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 $(\tilde{z}_k)_{k=0}^K$.

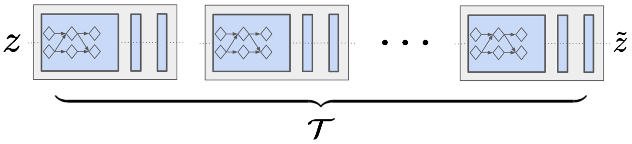

Figure 11: Overview of our invertible interpretation network consisting of multiple bocks, see Fig. 12.

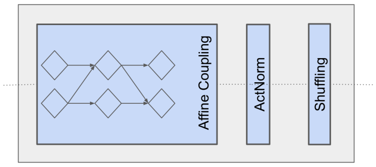

Figure 12: A single block of our interpretation network built from invertible layers described in Sec. A.1.

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

$$\tilde{h}_1 = h_1 \cdot s_1(h_2) + t_1(h_2) \tag{15}$$

$$\tilde{h}_2 = h_2 \cdot s_2(\tilde{h}_1) + t_2(\tilde{h}_1) \tag{16}$$

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.

E: **Encoder**	G: **Decoder**
Conv2d(3, 64, 4, 2, 1), ActNorm, LeakyReLU(0.2)	ConvT2d(z, 512, $h/16$, 1, 0), ActNorm, LeakyReLU(0.2)
Conv2d(64, 128, 4, 2, 1), ActNorm, LeakyReLU(0.2)	ConvT2d(512, 256, 4, 2, 1), ActNorm, LeakyReLU(0.2)
Conv2d(128, 256, 4, 2, 1), ActNorm, LeakyReLU(0.2)	ConvT2d(256, 128, 4, 2, 1), ActNorm, LeakyReLU(0.2)
Conv2d(256, 512, 4, 2, 1), ActNorm, LeakyReLU(0.2)	ConvT2d(128, 64, 4, 2, 1), ActNorm, LeakyReLU(0.2)
Conv2d(512, $2 \cdot z$, $h/16$, 1, 0)	ConvT2d(64, 3, 4, 2, 1), Tanh

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.

	Number of parameters per model $[10^6]$						
	ours			TwoStageVAE [4]			
input size	encoder	decoder	total	encoder	decoder	total	ratio
32×32, $z \in \mathbb{R}^{64}$	3.0	2.9	5.9	8.7	8.6	17.3	0.34
64×64, $z \in \mathbb{R}^{64}$	3.8	3.3	7.1	33.9	33.8	67.7	0.10
128×128, $z \in \mathbb{R}^{256}$	19.5	11.2	30.7	(135.1)	(134.9)	(270.0)	0.11

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.

E: **Encoder**	G: **Classification Head**
Conv2d(3, 64, 4, 2, 1), ActNorm, LeakyReLU(0.2)	Linear(z, $n_{classes}$)
Conv2d(64, 128, 4, 2, 1), ActNorm, LeakyReLU(0.2)	
Conv2d(128, 256, 4, 2, 1), ActNorm, LeakyReLU(0.2)	
Conv2d(256, 512, 4, 2, 1), ActNorm, LeakyReLU(0.2)	
Conv2d(512, z, $h/16$, 1, 0)	

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 autoenconder 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_{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.

input size	n_{flow}	H	D	σ_{ab}
$z \in \mathbb{R}^{64}$	12	512	2	0.9
$z \in \mathbb{R}^{256}$	12	512	2	0.9
$z \in \mathbb{R}^{2048}$	6	2048	2	0.9

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.

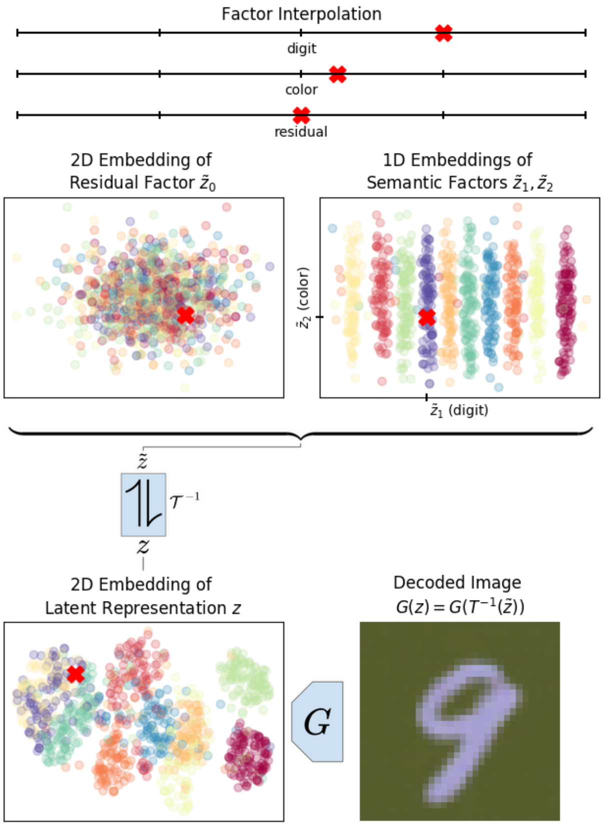

Figure 13: Semantic image modifications and embeddings: We interpolate within individual semantic concepts and visualize representations embedded onto semantically-meaningful dimensions. See Sec. B for details and https://compvis.github.io/iin/ for an animated version.

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/, 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 $G(T^{-1}(\tilde{z}))$.

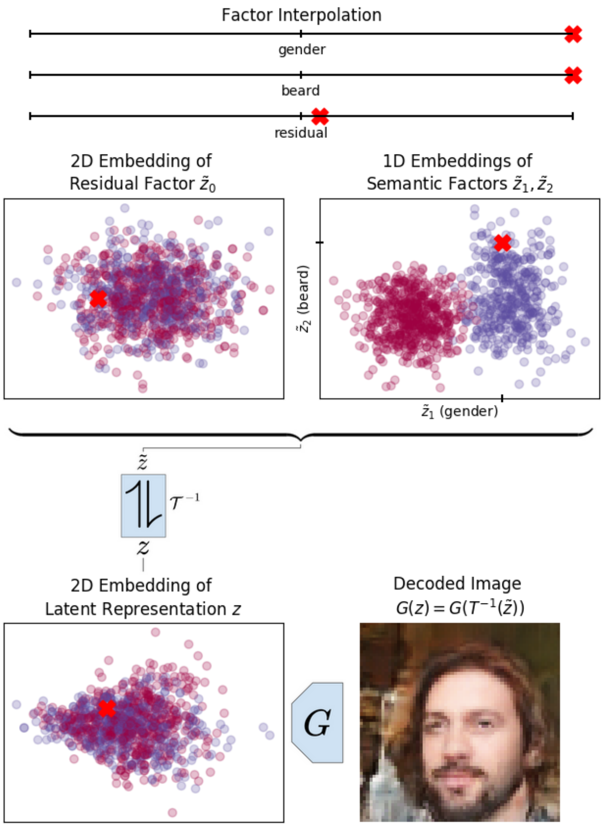
Factor Interpolation
gender
beard
residual
2D Embedding of Residual Factor $\tilde{z}_0$
1D Embeddings of Semantic Factors $\tilde{z}_1, \tilde{z}_2$
$\tilde{z}_2$ (beard)
$\tilde{z}_1$ (gender)
$\tilde{z}$
T^{-1}
z
2D Embedding of Latent Representation z
G
Decoded Image $G(z) = G(T^{-1}(\tilde{z}))$

Figure 14: Semantic image modifications and embeddings: We interpolate within individual semantic concepts and visualize representations embedded onto semantically-meaningful dimensions. See Sec. B for details and https://compvis.github.io/iin/ for an animated version.

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, *c.f*. Fig. 16. This process can be expressed as a simplified discretized version of a stochastic differential equation:

$$\tilde{z}_{k,t+1} = -\gamma \tilde{z}_{k,t} + \sigma W_t, \tag{17}$$

where t indexes the random sequence, starting at $\tilde{z}_k \equiv \tilde{z}_{k,0}$, $W_t \in \mathcal{N}(0, \mathbf{1})$ and γ and σ 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

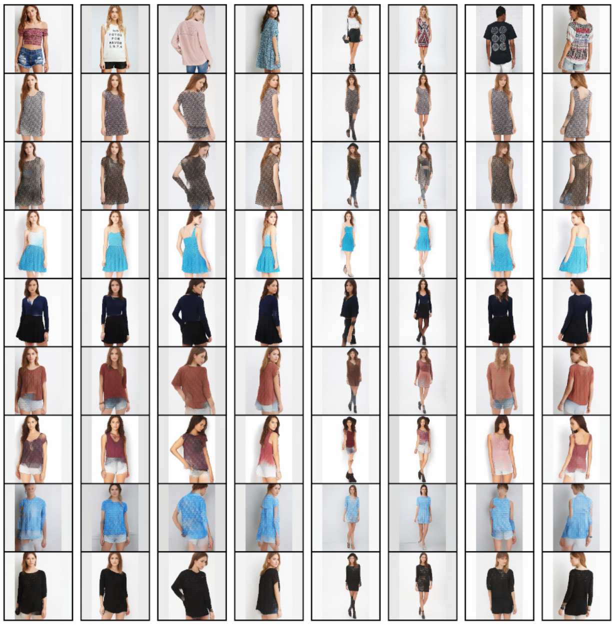

Figure 15: Sampling on DeepFashion: Instead of swapping factors between images, we use the ability of our interpretation network to explore a factor's variability by directly sampling it. In each row, we combine a randomly sampled appearance factor $\tilde{z}_1$ with the residual factor $\tilde{z}_0$ of the topmost image.

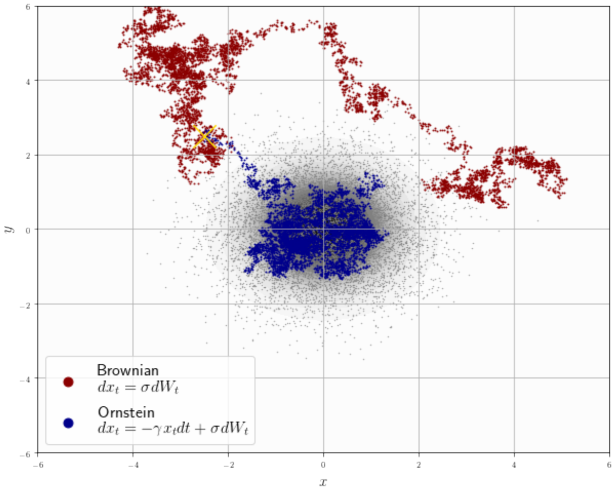

Figure 16: Brownian motion in flat space (red) and in a harmonic potential (blue). The latter is known as *Ornstein-Uhlenbeck process* and used to explore a factor in proximity to a given reference point.

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 $z \sim \mathcal{N}(0, \mathbf{1})$ and (ii) from the prior of the transformer network, i.e. $x = G(T^{-1}(\tilde{z}))$ for $\tilde{z} \sim \mathcal{N}(0, \mathbf{1})$. 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.

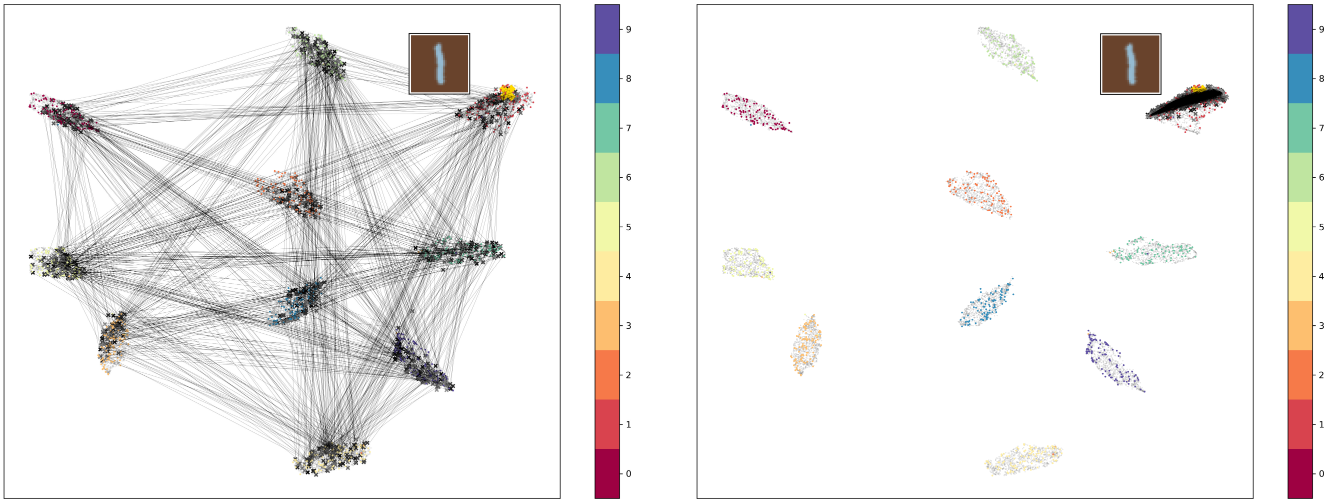

Figure 17: Same as Fig. 10 for a different input example. Left: Changing factor *digit* in ColorMNIST classifier's hidden representation by a random walk in a harmonic potential. Right: Changing factor *color* in ColorMNIST classifier's hidden representation by a walk in a harmonic potential.

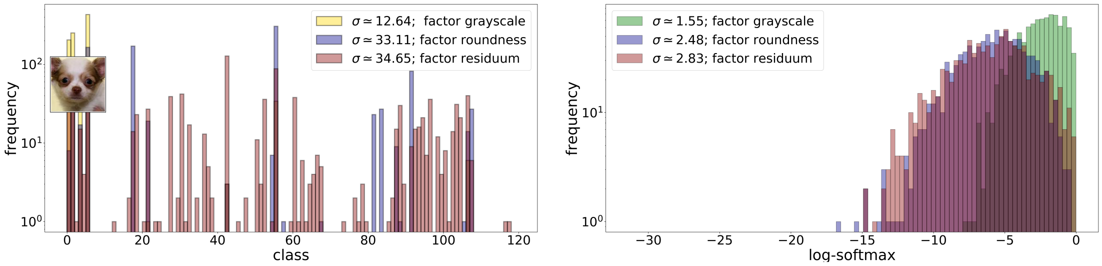

Figure 18: Output variance *per class* of a ResNet-50 classifier trained on AnimalFaces class identity, assessed via distribution of log-softmaxed logits and class predictions. T is trained to disentangle $\tilde{z}_0$ (residual), $\tilde{z}_1$ (roundness) and $\tilde{z}_2$ (greyscale). See Sec. 4.2.

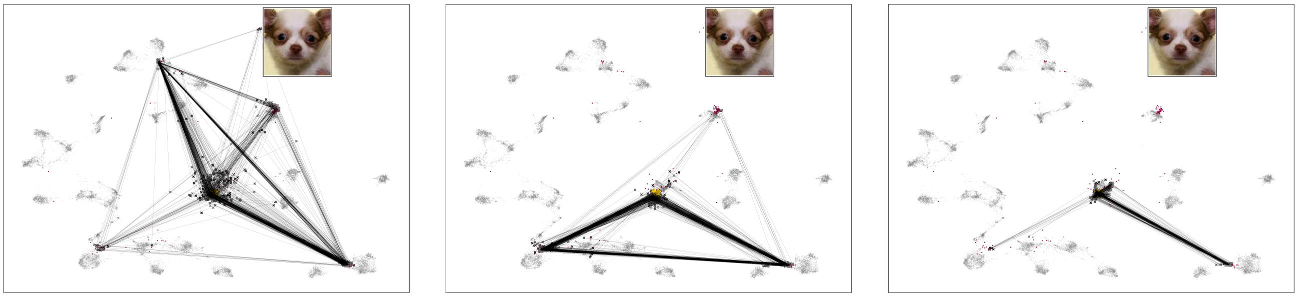

Figure 19: Left: Changing factor *residual* in AnimalFaces classifier's hidden representation. Middle: Changing factor *roundness*. Right: Changing factor *grayness*.

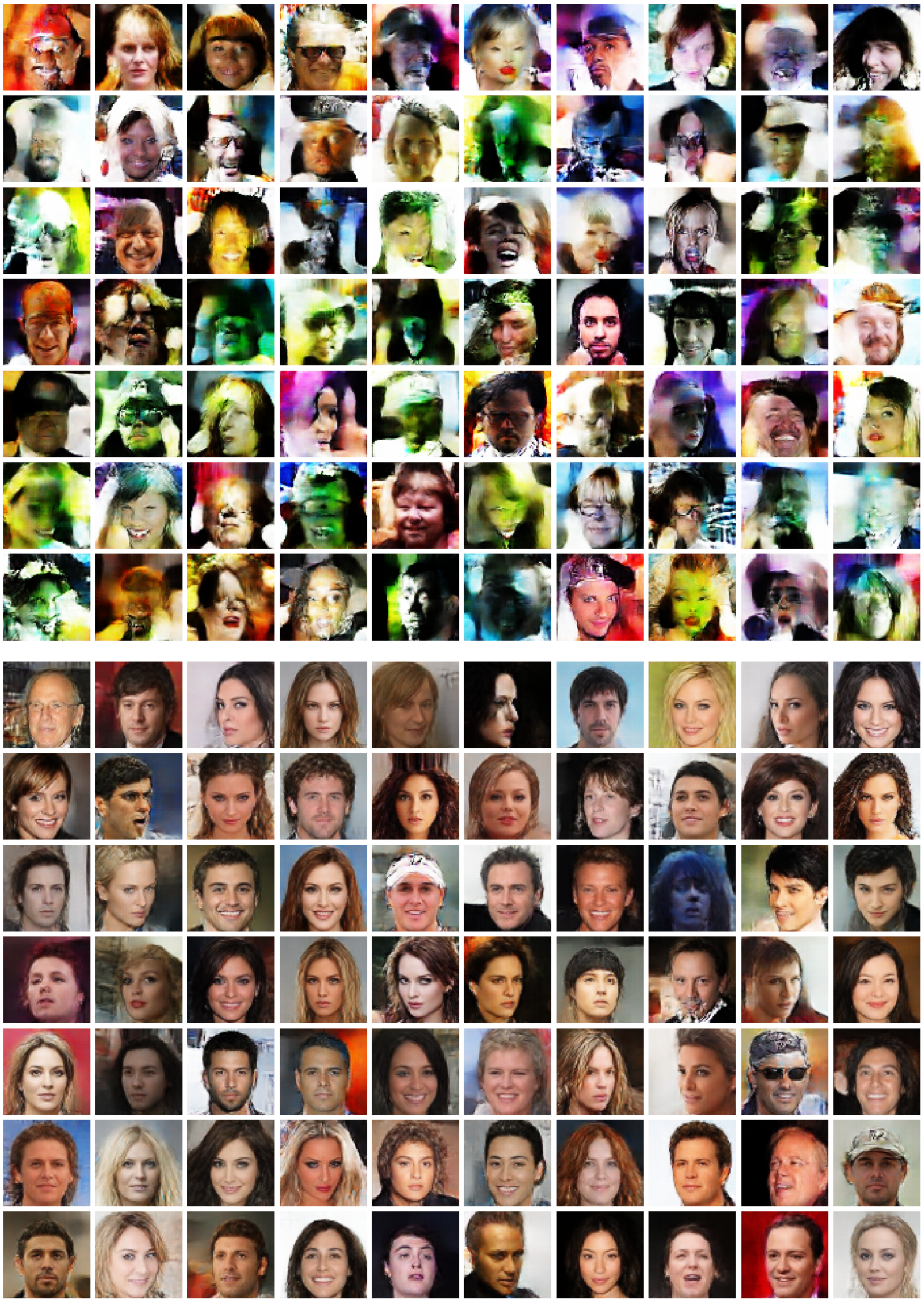

Figure 20: Samples on CelebA. Top: Decoded samples drawn from the prior of the autoencoder: $x = G(z),\ z \sim \mathcal{N}(0, \mathbf{1})$. Bottom: Decoded samples drawn from the prior of the transformer: $x = G(T^{-1}(\tilde{z})),\ \tilde{z} \sim \mathcal{N}(0, \mathbf{1})$.

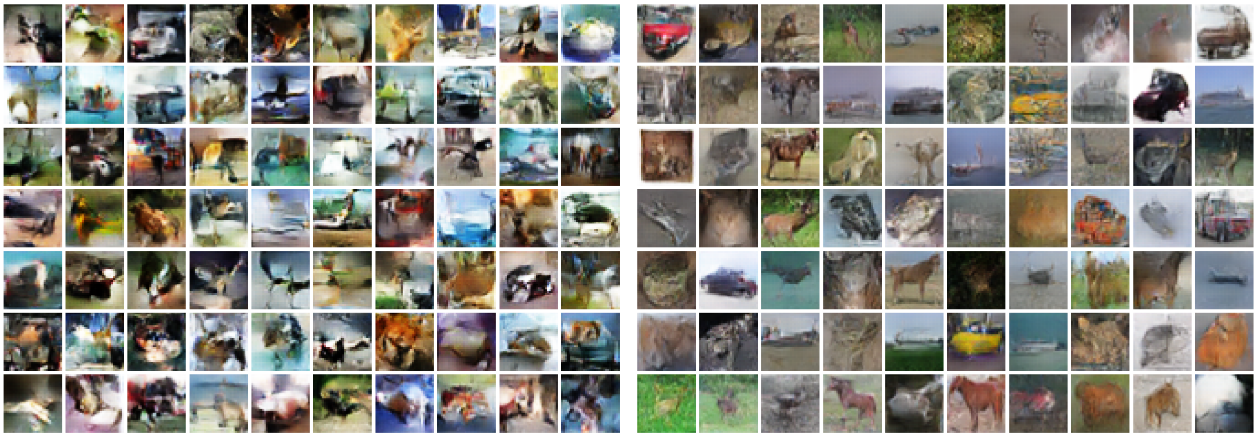

Figure 21: Samples on CIFAR-10. Left: Decoded samples drawn from the prior of the autoencoder: $x = G(z),\ z \sim \mathcal{N}(0, \mathbf{1})$. Right: Decoded samples drawn from the prior of the transformer: $x = G(T^{-1}(\tilde{z})),\ \tilde{z} \sim \mathcal{N}(0, \mathbf{1})$.

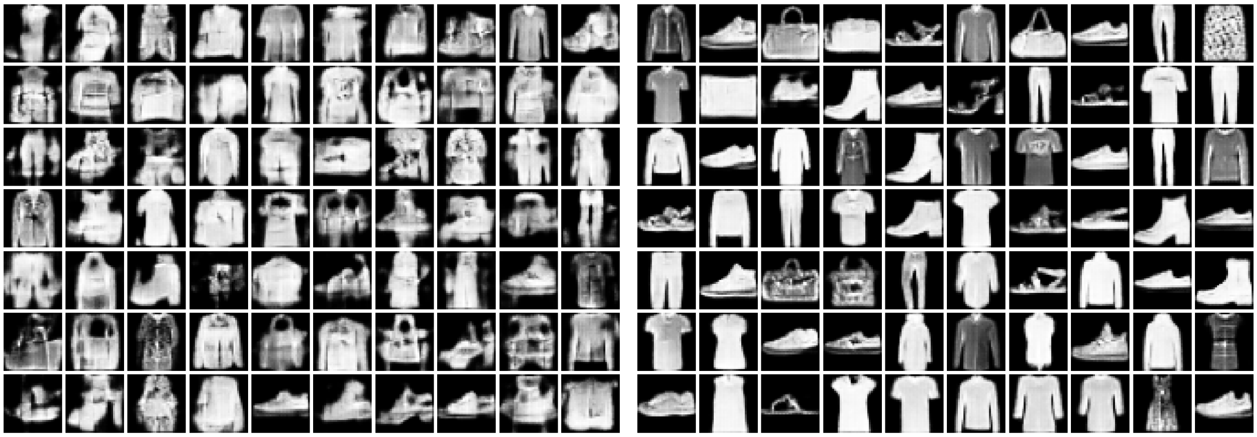

Figure 22: Samples on FashionMNIST. Left: Decoded samples drawn from the prior of the autoencoder: $x = G(z),\ z \sim \mathcal{N}(0, \mathbf{1})$. Right: Decoded samples drawn from the prior of the transformer: $x = G(T^{-1}(\tilde{z})),\ \tilde{z} \sim \mathcal{N}(0, \mathbf{1})$.