LEVERAGING RELATIONAL INFORMATION FOR LEARNING WEAKLY DISENTANGLED REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Disentanglement is a difficult property to enforce in neural representations. This might be due, in part, to a formalization of the disentanglement problem that focuses too heavily on separating relevant factors of variation of the data in single isolated dimensions of the neural representation. We argue that such a definition might be too restrictive and not necessarily beneficial in terms of downstream tasks. In this work, we present an alternative view over learning (weakly) disentangled representations, which leverages concepts from relational learning. We identify the regions of the latent space that correspond to specific instances of generative factors, and we learn the relationships among these regions in order to perform controlled changes to the latent codes. We also introduce a compound generative model that implements such a weak disentanglement approach. Our experiments shows that the learned representations can separate the relevant factors of variation in the data, while preserving the information needed for effectively generating high quality data samples.

1 INTRODUCTION

While trying to find a way to reproduce aspects of natural intelligence into artificial systems, researchers proposed the notion of *meta-priors*, first introduced by Bengio et al. (2013) and then further refined and expanded in Tschannen et al. (2018). A meta-prior is a generic assumption about the world that is expected to hold true for all possible tasks that an artificial agent might encounter in the future, thus providing a way to structure the learned representations in a useful way for possible downstream tasks. In the latest years, meta-priors have helped the representations learned by neural networks to reach levels of expressivity that were unthinkable just a few decades ago. Modern distributed representations can, for instance, disentangle factors of variations of the data, encode hierarchical features at different levels of abstractions, express the natural clustering organization of the data, and incorporate various types of supervised information (Tschannen et al., 2018).

Despite these achievements, finding a way for reliably enforcing different kinds of meta-prior is still an open research question. In particular, one of the most difficult meta-prior to impose on the learned representations is *disentanglement*. One of the challenges that arises when dealing with the disentanglement problem, is that a formal definition of what constitute a disentangled representation is still a matter of debate (Do & Tran, 2019). Many works just assume that a disentangled representation is a representation in which each latent dimension is responsible for encoding a single generative factor of the data. We argue that this intuitive definition can be too strict in general, as it is possible for distinct factors of variations to manifest themselves in the data only in an entangled way. Sometimes, only a subset of all the possible factors of variations is worth disentangling, while the others can be left entangled. Moreover, it has been shown that the imposition of this form of disentanglement on the learned representations can actually damage the overall performance on downstream tasks, instead of providing a clear benefit (Locatello et al., 2019a).

For these reasons, in this work we wish to introduce a different approach on disentanglement and disentangled representations, which we call *weak disentanglement*. A weakly disentangled representation is a representation where the generative factors are not encoded into specific separate dimensions. The information about the original values of generative factors is instead encoded into

different regions of the latent space, with each region identifying a specific combination of factors. Given a weakly disentangled representation, it is therefore possible to recover the original generative factors by checking in which region of the latent space that representation ends up.

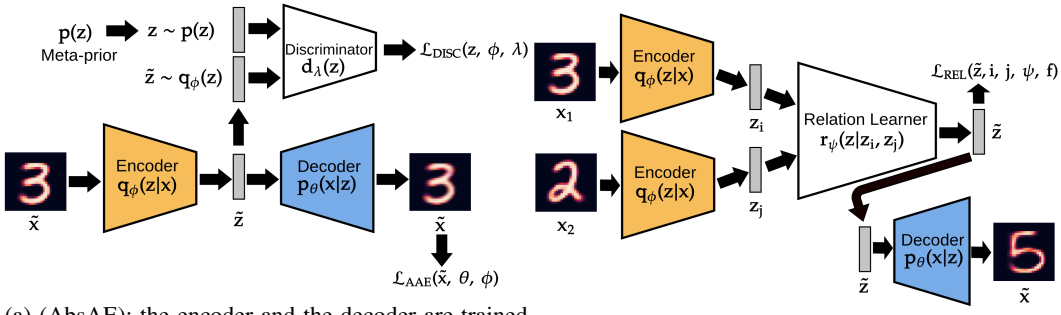
In particular, we propose a new generative neural model for the learning of weakly disentangled representations. The main components of this model are the Abstraction Autoencoder (AbsAE) and the Relational Learner (ReL). The AbsAE is an adversarial autoencoder (Makhzani et al., 2015) augmented with an adaptive prior distribution that is able to identify the regions of the latent space containing the relevant instances of generative factors, using only a minimal amount of supervised information. During training, the ReL learns how to navigate such structured latent space, moving the input representations into new regions of the same latent space according to a set of predefined relations. These modules, together, are able to learn representations that, while still being entangled from a "classic" point of view, allow for being easily manipulated in order to induce controlled changes on the chosen factors of variations. In the rest of this paper, we will show that this form of weak disentanglement can obtain representations that preserves all the relevant information for reconstructing the original data, while at the same time allowing for the manipulation of one or more factors of variations in a compositional way.

2 Related Works

After the inital introduction of the concept of meta-priors (Bengio et al., 2013; Tschannen et al., 2018), many works explored different ways to enforce meta-priors on the representations learned by autoencoders. It is possible to identify approximately three main approaches: i) Using regularization constraints on the encoder's posterior distribution $q_\phi(z|x)$, ii) Using architectural constraints on either the encoder $q_\phi(z|x)$ or the decoder $p_\theta(x|z)$ (or both) and iii) Choosing flexible prior distributions $p(z)$. Many works combine these approaches in order to force specific properties on the learned representations.

Focusing on disentanglement, the early methods are mainly concerned on re-weighting the second term of the ELBO, such as the β-VAE (Higgins et al., 2016), and the β-VAE-2 (Burgess et al., 2017). The main shortcoming of this approach is that disentanglement is achieved at the expense of reconstruction accuracy, hampering the performance on subsequent downstream tasks. Another line of works builds upon the ELBO decomposition provided by Hoffman & Johnson (2016) to separately penalize different terms, such as FactorVAE (Kim & Mnih, 2017), β-TCVAE (Chen et al., 2018), InfoVAE (Zhao et al., 2019) and DIP-VAE I & II (Kumar et al., 2017). They all apply several weighting factors on different parts of the ELBO in order to emphasize specific properties on the latent representations. For example, (Kim & Mnih, 2017) and (Chen et al., 2018) try to achieve disentanglement by encouraging the total correlation between the latent dimensions to be as low as possible. Other works such as HSIC-VAE (Lopez et al., 2018) and HFVAE (Esmaeili et al., 2019) try to enforce independence between groups of latent variables. While being able to isolate simple generative factor to some degree, in general such models struggles to achieve a reasonable disentanglement when the factor of variation cannot be identified by straightforward mathematical notions such as statistical independence (which is often the case with real-world data).

An interesting line of works tries to leverage different degrees of supervised information in order to achieve disentanglement (Louizos et al., 2015; Kulkarni et al., 2015; Lample et al., 2017; Locatello et al., 2019b; Gabbay et al., 2021). For example, Locatello et al. (2019b) relies on complete supervision of a small subset of training data. Gabbay et al. (2021) further relaxs these constraints by requiring only partial annotations on a subset of generative factors. However, the lack of a rich-structure in the latent space makes it impossible to associate a confidence level to the models' predictions. Some works in the field of concept learning also try to build a structured latent space distribution in order to isolate relevant high-level concepts associated with the data samples (Rostami et al., 2020) (Koh et al., 2020). In particular, Rostami et al. (2020) use an additional classification network on the latent space in order to cluster together representations associated to the same concept in a continual learning setting. Hosoya (2018) the authors strongly disentangle the latent space into group-common "content" variables and instance-specific "transformation" variables. These approaches are suitable for identifying the different values of a single relevant generative factor (i.e. the concept, or the data group), but cannot be applied when more than one factor needs to be disentangled from the data.



(a) (AbsAE): the encoder and the decoder are trained to learn the mapping between the data space and the latent space. The latent codes are forced to follow the prior distribution $p(z)$.

(b) (ReL): The relation learner is trained in the latent space. The encoder and the decoder are the same as AbsAE.

Figure 1: Overall proposed architecture, composed of the Abstraction Autoencoder (AbsAE) and the Relational Learner (ReL).

Finally, there exists a few works that focus on leveraging relational information among data samples. Locatello et al. (2020) uses pairs of images where the value of a random subset of generative factors is different. On the other hand, Chen & Batmanghelich (2020) uses weak supervision between a pair of images consisting in a similarity score about a factor to be disentangled. Bai et al. (2021) strongly disentangle representation of sequential data into "static" factors, that are constant along all the duration of the sequence, and "dynamic" factors, that vary across the timesteps. These approaches, while powerful in principle, typically require the training data to be structured in very specific ways in order to be able to make use of the supervised information available. This can limit their application to a wide variety of tasks.

3 Weak Disentanglement of Latent Representations

The section introduces our approach implementing the *weak disentanglement* meta-prior. A schematic view of the model's architecture is shown in Figure 1. The model loss is trained using the following objective:

$$\mathcal{L}(x, \theta, \phi, \lambda, \psi, f) = \mathcal{L}_{AE}(x, \theta, \phi) + \beta \mathcal{L}_{DISC}(z, \phi, \lambda) + \gamma \mathcal{L}_{REL}(z, \psi, f), \tag{1}$$

where θ, ϕ, λ and ψ are the parameters of the encoder, decoder, discriminator and relational learner respectively, x is a data sample and z is its corresponding latent representation, f is the particular relation that we would like to learn. The hyperparameters β and γ are used to adjust the importance of the different terms. The first two terms of Eq.1 correspond to the auto-encoding part of the model, while the last term is the relational part. The AbsAE's task is to learn a mapping from the data space to an abstract latent space, and vice-versa. The latent space is encouraged to follow a specific *meta-prior distribution* $p(z)$, an adaptive gaussian mixture (GM) distribution built ad-hoc for the task. This distribution is able to clusters different instances of the same generative factors into a similar region of the latent space. At the same time, the ReL is trained to learn relations between such regions of the latent space, exploiting the prior distribution $p(z)$ learned by the AbsAE. We alternate a training iteration of the AbsAE with a training iteration of the ReL in order to gradually learn both the relevant combinations of generative factors and the relations between them. In the rest of this section, we provide a detailed description of each of the modules.

3.1 Abstraction Autoencoder

The AbsAE, depicted in Figure 1a, is composed of two sub-networks: the *encoder* $q_\phi(z|x)$ and *decoder* $p_\theta(x|z)$, parameterized by ϕ and θ, respectively. The mapping between data space and

latent space is learned by optimizing the following maximum likelihood objective:

$$\max_{\theta,\phi} \quad \mathcal{L}_{AE}(x,\theta,\phi) + \beta\mathcal{L}_{DISC}(z,\phi) = \tag{2}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \beta D\left(q_\phi(z)||p(z)\right), \tag{3}$$

where D is an arbitrary divergence (such as the Kullback-Leibler divergence) and β is an hyperparameter controlling the amount of desired regularization. The first term of Eq.3 encourages the latent codes z to be an informative representation of the corresponding original input x, while the second term encourages z to follow a desired prior distribution $p(z)$. Since this second term is generally not computable for an arbitrary choice of $q_\phi(z)$ and $p(z)$(Mescheder et al., 2017), we estimate it using an additional *discriminator* network $d_\psi(z)$, parameterized by λ. Thus, the second term of Eq. 3 is optimized in an adversarial way via the following objective

$$\min_{\phi}\max_{\lambda} \quad \mathcal{L}_{DISC}(z,\phi,\lambda) = \tag{4}$$

$$= \mathbb{E}_{q_\phi(z)}[\log d_\lambda(z)] + \mathbb{E}_{p(z)}[\log(1-d_\lambda(z))]. \tag{5}$$

The flexibility introduced by the adversarial estimation of the divergence D allows us to chose any distribution $p(z)$ that best fits our needs. Since our goal is to identify and disentangle the different combinations of generative factors that appear in the data, we choose the prior distribution $p(z)$ to be a a gaussian mixture (GM)

$$p(z) = \frac{1}{N}\sum_{i=1}^{N} p_i(z) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{N}(\mu_i,\Sigma_i^2), \tag{6}$$

where N is the number of the distinct factor combinations in the data. The mean μ_i and the covariance Σ_i^2 of each prior component are estimated empirically from a small subset of supervised samples, containing ancillary information that describes properties of the data that are relevant for the relations we wish to learn (for practical examples of generative factor values see Section 4.1):

$$y = \{(y_{g_1}, y_{g_2}, ..., y_{g_K})\}_i^{N_y}, \tag{7}$$

where g_i is the i-th generative factor, y_{g_i} is the label associated to g_i, K is the total number of factors and N_y is the number of labelled data. We leverage the fact that the auto-encoding training procedure of AbsAE tends to naturally organize the latent space in an efficient way, with similar samples (i.e. samples for which the generative factors have the same value) that ends up to be encoded in the same region of the latent space. The labelled samples are a way to identify the relevant regions of interest, and the prior distribution $p(z)$ helps into shaping those regions into a GM distribution that is easy to model and manipulate in a meaningful way. This setting grants a high flexibility on what the relevant factors can be. For example, it is possible to specify only a subset of such factors of variations, so that the remaining factors will be treated as nuisances.

3.2 Relational Learner

The AbsAE is trained so that the encoder $q_\phi(z|x)$ and the decoder $p_\theta(x|z)$ provide a mapping between the raw data samples and the corresponding generative factors. This can be exploited by the ReL in order to efficiently learn relations between those factors. The ReL model (Figure 1b) is composed of the *relational learner* sub-network $r_\psi(z|z_1,..,z_N)$ that, in the case of a binary relation, is trained according to the following objective:

$$\max_{\psi} \quad \mathcal{L}_{REL}(z,i,j,\psi,f) = p_{f(i,j)}(z) \tag{8}$$

where $z \sim r_\psi(z|z_i,z_j)$, z_i and z_j representing encoded data samples belonging, respectively, to the i-th and the j-th gaussian of $p(z)$. The function $f : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ is any function with domain and range in $[1,N]$ that characterizes the specific relation to be learned. Note the the number of arguments of $\mathcal{L}_{REL}$ actually depends on the arity of the desired relation. For example, when considering images of natural numbers, assuming that a relevant factor of variation is the number identity, the *sum* relation can be learned by setting $f(i,j) = i+j$. Note that thus the prior distribution $p(z)$ is used to guide the learning process of the ReL, encouraging the model to encode the result into the desired component of the GM. This training can be done without the need of using additional data, as new samples can be drawn directly from the corresponding components of $p(z)$. Having the ReL

to operate in a structured latent space yields several advantages. First, in the latent space, a relation between generative factors is directly translated into a relation between components of $p(z)$. This means that the ReL can easily identify values of the generative factors of a data sample x just by checking which component of $p(z)$ is the most active for encoding x. Additionally, since $p(z)$ is known, it is always possible to associate a probability threshold α to each input sample and each model prediction. This is useful in several ways. For input samples, it allows to identify potentially adversarial samples that are too far-away from the empirical data distribution (i.e. samples that ends up getting encoded into very low-probability regions of the latent space). For model's predictions, it provides additional useful information about the confidence level of the predictions.

4 EXPERIMENTS

We designed a set of experiments in order to inspect the following questions: 1) How well the AbsAE is capable of correctly clustering the values of generative factors in the latent space? 2) How well the ReL is capable of manipulating the latent representation in order to implement the desired relations? 3) How much the learned representations can be considered disentangled? In the rest of this section we describe accurately describe the experimental setting. First, we give a detailed account of the preprocessing procedures that is common to all the experiments, Then, we describe the crucial design choices made to implement the specific experiments[1].

4.1 DATASETS AND PREPROCESSING

We consider three datasets: the newly introduced Hand-Written Formulas (HWF) dataset (Li et al., 2020), and the well-known dSprites (Higgins et al., 2016) and Shapes3D (Kim & Mnih, 2017) datasets. The HWF dataset contains images of hand-written math formulas, consisting of the ten digits and three basic math operators. The dSprites dataset contains images of various 2-dimensional shapes, in different positions, scales and orientations. The Shapes3D dataset contains images of various 3-dimensional shapes in different colors combinations of (floor, shape, background) and rotations. In the case of HWF, the only relevant generative factor considered is the *digit/operator identity*, for a total of 13 values (10 digits plus the sum, subtraction and multiplication operators), while everything else is considered a nuisance. For dSprites, we keep 3 values for the *horizontal position* (left, center, right), 3 values for the *vertical position* (up, center, down), and 3 values for the *shape* (ellipse, square, heart). The *scale* and *orientation* are nuisances factors. Finally, for Shapes3D, we keep 10 values for *object color*, 4 values for *shape* (cude, sphere, cylinder, ellipsoid) and 3 values for *scale* (small, medium, big), while considering *floor color*, *background color* and *orientation* nuisance factors. Thus, we end up having 13 factor combinations for HWF, 27 for dSprites and 120 for Shapes3D. Each of these combinations is represented as a single gaussian of the prior distribution $p(z)$. Regarding relations, on the HWF dataset we consider the *sum*, *subtraction* and *multiplication* binary relations. For dSprites, we consider 5 relations: *move_left*, *move_right*, *move_up*, *move_down* and *change_shape*. In Shapes3D, 5 relations are considered as well: +*_hue*, -*_hue*, *change_shape*, +*_scale*, -*_scale*.

In the case of HWF, the relations are chosen in order to reflect our intuitive understanding of the corresponding math operators. In dSprites and Shapes3D, on the other hand, the chosen relation have the effect of changing the value of a single factor of variation, while leaving the others unchanged. No restriction is imposed on the nuisance factors, that are able to vary freely when applying relations on the latent codes. We also perform data augmentation, corrupting the data samples by adding either bernoullian or gaussian noise to the original images. We split each dataset in training, validation and test set. The validation set is used to select the best values of the hyperparameters of the models, while the test set is used to compute the final results. The validation set is created by taking 10% of the available training set. Similarly, the test set is created from 20% of the total available data.

4.2 TRAINING SETTINGS

The encoder and decoder modules of AbsAE are implemented as a multilayer CNN architecture. Complete details about the chosen architecture are reported in Appendix A. The hyperparameter

[1]The source code of the project is included in the supplementary files, and will be publicly released upon acceptance.

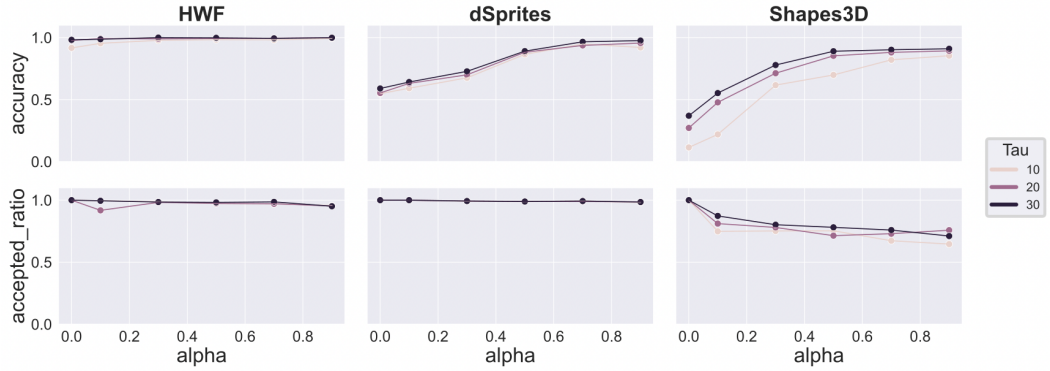


Figure 2: Accuracy and accepted ratio of the AbsAE for different α thresholds and different amounts of supervision. The results are computed over the HWF, dSprites and Shapes3D datasets. τ is the amount of supervision, measured in number of samples.

have been tuned using trial and error, selecting the combination yielding the best performance on the validation set. All the networks used in the experiments are deterministic, i.e. $q_\phi(z|x)$ and $p_\theta(x|z)$ are Dirac's delta functions. The discriminator and the relational networks are implemented as a 3-layer MLP with 1024 units each. The hidden neurons use hyperbolic tangent non-linearities, while the output neurons use the sigmoid. In the experiments, we set the number of latent factors $N_z = 8$ for HWF and dSprites, $N_z = 16$ for Shapes3D. All tasks use a batch size of 1024 for the AbsAE's training and 128 for the ReL's training. We use the Adam optimizer with learning rate of 10^{-4} for HWF and dSprites, 10^{-5} for Shapes3D.

Initially, the training starts in a *warmup phase*, were only the AbsAE is active. In this phase we set $p(z) \sim \text{Uniform}(-1, 1)$, to encourage the latent codes to spread evenly across the latent space. During this phase only the AbsAE is trained. After 1000 epochs (5000 for Shapes3D), the *full training phase* begins: the prior distribution is changed to the GM prior $p(z) \sim \frac{1}{N}\sum_{i=0}^{N}\mathcal{N}(\mu_i, \Sigma_i)$ described in Section 3.1. In this phase we also start the training of the ReL: the first step is to construct a training sample with the following structure:

$$(z_{in_1}, ..., z_{in_R}, z_{rel}, z_{out}) \tag{9}$$

where $z_{in_1}, ..., z_{in_R}$ are the input latent codes of the relation, R is the arity of the chosen relation, z_{out} is the target latent code, and z_{rel} is a code that identifies the relation. z_{rel} can either be a symbolic code (such a categorical variable) or a latent code representing the specific relation. Therefore, a training sample for the HWF dataset can be (z_2, z_3, z_+, z_5), where z_2 and z_3 are sampled from the prior's components corresponding to the digits "2" and "3", z_+ is sampled from the "+" component, and z_5 is sampled from the "5" component. On the other hand, when training the *move_up* relation on the dSprites dataset, a possible training sample will have the form $(z_{(\text{center,center,square})}, z_{\text{move\_up}}, z_{(\text{center,up,square})})$, where $z_{(\text{center,center,square})}$ is obtained by sampling from the prior component corresponding to the factor combination {x_position=*center*, y_position=*center*, shape=*square*}, $z_{(\text{center,up,square})}$) is sampled from the gaussian corresponding to {x_position=*center*, y_position=*up*, shape=*square*}, and $z_{\text{move\_up}}$ is a categorical variable that identifies the *move_up* relation. Unlike HWF, the dSprites dataset does not contain a way to identify the relations directly in the data, hence the need for an additional categorical variable for encoding relations.

Thus, the ReL learns how to perform changes to the latent codes from the starting region of the latent space to another one, according to the specific relation. Note that the training of the ReL can be done without the need of additional data, as the training samples can be constructed by directly sampling from $p(z)$. The elements of the training tuple are then concatenated together and sent in input to the ReL. We alternate a training iteration of the AbsAE with a training iteration of the ReL in order to learn both objectives at the same time. Training is carried on for 5000 more epochs (10000 for Shapes3D), for a total of 6000 epochs (15000).

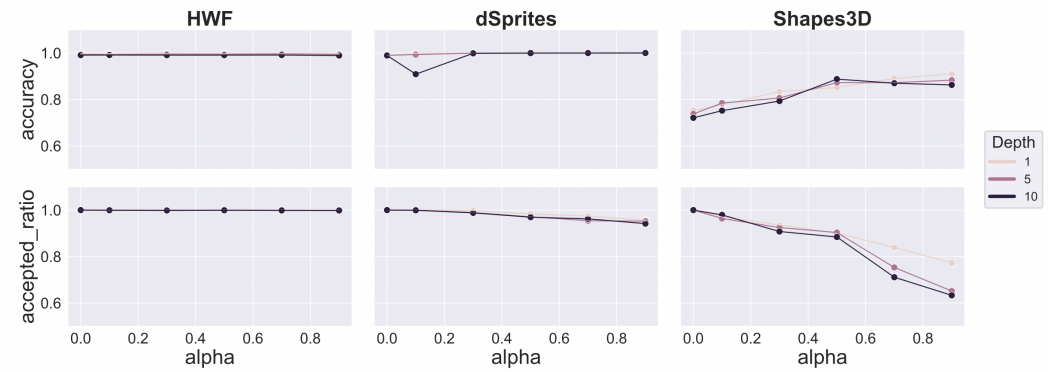


Figure 3: Accuracy and accepted ratio of the ReL for different α thresholds and different relational depths. The results are computed over the HWF, dSprites and Shapes3D datasets.

4.3 Structure of the Latent Space

The first set of experiments is meant to assess the capability of the prior distribution $p(z)$ to effectively identify and cluster the relevant regions of the latent space that identify a particular combination of generative factors. In Figure 2 are reported the clustering accuracy and the accepted ratio of the AbsAE on the test set of HWF, dSprites and shapes3D datasets. The results are obtained by first encoding a test sample x to get its latent representation z. The classification is then performed by selecting the prior component that is more likely to have generated z. We report the results for different values of α, that is, we compute the accuracy only on the test samples that reach a certain probability threshold α for at least one component of $p(z)$. If a sample does not reach the desired probability for any component, it is rejected, and the classification is not performed. We also report the ratio of test samples that the model does not reject (i.e. the *acceptance ratio*) for each α threshold. A high accepted ratio means that a high proportion of the test samples has a high probability under the prior.

The results in Figure 2 show that the clustering accuracy increases as α gets higher. For the HWF dataset, the model reaches over 90% accuracy for each α thresholds. In the dSprites dataset, it takes longer to exceed 90% accuracy, but the acceptance ratio stays very high for each α thresholds, meaning that the model is quite confident in its classifications. Shapes3D is, perhaps unsurprisingly, the most challenging dataset. Nevertheless, the AbsAE is still able to reach 90% clustering accuracy for $\alpha \geq 0.7$, while keeping an acceptance ratio of over 75%. We compare our results on the HWF dataset with the work of Li et al. (2020). Despite the more challenging setting (as the AbsAE is a generative model, our representations needs to also keep all the information needed for a good reconstruction the original data, whereas Li et al. (2020) are only concerned with symbol classification) our model yields better results, obtaining higher accuracies for any values of α. In Appendix B we include some generated images for the different datasets, while in Appendix C we report the data used to construct Figure 2.

4.4 Manipulation of Latent Codes

The second set of experiments has the goal to test how well the ReL is capable of manipulating the learned latent representations in order to implement the desired relations. The relation accuracy of the model is computed by first sampling a latent code z_{in} (or two, in the case of the binary relations of HWF) from the prior $p(z)$. After that, we choose a random relation z_{rel} among the one that are available for that dataset and we feed both the z_{in}s and z_{rel} in input to the ReL. If the *depth* parameter is more than 1, we repeat this process accordingly, using the output of the ReL at the current step as input for the next step. The final output of the ReL z_{out} is then classified by selecting the component of the GM prior with the highest probability of having generated z_{out}. Results are reported in Figure 3. We take into consideration different α thresholds and different depths of the relations.

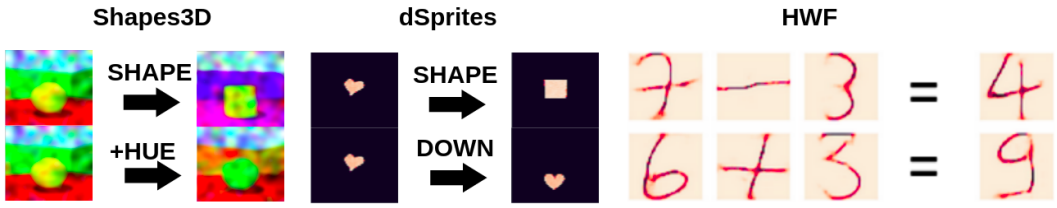


Figure 4: Samples of relations learned by the ReL on different datasets (additional samples are available in Appendix B).

Table 1: Disentanglement scores of latent representations on different datasets.

	dSprites			**Shapes3D**		
	DCI	MIG	SAP	DCI	MIG	SAP
β-VAE	0.4566	0.602	0.67	0.153	0.270	0.131
FactorVAE	0.8942	0.98	0.61	0.371	0.370	0.402
(Locatello et al., 2019b)	0.533	0.01	0.01	0.48	0.05	0.08
(Gabbay et al., 2021)	0.8366	0.14	0.57	**1.0**	0.3	**1.0**
Ours	**0.9543**	**0.994**	**0.7728**	0.6921	**0.6897**	0.5007

The results shows that the performance of the ReL are only marginally affected by the specific α thresholds. There is a general tendency for the accuracy to increase, and the accepted ratio to decrease, as α get higher, but this mainly happens on the more challenging Shapes3D dataset. In the case of HWF and dSprites datasets, the accuracy stays at around 99% and the accepted ratio is above 95% for different values of α and different depths. This is a sign that the ReL can reliably learn the desired relations with high accuracy. The performance do not seem to be much affected by *depth* parameter, meaning that the ReL is able to applying in cascade more than one relation to the same latent code without losing accuracy. Therefore, the learned relations can be combined compositionally in order to perform complex transformations of the initial latent representation. In Figure 4 we report qualitative samples obtained from the ReL on different datasets (additional samples are available in Appendix B). In Appendix C we report the data used to construct Figure 3.

4.5 Disentanglement of Learned Representations

Lastly, we wish to investigate how much the representations learned by our model can be considered disentangled by the "classic" standards, while still keeping all the information needed to reconstruct the original data sample. We trained a β-VAE (Higgins et al., 2016) and FactorVAE (Kim & Mnih, 2017) on both the dSprites and Shapes3D datasets for comparison (the HWF dataset is not used, as disentanglement can only be measured when there are two or more relevant factors of variation in the data). In order to ensure comparability of results, we use the same encoder-decoder architectures as the one described in Appendix A. We also keep the same learning rates and train for the same number of epochs as the one described in 4.2. We then repeats the experiments for different values of β (note that, in FactorVAE, we consider β to be the hyperparameter controlling only the total correlation term of the loss function). The ideal model would score highly in the various metric, while keeping the reconstruction error as low as possible for different values of β. Hence, the better models are the ones ending up in the upper-left region of the plots. In Figure 5 we plot the scores of popular disentangled metrics: SAP score (Kumar et al., 2017), MIG score Chen et al. (2018), and DCI score (Eastwood & Williams, 2018) against the reconstruction accuracy of our model on the dSprites and Shapes3D datasets (in the case of DCI, we report the average of *disentanglement*, *completeness* and *informativeness* scores). Since the above metrics assume to receive an input representations that follows the classic notion of disentanglement (i.e. where each individual dimension is responsible for encoding a single factor of variation), we transform each latent codes into its corresponding generative factors before computing the metrics for our model. This step can be done efficiently, as all the information about the factor of variations can be inferred just by classifying the latent code as described in Section 3.2.

Figure 5: Disentanglement/reconstruction trade-off of the models on the considered datasets. The disentanglement metric (y-axis) is plotted against the reconstruction error (x-axis).

The results of Figure 5 shows that our model's representations offer the best tradeoff between reconstruction and disentanglement. Our representation overall is not losing much reconstruction information as β is increased. On the other hand, the β-VAE's disentanglement scores can only be improved at the expense of reconstruction error, which begin to increase quickly as β becomes larger. FactorVAE's representation ends up in between, still not able to reach the same disentanglement/reconstruction tradeoff as our model. Finally, in Table 1 we directly compare the disentanglement performance of our model against the recent works of Locatello et al. (2019b) and Gabbay et al. (2021), as well as the β-VAE and FactorVAE baseline models. The results show that our model is able to reach superior disentanglement performance on the dSprites dataset, while still being competitive with the other state-of-the-art models on Shapes3D. Overall, it seems that our disentanglement disentanglement approach is able to identify and encode the relevant factors of variation without affecting the reconstruction power of learned representations.

5 Conclusion

We proposed the *weak disentanglement* meta-prior, a method for implementing disentanglement of latent representations of generative models by leveraging additional relational information. We presented a new generative model that implements our approach, divided into an auto-encoding part (AbsAE) and a relational learning part (ReL). We tested our approach on three different datasets of increasing complexity. The experiments shows that the AbsAE is able to identify and isolate the relevant regions of the latent space with high accuracy. The ReL is able to correctly manipulate the latent representations, even when applying multiple relations in sequence on the same representation. Finally, the learned representations yields better disentanglement scores when tested against similar models that rely on the "classic" notion of disentanglement, while preserving the information needed to achieve a good reconstruction of the original data sample, showing that our approach can be a viable option for disentanglement. The imposed structure of the latent space makes the model robust to potentially adversarial sample, as well as providing additional information about the confidence of individual predictions. In the future, we plan to further refine the structure of latent space learned by the AbsAE. It could be useful to encode the generative factors of the data in different latent spaces to encourage modularity of learned representations and to prevent the number of gaussian component of $p(z)$ to become too large when the number of values of generative factors increases. Another research direction is in finding a more expressive prior distribution $p(z)$. We also plan to enhance the ReL's overall architecture, possibly employing a graph neural network (Bacciu et al., 2020) to learn more expressive relations over the data.

ETHIC STATEMENT

The authors declare that they have no conflicts of interest. This article does not contain any studies involving animals or humans participants performed by any of the authors.

REPRODUCIBILITY STATEMENT

The theoretical description of the model is given in Section 3. In Sections 4.1 and 4.2 are accurately reported the data preprocessing and training procedures. We include references to all datasets used in this paper. The individual experiments are described in detail in Sections 4.3, 4.4 and 4.5. Appendix A reports additional information about the architecture of the model's components. The full source code of the project is included in the supplementary files, and will be publicly released upon acceptance.

REFERENCES

Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203 – 221, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.06.006. URL http://www.sciencedirect.com/science/article/pii/S0893608020302197.

Junwen Bai, Weiran Wang, and Carla Gomes. Contrastively disentangled sequential variational autoencoder. *NeurIPS2021*, 2021.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *2017 NIPS Workshop on Learning Disentangled Representations*, 2017.

Junxiang Chen and Kayhan Batmanghelich. Weakly supervised disentanglement by pairwise similarities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3495–3502, 2020.

Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pp. 2610–2620, 2018.

Kien Do and Truyen Tran. Theory and evaluation metrics for learning disentangled representations. *arXiv preprint arXiv:1908.09961*, 2019.

Cian Eastwood and Christopher KI Williams. A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*, 2018.

Babak Esmaeili, Hao Wu, Sarthak Jain, Alican Bozkurt, Narayanaswamy Siddharth, Brooks Paige, Dana H Brooks, Jennifer Dy, and Jan-Willem Meent. Structured disentangled representations. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2525–2534, 2019.

Aviv Gabbay, Niv Cohen, and Yedid Hoshen. An image is worth more than a thousand words: Towards disentanglement in the wild. *arXiv preprint arXiv:2106.15610*, 2021.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR2016*, 2016.

Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1, pp. 2, 2016.

Haruo Hosoya. Group-based learning of disentangled representations with generalizability for novel contents. *IJCAI2019*, 2018.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *Learning Disentangled Representations 2017 NIPS Workshop*, 2017.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International Conference on Machine Learning*, pp. 5338–5348. PMLR, 2020.

Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pp. 2539–2547, 2015.

Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *ICLR 2018*, 2017.

Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. Fader networks: Manipulating images by sliding attributes. In *Advances in neural information processing systems*, pp. 5967–5976, 2017.

Qing Li, Siyuan Huang, Yining Hong, Yixin Chen, Ying Nian Wu, and Song-Chun Zhu. Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In *International Conference on Machine Learning*, pp. 5884–5894. PMLR, 2020.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4114–4124. PMLR, 2019a.

Francesco Locatello, Michael Tschannen, Stefan Bauer, Gunnar Rätsch, Bernhard Schölkopf, and Olivier Bachem. Disentangling factors of variation using few labels. *arXiv preprint arXiv:1905.01258*, 2019b.

Francesco Locatello, Ben Poole, Gunnar Rätsch, Bernhard Schölkopf, Olivier Bachem, and Michael Tschannen. Weakly-supervised disentanglement without compromises. In *International Conference on Machine Learning*, pp. 6348–6359. PMLR, 2020.

Romain Lopez, Jeffrey Regier, Michael I Jordan, and Nir Yosef. Information constraints on auto-encoding variational bayes. In *Advances in Neural Information Processing Systems*, pp. 6114–6125, 2018.

Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *Proceedings of the 34th International Conference on Machine Learning, PMLR 70:2391-2400*, 2017.

Mohammad Rostami, Soheil Kolouri, Praveen Pilly, and James McClelland. Generative continual concept learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5545–5552, 2020.

Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *Workshop on Bayesian Deep Learning (NeurIPS 2018)*, 2018.

Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Balancing learning and inference in variational autoencoders. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 5885–5892, 2019.

A ARCHITECTURAL DETAILS

Due to the different image size and number of channels, each convolutional architecture is slightly adapted to each dataset. HWF uses 45x45 black and white images, dSprites 64x64 black and white images, Shapes3D 64x64 RGB images. For each dataset, the discriminator and the ReL modules are implemented as MLP with 3 tanh layers of 1024 neurons each. We report the detailed architectures of the encoders in Table 2 and of decoders in Table 3.

Table 2: Architectures of the encoder modules of the AbsAE, for the different datasets.

Dataset	Encoder
HWF	Conv2D(in_channels=1, out_channels=32, kernel_size=(5, 5), stride=2, padding=0) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Conv2D(in_channels=32, out_channels=32, kernel_size=(5, 5), stride=2, padding=0) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Conv2D(in_channels=32, out_channels=32, kernel_size=(3, 3), stride=2, padding=0) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Flatten() Linear(n_neurons=1024) LeakyReLU(α=0.1) Linear(n_neurons=8)
dSprites	Conv2D(in_channels=1, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Conv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Conv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Conv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Flatten() Linear(n_neurons=1024) LeakyReLU(α=0.1) Linear(n_neurons=8)
Shapes3D	Conv2D(in_channels=3, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Conv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Conv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Conv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Flatten() Linear(n_neurons=1024) LeakyReLU(α=0.1) Linear(n_neurons=16)

Table 3: Architectures of the decoder modules of the AbsAE, for the different datasets.

Dataset	Decoder
HWF	Linear(n_neurons=1024) LeakyReLU(α=0.1) Linear(n_neurons=32*4*4) Reshape(32, 4, 4) Deconv2D(in_channels=32, out_channels=32, kernel_size=(3, 3), stride=2, padding=0) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Deconv2D(in_channels=32, out_channels=32, kernel_size=(5, 5), stride=2, padding=0) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Deconv2D(in_channels=32, out_channels=32, kernel_size=(5, 5), stride=2, padding=0)
dSprites	Linear(n_neurons=1024) LeakyReLU(α=0.1) Linear(n_neurons=32*6*6) Reshape(32, 6, 6) Deconv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Deconv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Deconv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=32) LeakyReLU(α=0.1) Deconv2D(in_channels=32, out_channels=32, kernel_size=(4, 4), stride=2, padding=1)
Shapes3D	Linear(n_neurons=1024) LeakyReLU(α=0.1) Linear(n_neurons=64*6*6) Reshape(64, 6, 6) Deconv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Deconv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Deconv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1) Batchnorm(n_channels=64) LeakyReLU(α=0.1) Deconv2D(in_channels=64, out_channels=64, kernel_size=(4, 4), stride=2, padding=1)

B QUALITATIVE SAMPLES

In this appendix we provide a set of qualitative samples generated from our models. In Figure 6, Figure 7 and Figure 8 are reported some generated images for the HWF, dSprites and Shapes3D datasets, respectively. The images have been generated by decoding the means of the gaussians that composed the prior distribution $p(z)$. Each image therefore shows a particular combination of values of generative factors that has been identified and isolated by the AbsAE. In Figure 9, Figure 11 and Figure 12 are reported some manipulations performed by the ReL on the different datasets. In Figure 10 we show some relational samples obtained by modifying the ReL network in various ways. It is possible to see that the ReL is able to change only the selected generative factor without affecting the others, possibly letting all other nuisance factors change freely (e.g. the rotation of dSprites samples, or the floor color of Shapes3D samples).

Figure 6: Samples generated from the HWF dataset, obtained by decoding the means of the 13 gaussians that compose the prior $p(z)$. In HWF there is only a generative factor, that is, the identity of the digit/math operator (13 total values).

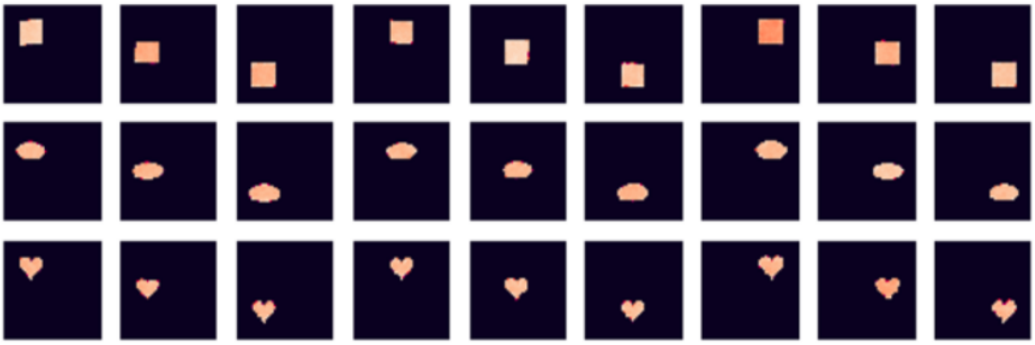


Figure 7: Samples generated from the dSprites dataset, obtained by decoding the means of the 27 gaussians that compose the prior $p(z)$. The dSprites dataset is the combination of three generative factors: *shape* (3 values), *x_position* (3 values), *y_position* (3 values).

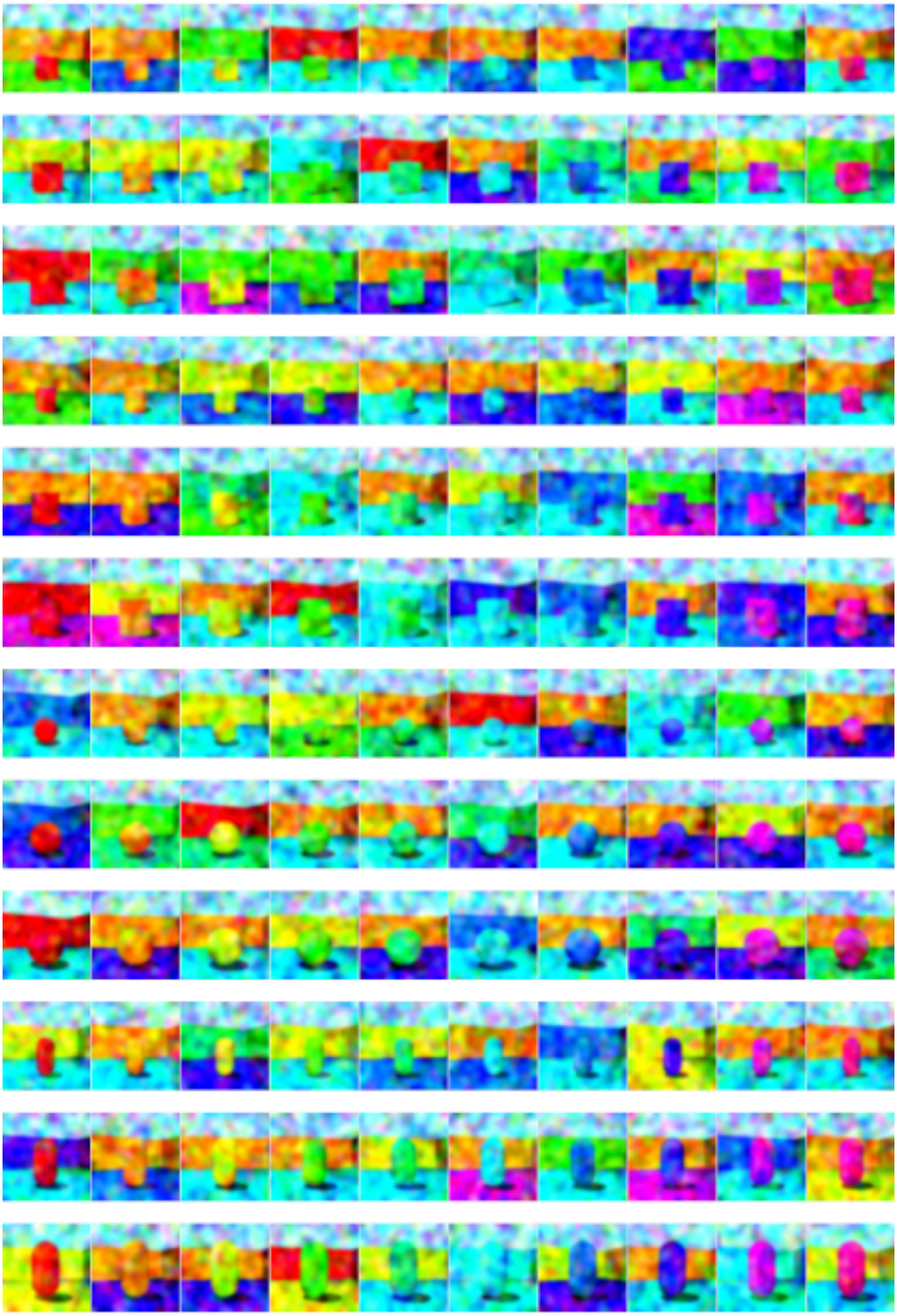


Figure 8: Samples generated from the Shapes3D dataset, obtained by decoding the means of the 120 gaussians that compose the prior $p(z)$. The Shapes3D dataset is the combination of three generative factors: *object_hue* (10 values), *object_scale* (3 values), *object_shape* (4 values). The remaining factors are considered nuisances.

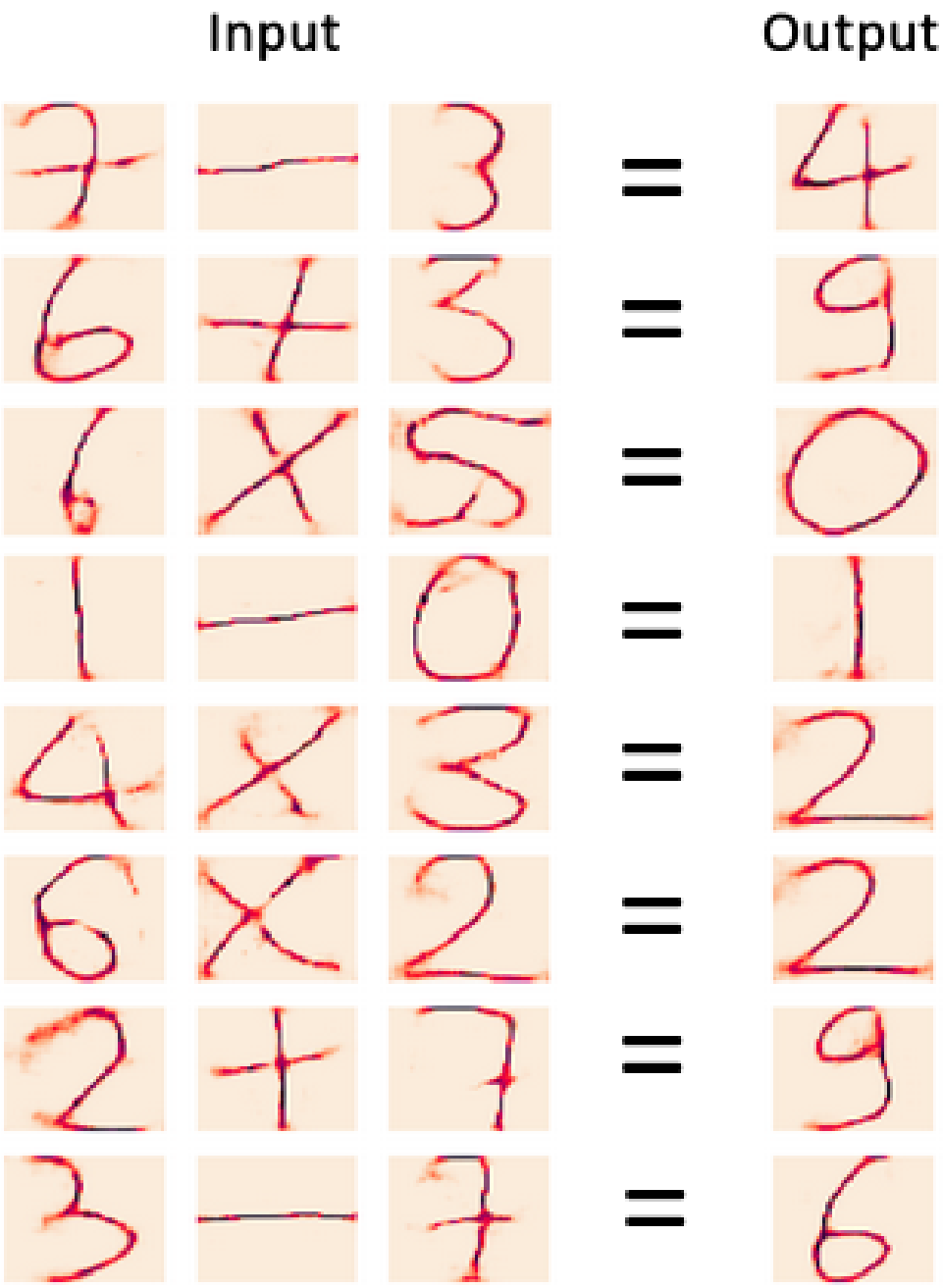


Figure 9: Examples of latent codes manipulation by the ReL on the HWF dataset. The samples have been obtained by feeding in input to the ReL the concatenation of the encoded inputs, and then decoding the corresponding output (please note that all the math relations are to be intended as module 10).

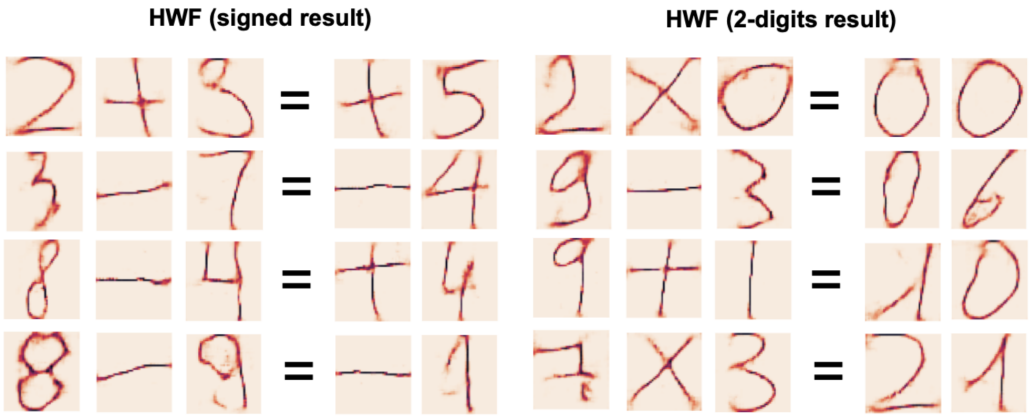


Figure 10: Examples of latent codes manipulation by the ReL on the HWF dataset. Here we show two different variations of the ReL. On the left hand side, the ReL is modified to output the sign of the results. On the right hand side, the ReL is modified to output 2-digits results.

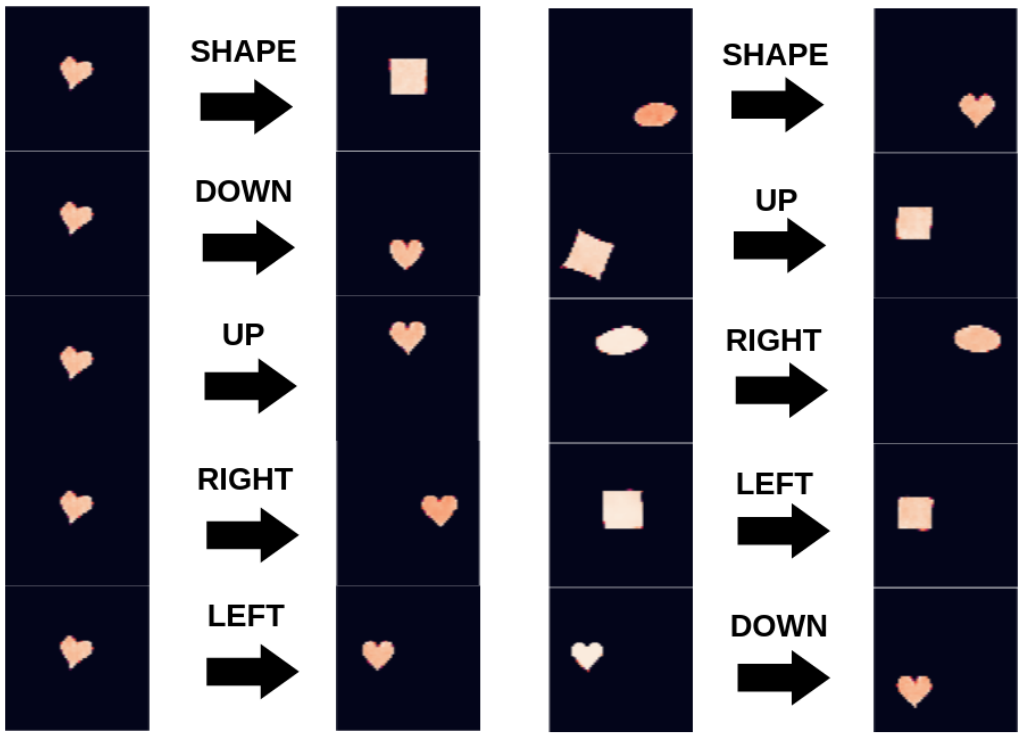


Figure 11: Examples of latent codes manipulation by the ReL on the dSprites dataset. The samples have been obtained by feeding in input to the ReL an encoded image, selecting a relation (represented as a categorical variable), and decoding the corresponding output.

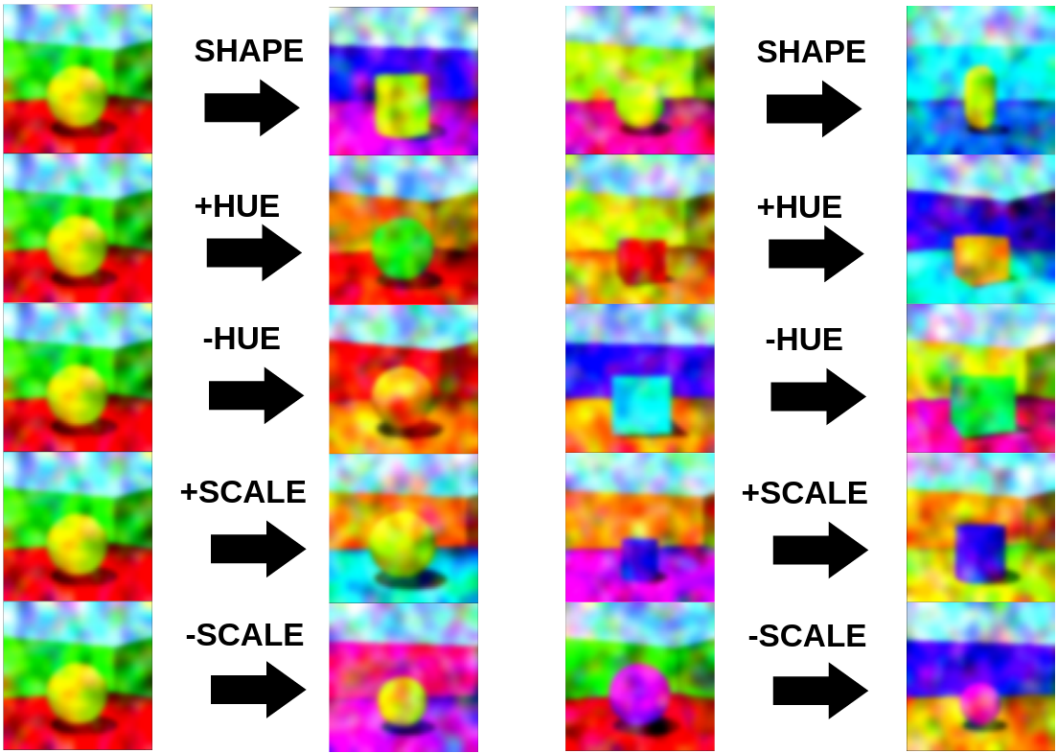


Figure 12: Examples of latent codes manipulation by the ReL on the Shapes3D dataset. The samples have been obtained by feeding in input to the ReL an encoded image, selecting a relation (represented as a categorical variable), and decoding the corresponding output.

C ADDITIONAL TABLES

This appendix contains the data tables used to produce Figure 2 and Figure 3. Table 4 reports accuracy and accepted ratio of the AbsAE for different α thresholds and different amounts of supervised samples. Table 5 reports accuracy and accepted ratio of the ReL for different α thresholds and different relational depths.

Table 4: Latent space classification accuracy of the AbsAE on the HWF, dSprites and Shapes3D datasets. ACC denotes the accuracy, AR the accepted ratio. τ is the supervision amount, measured in number of samples.

		HWF		**dSprites**		**Shapes3D**	
α	τ	ACC	AR	ACC	AR	ACC	AR
Li et al. (2020)	–	0.997	1.0	–	–	–	–
AbsAE, α=0.0	10	0.917	1.0	0.548	1.0	0.115	1.0
	20	0.977	1.0	0.554	1.0	0.272	1.0
	30	0.982	1.0	0.590	1.0	0.370	1.0
AbsAE, α=0.1	10	0.954	0.995	0.592	1.0	0.220	0.749
	20	0.992	0.918	0.631	0.999	0.478	0.811
	30	0.987	0.995	0.642	1.0	0.553	0.873
AbsAE, α=0.3	10	0.978	0.988	0.676	0.991	0.617	0.752
	20	0.990	0.982	0.700	0.993	0.713	0.780
	30	1.0	0.985	0.728	0.993	0.779	0.802
AbsAE, α=0.5	10	0.983	0.973	0.865	0.988	0.699	0.753
	20	0.993	0.974	0.882	0.990	0.853	0.714
	30	0.998	0.982	0.891	0.989	0.890	0.781
AbsAE, α=0.7	10	0.982	0.970	0.948	0.994	0.821	0.673
	20	0.995	0.971	0.937	0.989	0.881	0.738
	30	0.994	0.986	0.966	0.993	0.902	0.759
AbsAE, α=0.9	10	0.993	0.957	0.921	0.981	0.854	0.646
	20	0.999	0.953	0.956	0.987	0.893	0.758
	30	0.999	0.951	0.976	0.985	0.910	0.711

Table 5: Relational accuracy of the ReL on the HWF, dSprites and Shapes3D datasets. ACC denotes the accuracy, AR the accepted ratio.

		HWF		**dSprites**		**Shapes3D**	
α	Depth	ACC	AR	ACC	AR	ACC	AR
Li et al. (2020)	1	0.985	1.0	–	–	–	–
ReL, α=0.0	1	0.9966	1.0	0.9896	1.0	0.7521	1.0
	5	0.9939	1.0	0.9898	1.0	0.7378	1.0
	10	0.9909	1.0	0.9894	1.0	0.7210	1.0
ReL, α=0.1	1	0.9971	0.9984	0.9986	0.9998	0.7774	0.9702
	5	0.9930	0.9994	0.9932	1.0	0.7845	0.9638
	10	0.9913	0.9993	0.9891	0.9993	0.7519	0.9793
ReL, α=0.3	1	0.9985	0.9989	0.9993	0.9987	0.8342	0.9361
	5	0.9949	0.9992	0.9992	0.9893	0.8062	0.9250
	10	0.9909	0.9985	0.9987	0.9881	0.7933	0.9078
ReL, α=0.5	1	0.9980	0.9989	1.0	0.9832	0.8518	0.9011
	5	0.9945	0.9993	0.9997	0.9711	0.8728	0.9034
	10	0.9909	0.9993	0.9995	0.9695	0.8877	0.8843
ReL, α=0.7	1	0.9980	0.9988	0.9999	0.9730	0.8902	0.8392
	5	0.9962	0.9990	0.9998	0.9543	0.8726	0.7531
	10	0.9912	0.9987	0.9998	0.9623	0.8699	0.7111
ReL, α=0.9	1	0.9979	0.9990	1.0	0.9566	0.9102	0.7734
	5	0.9938	0.9987	1.0	0.9523	0.8830	0.6517
	10	0.9892	0.9984	1.0	0.9419	0.8627	0.6333