribution of log-softmaxed logits and class predictions. $T$ is trained to disentangle $\tilde{z}_0$ (residual), $\tilde{z}_1$ (roundness) and $\tilde{z}_2$ (greyscale). See Sec. 4.2.

Figure 19: Left: Changing factor residual in AnimalFaces classifier’s hidden representation. Middle: Changing factor roundness. Right: Changing factor grayness.
Figure 20: Samples on CelebA. Top: Decoded samples drawn from the prior of the autoencoder: \( x = G(z), \ z \sim \mathcal{N}(0, 1) \). Bottom: Decoded samples drawn from the prior of the transformer: \( x = G(T^{-1}(\tilde{z})), \ \tilde{z} \sim \mathcal{N}(0, 1) \).
Figure 21: Samples on CIFAR-10. Left: Decoded samples drawn from the prior of the autoencoder: $x = G(z), \ z \sim \mathcal{N}(0, 1)$. Right: Decoded samples drawn from the prior of the transformer: $x = G(T^{-1}(\tilde{z})), \ \tilde{z} \sim \mathcal{N}(0, 1)$.

Figure 22: Samples on FashionMNIST. Left: Decoded samples drawn from the prior of the autoencoder: $x = G(z), \ z \sim \mathcal{N}(0, 1)$. Right: Decoded samples drawn from the prior of the transformer: $x = G(T^{-1}(\tilde{z})), \ \tilde{z} \sim \mathcal{N}(0, 1)$. 