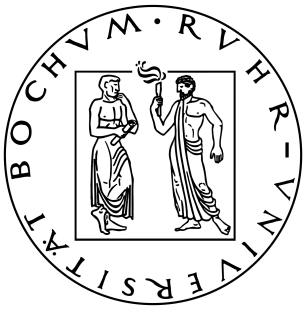


Semantic Representations in Variational Autoencoders as a Model of the Visual System

Schriftliche Prüfungsarbeit für die Master-Prüfung des Studiengangs Angewandte Informatik an der Ruhr-Universität Bochum

vorgelegt von

Leonard Papenmeier

108017257755

21.07.2020

Prof. Dr. Laurenz Wiskott

M.Sc. Zahra Fayyaz

Abstract

Unsupervised convolutional neural networks have been successfully applied to various tasks, but their biological plausibility is underexplored. Previous research has developed both biologically plausible networks and networks able to generate natural images, but the gap between these network types remains. This thesis aims at answering two questions: First, whether Variational Autoencoders (VAEs) - a specific type of convolutional neural networks - are a biologically plausible model of the visual cortex, and, second, whether the latent representations of these networks are related to semantic representations in the brain. Regarding the first question, it is studied whether they learn similar features as the primary visual cortex and whether they employ sparseness similarly to the brain. For answering the second question, it is investigated how the model maps categorical and continuous attributes in the latent space.

For this purpose, six different network types are developed and evaluated on different established datasets. The findings show that hierarchical and non-hierarchical VAEs do not learn the same low-level visual features as the human brain and that VAEs do not seem to employ sparseness. In conclusion, feed-forward VAE-models trained on static images are probably an inapt model of the visual cortex. Furthermore, it is argued that the latent space represents many attributes of datasets in a highly non-linear manner. This asks to revise common assumptions of the latent space structure.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, research in artificial neural networks has risen due to their success in a large variety of tasks. The increase in research has led to progressively better network architectures with higher performance. However, the improvement of network architectures is mainly driven by the question of whether a new architecture will lead to better results [45].

Biological plausibility, an essential consideration in computational neuroscience, is often of no interest to researchers aiming at solving complex problems. Initially, neural networks were inspired by biology. Modern neural networks operating on images, mainly convolutional neural networks (CNNs), indeed share many features with the visual system. However, recently, neural network research has become more disconnected from the biological example.

Even though neural networks are trained differently than how the brain learns and builds memories, it has been shown that image classification networks are related to the visual system. This relatedness shows not only in areas where it has been intentionally built into the model.

Both the biological foundation of CNNs and the recent insights on their indirect relatedness to the visual system qualify them as a potentially useful model of the visual system.

Unfortunately, the neural networks where the relatedness has been discovered are trained in a supervised manner, requiring lots of labeled data. This kind of learning is disconnected from human perception, where often one single example is sufficient when it comes to grouping objects into classes.

The Variational Autoencoder (VAE) is one important representative of *generative models*. This class of models allows a different training procedure. They still require many samples, yet no labels. Furthermore, VAEs build generalizable latent representations of inputs, which might be similar to the abstract representation found in the brain when perceiving the world.

VAEs, therefore, could be an important step towards a more realistic model of the visual system. A vital consideration in the context of computational neuroscience is the representation of visual input. By design, VAEs learn an input representation that generalizes to some degree. Furthermore, it has been shown that the latent space learns some semantic relationship. Nonetheless, the structure of the latent space is mainly a black box.

This thesis investigates whether VAEs could be a good model of the visual system, focussing on the representation of data in the latent space.

The remainder of this work is structured as follows.

Section 2 introduces the theoretical background and related work. Starting points for further studies and open questions are stated as well. Section 3 describes the methods and means of work. Section 4 describes and discusses studies based on the open questions raised in Section 2.

2 Theoretical Background

This thesis discusses whether Variational Autoencoders (VAEs) are a good model of the visual system. The following sections, therefore, first introduce the human visual system and, subsequently, work related to VAEs as a model of this.

2.1 Visual Cognition

The visual system allows humans and other developed animals to make sense of visual stimuli. The following sections describe how the visual systems' hierarchical structure enables the perception of gradually increasing complex visual features.

2.1.1 Cells

The brain, together with the spinal cord, constitutes the central nervous system (CNS) [49, p. 340]. The CNS comprises two types of cells: *neurons* and *glial cells* [49, p. 71].

Even though glial cells occur two to ten times more often in the nervous system, neurons are the basic units allowing signal exchange in the nervous system [49, p. 24]. In contrast, glial cells surround neurons and play a supportive role but are not directly involved in this signal exchange [49, p. 26].

Neurons take different forms in different brain regions. However, regardless of their specific configuration, they have four defined regions [49, p. 22]. The (1) cell body (*soma*) performs metabolism and contains genetic cell information. (2) *Dendrites* are attached to the cell body. By branching out, they receive signals from other neurons. Through their (3) *presynaptic terminals*, (4) *axons* transmit signals, the *action potentials*, to other cells [49, pp. 22, 23].

2.1.2 Human Brain Structure

The human brain is subdivided into six regions of different forms and functions: *medulla*, *pons*, *midbrain*, *cerebellum*, *diencephalon*, and the *cerebral hemispheres* [49, p. 340] (see Figure 1).

Medulla, pons, and midbrain constitute the *brain stem*. The brain stem receives input from some senses (but not vision) and plays a role in motor control [49, p. 341].

The *cerebellum* is mostly responsible for motor skills but is also involved in cognition [49, p. 341]. It contains more neurons than other brain divisions but only a few different types and is well explored [49, p. 341].

The *diencephalon* contains the *thalamus* and *hypothalamus*. The thalamus acts as a filter, deciding which information is forwarded to the *neocortex* [49, p. 341]. The hypothalamus plays a vital role in controlling body functions such as eating or drinking and initiating behaviors [49, p. 341].

The *cerebral hemispheres* [49, p. 341] is the brain region the most relevant for vision. It can be further subdivided into the *cerebral cortex*, *white matter*, *basal ganglia*, *amygdala*, and *hippocampus* [49, p. 341]. The latter three are "concerned with the expression of emotion (amygdala), [...] memory formation (hippocampus), and ... control of movement and aspects of motor learning (basal ganglia)" [49, p. 342]. Underlaid by the *white matter*, the neocortex is the brain region that enables cognition [49, pp. 341, 392]. It is the region of the cerebral cortex closest to the brain surface [49, p. 345].

The neocortex is structured into six layers and columns [49, p. 345]. Neurons within a column are assumed to form a *local processing network* and are understood as "the fundamental computational modules of the neocortex" [49, p. 348]. Depending on the layer, neurons show

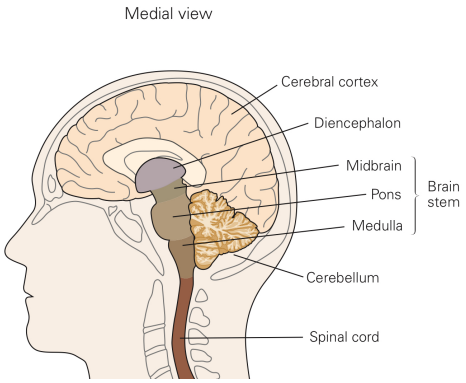


Figure 1: Five divisions of the human brain in the CNS, the cerebral cortex as part of the cerebral hemispheres, and the spinal cord. Taken from Mack et al. [49, p. 340].

different kinds of connectivity. For example, Layer I mainly contains dendrites of cells in deeper layers, whereas Layers II and III contain *pyramidal neurons* whose axons project onto other neurons [49, p. 346].

Furthermore, the neocortex is structured *topographically*, i.e., cells within a sensory area (e.g., the skin or retina) map onto the neocortex such that neighboring cells in the sensory area ultimately map onto neighboring regions in the neocortex [49, p. 343].

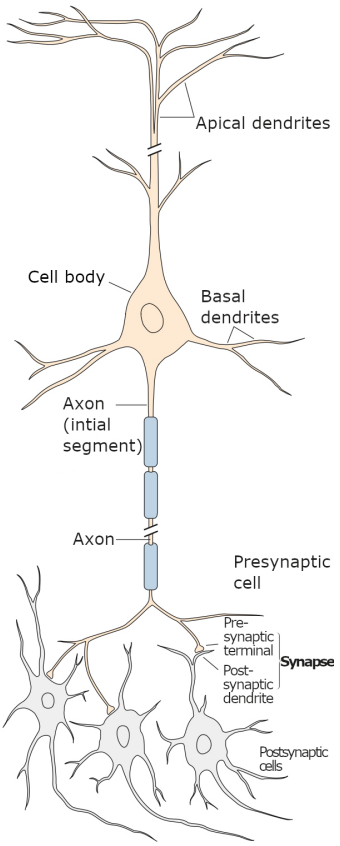


Figure 2: The morphological structure of a neuron, taken from Mack et al. [49, p. 22] (revised)

2.1.3 Visual Cortex

Through the retina, signals of visual stimuli travel through the *primary visual pathway* that includes the lateral geniculate nucleus (LGN) [49, p. 559] (see Figure 3). The primary visual pathway transports the signals to primary visual cortex (V1), the first region of the visual cortex [49, p. 559].

The LGN contains *on-center* and *off-center* cells that respond strongly to stimuli having either a bright center and a dark surrounding or a dark center and a bright surrounding in their receptive field [49, pp. 564-566]. In the context of vision, the receptive field of a cell is the area of the visual field, the neuron is responsive to [49, p. 564]. Although signals travel primarily from LGN to V1, the LGN receives strong feedback from V1. However, the function of these feedback connections, outnumbering the number of feedforward neurons from the LGN by factor ten, is mostly unknown [49, p. 573].

From V1, information is propagated to other brain regions via the *ventral* and *dorsal* pathways [49, pp. 563, 563]. The dorsal pathway is responsible for the pass of information regarding the direction of movements, whereas the ventral pathway is concerned with object recognition [49, p. 564]. Figure 3 shows the two pathways and the flow of information. From V4, the ventral pathway has feedback and feed-forward connections from and to the temporo-occipital area (TEO), and from the TEO it has feedback and feed-forward connections from and to inferior temporal cortex (IT), which, in turn, has feedback connections to

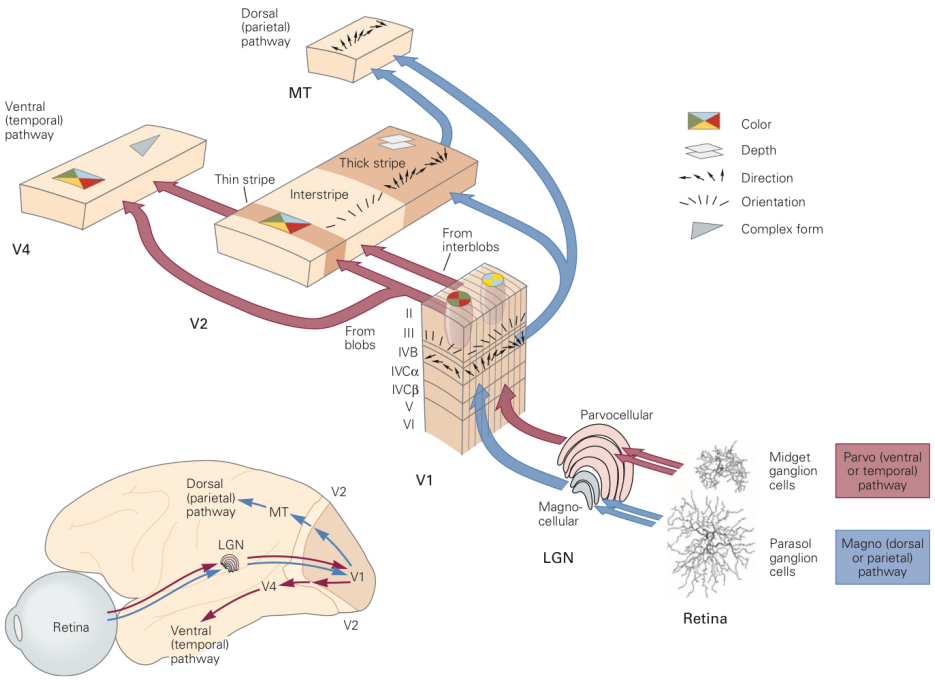


Figure 3: The ventral and dorsal pathways, carrying information from V1 to other brain regions. Taken from Mack et al. [49, p. 571]

V1 [49, p. 563].

The visual cortex is mainly structured in a feed-forward manner. Lower regions in the visual cortex detect lower level features. In contrast, higher regions detect higher-level features [19, 17]: The primary visual cortex (V1) detects edges [32, 19], while the secondary visual cortex (V2) does not respond to such basic shapes [22]. Instead, it responds to naturalistic textures that are combinations of features V1 is sensitive to [22]. The function of V4 is manifold - it "respond[s] to more complex geometric shapes, color, and a large number of other stimulus characteristics" [19] and it is assumed to perform foreground and background segmentation [58].

Furthermore, the quaternary visual cortex (V4) is assumed to play a role in *visual attention*. Attentional mechanisms are assumed to be enabled by top-down connections. They allow the suppression of irrelevant stimuli and focus on relevant ones in the current context [17, 58]. V4 receives input from top-down connections and is assumed to enable attentional mechanisms [58]. The IT, finally, responds to high-level features such as faces and complex objects [47, 19].

The two-stream hypothesis [25] states that the two pathways are concerned with the *where* (dorsal stream) and *what* (ventral stream) in a visual scene [49, p. 520]. However, Mack et al. [49, p. 564] note that the two pathways can exchange information. Nevertheless, they still encode two different qualities of a stimulus: the identity and the location.

2.1.4 Visual Object Perception

Recognizing an objects' identity is different from the ability to see an object or make a copy of it. Rubens and Benson [60] report the case of a 47-year-old man who showed an inability to recognize objects, and in cases where he was unable to recognize an object, he also could not describe its use. When given the category of an object, "identification improved very slightly": He recognized the item after being told the name. When shown sketches of items, he was generally unable to recognize the items. However, he was able to name geometric shapes such as circles or squares present in the sketch. Even though the man did not recognize the objects,

he could make copies of them (see Figure 4). Rubens and Benson [60] report that the patient "was unable to identify any [items] before copying." However, he was able to contain some of the categories of the objects after copying them.

This example shows that the ability to see and reproduce an object is different from the ability to *recognize* it.

For monkeys, the IT is assumed to be the brain region being crucial for object recognition [65, pp. 1070, 1071]. Bilateral lesions of the IT in monkeys affect their ability to "distinguish between different visual patterns or objects, and in retaining previously acquired visual discriminations" [65, p. 1070]. They are no longer able to generalize from tasks learned in one half of the visual space to the other half, presumably because the invariance of representations is lost [65, p. 1070]. Squire et al. [65, p. 1071] explicitly point out the importance of the IT during object recognition.

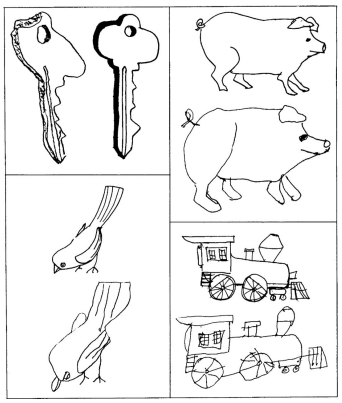


Figure 4: "Copies of line drawings." taken from Rubens and Benson [60]

2.1.5 Sparse Representations

When shown natural images, between 1.8% and 3.0% (with $p < 0.01$) of mice V1 neurons are active across planes [74]. The overlap of responsive neurons for different images is quite small. Between 4.8% and 6.0% of neurons overlap for different natural images. The brain uses sparseness to represent information ([74], [69, pp. 356, ff.]).

Sparse representations (or sparse codes) are a trade-off between *local* and *dense* codes [21]. Assume a region in the brain has N neurons that can either be *on* or *off*, depending on the input. For a local code, one and only one neuron is active for a particular input. This coding scheme allows to encode only N distinct inputs but representations of different inputs can be superposed. Dense codes, in contrast, activate about one half of the neurons for an input. One example is binary coding that can encode 2^N different inputs. However, dense codes do not allow to superpose representations of different inputs [21].

Sparse codes combine advantages of dense and local codes while avoiding their drawbacks [21]. Unsupervised models trained to represent natural images from a sparse set of basis functions have shown to learn Gabor-like basis function [53].

2.2 Types of Learning

In machine learning, one often distinguishes between the three fundamentally different types of learning that are roughly described in the following.

Supervised Learning Algorithms employing supervised learning learn to predict the label for samples representative for the training set. Datasets for this type of learning usually consist of sets of *samples* $\boldsymbol{x}_i$ and *labels* y_i, where the label describes the desired outcome of the algorithm on the sample. An example of supervised learning is *object detection*. Here, one is interested in predicting what is present in an image or a sequence of images, e.g., images showing a dog or a cat.

However, it is not always the case that a pre-defined and discrete set of samples and labels exist. In *active learning*, the learning algorithm has no beforehand-dataset but has to query labels for self-chosen data points in the sample space. In this setting, the learning algorithm chooses itself what samples to use for learning.

In the example of a classifier distinguishing cats and dogs, the dataset of samples and labels is available beforehand. In such settings, the learning algorithm learns the mapping function and does not update it once the learning is finished. Contrarily, in *online learning*, the mapping function is updated during the lifetime of a system, e.g., in recommender systems for streaming services. Users get recommendations while using the service but also provide new labels for the algorithm as they rate watched movies.

Self-supervised learning is a special case of supervised learning, where the learning algorithm creates the labels itself. An example of this type of learning is the prediction of the next frame in a video [72]. The dataset has no labels. However, given a sequence $\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, \ldots, \boldsymbol{x}_{i,n}$ of images in a video, the sample at time t can be considered as the supervised training set $((\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, \ldots, \boldsymbol{x}_{i,t}), \boldsymbol{x}_{i,t+1}), \quad 1 \leq t \leq n-1$. Self-supervised learning is also a subtype of unsupervised learning. However, algorithms employing this approach usually use classical supervised learning techniques. Another example of self-supervised learning is word embeddings [52].

Unsupervised Learning Like in self-supervised learning, unsupervised learning works with unlabeled data, i.e., the dataset only consists of samples $\boldsymbol{x}_i$. A typical example of unsupervised learning is clustering, an approach that can be used if the data points can be compared in terms of a distance measure. Consider Figure 5. A clustering algorithm assigns the points labels as indicated by the color.

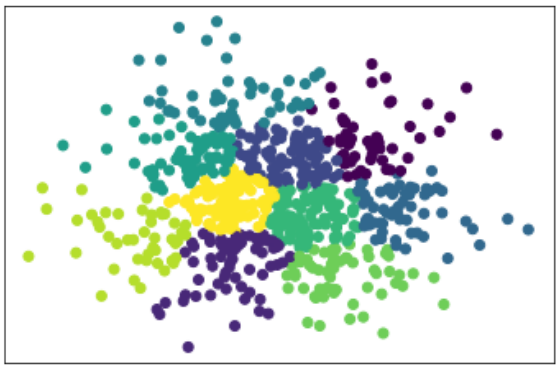


Figure 5: Data points colored by their cluster correspondence.

Cluster algorithms are often employed in cases where the class identity of data points is unknown, i.e., they aim to find the labeling function by identifying clusters of data points. Figure 5 shows an example where this can be feasible because adjacent data points tend to have the same class.

Reinforcement Learning Reinforcement learning is quite a different paradigm from supervised and unsupervised learning. Here, the aim is to train an *agent* to achieve a goal.

An example is rats in a maze. A piece of cheese is located in one corner of the maze. The different positions in the maze are the possible *states*; the set of all states is the *state space*. The rat has multiple possible actions, going forward, backward, left, or right; the set of all actions is the *action space*. In this setting, the rat is the agent, trying to maximize its reward. It is rewarded for each action in a state (the reward can also be zero or even negative). By rewarding good sequences of actions, the rat eventually shows a reasonable behavior.

Reinforcement learning algorithms resemble such settings by modeling a state space, an action space, rewards, and a reward function [66, pp. 1, ff.].

2.3 Semantic Representations

Section 2.1.4 described a subject who was able to copy drawings of objects but not capture their *meaning*. Squire et al. [65, pp. 1069, 1070] hypothesize that the subjects' LT was impaired, leading to an "inability to generate a high-level representation of the object."

The term *Semantic Representation* refers to these high-level representations that allow to capturing the *meaning* of a stimulus.

Albeit, a universally accepted definition for this term yet has to be found. *Semantics* is defined as [44]

> The branch of linguistics and logic concerned with meaning. There are a number of branches and subbranches of semantics, including formal semantics, which studies the logical aspects of meaning, such as sense, reference, implication, and logical form, lexical semantics, which studies word meanings and word relations, and conceptual semantics, which studies the cognitive structure of meaning.

This thesis is less concerned with linguistics but with *meanings* and *relations*, not of language but *images*. A general definition of *Semantic Representations*, however, should not be restricted to words or images. It should incorporate all kinds of concepts a human can form of perception.

In this thesis, *Semantic Representations* are somewhat related to *Word Embeddings* in natural language processing (NLP). Word embeddings [52] are a vectorized representation of words in a vector space. The training procedure learns a mapping from word into the latent space so that the words' position itself has meaning. One prominent example is that the vector space allows "simple algebraic operations[...]. [I]t was shown for example that *vector*('King') - *vector*('Man') + *vector*('Woman') results in a vector that is closest to the vector representation of the word *Queen* [...]" [52]. It has been shown that for word representations, they correlate with brain activity fMRI data [59, 1].

Just like the brain maps similar visual stimuli to neurons close to one another, a semantic representation should encode concepts such that encodings of similar concepts are close to one another according to some distance metric. Unfortunately, a dense vector (as in word embeddings) as a semantic representation is somewhat unrelated to active neurons subpopulation.

Another consideration is the *mode* of perception. Section 2.1.4 discusses that the IT seems to play a crucial role in building and accessing semantic representations for visual input. Different brain areas may be activated by different modes of presentation (e.g., *visual* vs. *auditory*). However, only the brain regions that are active regardless of the mode of presentation are candidate regions for amodal representations [20]. Fairhall and Caramazza [20] attempt to find these brain regions by analyzing which brain regions are active for textual and visual stimuli (both presented visually).

All in all, the question of how to obtain and assess good semantic representations remains. One minimum requirement, albeit, is that those (amodal) semantic representations should behave like biological examples.

This thesis focuses on visual-modal semantic representations.

For visual input, it has been shown that higher-layer activations of supervised convolutional neural networks (CNNs), described in the following section, explain IT cortical representation [36, 10]. For unsupervised models operating on image data, in contrast, there is some evidence that these models do not explain IT cortical representation (or by far worse than supervised models) [36] - even though not explicitly for CNNs.

Therefore, CNNs seem to be a promising candidate for visual-modal semantic representations.

2.4 Models of the Visual System

The following section discusses established models of the visual system.

2.4.1 Simple and Complex Cells in the Primary Visual Cortex

Hubel and Wiesel [32] distinguish two main types of cells in the V1: simple and complex cells. Both cell types are orientation sensitive, i.e., different cells are tuned towards different

orientations of stripes in a stimulus. Unlike simple cells, complex cells are more invariant towards the stripes' location (but not its orientation) in the receptive field. Complex cells are assumed to receive input from simple cells [32]. Based on this assumption, complex cells can be modeled by receiving input from many simple cells, tuned towards a specific orientation but not location. If any of the afferent simple cells fire, the complex cells fires, leading to a translation-invariant behavior [32].

It can be shown that a two-dimensional Gabor wavelet, a (co-)sine function, modulated by a Gaussian function, is an excellent stimulus for a simple cell in V1 in terms of the neuronal action potential [34].

2.4.2 Neocognitron

The Neocognitron [23] is a model of object perception based on the findings of Hubel and Wiesel [32]. It is a neural network consisting of models of simple ("S-cells") and complex ("C-cells") cells, alternatingly arranged in multiple layers. It can be trained in an unsupervised manner by reinforcing connections leading to high cell activations in the next layer and has shown to be effective in pattern recognition [23].

2.4.3 Convolutional Neural Networks

CNNs [42] can be understood as a successor of the Neocognitron [45] and is a network type commonly used on image data [26, p. 326]. The convolutional layers of a CNN act as a pattern detector, similar to the S-cells in the Neocognitron [45]. After an activation layer, CNNs usually perform an operation called (max-)pooling [26, pp. 326, 339]. The pooling operation introduces invariance to the network, similar to C-cells [45].

Convolution Assume I is a two-dimensional grayscale image, and K is a convolutional kernel. The convolution operation then is [26, p. 327]:

$$S(i,j) = (I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n) \tag{1}$$

The result of the convolution is S, a *feature map*.

The kernel is usually implemented as a two-dimensional array [26, p. 327]. Values outside of this array, as they are assumed in the summation in Equation 1, are assumed to be zero. These values then do not contribute to the sum, and the convolution effectively only has to use a finite number of summations. In case of a 3×3 kernel, Equation 1 simplifies to:

$$S(i,j) = (I * K)(i,j) = \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} I(m,n)K(i-m, j-n) \tag{2}$$

Equation 1 holds if indexing starts at zero for the image and at minus one for the kernel.

The output of a convolution is maximum in image areas where the image contains a flipped version of the kernel. Unlike the cross-correlation[1], the convolution is commutative. However, this property is not relevant in practice, and "many neural network libraries implement [...] the cross-correlation [instead]" [26, p. 329].

[1] The cross-correlation is defined as $S(i,j) = (I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(i+m, j+n)K(m,n)$ [26, p. 329].

Padding The convolution operation applies the kernel at all image locations. At border pixels, however, this operation is problematic because the sum in Equation 2 also considers image pixels with an index smaller than i or j. There are multiple options to deal with this problem.

One option is not to pad the image at all and to start and end the convolution at indices such that the sum in Equation 2 is never applied to invalid image indices [26, p. 350]. However, this leads to feature maps smaller than the input image because the border pixels are not considered. This can be problematic for deep networks as this happens on all convolutional layers. Also, the network does not consider border pixels equally often compared to non-border pixels.

Another option is *zero-padding* [26, p. 350], i.e., assuming values outside the image to be zero. Here, the feature map is of the same size as the input. However, the network still handles border pixels differently from inner pixels. The feature maps values for border pixels tend to be smaller due to the multiplication with zero in Equation 2.

Other padding techniques mirror the image at the borders or assume that all pixel values outside the image are the same as at the borders [2].

Max-Pooling Like the convolution, pooling operations use a sliding window over the feature maps, often with an offset (*stride*) such that the sliding window sees every pixel only once. A stridden max-pooling has multiple effects. First, it leads to the downsampling of the image. Downsampling is a useful property as it allows us to use more convolutional kernels in higher layers without exceeding a systems' memory[3]. Not less importantly, max-pooling makes the network more invariant to small translations [26, p. 342]. In cases where translation invariance is not desired[4], but the image should be down-sampled, the max-pooling layer can be omitted, and a stridden convolution can be used instead [26, p. 337].

Stridden Convolutions The *stride* of a convolution operation refers to the number of pixels being skipped as the sliding window moves over the feature map [26, p. 348]. For example, a stride of three with a 3×3 kernel leads to each pixel of the convolution input being evaluated only *once*. A *stride* in the convolution usually replaces the max-pooling operation. Just like max-pooling, it leads to a down-sampling of the image because it is evaluated fewer times. Replacing max-pooling by strides seems to have no significant effect on network classification performance [64], but it requires fewer evaluations than a full convolution with a subsequent max-pooling step.

Transposed Convolutions Suppose we have a 4×4 input image which we are convolving with a 3×3 kernel without padding and no stride, i.e., a stride of one. The output of this is a 2×2 feature map. In this example, the *transposed convolution* is the operation that transforms a 2×2 *input* into a 4×4 *output* [18]. This can be achieved by padding the 2×2 input with two pixels on the borders. Convolving this padded input with a 3×3 kernel results in a 4×4 output. Therefore, a transposed convolution performs an upsampling of the image. Usually, it is used to inverse the convolution operation. Even though it could be implemented as described above, it would be rather slow. Convolutions are mostly implemented as matrix multiplications [24]. The transposed convolution is then implemented multiplying with an (eponymous) transposed matrix [18].

Convolutional Neural Networks as Models of the Visual System By design, CNNs share some features with the visual system, namely the convolution as a model of S-cells and

[2] https://www.tensorflow.org/api_docs/python/tf/pad, last access: 2020/06/18

[3] Of course, the convolutional kernels itself are almost certainly unproblematic as they are very small. Each convolution, however, produces another memory-consuming feature map.

[4] This is often the case for generative models.

max-pooling as a C-cell model. However, CNNs show further unintentional parallels to the visual system:

AlexNet [40] is a deep CNN trained on image classification. Krizhevsky, Sutskever, and Hinton [40] have shown that this network learns Gabor wavelets in the first layers' kernels. A Gabor wavelet is the optimal stimulus for a Gabor-like kernel (see Section 2.4.3), resulting in the strongest excitement. It is also the optimal stimulus for cells in V1 (see Section 2.4.1, and Jones and Palmer [34]).

Furthermore, Khaligh-Razavi and Kriegeskorte [36] have shown that CNNs trained in a supervised manner show similar representational dissimilarity matrices (RDMs) like the human IT. An RDM is a matrix encoding the correlation of activity patterns for a given model. Rows and columns refer to different input stimuli. A cell in an RDM then contains the correlation of the activity patterns for the given input stimuli and the given model.

Eickenberg et al. [19] explicitly discuss CNNs as models of the visual system. Their findings show that CNNs reproduce the hierarchy of semantic representations in the visual cortex [19, 73]. The similarity of the RDMs is evidence that supervised CNNs and the human IT have a similar high-level encoding of information.

Khaligh-Razavi and Kriegeskorte [36] also show that this is not true for a variety of unsupervised models, yet, not for VAEs, the models mainly used in the course of this thesis. VAEs are introduced in Section 2.5.2. The question of whether VAEs show similar RDMs as the human IT, therefore, remains.

It has been shown that VAEs, trained in a self-supervised manner, partially learn Gabor wavelets when trained to predict the next image in a sequence of images [54]. Also, some unsupervised models learn Gabor wavelets [53, 4] but whether VAEs learn Gabor wavelets in lower layers remains unanswered.

2.5 Generative Methods

Section 2.2 used the example of a "cat-and-dog-classifier" to introduce supervised learning. Such a classifier is also an example of a *discriminative model* because it "model[s] the posterior probabilities directly" [5, p. 43]. Therefore, a discriminative model, for example, does not allow to predict the probability density of a data point in the feature space [5, pp. 43, 44]. *Generative methods* model both the input and the output probabilities. Access to the input probability allows sampling from the input space to generate new data points [5, p. 43]. The following sections give an overview of existing generative methods.

2.5.1 Generative Adversarial Networks

The idea behind generative adversarial networks (GANs) [27] is to use two networks, one *generator network* G to approximate the data distribution, and another *discriminator network* $D \mapsto [0, 1]$ to discriminate between true data points and data points generated by the generator network. This interplay can be formalized as [27]:

$$V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{3}$$

where $\boldsymbol{z}$ is random noise[5].

GANs are commonly used to generate new data points. Therefore, one is usually interested in the generator network. It can be found by [27]:

$$G^* = \arg\min_G \max_D V(D, G). \tag{4}$$

[5] Usually Gaussian white noise.

Figure 6: GAN-generated samples, taken from Brock, Donahue, and Simonyan [6].

The discriminator D in Equation 4 is trained towards maximizing $V(D, G)$. A perfect discriminator achieves this by labeling all true samples $\boldsymbol{x} \sim p_{\text{data}}$ with 1 and all generated samples $G(\boldsymbol{z}), \boldsymbol{z} \sim p_{\boldsymbol{z}}$ with 0. The correct labeling of all samples results in a value $V(D, G) = 0$, which is the maximum.

The generator G is trained towards minimizing $V(D, G)$. As the discriminator output is the only variable term in $V(D, G)$, G can only minimize $V(D, G)$ by misleading the discriminator to make a more significant error. If the generator reproduces the data distribution p_{data}, the discriminator would no longer distinguish the generated from the real samples as they come from the same distribution. Reproducing the data distribution is what is desired in generative methods.

Recent improvements on GANs have led to syntheses of highly natural images, as shown in Figure 6.

The main disadvantage of GANs is training stability. When training GANs, one commonly encounters *mode collapse* or *loss oscillation*. For mode collapse, the generator produces only a few or even just one sample [12]. Mode collapse indicates a failure to match the posterior distribution, which is the GAN training objective. Loss oscillation occurs when the generator and the discriminator alternatingly become stronger and weaker [28], leading to a non-convergence of the model.

2.5.2 (Variational) Autoencoders

Autoencoders are models transforming input into a lower dimensional representation [26, p. 146]. Unlike the regular Autoencoder, VAEs furthermore require the lower dimensional representation to follow a defined distribution. Since Variational Autoencoders (VAEs) are a specialization of the autoencoder, the traditional autoencoder is introduced first.

Autoencoders Autoencoders are neural networks trained to reconstruct their input [26, p. 499]. They consist of an *encoder* and a *decoder*. The encoder $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ transforms an input $\boldsymbol{x}$ to a hidden representation $\boldsymbol{r} = f(\boldsymbol{x})$. Usually, the encoder transforms the input into a lower-dimensional representation ($m < n$). This mapping can be used for dimensionality reduction or feature learning [26, p. 499]. The decoder $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ transforms the hidden representation back into the original feature space. Usually, one wants the reconstruction $\tilde{\boldsymbol{x}}$ to be close to the original feature $\boldsymbol{x}$ ($\tilde{\boldsymbol{x}} \approx \boldsymbol{x}$). In order to achieve this, the autoencoder is usually trained by minimizing the loss $\mathcal{L}(\boldsymbol{x}, g(f(\boldsymbol{x})))$ with

$$\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}. \tag{5}$$

One common choice for $\mathcal{L}$ is the mean squared error (MSE) which is defined as [26, p. 106]:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum (\boldsymbol{x}_i - \boldsymbol{y}_i)^2. \tag{6}$$

Note that for linear activations in the autoencoder, the subspace spanned by the first m principal components is the optimal solution for equation 6 [15].

Variational Autoencoders Autoencoders transform an input samples $\boldsymbol{x}$ to a *hidden representation* $f(\boldsymbol{x})$. However, the distribution over $f(\boldsymbol{x})$ is generally unknown. Let $\boldsymbol{z} = f(\boldsymbol{x})$. Assume $\tilde{\boldsymbol{z}} = \boldsymbol{z} + \boldsymbol{\epsilon}$ is a slightly perturbed version of $\boldsymbol{z}$, created by adding a small amount of noise $\boldsymbol{\epsilon}$. Even though $\tilde{\boldsymbol{z}} \approx \boldsymbol{z}$, in terms of equation 6, the result after decoding can be very different, i.e., $g(\tilde{\boldsymbol{z}}) \not\approx g(\boldsymbol{z})$. This reconstruction-mismatch can occur because the distribution $p(\boldsymbol{z})$ can take any arbitrary form. Values of $\boldsymbol{z}$ that are close to another[6] can be likely for very different values of $\boldsymbol{x}$.

To *generate* new images, it would be advantageous to enforce $p(\boldsymbol{z})$ to follow a particular distribution. A common choice is an independent, multivariate normal distribution [38, pp. 24, 25]. VAEs [39] enforce a specific (usually Gaussian) distribution over $\boldsymbol{z}$.

VAEs are trained to generate samples that are likely to occur in the training set. Therefore, they aim to maximize $\log p(\boldsymbol{x})$[7] [38, p. 18]. Assuming a latent *prior* distribution $p(\boldsymbol{z})$ over $\boldsymbol{z}$, $p(\boldsymbol{x})$ can be written as the marginal distribution

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \tag{7}$$

$$= \int p(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z}) d\boldsymbol{z}. \tag{8}$$

Unfortunately, due to the integral w.r.t. $\boldsymbol{z}$ in equations 7 and 8, computing $p(\boldsymbol{x})$ is intractable [38, p. 13]. However, if the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ was given, $p(\boldsymbol{x})$ could be obtained by

$$p(\boldsymbol{x}) = \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}, \tag{9}$$

since computing $p(\boldsymbol{x}, \boldsymbol{z})$ is tractable [38, p. 14].

VAEs approximate the *posterior* by an *inference model* $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$, also called the *encoder* [38, p. 15].

To minimize the difference between q and p, usually, the Kullback-Leibler divergence (KL-divergence) plus the reconstruction error is minimized. The reason for this is as follows. The starting point is that we cannot compute the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Therefore, we approximate it by another distribution q_ϕ. The KL-divergence is used o minimize the difference between p_θ and p_ϕ: $D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p_\theta(\boldsymbol{z}|\boldsymbol{x}))$.

[6] For example in terms of Euclidean distance

[7] Note that $\max \log p(\boldsymbol{x}) \equiv \max p(\boldsymbol{x})$.

$$
\begin{aligned}
D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p_\theta(\boldsymbol{z}|\boldsymbol{x})) &= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} && (10)\\
&= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{p_\theta(\boldsymbol{x})}}{\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{1}} && (11)\\
&= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \frac{1}{p_\theta(\boldsymbol{x})} && (12)\\
&= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \left[\log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} - \log p_\theta(\boldsymbol{x})\right] && (13)\\
&= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} + \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}) && (14)\\
&= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} + \log p_\theta(\boldsymbol{x}) \underbrace{\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x})}_{=1} && (15)\\
&= -\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} + \log p_\theta(\boldsymbol{x}) && (16)
\end{aligned}
$$

Then

$$
\underbrace{\log p_\theta(\boldsymbol{x})}_{\text{constant}} = \underbrace{D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p_\theta(\boldsymbol{z}|\boldsymbol{x}))}_{\downarrow} + \underbrace{\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}}_{\uparrow}. \tag{17}
$$

Because $\log p_\theta(\boldsymbol{x})$ is assumed to be fixed, instead of minimizing $D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p_\theta(\boldsymbol{z}|\boldsymbol{x}))$, we can maximize $\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}$, as both terms always sum up to a constant. $\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}$ is called *variational lower bound* or *evidence lower bound* (ELBO) [38, p. 18].

The variational lower bound can be written as

$$
\begin{aligned}
\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} &= \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x}) p_\theta(\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} && (18)\\
&= \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \left[\log p_\theta(\boldsymbol{x}) + \log \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] && (19)\\
&= \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}) + \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} && (20)\\
&= \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x})]}_{=\log p_\theta(\boldsymbol{x})\text{, independent of } \boldsymbol{z}} + \underbrace{\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}}_{=D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p_\theta(\boldsymbol{z}|\boldsymbol{x}))}, && (21)
\end{aligned}
$$

but also as

$$
\begin{aligned}
\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} &= \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} && (22)\\
&= \sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] && (23)\\
&= \underbrace{\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}|\boldsymbol{z})}_{=\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})]} + \underbrace{\sum_{\boldsymbol{z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}}_{=D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p_\theta(\boldsymbol{z}))}. && (24)
\end{aligned}
$$

Equations 21 and 24 allow two different interpretations of the ELBO. Equation 21 justifies the name *evidence lower bound*. If the KL-divergence nears zero, it approaches the likelihood of the data [38, p. 18]. However, $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ cannot be computed with a VAE.

Equation 24, however, is more useful for implementing the error function of a VAE. Consider the term inside the expectation of the first term. Usually, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is chosen to be a normal distribution with probability density function (PDF):

$$f_{\sigma,\mu}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{25}$$

Then

$$\log f_{\sigma,\mu}(x) = \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}\right) \tag{26}$$

$$= -\log\sqrt{2\pi}\sigma - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2. \tag{27}$$

Since μ, in this case, is a function of $\boldsymbol{z}$ (parametrized by θ), we can consider this as the reconstruction $\hat{\boldsymbol{x}} = f_\theta(\boldsymbol{z})$. Then, except for subtracting a constant and scaling by a constant, equation 27 is the MSE. The sum on the left of Equation 24 is subsequently called *reconstruction term*, the sum on the *Kullback-Leibler (KL)-term*.

Since $p(\boldsymbol{x}|\boldsymbol{z})$ is a Gaussian, maximizing $\mathbb{E}_{q(z)}[p(x|z)]$ is equivalent to minimizing the mean squared error between x and the *reconstruction* $\hat{x}$. $\hat{x}$ can be written instead of z because $p(z) = \hat{x}$ and minimizing does not change the relation.

When implementing a VAE, q_ϕ and p_θ are realized by neural networks. The encoder predicts the mean and variances for an independent, multivariate normal distribution, whereas the decoder reconstructs the input. The usage of neural networks poses a challenge for computing the first term in equation 24. Usually, one would draw a sufficient number of samples from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to approximate the expectation. However, sampling is impossible if one wants to compute the gradient to train the model by backpropagation.

Backpropagation for the encoder would be intractable if $\boldsymbol{z}$ were realized as a random variable, i.e., $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$. This is because the gradients w.r.t. ϕ cannot merely be calculated for one $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$ as this would be biased towards this $\boldsymbol{z}$ [38]. Instead, one would have to sample many $\boldsymbol{z}$s to approximate the expectation w.r.t. $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. This is not true for the decoder as its gradients w.r.t. θ are independent of this expectation. To solve this problem, implementations of VAEs do not sample from the posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Instead, they use the *resampling trick* [38]. Under this regime, $\boldsymbol{z}$ is the value of a function $g(\phi, \boldsymbol{x}, \epsilon)$ where $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is a random variable [38]. In an intermediate step, the encoder predicts values $\boldsymbol{\mu}$ and $\log\boldsymbol{\sigma}$ and $\boldsymbol{z}$, eventually, is realized as $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \epsilon$ [38]. Using this trick, the encoder gradients are w.r.t. the expectation of ϵ. Backpropagation on a minibatch of data, therefore, is unbiased.

Unlike regular autoencoders, VAEs produce an interpretable latent space. Enforcing randomness in the positioning of points in the latent space guides the model to predict an area in the latent space that is likely for a given input $\boldsymbol{x}$ [26, p. 701]. Eventually, this allows to sample from the random space to generate new images, employing the decoder. Furthermore, it allows simple arithmetic in the latent space (see Section 2.5.3).

Even though the latent space is forced to follow a prior distribution, it remains unclear how (lower-level) attributes of an image are encoded in this latent space.

Disadvantages One problem arising when training VAEs is *posterior collapse*, meaning that one or more of the latent dimensions resemble the prior ([26, p. 694], [48]), i.e. [48]:

$$\exists i : \forall \mathbf{x} q_\phi \left(z_i | \mathbf{x} \right) \approx p \left(z_i \right) . \tag{28}$$

In the most extreme case, this happens for *all* latent dimensions. In that case, the generator cannot make use of $\boldsymbol{x}$ at all, because generated images are unrelated to the input image. The KL-term is probably the reason for the posterior collapse as it is the term forcing the posterior to match the prior distribution [48]. One technique to avoid posterior collapse is to lower the weight of the KL-term compared to the reconstruction term [48].

Another disadvantage of VAEs are "blurry" and "fuzzy" reconstructions [76]. According to Zhao, Song, and Ermon [76], these reconstructions occur for $\boldsymbol{z}$s' that are encodings of a variety of different $\boldsymbol{x}$s'. They propose that blurry reconstructions are caused by the pixel-wise loss and a too simple prior, e.g., a standard multivariate normal distribution [76].

2.5.3 Representation Learning

Representation learning originally addresses the learning of representations of input data that makes models operating on this data more efficient [3]. In the context of NLP, for example, sentences are often represented by *Bag-of-Words* vectors. In a Bag-of-Words vector, each cell stands for one word in the vocabulary. The representation for one specific sentence is obtained by assigning each cell the number of occurrences of its word in the sentence.

However, representation learning can also be considered in semantic representations (see Section 2.3). As discussed earlier (see Section 2.1.3), different regions of the visual cortex learn representations of the input in different regions. Earlier regions learn representations of simple features, whereas later regions respond to sophisticated features of the input. RTherefore, representations of inputs are learned in a *hierarchy* of simple to complex shapes [57].

The following paragraphs introduce different hierarchical representation learning approaches.

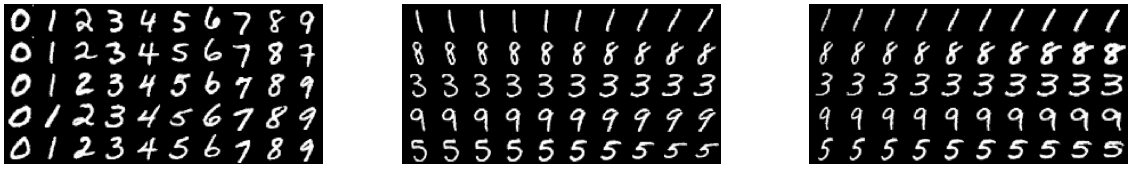
InfoGAN InfoGANs [14] address feature learning in GANs by "decomposing the input noise vector into two parts" [14]. Instead of only the noise z, the generator G receives an additional input of "structure latent variables" $\boldsymbol{c} = (\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_L})$. The additional variables are assumed to be independent of each other. To force G not just to ignorce c, Chen et al. [14] modify the training objective such that "there should be high mutual information between latent codes c and the generator distribution $G(z, c)$" [14]. The new loss function takes the form

$$\min_G \max_D V_I(D, G) = V(D, G) - \lambda I(\boldsymbol{c}; G(\boldsymbol{z}, \boldsymbol{c})), \tag{29}$$

where $V(D, G)$ is the training objective defined in Equation 4, λ is a trade-off hyperparameter, and $I(\boldsymbol{x}; \boldsymbol{y})$ is the mutual information between $\boldsymbol{x}$ and $\boldsymbol{y}$.

"To disentagle digit shape styles on MNIST," Chen et al. [14] choose $\boldsymbol{c}$ as three-element set $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3)$ with $\boldsymbol{c}_1$ drawn from a categorical distribution with ten categories (equal to the number of classes in MNIST) and event probability $p_i = 0.1 \, \forall i \in [0, 10]$. $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$ are drawn from a uniform distribution over $[-1, 1]$.

Figure 7 shows the latent space separability by varying the $\boldsymbol{c}_i$-values. In conclusion, InfoGAN seems to generalize as $\boldsymbol{c}_2$, and $\boldsymbol{c}_3$ vary between $[-2, 2]$, even though these values were in $[-1, 1]$ at training time. Generalization is a critical property for a model of the visual context. Also, InfoGAN addresses the problem of hierarchical learning. As discussed in Section 2.1.3, the different regions in the visual cortex learn increasingly complex representations. Therefore, InfoGAN, as a model employing hierarchical representations, addresses one more important property of models of the visual system.



(a) Varying $\boldsymbol{c}_1 \in [0, 10]$ (b) Varying $\boldsymbol{c}_2 \in [-2, 2]$ (c) Varying $\boldsymbol{c}_3 \in [-2, 2]$

Figure 7: Exploration of $\boldsymbol{c}$ for InfoGAN on MNIST by traversing either, $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, or $\boldsymbol{c}_3$. Rows correspond to different noise values $\boldsymbol{z}$ but are equal for the three figures, columns correspond to different values for $\boldsymbol{c}_i$. $\boldsymbol{c}_1$ takes discrete values, $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$ take continuous values. Taken from Chen et al. [14].

StyleGAN StyleGAN [35] is a type of GAN that explicitly addresses *latent space separability* (see Section 2.7). The term *style* refers to latent space separability, i.e., the model is can learn a disentanglement of images into different styles. Another term for style is factor of variation. For human faces, for example, one style or factor of variation is the hair color.

The styles of an image are learned at different levels in a self-supervised manner. Therefore, it is not possible to explicitly force the model to learn pre-defined aspects of an image. A posterior analysis allows us to identify which aspects were learned on which level.

Karras, Laine, and Aila [35] train their model on a dataset of human faces. In this context, coarse styles correspond to gender, age, or glasses. Middle styles correspond to skin color, face form, or mouth open/closed. Fine styles correspond mainly to hair color and lightning.

GANs generate new images using random noise $\boldsymbol{z} \in \mathcal{Z}$ as input (see Section 2.5.1). Besides, StyleGAN uses a mapping network $f : \mathcal{Z} \mapsto \mathcal{W}$ to map the random noise to a vector $\boldsymbol{w} \in \mathcal{W}$. With an affine transformation, the vector w is then mapped to a vector $\boldsymbol{y}_i = (\boldsymbol{y}_{s,i}, \boldsymbol{y}_{b,i})$. Different vectors $\boldsymbol{y}_i$ are fed into different layers i of the generator network to control the style on this layer.

The generator network g has a constant input $\boldsymbol{x}_0$ that is lower-dimensional than the final output image. $\boldsymbol{x}_0$ is of size $4 \times 4 \times 512$ [35]. On each layer, a noise vector $\boldsymbol{\epsilon}_i$ is added, followed by an *adaptive instance normalization (AdaIN)* operation. The AdaIN operation of SyleGAN is defined as [35]:

$$\text{AdaIN}(\boldsymbol{x}_i, \boldsymbol{y}) = \boldsymbol{y}_{s,i} \frac{\boldsymbol{x}_i - \mu(\boldsymbol{x}_i)}{\sigma(\boldsymbol{x}_i)} + \boldsymbol{y}_{b,i}, \tag{30}$$

where $\mu(\cdot)$ gives the mean and $\sigma(\cdot)$ the (empirical) standard deviation. Importantly, this operation is applied separately for each feature map of $\boldsymbol{x}_i$.

On each resolution, g first applies noise addition, AdaIN, a 3×3 convolution, noise addition, and one last AdaIN operation. Then, the image is upsampled to double size. This process is repeated until the image has the desired output size. Karras, Laine, and Aila [35] use nine of these blocks, resulting in an output size of 1024×1024.

In StyleGAN, latent space separability is achieved by a technique called *mixing regularization*. Here, for a subset of the training images, two input vectors $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are drawn. During the training, a point (or layer) of g up to which layer $\boldsymbol{z}_1$ controls the style is chosen at random. After this point, the style information from $\boldsymbol{z}_2$ is fed into the network. Randomly choosing this point "prevents the network from assuming that adjacent styles are correlated" [35].

The training procedure allows for using two sources for the generation. During inference time, the crossover point can be chosen arbitrarily, i.e., it can be controlled what styles are taken from the first and what styles are taken from the second source.

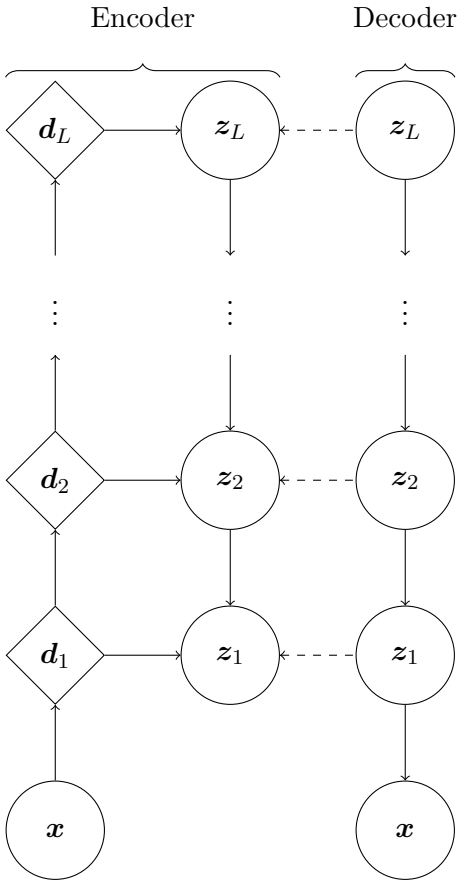


Figure 8: Ladder Variational Autoencoder structure, adapted from Sønderby et al. [68]. Solid arrows indicate feed-forward connections, dashed arrows indicate weight sharing. Diamonds indicate stochastic variables, circles deterministic variables. $\boldsymbol{z}$s are latent spaces, $\boldsymbol{x}$s are model inputs/outputs.

$\boldsymbol{y}_i$s fed into the network at a low resolution (i.e., i is small) control the coarse styles, whereas $\boldsymbol{y}_i$ at higher layers control finer styles.

Finally, Karras, Laine, and Aila [35] analyze how disentangled generations from $\mathcal{W}$-space are compared to $\mathcal{Z}$-space employing *perceptual path length* (see Section 2.6). They show that the perceptual path length is lower for interpolating in $\mathcal{W}$ than for interpolation in $\mathcal{Z}$ [35]. Therefore, they conclude that $\mathcal{Z}$ is more entangled than $\mathcal{W}$ in their setup.

Disentanglement in the latent space has important ramifications in the context of semantic representations (see Section 2.3). It has to be considered if the interpolation between two points in the latent space requires a curved instead of linear path.

Ladder Variational Autoencoder Ladder Variational Autoencoders [68] address the problem of hierarchical learning in VAEs. Classical VAEs, in contrast, learn one latent representation in one layer.

Hierarchical VAEs, also, only use the first few layers to learn meaningful semantics of an input. In the case of Sønderby et al. [68], the first two layers are sufficient. Zhao, Song, and Ermon [75] state that, given a sufficiently large encoder network in the first layer, the first layer learns all the semantics. While this is not always true, they "demonstrate that this phenomen occurs in practice [...]".

Ladder Variational Autoencoder (LVAE) is a model designed to use all embedding layers to

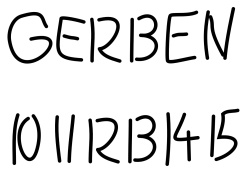


Figure 9: Illustration of context-dependent semantic ambiguity. Taken from Broeke [7, p. 61]

learn a representation instead of just the first few layers. They do this by passing information top-down in the inference network. Figure 9 gives an intuition of why passing information top-down can be advantageous. Here, the two rows show two different alphanumeric strings.

Apparently, the first row shows a sequence of alphabetic characters ("GERBEN"), whereas the second row shows a sequence of numeric characters ("10 11 12 13 14 15"). However, "R" and "B" in the first row are the same symbols as "12" and "13" in the second one. A simple feed-forward system would be unable to discriminate between the low-level features *R* and *13*. Incorporating high-level features like *alphabetic string* helps to improve the posterior low-level feature representation. Furthermore, the top-down pass is more biologically plausible than a pure feed-forward network and enables the model to focus on relevant parts of the input (see Section 2.1.3).

Unlike hierarchical VAEs, LVAEs do not use lower stochastic embedding layers as input for higher layers. Instead, they use a deterministic multi-layer network in the encoder (see Figure 8) and pass information from intermediate layers to the stochastic embedding layers.

$$\boldsymbol{d}_i = \mathrm{NN}(\boldsymbol{d}_{i-1}) \tag{31}$$

$$\hat{\boldsymbol{\mu}}_{q,i} = \mathrm{NN}(\boldsymbol{d}_i) \tag{32}$$

$$\hat{\boldsymbol{\sigma}}^2_{q,i} = \mathrm{NN}(\boldsymbol{d}_i) \tag{33}$$

The output from higher stochastic embedding layers is passed to lower layers via the decoder p:

$$\boldsymbol{\sigma}_{q,i} = \frac{1}{\hat{\boldsymbol{\sigma}}^{-2}_{q,i} + \hat{\boldsymbol{\sigma}}^{-2}_{p,i}} \tag{34}$$

$$\boldsymbol{\mu}_{q,i} = \frac{\hat{\boldsymbol{\mu}}_{q,i}\hat{\boldsymbol{\sigma}}^{-2}_{q,i} + \hat{\boldsymbol{\mu}}_{p,i}\hat{\boldsymbol{\sigma}}^{-2}_{p,i}}{\hat{\boldsymbol{\sigma}}^{-2}_{q,i} + \hat{\boldsymbol{\sigma}}^{-2}_{p,i}}. \tag{35}$$

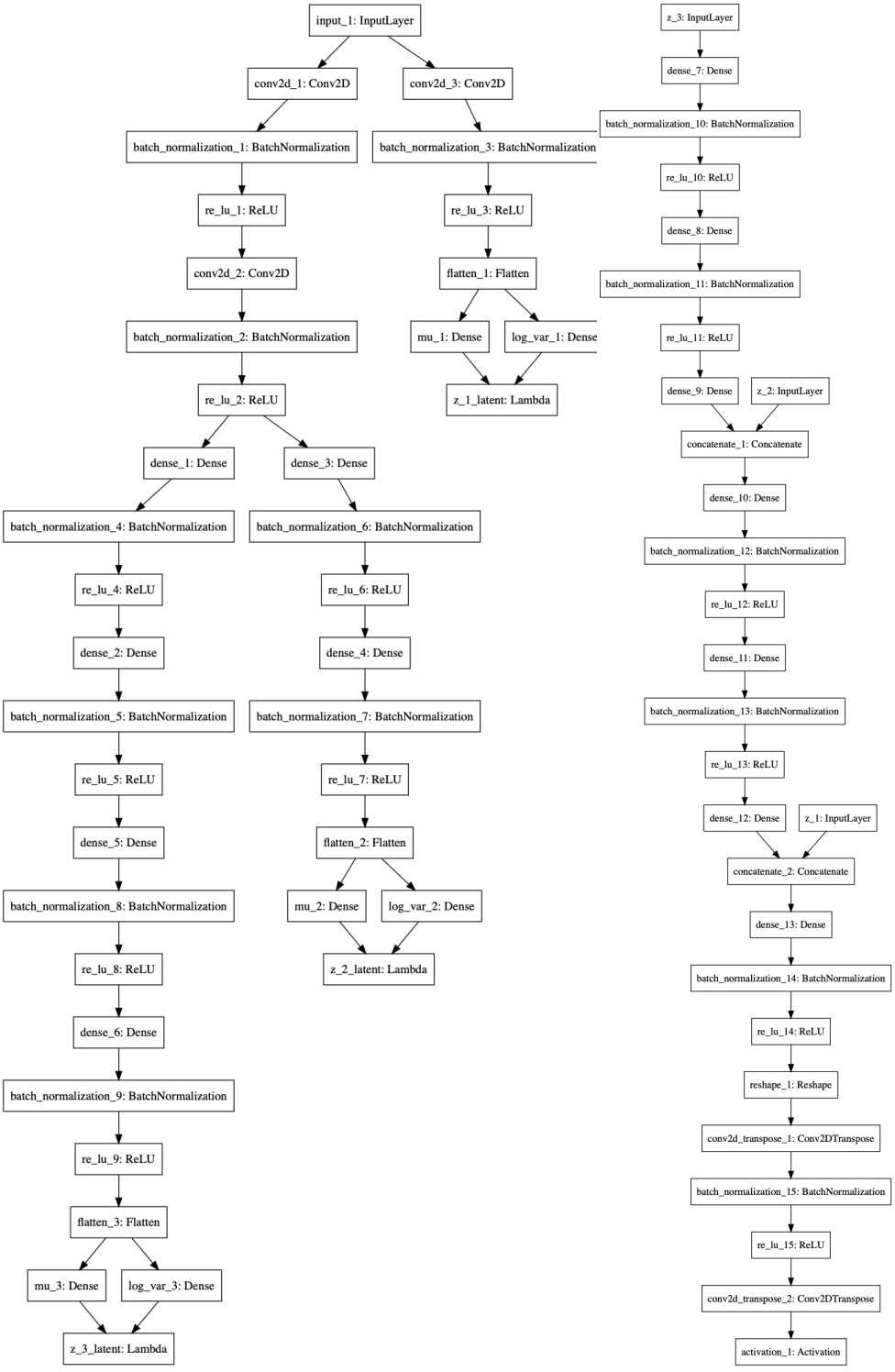
The latent variables then are sampled conditionally:

$$q(\boldsymbol{z}_i|\cdot) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_{q,i}, \boldsymbol{\sigma}^2_{q,i}). \tag{36}$$

Sønderby et al. [68] show that gradually increasing the weight of the KL-term in the VAE training criterion leads to better usage of the latent spaces in terms of active units. Without this *warm-up* phase, VAEs show many inactive units. Sønderby et al. [68] discuss that VAEs show this behavior because it allows them to minimize the KL-term in the VAE loss function quickly.

Using batch-normalization and warm-up alone already leads to more meaningful representations in the higher layers of a hierarchical VAE [68]. Using all these techniques on the LVAE model allows meaningful representations in even higher layers than a hierarchical VAE[8].

[8] In Sønderby et al. [68], a hierarchical VAE with batch-normalization and warm-up learns meaningful representations up to the fourth layer where as 5-layer LVAE also learns meaningful representations in the fifth layers. Deeper models were not investigated.



(a) Varying $\boldsymbol{c}_1 \in [0, 10]$ (b) Varying $\boldsymbol{c}_2 \in [-2, 2]$

Figure 10: Structure of the VLAE model

Variational Ladder Autoencoder Unlike the LVAE, the Variational Ladder Autoencoder (VLAE) [75] is a pure feed-forward network. Its design is driven by the idea that "[i]f $\boldsymbol{z}_i$ is more abstract than $\boldsymbol{z}_j$, then the inference mapping $q(\boldsymbol{z}_i|\boldsymbol{x})$ and generative mapping when other layers are fixed $p(\boldsymbol{x}|\boldsymbol{z}_i, \boldsymbol{z}_{\neg i} = z^0_{\neg i})$ requires a more expressive network to capture" [75]. In simplified terms, learning more complex input-features requires as deeper network than for less complex input-features. Figure 10 shows an exemplary network structure of a VLAE. Early embedding layers like $\boldsymbol{z}_1$ ($\boldsymbol{z}_1$ corresponds to *z_1_latent* in Figure 10) are equipped with a less powerful network that. According to Zhao, Song, and Ermon [75], early layers should not learn abstract features[9].

Equipping lower layers with less powerful networks allows VLAEs to encode lower-level factors of variation (for example *stroke thickness* in the context of a dataset of written digits) in lower layers and higher-level factors of variation (e.g., *digit identity*) in higher layers. Zhao, Song, and Ermon [75] claim that the VLAE, for such a dataset, learns different factors of variation independently in different layers.

The hierarchical structure of embedding layers in the VLAE make it a candidate model for the visual cortex. However, the proposition that different layers learn different factors of variation independently seems to be mostly based on an analysis on model-generated samples.

***β*-VAE** Consider Equation 24, describing the loss function in a VAE. Higgins et al. [29] discovered that weighting both, the KL-divergence and the reconstruction loss, equally leads to suboptimal results in terms of *feature disentanglement*. Therefore, they added a hyperparameter β to the loss function, controlling the relative weights of the two terms in the β-VAE learning objective [29]:

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \beta D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})). \tag{37}$$

A value of $\beta > 1$ puts more emphasis on the KL-term than the original VAE loss function ($\beta = 1$). Higgins et al. [29] state that this could lead to better feature disentanglement because the KL-term "encourages conditional independence in $q_\phi(\boldsymbol{z}|\boldsymbol{x})$" [29]. A proper disentanglement requires that some factors of the data are conditionally independent. However, Higgins et al. [29] empirically show that disentanglement also succeeds for data not having this property.

Setting $\beta > 1$ also lowers the influence of the reconstruction error in the VAE training objective. A larger β thus leads to worse reconstructions. The right parameter choice is therefore context-dependent.

Burgess et al. [9] show that disentangled representations and a good reconstruction quality can be achieved by giving the model "control of the encoding capacity." Controlling encoding capacity is done by introducing a parameter C to Equation 37 [9]:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right] - \gamma \left| D_{KL}\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p(\boldsymbol{z})\right) - C \right|. \tag{38}$$

During training, C is gradually increased from zero to a higher value (25 for dSprites with $\gamma = 1000$ [9]).

VAE-GAN VAE-GANs [41] are a variation of VAEs specifically designed to replace the pixel-wise reconstruction loss. Even on highly curated datasets such as MNIST (see Section 3.3.3), VAEs tend to generate blurry images [39]. The design of VAE-GANs is based on the assumption that the pixel-wise loss in the VAE training objective (see Equation 27) is the reason for blurry

[9] In case of MNIST, more abstract features are for example the digit identity.

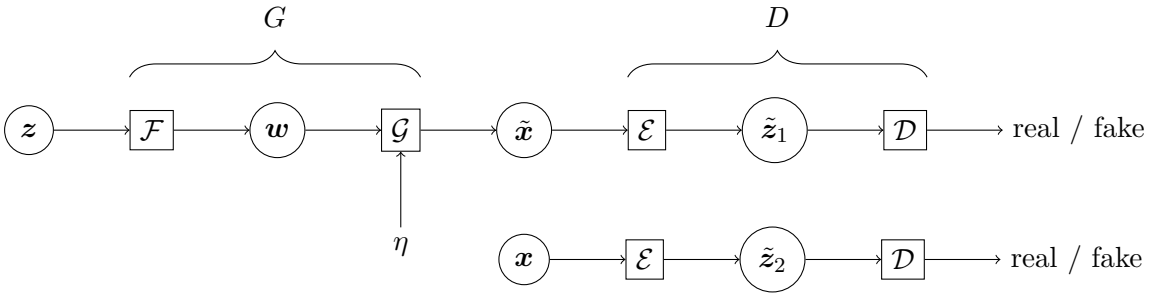


Figure 11: Training of ALAE, adapted from Pidhorskyi, Adjeroh, and Doretto [55]

output images. Larsen et al. [41] motivate this by stating that even small translations result in a high pixel-wise loss[10].

Instead of measuring the MSE between the true and the generated image, VAE-GANs train a discriminator network to distinguish between generated and real images. They maximize

$$\mathcal{L}_{\text{GAN}} = \log(D(\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z}))), \tag{39}$$

where D and G are the discriminator/generator networks, $\boldsymbol{x}$ an input image and $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ a random Gaussian sample. For VAE-GANs, the generator network is nothing but the decoder network, whereas the discriminator network is an additional network.

Equation 39 is sufficient to train the discriminator network. However, it does not force $p(\boldsymbol{x})$ to resemble a standard multivariate normal distribution, nor does it force reconstructions to resemble the input image.

The latter is achieved by introducing an additional loss term

$$\mathcal{L}_{\text{llike}}^{\text{D}} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})} \left[\log p \left(\text{D}_l(\boldsymbol{x}) | \boldsymbol{z} \right) \right], \tag{40}$$

penalizing differences of intermediate discriminator layer activations between $\boldsymbol{x}$ and the reconstruction $\tilde{\boldsymbol{x}}$. Forcing $p(\boldsymbol{x}) \approx \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is achieved by minimizing the KL-divergence as in Equation 24:

$$\mathcal{L}_{\text{prior}} = D_{KL}(q(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})). \tag{41}$$

The encoder minimizes $\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{D}}$, the decoder minimizes $\gamma \mathcal{L}_{\text{llike}}^{\text{D}} - \mathcal{L}_{\text{GAN}}$[11], and the discriminator minimizes $-\mathcal{L}_{\text{GAN}}$.

Larsen et al. [41] show that this training procedure leads to less blurry reconstructions. Adapting the VAE-GAN training objective to the VLAE-model could lead to improvements for that model as well.

Furthermore, by training a discriminative network to distinguish between prior and posterior samples, Makhzani et al. [50] have enforced arbitrary distributions in the latent space .

Adversarial Latent Autoencoder Similar to VAE-GANs, adversarial latent autoencoders (ALAEs) [55] are a hybrid model between VAEs and GANs. ALAEs are not based on variational inference but on the GANs training objective.

The starting point of the ALAE model is to decompose both the generator G and discriminator D into two parts, i.e., $G = \mathcal{G} \circ \mathcal{F}$ and $D = \mathcal{D} \circ \mathcal{E}$ (see Figure 11). $\mathcal{G}$ uses noise η as an additional

[10] This depends on the images' frequency as well as the variance of the pixel intensities.
[11] Choosing $\gamma = 0.75$ led to training convergence for the experiments performed for this thesis.

input. $\mathcal{F}$ maps the Gaussian random noise $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to another latent representation $\boldsymbol{w}$. The loss function requires the outputs of $\mathcal{F}$ and $\mathcal{E}$ to be similar in terms of the l^2-norm. Apart from this, the $\boldsymbol{w}$-space is unregularized. In addition to the l^2-loss, the model is optimized towards minimizing the discriminator and the generator loss.

Because $p_{\mathcal{E}}(w) \approx p_{\mathcal{F}}(w)$, ALAE can generate reconstructions by

$$\boldsymbol{x} \approx (\mathcal{E} \circ \mathcal{G}_\eta)(\boldsymbol{x}). \tag{42}$$

The notion $\mathcal{G}_\eta$ indicates that $\mathcal{G}$ uses η as an additional input.

Furthermore, just like for the VAE, the Gaussian latent space Z can be traversed to generate new samples by

$$\boldsymbol{x}_{\text{new}} = (\mathcal{F} \circ \mathcal{G}_\eta)(\boldsymbol{z}). \tag{43}$$

Pidhorskyi, Adjeroh, and Doretto [55] show that (linearly) interpolating in $\boldsymbol{w}$-space yields smoother transitions than linear interpolation between two corresponding points in $\boldsymbol{z}$-space. This indicates that the $\boldsymbol{w}$-space is less entangled than the $\boldsymbol{z}$-space [62, 2]. However, traversing the $\boldsymbol{w}$-space directly is not easily possible as its structure is unknown. Compared to VAEs, where a linear interpolation between points in the latent $\boldsymbol{z}$-space leads to an almost geodesic path in $\boldsymbol{x}$-space[12], the mapping function $\mathcal{F}$ seems to be non-smooth. Therefore, traversing $\boldsymbol{z}$ in an ALAE seems to lead to a non-smooth path in the feature space.

Compared to the VAE that uses a pixel-wise loss on the reconstructions, ALAE penalizes the l^2-norm of $\boldsymbol{w}$ and $\tilde{\boldsymbol{w}}$. Except for this loss, ALAE by no means forces the input to be similar to the reconstruction. This mitigates the problem of the pixel-wise loss in VAEs that is assumed to be the reason for blurry reconstructions.

For an in-depth discussion on representation learning with autoencoders, the reader is referred to Tschannen, Bachem, and Lucic [70].

Feature Consistency One important property of the VAE latent space is that it captures image semantics up to a certain degree. For a model trained on CelebA[13] it is possible to add sunglasses to a randomly generated face by adding the mean latent vector of images containing sunglasses [41, 56, 31]. Hou et al. [31] train a model particularly tailored to have this property by using the *perceptual loss* (the between hidden layer activations of a feature extraction network).

Feature consistency as well as an interpretable latent space qualify VAEs as models of the visual system.

2.6 Latent Space Disentanglement

Assume we have a dataset of images. Each image consists of a shape (e.g., *Square*, *Heart*, and *Ellipse*) at different x- and y-positions. Then, *shape*, *x-position*, and *y-position* are the "factors of variation" [63, 30], say S_1, S_2, and S_3 with $S_i \in \mathcal{S}^n$ [63]. Each image $\boldsymbol{x} \in \mathcal{X}^m$ can be generated from the factors of variation by a data-generating process $g : \mathcal{S}^n \times \mathcal{S}^n \times \mathcal{S}^n \mapsto \mathcal{X}^m$.

In the context of VAEs, the generator maps $\boldsymbol{x}$ to a vector $\boldsymbol{z}$ in the latent space. In a disentangled latent space, if a single factor of variation (e.g., the shape) is changed, one and only one subspace in the latent space is affected [30].

[12] Therefore, points that are close in $\boldsymbol{z}$-space of an VAE lead to images that are close in the feature space.
[13] See Section 3.3.1.

(a) "Distribution of features in the training set" [35]

(b) Mapping from latent space to feature space

Figure 12: Unbalanced regions of probability density as one problem of latent space entanglement (taken from Karras, Laine, and Aila [35])

A stricter definition additionally requires these subspaces to be linear subspaces of the latent space [30]. Furthermore, it is often desired that latent space sampling leads to the data distribution [35].

Latent space disentanglement is supposed to have two properties: *Consistency* and *Restrictiveness*, as defined in Shu et al. [63]. Consistency means that if only one factor of variation is varied (e.g., the factor that attributes to an objects' size), generated items only change for this very factor but not to other factors (i.e., only the objects' size is changing but not its shape). Restrictiveness means that if again, only one factor i of variation is varied, the choice of the other factors should not affect the models' measurement of the ith factor. For example, if factor i again encodes an objects' size, the same value of factor i should always encode the same size, irrespective of the choice of the other factors [63].

Consider Figure 12a and assume a dataset of voice records of males containing only two factors of variation: age (x-axis from older to younger) and voice level (y-axis, bottom-up from lower to higher). The missing square in the upper left is the combination "old and high voice level" that is not present in the dataset. Now, consider Figure 12b. The latent space is forced to follow a prior distribution. Figure 12b shows a mapping from a standard normal latent space to the feature space. The upper left square in Figure 12a is not present in this latent space. Therefore, there are regions of higher (or lower) probability density in the latent space than in the data space.

Different methods have been proposed to measure the degree of latent space entanglement. One approach is to measure the degree by which a generated image changes as the latent space is traversed (perceptual path length (PPL)) [35]. The PPL measures by how much reconstructions change for small variation in the latent space. It is calculated by moving by a small value ϵ on a path obtained by interpolating between two random $\boldsymbol{z}_1$, $\boldsymbol{z}_2$ (or $\boldsymbol{w}_1$, $\boldsymbol{w}_2$)[14], and taking the perceptual loss [33] between the image before moving by ϵ and after. Finally, the mean of this value is obtained by applying the procedure for sufficiently many samples. If the latent space is entangled, this value should be on average higher compared to a disentangled space.

Another approach is *Linear Separability*, measuring how well a linear hyperplane can separate sets of points. For this purpose, a linear support vector machine (SVM) is trained to predict class labels based on the latent points [35].

Kim and Mnih [37] propose to generate data by fixing one factor of variation (for example,

[14] Interpolation is spherical for $\boldsymbol{z}$ and linear for $\boldsymbol{w}$.

one dimension in the latent space) and randomly sampling from others. If the latent space is disentangled, there should be one or multiple almost invariant dimensions in the generated data that account for the fixed factor of variation.

2.7 Latent Space Separability

Latent Space Separability is related to Latent Space Disentanglement, but it is used in a stricter sense in the course of this thesis. Whereas Latent Space Disentanglement was mainly motivated by the disentanglement of one latent space, Latent Space Separability is concerned with splitting up the latent space representation of features into multiple levels of latent spaces. If successful, Latent Space Separability allows to directly control the factors of variation without the need to find the subspace in one latent space controlling this factor. Besides this, latent space separability has the same properties as latent space disentanglement. If there are less layers than factors of variation, the latent spaces in one layer also have to be disentangled.

For example, *InfoGAN* (see Section 2.5.3) and VLAEs are based on this principle.

3 Methods

The following Sections describe the methods used in the course of this thesis.

3.1 Research Questions

The studies in Section 4 are guided by the following research questions:

Number		Question
RQ1		Are VAEs or VLAEs related to the visual cortex in terms of ...
	a)	... the emergence of Gabor wavelets?
	b)	... sparse coding?
RQ2		Do VAEs or VLAEs fulfil the requirements of latent space disentanglement or latent space separability?
RQ3		Can VAEs or VLAEs represent both continuous and categorical factors of variation in the latent space?
RQ4		How do VAEs and VLAEs represent lower factors of variation in the latent space?
RQ5		Do VAEs or VLAEs learn independent factors of variation independently in the latent space?
RQ6		Are the latent spaces of VLAEs independent in terms of generated images?
RQ7		Do VAEs/VLAEs-generated images resemble the data distribution?
RQ8		Is the discriminative loss superior to the pixel-wise loss in terms of the previous research questions?

Table 1: Research Questions

3.2 Implementation Details

All models are implemented with Keras[15] in Version 2.2.4 using the TensorFlow[16] backend in Version 1.15.. The models are trained on Tesla V100-DGXS GPUs with 16GB of RAM. The model code can be found under https://github.com/LeoIV/master-thesis-leonard.

3.3 Datasets

Five different datasets were used to train the models. Four of the datasets contain images of different sizes, the fifth dataset provides additional labels for one of the datasets. The images were resized to match the models' expected input sizes using Lanczos interpolation [8, pp. 223, ff].

3.3.1 CelebA

The *CelebA* dataset [46] consists of 202,599 RGB images of size 178 x 218 pixels representing celebrities, as well as 40 binary attributes. The images belong to 10.177 unique identities[17] as well as five "landmark annotations". They are aligned and cropped resulting in images of same size always showing only one face (see Figure 13 for an example). The landmark annotations give the positions of facial attributes in the image; the left and right eye, the nose, and the

[15] https://keras.io/, last access: 07/01/2020
[16] https://www.tensorflow.org/, last access: 07/01/2020
[17] The identities are not revealed.

Figure 14: Examples from the MNIST dataset.

left and right corner of the mouth. The binary attributes indicate if the image has certain characteristics, for example if the person wears eyeglasses, has black hair, is smiling[18].

3.3.2 ImageNet

ImageNet[19] is a large-scale "image database organized according to the WordNet hierarchy" [16]. It contains of over 14 million images as of February 2020. According to WordNet[20], the images are subdivided into groups called "synsets" [16] on different levels of granularity. For example, the group *woman, adult female* is subordinated to *person, individual, someone, somebody, mortal, soul* and is further subdivided into groups like *old woman* or *lady* [21]

Figure 13: A sample image from the CelebA dataset.

A smaller version of ImageNet, commonly called *ILSVRC2012* has been used for Large Scale Visual Recognition Challenge 2017 (ILSVRC2017) [61], consisting of approximately 1,3 million images from 1000 different classes, that were selected, such that "there is no overlap between synsets: for any synsets i and j, i is not an ancestor of j in the ImageNet hierarchy" [16].

This curated version is commonly used as a baseline [40, 67][22].

3.3.3 Mnist

MNIST[23] [43] is a widely-used dataset of hand-written digits. Figure 14 shows ten examples from this dataset. The data is subdivided into a training set of 60.000 images and a test set containing 10.000 images. The digits are all of the same size and centered. The samples are grayscale images of size 28×28pixels.

[18] See https://www.kaggle.com/jessicali9530/celeba-dataset#list_attr_celeba.csv for a complete list of the attributes, login required. Last access: 12/02/2020.

[19] http://image-net.org/, last access: 12/02/2020.

[20] See https://wordnet.princeton.edu/, last access: 12/02/2020.

[21] ImageNet 2011 Fall Release, http://image-net.org/explore, last access: 12/02/2020.

[22] See https://paperswithcode.com/sota/image-classification-on-imagenet for an overview of models on ImageNet. Last access: 07/16/2020

[23] http://yann.lecun.com/exdb/mnist/, last access: 23/04/2020

3.3.4 Morpho-Mnist

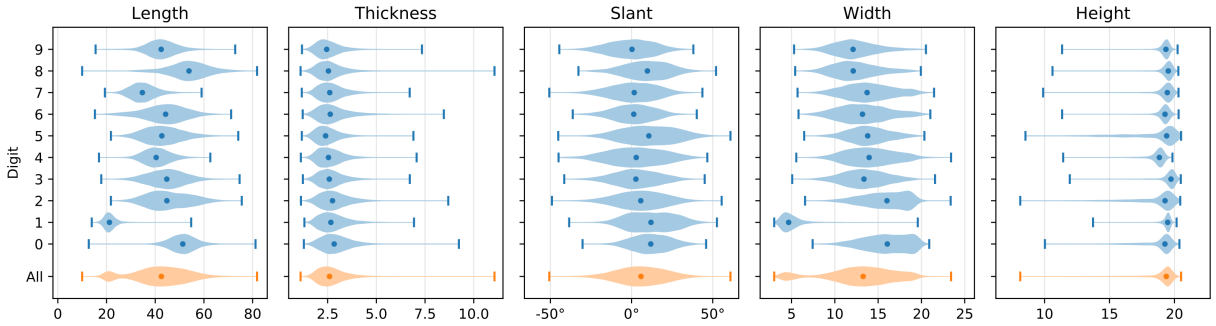


Figure 15: Distribution of the Morpho-MNIST attributes for the different digits. Taken from [11].

Morpho-MNIST [11] is an extension of the MNIST dataset that addresses the question: "(T)o what extent has my model learned to represent specific factors of variation in the data?" [11]. To address this questions, Morpho-MNIST provides the following (continuous) labels of morphological attributes of the MNIST samples: stroke length, stroke thickness, slant, width, and height.

Besides providing additional labels of low-level MNIST attributes, Morpho-MNIST provides a toolbox to measure (i.e calculate the morphological labels) and perturb MNIST images. The perturbation toolbox allows it to thin, thicken, swell, and to add fractures to an image. Morpho-MNIST also provides pre-computed datasets that were built using the perturbation toolbox.

Importantly, the distribution of the morphological attributes partly is highly skewed (for example Thickness and Height, see Figure 15).

3.3.5 dSprites

dSprites[24] [51] is a dataset designed "to assess the disentanglement properties of unsupervised learning methods.". It contains 737,280 grayscale images of size 64×64 pixels. The images were generated from "6 ground truth independent latent factors": color, shape, scale, orientation, x-position, and y-position. The color is white in all images. The shapes are: square, ellipse, and heart. For the other factors, points are chosen evenly along their support: six values in $[0.5, 1]$ (scale), 40 values in $[0, 2\pi]$ (orientation), 32 values in $[0, 1]$ (x-position and y-position). Each factor combination only occurs once in the data set. The dataset also contains the factor labels for each image.

3.4 Models

Eight different VAE, six different VLAE, four different VAE-generative adversarial network (GAN), and four different VLAE-GAN were evaluated in the course of this thesis. Furthermore, two "AlexNet" models were used for some additional experiments.

The models vary depending on the dataset and are described in the following. An overview is given in Table 2. A more detailed description can be found in Appendix A.

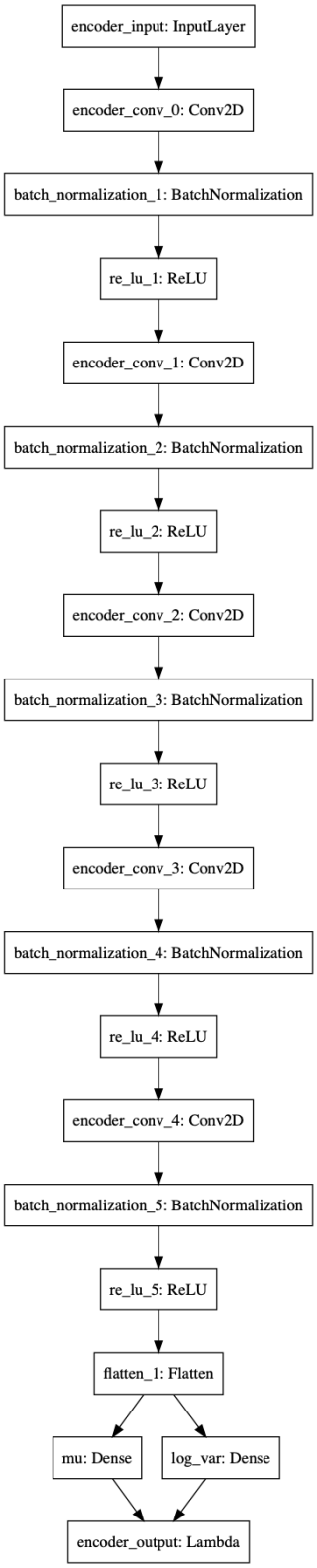
3.4.1 VAE Models

The VAE model (see Section 16) consists of an encoder and a decoder. The encoder is made up of multiple "Convolution, Activation, Batch-Normalization"-blocks, followed by the embedding

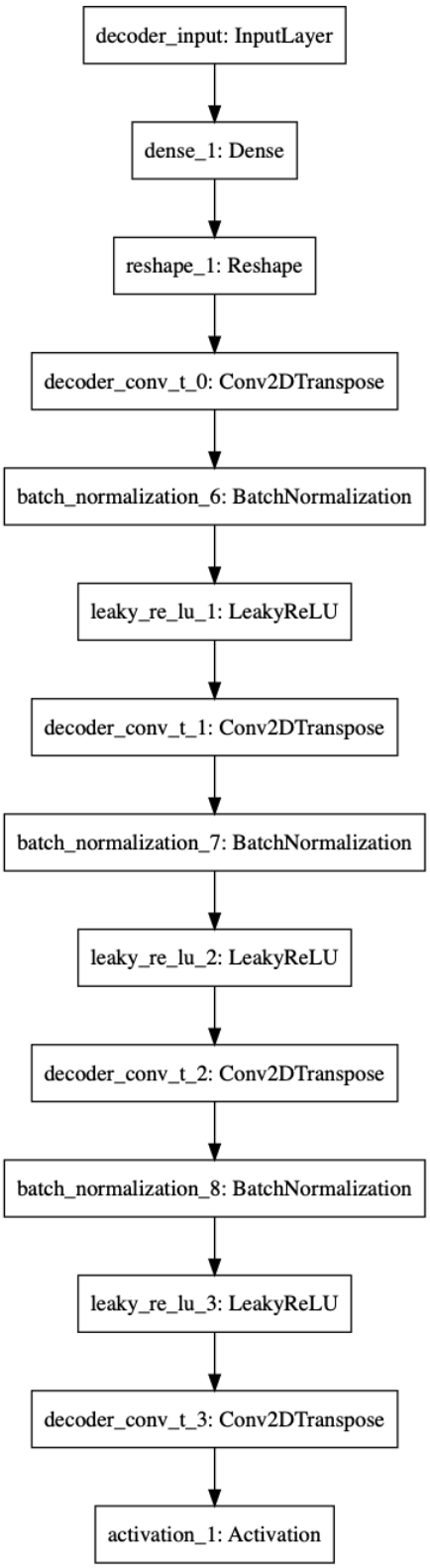
[24] https://github.com/deepmind/dsprites-dataset/, last access: 5/28/2020

model name	dataset	input/output size	latent space size	reconstruction term weight	feature map reduction factor
MNIST-VAE	MNIST	$28 \times 28 \times 1$	2	10,000	1
(dSprites/10,000)-VAE	dSprites	$64 \times 64 \times 1$	10	10,000	1
7,500-VAE	dSprites	$64 \times 64 \times 1$	10	7,500	1
6,250-VAE	dSprites	$64 \times 64 \times 1$	10	6,250	1
5,000-VAE	dSprites	$64 \times 64 \times 1$	10	5,000	1
3,750-VAE	dSprites	$64 \times 64 \times 1$	10	3,750	1
dSprites-VAE-dim6	dSprites	$64 \times 64 \times 1$	6	10,000	1
CelebA-VAE	CelebA	$128 \times 128 \times 3$	8	3,750	1
MNIST-VLAE(-factor-1)	MNIST	$28 \times 28 \times 1$	2,2,2	10,000	1
MNIST-VLAE-factor-2	MNIST	$28 \times 28 \times 1$	2,2,2	10,000	2
MNIST-VLAE-factor-3	MNIST	$28 \times 28 \times 1$	2,2,2	10,000	3
dSprites-VLAE	dSprites	$64 \times 64 \times 1$	4,4,4	10,000	1
dSprites-VLAE-dim2	dSprites	$64 \times 64 \times 1$	2,2,2	10,000	1
CelebA-VLAE	CelebA	$128 \times 128 \times 3$	2,2,2	10,000	1
MNIST-VAE-GAN	MNIST	$28 \times 28 \times 1$	2	10,000	1
dSprites-VAE-GAN	dSprites	$64 \times 64 \times 1$	10	10,000	1
CelebA-VAE-GAN	CelebA	$128 \times 128 \times 3$	8	10,000	1
MNIST-VLAE-GAN	MNIST	$28 \times 28 \times 1$	2,2,2	10,000	1
dSprites-VLAE-GAN	dSprites	$64 \times 64 \times 1$	4,4,4	10,000	1
CelebA-VLAE-GAN	CelebA	$128 \times 128 \times 3$	2,2,2	10,000	1
AlexNet Classifier	ImageNet	$224 \times 224 \times 3$	–	–	1
AlexNet VAE	ImageNet	$224 \times 224 \times 3$	2000	10,000	1

Table 2: Overview of all models with important parameters.



(a) Encoder



(b) Decoder

Figure 16: VAE model structure

layer. The embedding layer predicts μ and $\log \sigma^2$ and performs the resampling by:

$$z = \mu + \epsilon\sigma \tag{44}$$
$$\epsilon \sim \mathcal{N}(0, \boldsymbol{I}). \tag{45}$$

The encoder input size is equal to the decoder output size and depends on the dataset. The number of "Convolution, Activation, Batch-Normalization"-blocks is chosen depending on the input size, as smaller input sizes require fewer layers to achieve a receptive field of the input size. The batch-normalization [26, pp. 317, ff.] can be omitted[25]. The activation can be either ReLU [26, p. 173] or LeakyReLU [26, p. 192] and is ReLU unless stated otherwise. The convolutions use zero-padding unless stated otherwise. Encoder and decoder use stridden convolutions for downsampling, unless stated otherwise.

The VAE model implements the loss function from Equation 24 but with a pre-factor for the reconstruction term. The reconstruction term pre-factor was determined empirically, observing reconstruction and generation quality.

The decoder uses similar blocks as the encoder but employs transposed convolutions [26, pp. 356, ff.] instead of convolutions to upsample feature maps. The output layer of the decoder uses a sigmoid activation instead of ReLU.

In total, eight VAE-models are used: "MNIST-VAE", "dsprites-VAE", "7,500-VAE", "6,250-VAE", "5,000-VAE", "3,750-VAE", "dsprites-VAE-dim6", and "CelebA-VAE". The model structures can be found in Appendix A.1.

Mnist-VAE MNIST-VAE uses an input- and output-size of $28 \times 28 \times 1$ (MNIST images are grayscale images). The model is trained with the Adam optimizer on the MNIST training set with a batch size of 128 and a learning rate of 0.001 for 200 epochs. The reconstruction loss factor is 10,000. The latent space is two-dimensional. The inner activation function is ReLU.

dSprites-VAE dSprites-VAE uses an input- and output-size of $64 \times 64 \times 1$. The model is trained with the Adam optimizer on a training set consisting of 90% of the dSprites dataset with a batch size of 128 and a learning rate of 0.001 for 200 epochs. The reconstruction loss factor is 10,000. The latent space is ten-dimensional. The inner activation function is ReLU. For dSprites, four additional models have been trained with a different reconstruction term weight: 7,500-VAE, 6,250-VAE, 5,000-VAE, and 3,750-VAE. These models differ from dSprites-VAE only in the reconstruction term weight.

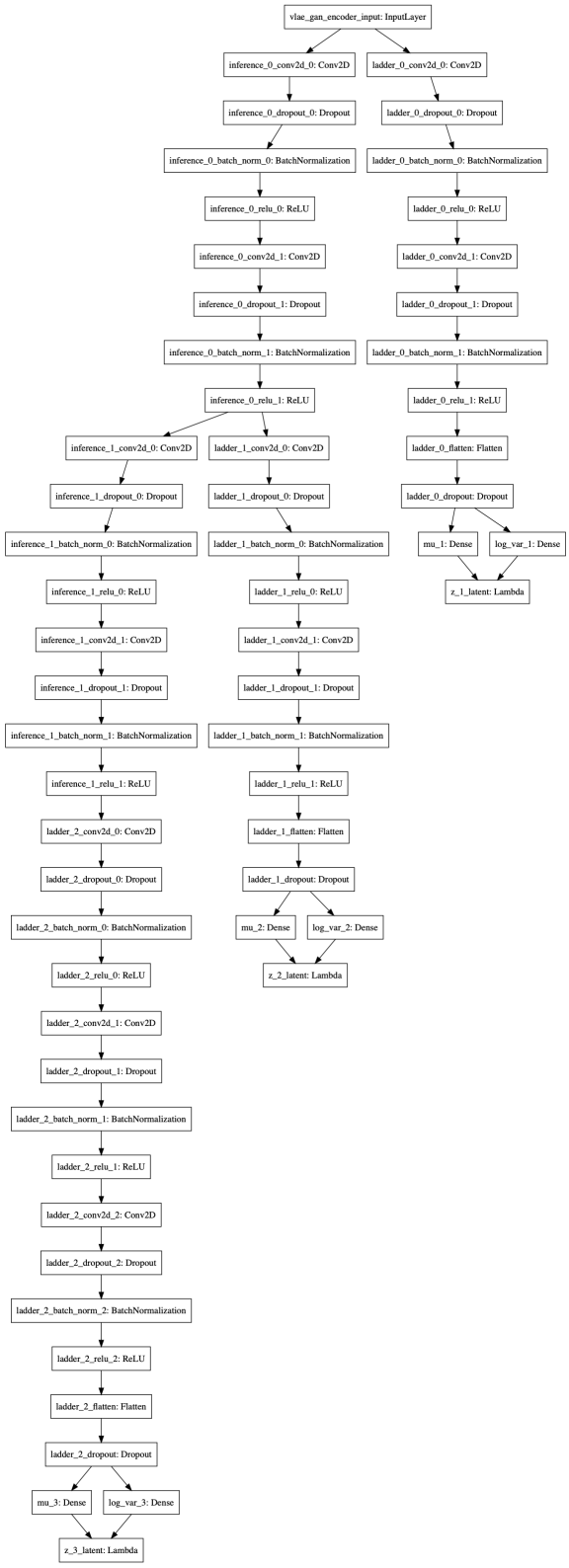
dsprites-VAE-dim6 The dsprites-VAE-dim6 is equivalent to the dSprites-VAE model but uses a six-dimensional latent space.

CelebA-VAE CelebA-VAE uses an input- and output-size of $128 \times 128 \times 3$. The model is trained with the Adam optimizer on a training set consisting of 90% of the CelebA dataset with a batch size of 128 and a learning rate of 0.001 for 200 epochs. The reconstruction loss factor is 10,000. The latent space is eight-dimensional. The inner activation function is ReLU.

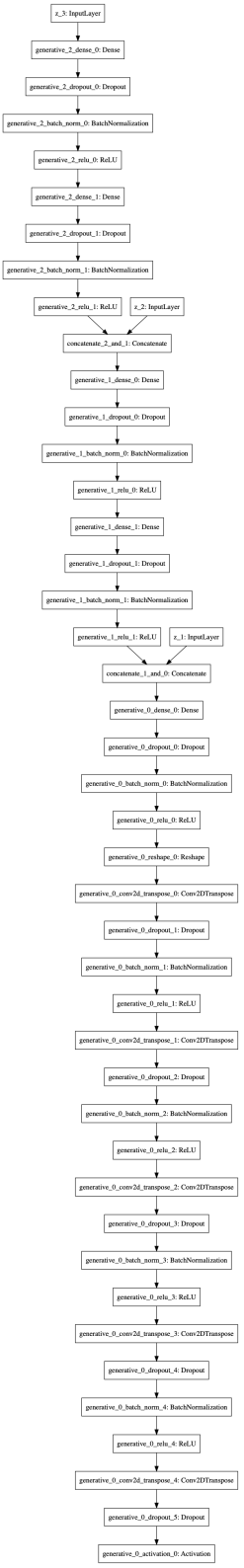
3.4.2 VLAE Models

Figure 17 shows the VLAE model structure. Like the VAE, it consists of an encoder and a decoder. The encoder has three latent spaces[26] A re-sampling according to Equation 45 is performed for each of the latent spaces. Lower latent spaces are equipped with a less powerful

[25] It is stated in the experiments if batch-normalization is omitted.
[26] *z_1_latent*, *z_2_latent*, and *z_3_latent* in Figure 17a.



(a) Encoder



(b) Decoder

Figure 17: VLAE model structure

encoder (e.g., *z_1_latent* in Figure 17a), higher latent spaces with a more powerful encoder. Again, the network is composed of multiple "Convolution, Activation, Batch-Normalization"-blocks. The number of these blocks is variable and chosen depending on the dataset. Batch-normalization can be omitted (default), the inner activation can be either ReLU (default) or LeakyReLU. The convolutions use-zero padding, and encoder and decoder use stridden convolutions for downsampling.

The VAE model implements the loss function from Equation 24 but with a pre-factor for the reconstruction term. The Kullback-Leibler (KL)-terms of different layers are totalized.

The decoder has three inputs, where the first input of the decoder[27] receives input from the last output of the encoder[28]. The decoder uses blocks similar to the encoder but with transposed convolutions instead of regular convolutions.

In total, six VLAE-models are used: "Mnist-VLAE-factor-1", "Mnist-VLAE-factor-2", "Mnist-VLAE-factor-3" "dSprites-VLAE", "dSprites-VLAE-dim2", and "CelebA-VLAE". The model structures can be found in Appendix A.2.

Mnist-VLAE The three Mnist-VLAEs[29] use an input- and output-size of $28 \times 28 \times 1$. The models are trained with the Adam optimizer on the Mnist training set with a batch size of 128 for 200 epochs. The reconstruction loss factor is 10,000. The latent spaces are two-dimensional. The inner activation function is ReLU. The models use no batch-normalization. Mnist-VLAE-factor-1 is the model with the original number of feature maps, for Mnist-VLAE-factor-2 and Mnist-VLAE-factor-3, the number of feature maps is reduced according to the factor. Mnist-VLAE-factor-1 is trained with a learning rate of 0.005. Mnist-VLAE-factor-2 and Mnist-VLAE-factor-3 are trained with a learning rate of 0.001.

dSprites-VLAE dSprites-VLAE uses an input- and output-size of $64 \times 64 \times 1$. The model is trained with the Adam optimizer on a training set consisting of 90% of the dSprites dataset with a batch size of 128 and a learning rate of 0.001 for 200 epochs. The reconstruction loss factor is 10,000. The latent spaces are four-dimensional. The inner activation function is ReLU.

dSprites-VLAE-dim2 The dSprites-VLAE-dim2 model is equivalent to dSprites-VLAE but uses a two-dimensional latent space.

CelebA-VLAE CelebA-VLAE uses an input- and output-size of $128 \times 128 \times 3$. The model is trained with the Adam optimizer on a training set consisting of 90% of the CelebA dataset with a batch size of 128, a learning rate of 0.001 with an additional learning rate decay of 0.01 for 200 epochs. The reconstruction loss factor is 10,000. The latent spaces are two-dimensional. The inner activation function is ReLU.

3.4.3 VAE-GAN Models

The VAE-GAN-model is similar to the VAE-model. However, it implements the VAE-GAN loss function (see Section 2.5.3) instead of Equation 24. Therefore, the VAE-GAN has an additional *discriminator* network. The feature loss compares inner activations in the discriminator of true and generated samples. The discriminator loss signifies by how much the discriminator violates the GAN training objective. See Section 2.5.3 for more details.

[27] *z_3* in Figure 17b
[28] *z_3_latent* in Figure 17a
[29] "Mnist-VLAE-factor-1", "Mnist-VLAE-factor-2", "Mnist-VLAE-factor-3"

For the encoder, the KL-term is weighted ten-times more strongly than the feature loss. The decoder weights the discriminator loss with factor 1 and the feature loss with factor 0.75.

In total, seven VAE-GAN-models are used: "MNIST-VAE-GAN", "dSprites-VAE-GAN", and "CelebA-VAE-GAN". The model structures can be found in Appendix A.3.

Mnist-VAE-GAN MNIST-VAE-GAN uses an input- and output-size of $28 \times 28 \times 1$. The model is trained with the Adam optimizer on the MNIST training set with a batch size of 128 and a learning rate of 0.0001 for 200 epochs. The reconstruction loss factor is 10,000. The latent space is two-dimensional. The inner activation function is ReLU.

dSprites-VAE-GAN dSprites-GAN uses an input- and output-size of $64 \times 64 \times 1$. The model is trained with the Adam optimizer on a training set consisting of 90% of the dSprites dataset with a batch size of 128 and a learning rate of 0.0001 for 200 epochs. The reconstruction loss factor is 10,000. The latent space is ten-dimensional. The inner activation function is ReLU.

CelebA-VAE-GAN CelebA-VAE-GAN uses an input- and output-size of $128 \times 128 \times 3$. The model is trained with the Adam optimizer on a training set consisting of 90% of the CelebA dataset with a batch size of 128 and a learning rate of 0.001 with an additional learning rate decay of 0.02 for 200 epochs. The reconstruction loss factor is 10,000. The latent space is eight-dimensional. The inner activation function is ReLU.

3.4.4 VLAE-GAN Models

The VLAE-GAN-model is similar to the VAE-GAN-model in terms of the loss functions. However, it uses the structure of the VLAE-model and totalizes the three KL-losses from the different layers.

The model structures can be found in Appendix A.4.

Mnist-VLAE-GAN The three MNIST-VLAE-GAN uses an input- and output-size of $28 \times 28 \times 1$. The model is trained with the Adam optimizer on the MNIST training set with a batch size of 128 and a learning rate of 0.0001 for 200 epochs. The reconstruction loss factor is 10,000. The latent spaces are two-dimensional. The inner activation function is ReLU. The model uses no batch-normalization.

dSprites-VLAE-GAN dSprites-VLAE-GAN uses an input- and output-size of $64 \times 64 \times 1$. The model is trained with the Adam optimizer on a training set consisting of 90% of the dSprites dataset with a batch size of 128, a learning rate of 0.0001, and an additional learning rate decay of 0.01 for 200 epochs. The reconstruction loss factor is 10,000. The latent spaces are four-dimensional. The inner activation function is ReLU.

CelebA-VLAE-GAN CelebA-VLAE-GAN uses an input- and output-size of $128 \times 128 \times 3$. The model is trained with the Adam optimizer on a training set consisting of 90% of the CelebA dataset with a batch size of 128, a learning rate of 0.0001 with an additional learning rate decay of 0.01 for 200 epochs. The reconstruction loss factor is 10,000. The latent spaces are two-dimensional. The inner activation function is ReLU.

3.4.5 AlexNet Classifier

The AlexNet Classifier resembles the architecture from Krizhevsky, Sutskever, and Hinton [40]. It uses dropout and a dropout rate of 0.3. The model is trained with the Adam optimizer and

a learning rate of 0.0001 using batch normalization and a batch size of 32 for ten epochs. The model structure can be found in Appendix A.5.

3.4.6 AlexNet-VAE

The AlexNet-VAE resembles the AlexNet classifier but uses a 2000-dimensional latent space with re-sampling (see Equation 45). For AlexNet-VAE, no dropout is used. The model is trained with the Adam optimizer and a learning rate of 0.0001 using batch normalization and a batch size of 32 for 100 epochs.. The model structure can be found in Appendix A.6.

4 Results and Discussion

4.1 Gabor Wavelets in Variational Autoencoders

Simple and cells in the primary visual cortex (V1) show the strongest excitation for Gabor wavelets (see Section 2.4.1). As discussed earlier, the optimal stimulus for a convolutional kernel resembling a Gabor wavelet is a Gabor wavelet itself (see Section 2.4.3). Therefore, the emergence of Gabor-like filters is considered evidence for a models' biological plausibility [4]. Krizhevsky, Sutskever, and Hinton [40] found that AlexNet learns Gabor-like features in the first layer. It was later found that supervised convolutional neural networks (CNNs) like AlexNet explain activity in the inferior temporal cortex (IT) [36].

Therefore, the emergence of Gabor wavelets in Variational Autoencoders (VAEs) would be evidence for their suitability as a model of the visual system. They would potentially even indicate that higher layers explain higher regions' activity in the visual cortex. To examine if VAEs or Variational Ladder Autoencoders (VLAEs) learn Gabor wavelets, the following experiment was conducted (RQ1 a, see Table 1).

Since it is known that AlexNet learns Gabor wavelets in lower-level filters, a VAE network with similar architecture (AlexNet-VAE, see Section 3.4.6) was designed. The hypothesis is that if VAEs can learn Gabor wavelets, this model should show them because it is very similar to the supervised model for which they are learned. First, it was verified that the supervised network (AlexNet Classifier, see Section 3.4.5) actually learns Gabor wavelets. The network was trained for ten epochs, the top-1 training accuracy at this time was at 0.85.

Figure 18 shows all 288 ($96 \cdot 3$) convolutional kernels of AlexNet Classifier (see Section 3.4.5). In many kernels, Gabor wavelet-like filters emerge.

However, the same effect cannot be observed in the kernels AlexNet-VAE (see Section 3.4.6). Figure 19 shows the kernels of the first layer in AlexNet-VAE. Moreover, no Gabor wavelets emerged in any of the models[30] or for any of the datasets.

Therefore, it is concluded that a simple feed-forward VAEs or VLAEs trained to generate dense representations of static images probably cannot learn Gabor-wavelets. This does not imply that VAEs or VLAEs do not explain cortical activity (this is discussed in Section 4.13). Furthermore, variations of the VAE-design are proposed that could lead to a more biologically plausible model (see Sections 4.11 and 4.15).

4.2 Sparseness in Generative Models

One hyperparameter to choose in the different models is the number of feature maps of different layers. During the implementation, it was observed that some feature maps show little activity compared to others. As the number of features maps is a model hyperparameter, it was investigated by how much the number of feature maps can be reduced without increasing the network loss.

Furthermore, sparseness has been found to be used in the visual system. It is therefore investigated (RQ1 b, see Table 1) whether the sparseness in VLAEs is related to the sparseness in the primary visual cortex (see Section 2.1.5). Even though VAEs and VLAEs do not learn Gabor wavelets in lower layers, they could superpose sparse sets of different basis functions to represent their input, similar to Olshausen and Field [53]. If they do, this would support the relatedness of VLAEs to the biological example.

The following experiment was conducted to examine if VLAEs employ sparseness or if the model

[30] All models described in Section 3.4 apart from AlexNet Classifier.

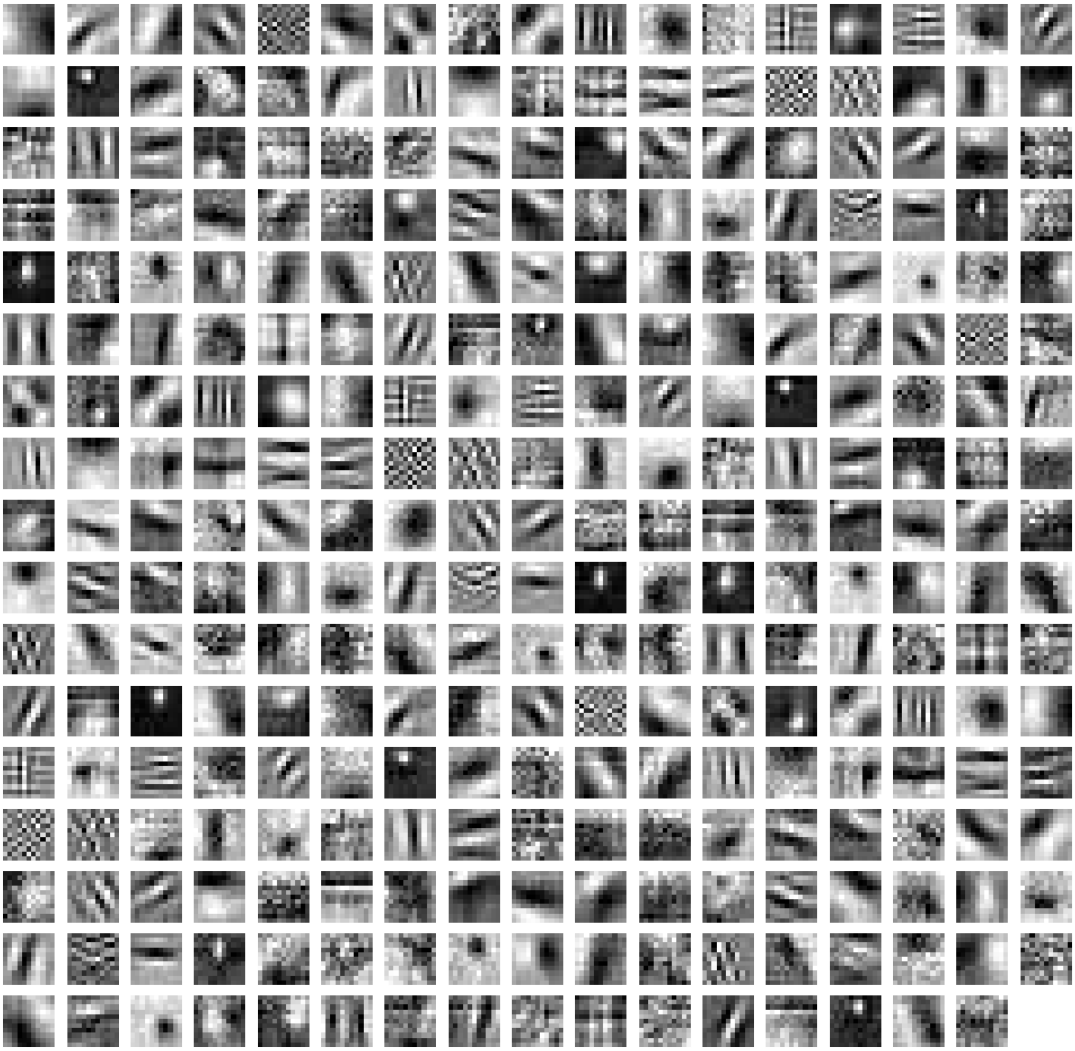


Figure 18: Convolutional Kernels in the first layer of the image classification network. The filters are shown in their original size (11x11).

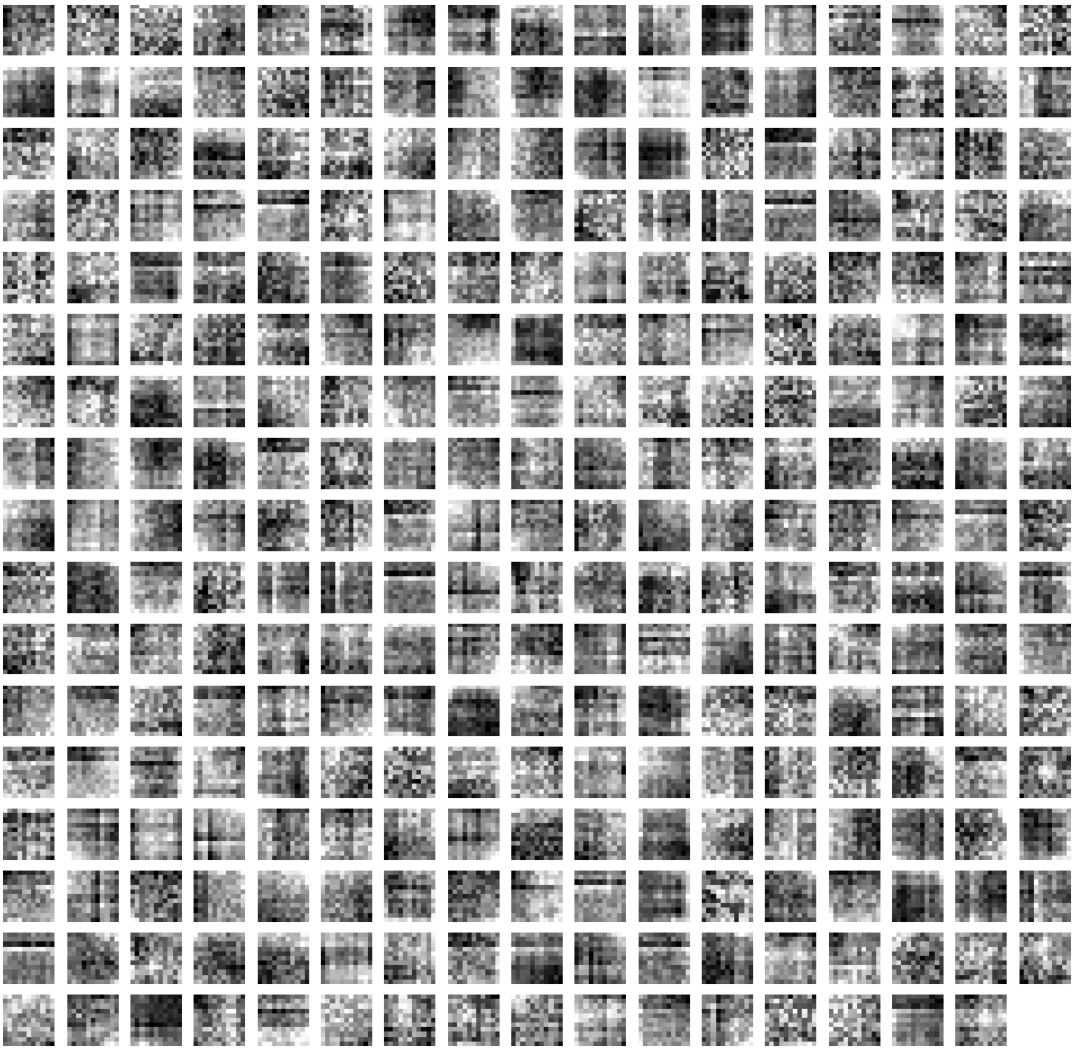
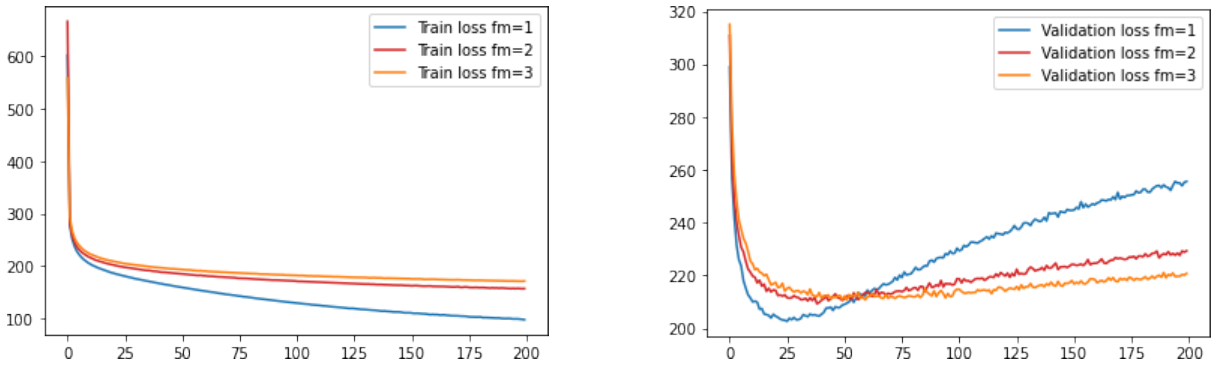


Figure 19: Convolutional Kernels in the first layer of the AlexNet VAE network. The filters are shown in their original size (11x11).



(a) Training loss

(b) Validation loss

Figure 20: Loss curves of models with different numbers of feature maps (see Section 3.4.1). fm is the reduction factor of the feature maps; $fm = 1$, therefore, is the original model, $fm = 2$ the model with half the number of feature maps, and $fm = 3$ the model with one third of the feature maps.

only has too much capacity. First, the feature map activities of VLAEs on MNIST with input size 28×28 and different numbers of feature maps were observed. Then, by gradually decreasing the network capacity, the evolvement of sparseness were analyzed.

In total, three models (MNIST-VLAE-factor-1, MNIST-VLAE-factor-2, and MNIST-VLAE-factor-3, see Section 3.4.2) were trained. Batch normalization was disabled for this experiment, because it re-scales the feature map activities.

Figure 20 shows the training and validation losses for the models. Firstly, increasing the number of feature maps increases convergence speed. Even though MNIST-VLAE-factor-1 is trained with a slightly lower learning rate, it converges faster than MNIST-VLAE-factor-2 and MNIST-VLAE-factor-3. It also achieves the lowest validation loss among the three models and achieves the minimum validation loss faster than the other models; it is minimal after 26 epochs for MNIST-VLAE-factor-1, 39 epochs for MNIST-VLAE-factor-2, and 52 epochs for MNIST-VLAE-factor-3. After achieving the minimum validation loss, however, all models overfit. MNIST-VLAE-factor-1 overfits stronger than MNIST-VLAE-factor-2 or MNIST-VLAE-factor-3, leading to the highest validation loss after 200 epochs.

Figure 21 shows the activities of the feature maps in MNIST-VLAE-factor-3. The bias terms are not considered: If the variations in the less-active feature maps are negligible, the feature maps carry no information except for the bias term. In that case, it should be possible to downscale the model without significantly impairing the model performance. Therefore, the network architecture was changed such that of n feature maps only the $\lfloor \frac{n}{4} \rfloor$ maps with the highest variance of feature map means remained active. Feature maps with lower variance are deactivated by a custom filtering layer. By appending a custom layer to each convolutional layer of the original network, the values of the less active feature maps were set to their mean values (as determined on the MNIST validation set, presented in Figure 21).

Figure 22 shows the result of the experimental setup for MNIST-VLAE-factor-3[31]. The feature map activities are high in lower layers. However, in higher layers, they are partially lower than in the original network (for example in *ladder_2_few_active_1*). In conclusion, activities of lower feature map are not negligible and cannot be replaced with bias terms because they influence higher layers.

A two-sided Wilcoxon rank-sum test with Benjamini/Hochberg correction for multiple tests was

[31] The plots for the other models can be found in Appendix B.

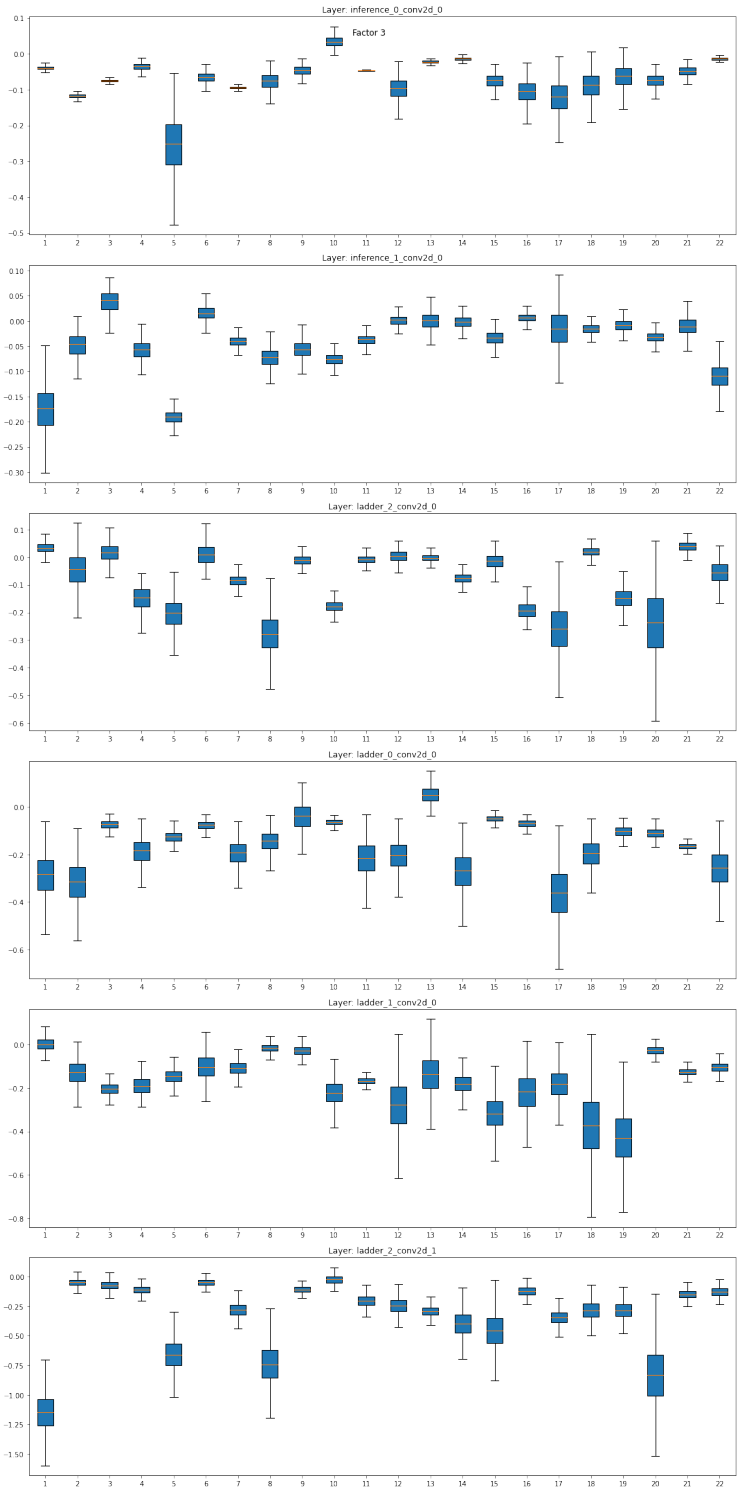


Figure 21: Feature map activities for MNIST-VLAE-factor-3. Each boxplot corresponds to the distribution of the mean value of one feature map after the convolution.

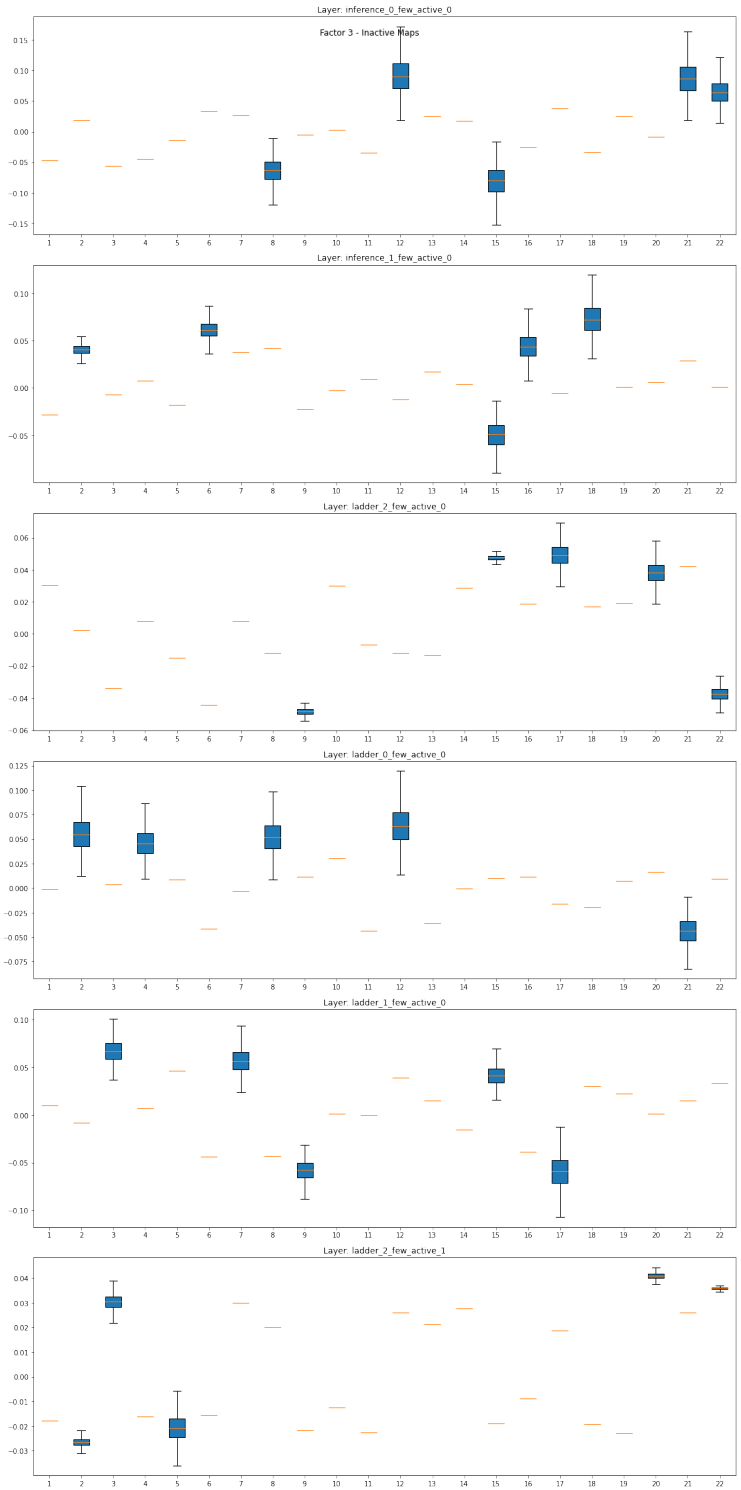


Figure 22: Activities of only the $\lfloor \frac{n}{4} \rfloor$ most active feature maps in MNIST-VLAE-factor-3. Each boxplot corresponds to the distribution of the mean value of one feature map after the convolution. Less active feature maps are set to their mean values.

Model	mean (SD) - Original	mean (SD) - Deactivated	p-value
MNIST-VLAE-factor-1	232.759 (8.136)	1190.616 (14.710)	< 0.001
MNIST-VLAE-factor-2	208.107 (6.958)	1111.337 (16.326)	< 0.001
MNIST-VLAE-factor-3	200.258 (6.822)	1106.544 (13.942)	< 0.001

Table 3: Reconstruction losses and p-values of the comparisons for the original MNIST-VLAEs and the versions with deactivated feature maps.

conducted to examine if deactivating feature maps influences the reconstruction quality. The MNIST validation set was split in 24 subsets and the reconstruction errors for each subset were compared between the original model and the model with deactivated feature maps. Table 3 shows the reconstruction losses, their standard deviations, and the p-values of comparing the different models. The p-values indicate that the reconstructions of the models with deactivated feature maps are significantly worse. However, the models still reproduce meaningful digits.

In conclusion, the models do not seem to employ true sparseness like the visual cortex does. Feature maps with small activities contribute substantially to the model performance. Even though some feature maps are less active than others, they still contribute more than just the bias term. Since the feature maps are not truly inactive, decreasing the model size does not lead to more *active* feature maps. Yet, it helps preventing overfitting.

4.3 Latent Space Entanglement and Categorical Factors of Variation

As discussed in Section 2.6 and Section 2.5.3 (β-VAE), in practice, there is a trade-off between reconstruction quality and prior-posterior matching. Again, the β-VAE loss function is defined as

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \beta D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})). \tag{46}$$

Increasing the weight of the reconstruction term in the VAE training objective (smaller β in Equation 46[32]) leads to reconstructions more similar to the training samples. The posterior distribution, however, matches the prior distribution less precisely. Contrarily, increasing β in Equation 46 leads to a posterior closer to the prior distribution at the expense of reconstruction quality. Moreover, Higgins et al. [29] claim that increasing β leads to a better latent space disentanglement, i.e., each dimension in the latent space is uniquely correlated with a single factor of variation.

As discussed in Section 2.6, the quality of feature space disentanglement can be assessed by measuring how fast reconstructions change as the latent space is traversed. Nevertheless, a disentangled feature space should be problematic for datasets with categorical factors of variation, such as dSprites, where features change fast by definition. Even though CelebA has binary labels as well (e.g., "brown hair"), there still is enough variation within hair colors to allow a smooth translation between hair colors [29]. For dSprites, there are only three distinct shapes ("square", "ellipse", and "heart"). To achieve a good reconstruction quality on these shapes, the model would have to place them in separate areas of the latent space, violating both feature space disentanglement and a Gaussian posterior.

To analyze how VAEs behave for categorical factors of variation (RQ3, see Table 1), a VAE with input size 64×64 (the size of dSprites images) and a ten-dimensional latent space was

[32]The β in β-VAE is the Kullback-Leibler (KL)-term weight. Therefore, decreasing β analog to increasing the reconstruction term weight.

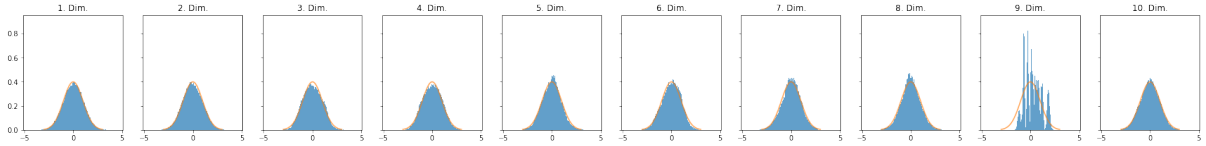
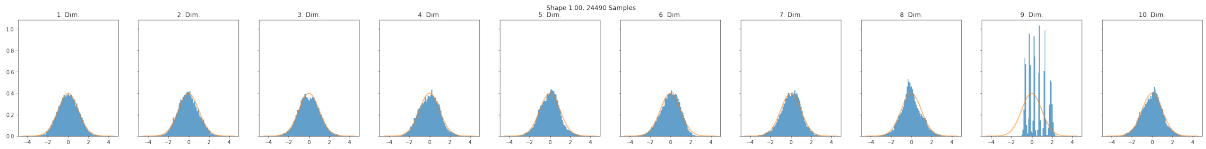
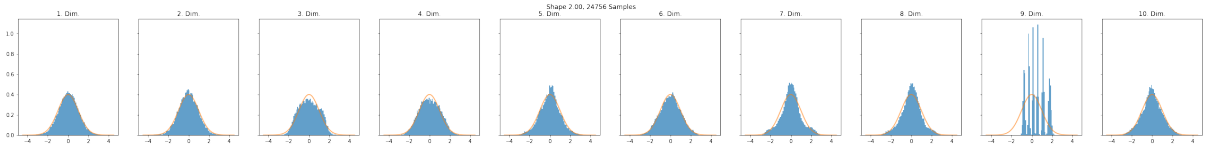


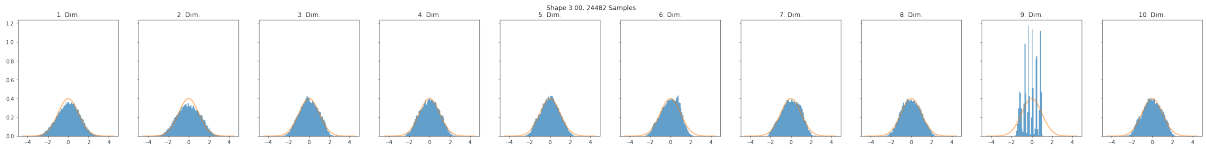
Figure 23: Posterior distribution of VAE with reconstruction term weight 10,000 on 73,728 dSprites images from the validation set in 100 bins



(a) Latent space distribution of images with shape *Square*



(b) Latent space distribution of images with shape *Ellipse*



(c) Latent space distribution of images with shape *Heart*

Figure 24: Histogram of dSprites images with a certain shape from the validation set in 100 bins

trained on dsprites with different reconstruction term weights: 10,000, 7,500, 6,250, 5,000, and 3,750 (see Section 3.4.1)[33].

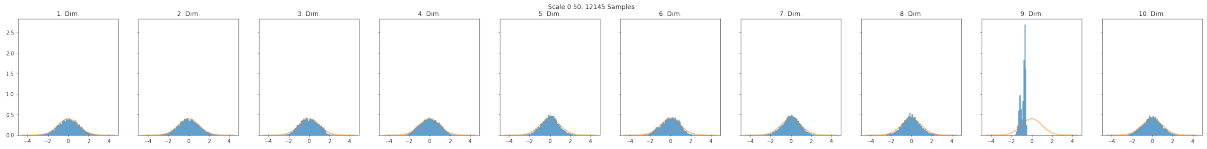
Figure 23 shows the posterior distributions of dSprites validation images. The ninth dimension is far less Gaussian than the other nine dimensions. As "object shape" is the only categorical factor of variation, one could assume that the ninth dimension encodes "object shape". However, the graph shows seven peaks in the ninth dimension, while there are only three distinct shapes in the dataset.

Figure 24 shows the distribution of images with a particular shape. Even though there is some difference between the plots, the μ-values in the ninth dimension still are assigned to different and distinct areas. The *shape* alone does not explain the sharp peaks in the ninth dimension.

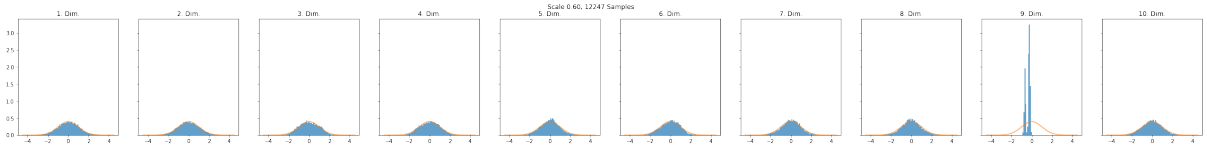
Consider Figure 25. It shows distributions of images grouped by their scale. Images in dSprites are scaled by six distinct factors evenly chosen from $[0.5; 1.0]$. The scale values in the dSprites dataset are six distinct values in the range from 0.5 to 1.0. A VAE model with a too strong reconstruction term does not learn to interpolate between these values but assigns them distinct areas in the latent space.

Grouping images by their scale *and* shape explains the distinct peaks in the ninth dimension (see Figure 26). Each shape has a distinct peak in the ninth dimension. However, the peaks of shapes *Ellipse* and *Square* superpose. When plotting all three shapes together, a small peak

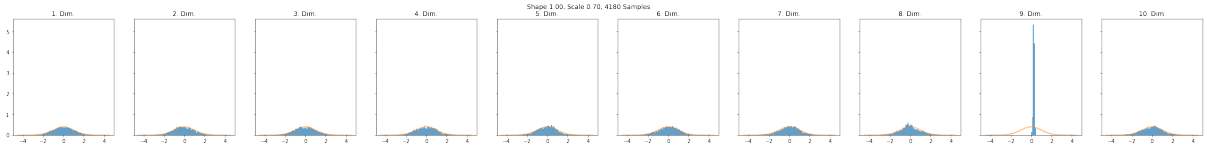
[33] It was first empirically validated that a reconstruction term weight of 10,000 leads to good reconstruction. The other values were chosen by reducing the reconstruction loss term by 1,250 for each model with a larger initial difference (10,000 vs. 7,500).



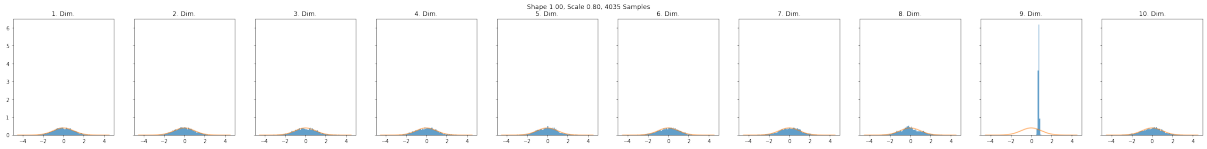
(a) Latent space distribution of images with scale = 0.5



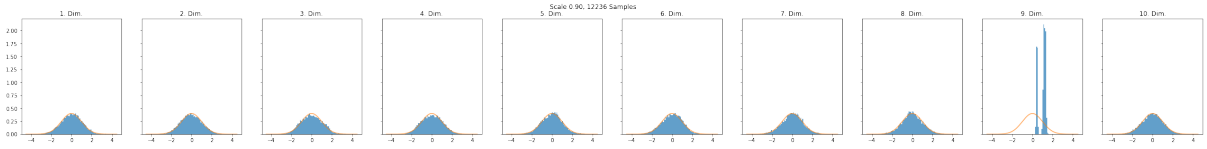
(b) Latent space distribution of images with scale = 0.6



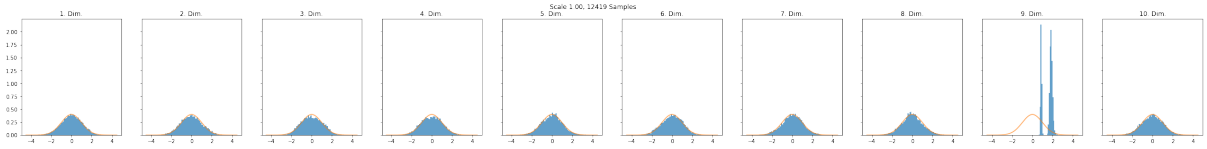
(c) Latent space distribution of images with scale = 0.7



(d) Latent space distribution of images with scale = 0.8

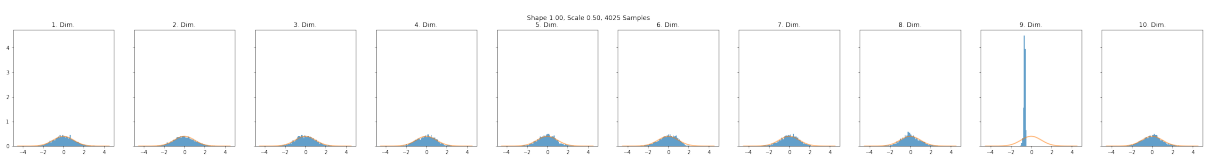


(e) Latent space distribution of images with shape scale = 0.9

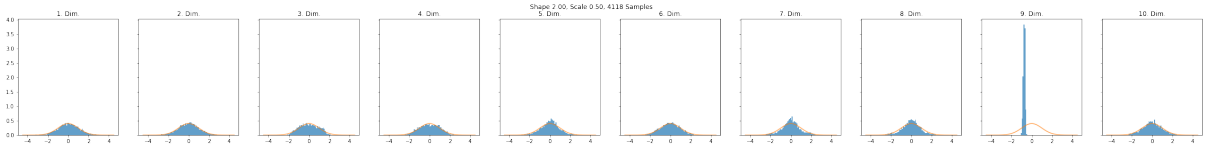


(f) Latent space distribution of images with shape scale = 1.0

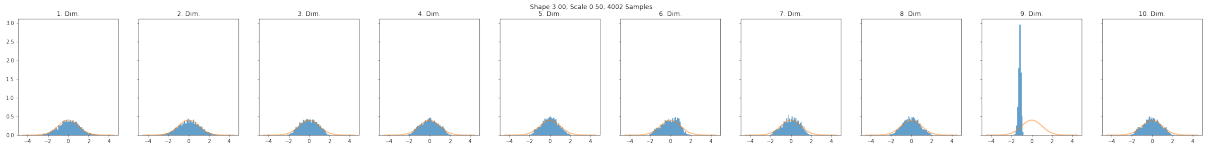
Figure 25: Posterior distribution for dSprites images with a certain scale from the validation set in 100 bins



(a) Latent space distribution of images with scale = 0.5 and shape *Square*



(b) Latent space distribution of images with scale = 0.5 and shape *Ellipse*



(c) Latent space distribution of images with scale = 0.5 and shape *Heart*

Figure 26: Posterior distribution for dSprites images with scale = 0.5 and different shapes from the validation set in 100 bins

(a) Latent space traversal from *Square* → *Ellipse*

(b) Latent space traversal from *Ellipse* → *Heart*

(c) Latent space traversal from *Heart* → *Square*

Figure 27: Latent space traversal between latent space representations of images with certain shapes for 10,000-VAE. Color-values were inverted for this plot.

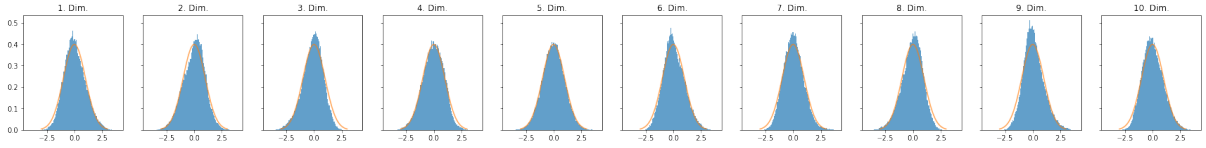
(*Heart*) and a big peak (*Ellipse* and *Square*) can be observed.

Furthermore, the model seems to learn a hierarchical clustering in the ninth dimension. *Shape* appears to be a sub-cluster of *Scale*. This sub-clustering leads to highly entangled latent space and a highly un-Gaussian distribution. However, it allows us to violate the KL-term in favor of the reconstruction term in just one instead of two dimensions.

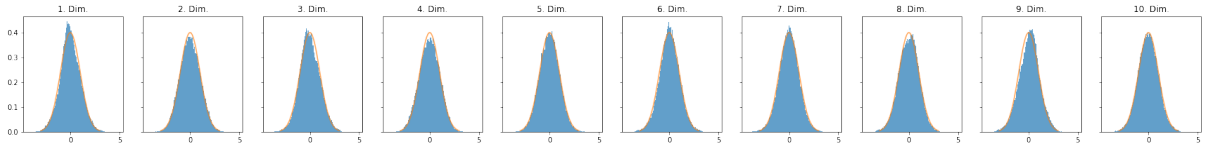
Figure 27 shows generated images for a latent space traversal between different shapes[34]. For the plot, the latent representations $\boldsymbol{z}_1, \boldsymbol{z}_2$ were obtained for two $\boldsymbol{x}_i$s with identical parameters except for the shape. Figure 27 shows the reconstructions of 21 evenly spaced points on the line segment between $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$. It shows the high degree of latent space entanglement: sudden position changes (Figure 27a) or the sudden emergence of new objects (Figure 62c).

Obtaining $\boldsymbol{z}$ by fixing only the shape and the scale for an $\boldsymbol{x}_i$ and averaging over all other configurations was unsuccessful. Based on the cluster analysis, not fixing the scale would easily result in regions with low probability density. It leads to empty images as it probably leads to $\boldsymbol{z}_i$s in regions of the latent space with low probability density. Furthermore, for the latent space traversal shown in Figure 27, it was possible to always find parameter combinations for which

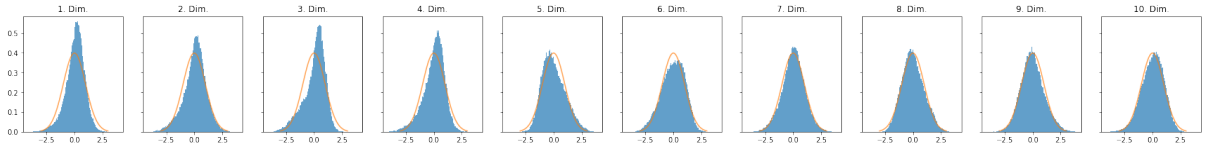
[34]The corresponding plots for the other model can be found in Appendix C.



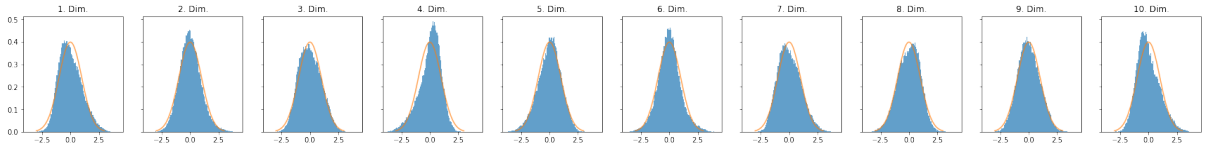
(a) Reconstruction term weight 7,500



(b) Reconstruction term weight 6,250



(c) Reconstruction term weight 5,000



(d) Reconstruction term weight 3,750

Figure 28: Posterior distribution of VAE with different reconstruction terms weights for 73,728 dSprites images from the validation set in 100 bins

the traversal lead to empty images in the middle of the line segment.

Figure 28 shows the latent space distribution of VAEs with a reduced reconstruction term weight, i.e., an increase in Kullback-Leibler divergence (KL-divergence) term weight. The histograms are more Gaussian, and the KL-divergence on the validation set decreased from 22.744 (10,000-VAE) to 22.271 (7,500-VAE), 19.399 (6,250-VAE), 18.172 (5,000-VAE), and 16.018 (3,750-VAE). The difference between 10,000-VAE and 7,500-VAE is small on the validation data, whereas the decrease in KL-divergence is higher for 6,250-VAE, 5,000-VAE, and 3,750-VAE.

To quantify the level of latent space disentanglement, the perceptual path length (PPL) (see Section 2.6) for the latent spaces of the VAEs (10,000, 7,500, 6,250, 5,000, 3,750) is computed. The perceptual loss is obtained by a CNN, trained to predict dSprites object categories[35]. The PPL was computed for 50 different random seeds. Each of these PPL-computations interpolated at 100 random steps (dependent on the random seed).

Table 4 shows the mean PPL values and their standard deviations. A two-sided Wilcoxon rank-sum test with Benjamini/Hochberg correction to account for repeated testing for the differences in the models' PPLs yielded significant p-values $< 10^{-29}$ in all cases.

Moreover, the 10,000-VAEs' PPL is lower than for 7,500-VAE or 6,250-VAE. This can be explained by the vast regions of low probability density in the posterior distribution. For 10,000-VAE, it is likely to produce images that are almost entirely black for some random point in the latent space. Only a few regions in the latent space produce reasonable images. A neighboring point in the latent space will probably also produce a black image, leading to a low perceptual

[35]The model architecture can be found in Appendix D, the features are extracted in "conv2d_94". The model was trained for one epoch using categorical crossentropy with the Adam optimizer with an initial learning rate of 0.1. The model achieved a top-1 accuracy of 0.8 on the validation set.

Model	mean PPL	standard deviation
10,000-VAE	936.257	779.098
7,500-VAE	2834.674	567.933
6,250-VAE	4498.156	1091.255
5,000-VAE	173.533	35.896
3,750-VAE	258.326	49.117

Table 4: Perceptual path lengths for the different models latent spaces

loss. As the reconstruction term weight is lowered (7,500-VAE and 6,250-VAE), the probability of generating some image and, therefore, the PPL increases. For 5,000-VAE and 3,750-VAE, finally, the latent space is sufficiently disentangled. This likewise leads to a low PPL, but this time due to an actual disentanglement rather than large regions of low probability density.

The standard deviation of the PPLs supports this interpretation. Consider 6,250-VAE. Here, generating some image for a random point in the latent space is already quite likely. Nevertheless, there are still "enough" regions producing black images. For these black regions, the PPL is low, just like for 10,000-VAE. Contrarily, for the points producing images, the PPL is high, because the latent space still is entangled. This causes a high standard deviation.

The previous observations have essential implications for the application of the PPL. First, the latent space needs to be approximately Gaussian for the PPL to produce interpretable results. This could be measured in terms of KL-divergence or by a visual analysis (as conducted for Figure 23). Second, if the latent space is Gaussian, the standard deviation should be observed. A high PPL standard deviation is evidence that there are regions of low probability density in the latent space.

4.4 Latent Space Analysis

VAEs and VLAEs force the latent space to approximate a standard normal distribution. However, latent space disentanglement (see Section 2.6) and latent space separability (see Section 2.7) require the latent space also to learn factors of variation in different (linear) sub-spaces. The following sections explore feature learning in the latent space and answer to what extent VAEs and VLAEs satisfy the requirements of latent space disentanglement and separability (RQ2, see Table 1).

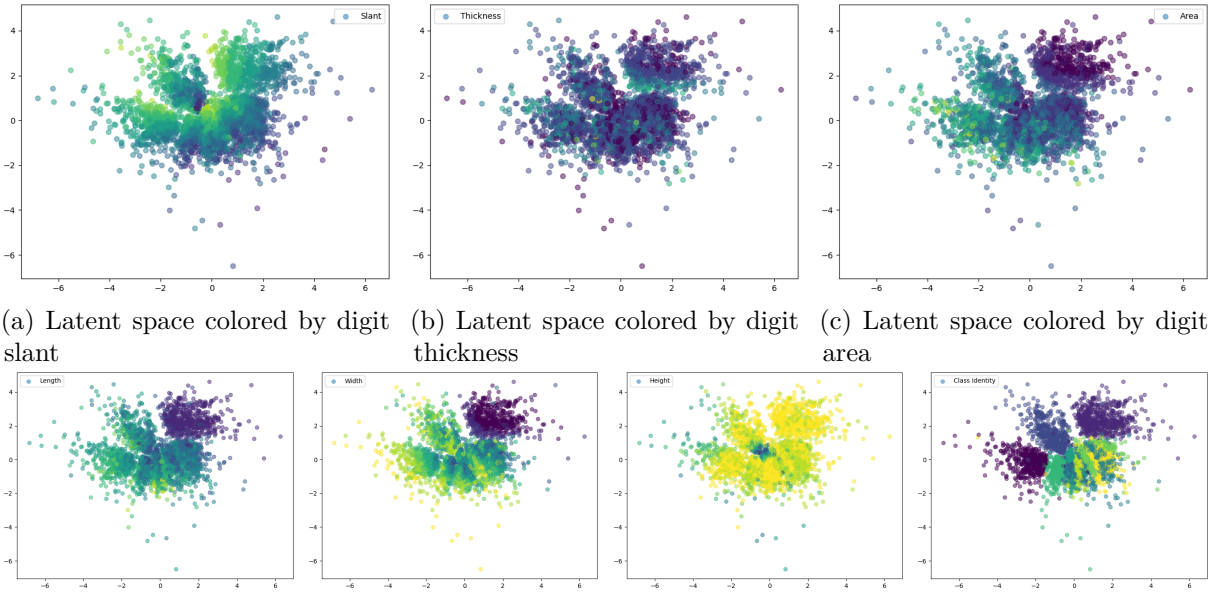
4.4.1 Latent Space Embeddings

Mnist Figures 29 and 30 show the latent spaces of VAE and VLAE trained on the Mnist dataset (Mnist-VAE and Mnist-VLAE, see Section 3.4)[36]. For Mnist-VAE, there is only one latent space, while there are three for Mnist-VLAE. The embeddings are colored by different factors of variation, employing information from Morpho-Mnist (see Section 3.3.4) and the digit identity information provided by Mnist itself.

As discussed in Section 2.5.3, the VLAE aims at learning "hierarchical disentangled representation" [75]. According to Zhao, Song, and Ermon [75], the lower embedding layers of such a model trained on Mnist (see Section 3.3.3) encode features such as stroke width, digit width, and digit tilt, whereas the highest layer should mainly learn high-level features like digit identity.

Incorporating the additional labels provided by Morpho-Mnist, we can analyze to what extent the lower layers learn which morphological features of MNIST (RQ4, see Table 1). The

[36] Plots for the other models can be found in Appendix E.1.



(a) Latent space colored by digit slant

(b) Latent space colored by digit thickness

(c) Latent space colored by digit area

(d) Latent space colored by digit length

(e) Latent space colored by digit width

(f) Latent space colored by digit height

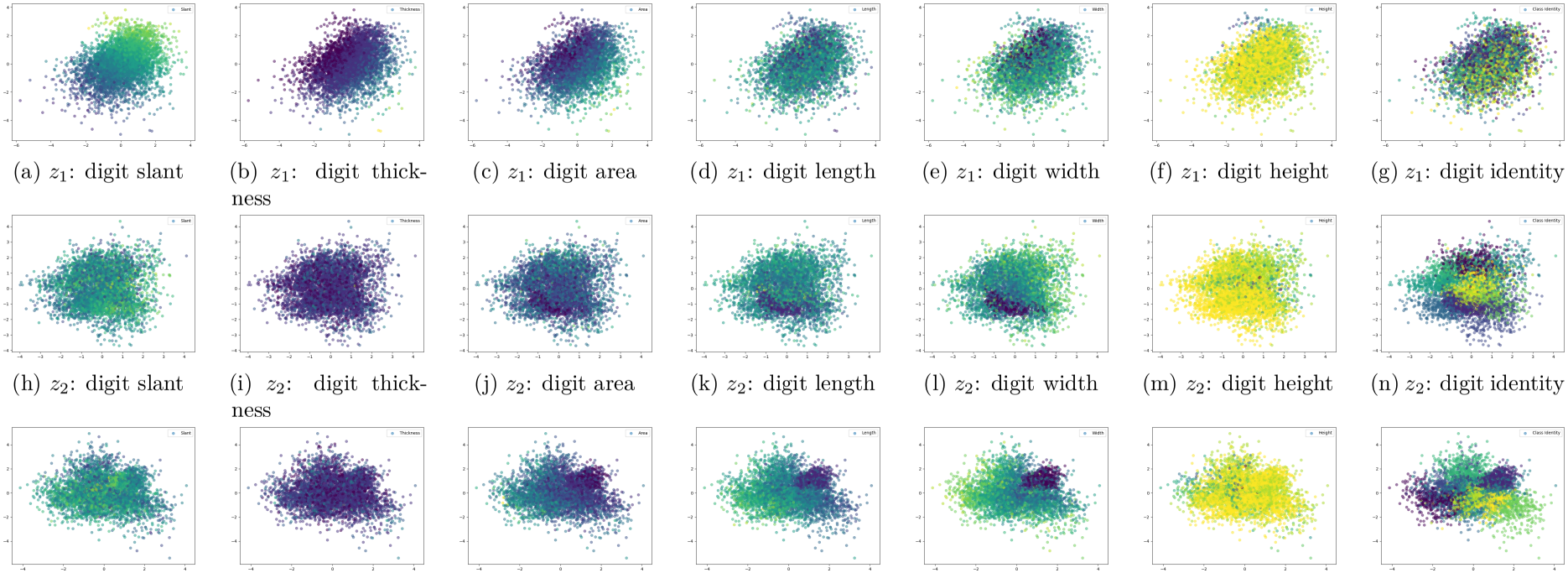
(g) Latent space colored by digit identity

Figure 29: Latent space colored by different means of MNIST-VAE

morphological attributes, however, are not equally distributed for all digits. The mean stroke length and the digit width, for example, have a low mean value for the digit "1" (see Figure 15). Therefore, it should be possible to almost uniquely identify digit "1" by just considering the stroke length or the digit width. Other attributes, such as stroke thickness, digit slant, and digit height, are more evenly distributed (see Figure 15).

Figure 30a shows the embedding layer $\boldsymbol{z}_1$ colored by digit slant. Figure 15 shows that the mean of the attribute *slant* is quite evenly distributed. The color gradient in Figure 30a, therefore, indicates that the VLAE learns the morphological attribute instead of just showing the class identity, encoded using another morphological attribute that correlates with class identity.

For Figure 30e, the situation is different. The noticeable dark-purple cluster in the top left correlates with dark-purple points in Figure 30g that encode images with label "1". For digit width, however, "1" is an outlier (see Figure 15), and a small digit width, therefore, is a reliable indicator for a digit identity of "1". Nonetheless, overall digit identity does not seem to be encoded strongly by $\boldsymbol{z}_1$ (see Figure 30g).

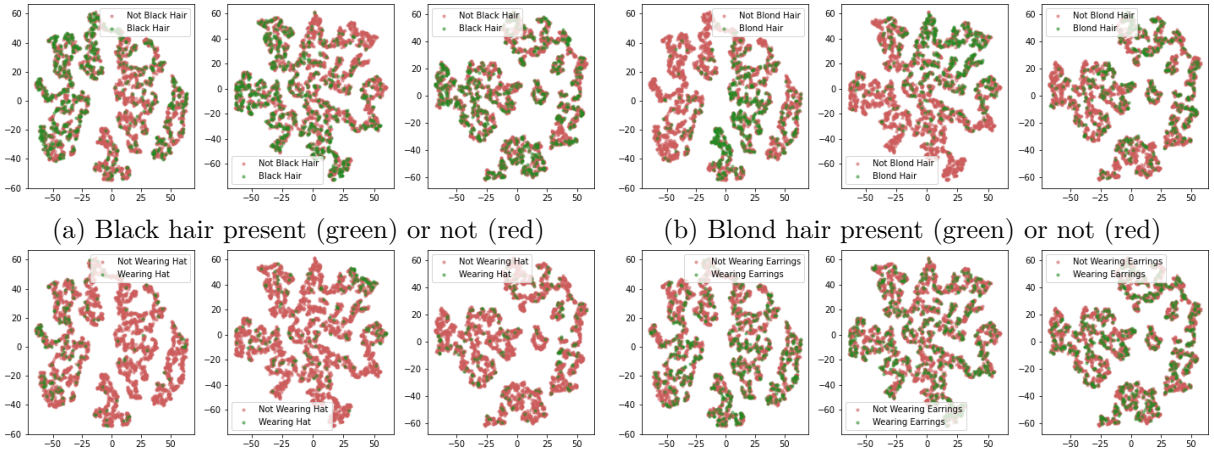


(a) z_1: digit slant (b) z_1: digit thickness (c) z_1: digit area (d) z_1: digit length (e) z_1: digit width (f) z_1: digit height (g) z_1: digit identity

(h) z_2: digit slant (i) z_2: digit thickness (j) z_2: digit area (k) z_2: digit length (l) z_2: digit width (m) z_2: digit height (n) z_2: digit identity

(o) z_3: digit slant (p) z_3: digit thickness (q) z_3: digit area (r) z_3: digit length (s) z_3: digit width (t) z_3: digit height (u) z_3: digit identity

Figure 30: Latent space colored by different means of MNIST-VLAE



(a) Black hair present (green) or not (red)

(b) Blond hair present (green) or not (red)

(c) Wearing a hat (green) or not (red)

(d) Wearing earrings (green) or not (red)

Figure 32: t-distributed stochastic neighbor embedding (t-SNE)-reduced Latent Space of a VLAE trained on CelebA, colored by curated Factors of Variation (present or not present)

Digit identity seems to be learned in $\boldsymbol{z}_2$ and $\boldsymbol{z}_3$ (see Figures 30n and 30m). However, learning one factor of variation in two layers would be a contradiction to [75], postulating that different factors of variation a learned in different layers. Albeit $\boldsymbol{z}_2$ has some influence on digit identity, this effect is not nearly as prominent as for $\boldsymbol{z}_3$ even though the clustering in Figure 30n seems to be almost as good as in Figure 30m. It seems that the digit identity learned in $\boldsymbol{z}_2$ only supports the model generation but has a far weaker influence than $\boldsymbol{z}_3$.

All in all, VLAE does not seem to learn entirely separable representations. However, the (less powerful) network below $\boldsymbol{z}_1$ indeed seems to learn lower-level representations that are employed when generating new images (see Figures 36a and 36b).

For VAE, digit identity defines the shape of the latent space (Figure 29g), leading to more prominent "main clusters." As VAE only has one latent space, it employs subspaces to learn the lower-level factors of variation, such as slant within the main clusters (Figure 29a).

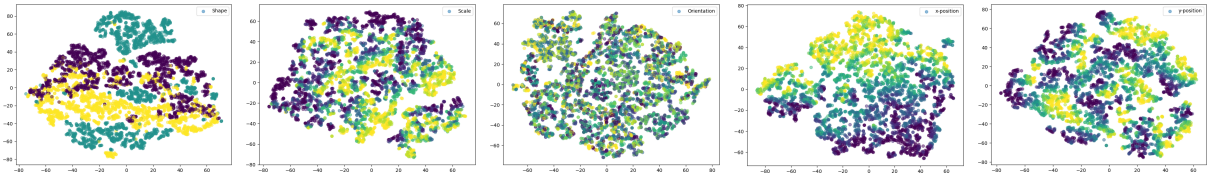
The VAE seems to learn an efficient encoding of most of the factors of variation as revealed by this kind of visualization even though it has a narrower information bottleneck with a two-dimensional latent space compared to the three two-dimensional latent spaces of VLAE.

CelebA The CelebA dataset provides binary labels for factors of variation. Figure 31 shows the distribution of the different features in the dataset. Figure 32 shows the t-SNE-reduced latent space of a CelebA-VLAE (see Section 3.4.2). A green dot in the latent space indicates that the feature is present for the data point, a red point indicates that it is not present[37].

The latent space plots allow to analyze which features are learned in which layers. For example, hair color is mainly learned in the first and second layers (see Figures 32a and 32b) because green and red dots build clusters within these layers. Other features do not seem to be learned at all (see Figure 32d).

Unlike dSprites and Morpho-MNIST, CelebA provides only binary attributes. This limits the latent space analysis because no gradients can be shown. Furthermore, the models learn factors of variation that are not present in the labels (see Section 4.4.2).

[37] The plots for all models and all factors of variation can be found in Appendix E.3.



(a) Latent space colored by object shape (b) Latent space colored by object scale (c) Latent space colored by object orientation (d) Latent space colored by object x-position (e) Latent space colored by object y-position

Figure 33: t-SNE-reduced latent space embeddings colored by different means of dSprites-VAE-dim6

dSprites The latent space embeddings for the dSprites dataset allow a detailed comparison of VAE and VLAE. Here, the dSprites-VAE-dim6 model (see Section 3.4.1) was given a six-dimensional embedding space, whereas the dSprites-VLAE-dim2 (see Section 3.4.2) model has three two-dimensional embedding spaces[38]. The VAE and VLAE approximately have the same model capacity under the assumption that VLAE uses lower embedding layers to learn lower-level features and higher embedding layers to only learn higher-level features and that the dataset can be split in this way. The VLAE latent spaces were chosen with two dimensions to allow for a direct visualization of the latent space without dimensionality reduction. As the VAE model employs a higher-dimensional latent space, the embeddings are visualized using t-SNE embeddings (see Figure 33).

One advantage of dSprites over MNIST is that all factors of variation are independent by design and appear equally often, allowing a more straightforward analysis.

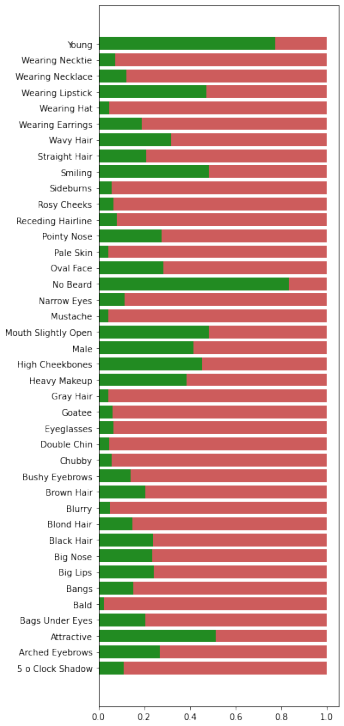


Figure 31: Distribution of the Binary Factors of Variation in the CelebA dataset. The green bar shows the fraction of the images where the attribute is present, the red bar where it is not present.

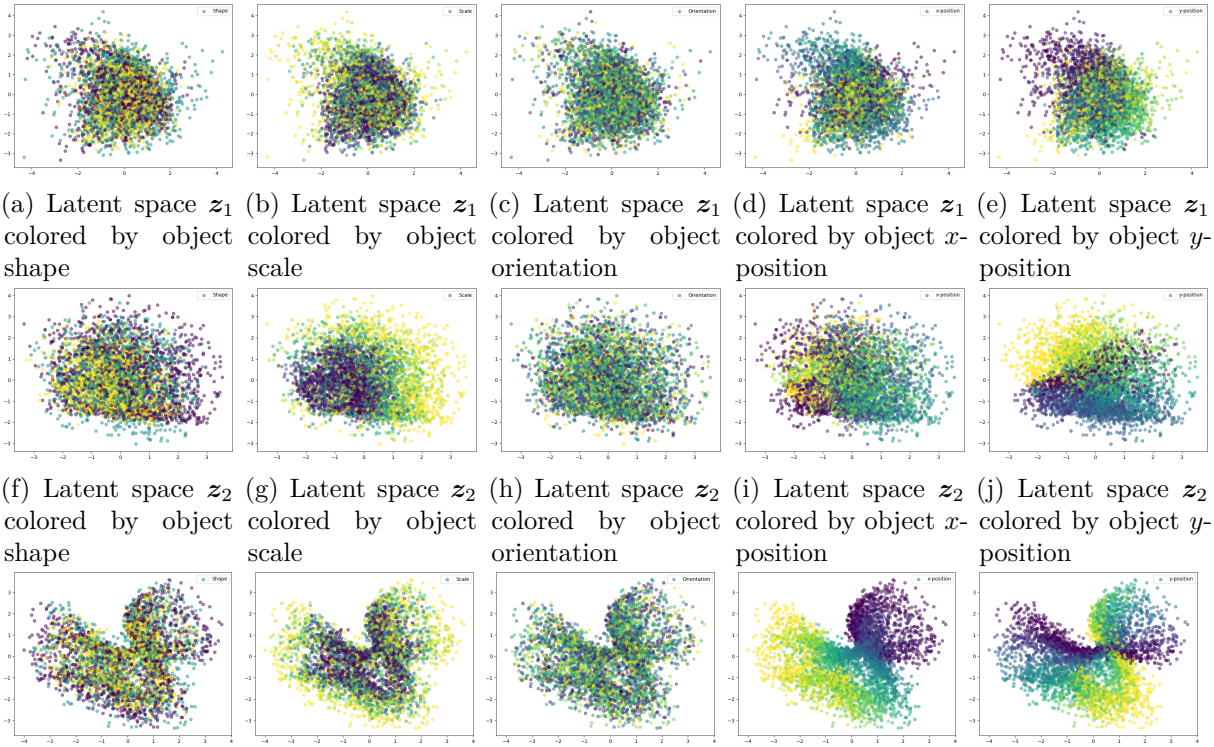
The VAE builds main clusters and seems to embed the lower-level features into main clusters. Especially the object scale seems to be a lower-level feature (see Figure 33b). Only object-orientation does not seem to be learned at all (see Figure 33d).

dsprites-VLAE-dim2 (see Figure 34) is worse compared to dSprites-VAE-dim6. The model learns main clusters, especially for object position (see Figures 34n and 34o). Furthermore, the model learns the object scale (see Figure 34l and 34g). Apart from that, the model does not seem to learn other factors of variation. The representations are not particularly disentangled. Object position and scale are learned in all layers, but best in z_3.

After training 200 epochs, dSprites-VAE-dim6 achieved a loss of 21.638 (reconstruction loss: 6.375, KL-loss: 15.264). In contrast, dSprites-VLAE-dim2 achieved a loss of 47.860 (reconstruction loss: 32.813, KL-loss: 15.047).

All in all, using a VAE with a higher-dimensional space seems superior to the VLAE in terms

[38] Plots for the other models can be found in Appendix E.2.



(a) Latent space z_1 colored by object shape (b) Latent space z_1 colored by object scale (c) Latent space z_1 colored by object orientation (d) Latent space z_1 colored by object x-position (e) Latent space z_1 colored by object y-position

(f) Latent space z_2 colored by object shape (g) Latent space z_2 colored by object scale (h) Latent space z_2 colored by object orientation (i) Latent space z_2 colored by object x-position (j) Latent space z_2 colored by object y-position

(k) Latent space z_3 colored by object shape (l) Latent space z_3 colored by object scale (m) Latent space z_3 colored by object orientation (n) Latent space z_3 colored by object x-position (o) Latent space z_3 colored by object y-position

Figure 34: Latent space colored by different means of dSprites-VLAE-dim2

of encoding efficiency.

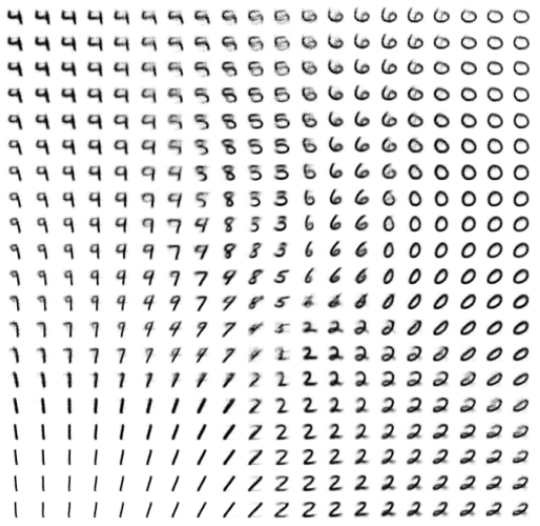
4.4.2 Latent Space Explorations

A latent space exploration is one way to see if and how a model learns a general representation of the dataset. Different from the analysis conducted in Section 4.4.1, *reconstructions* for defined positions in the latent space are analyzed. Especially for VLAEs, it also allows to examine if different attributes are learned independently in different layers (RQ5, see Table 1).

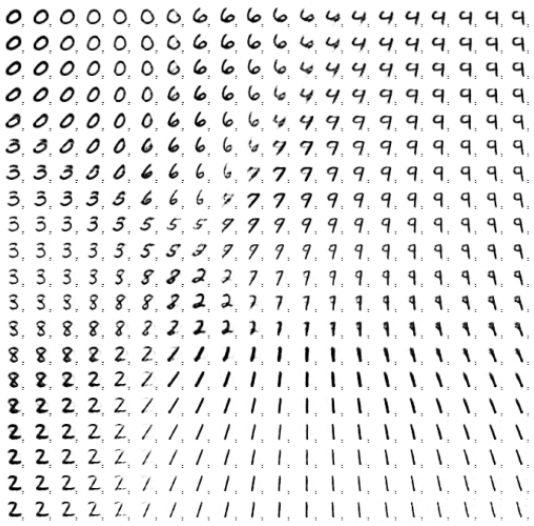
Mnist Figures 35 and 36 show the latent space traversal of Mnist-VAE (see Section 3.4.1), Mnist-VAE-generative adversarial network (GAN) (see Section 3.4.3), Mnist-VLAE (see Section 3.4.2), and Mnist-VLAE-GAN (see Section 3.4.4). For the models with only one latent layer (Mnist-VAE and Mnist-VAE-GAN), the plot is generated by traversing the latent space in equal steps from $z_i = -3$ to $z_i = 3$ in both dimensions. For the hierarchical models (Mnist-VLAE and Mnist-VLAE-GAN), one $\boldsymbol{z}$-layer was fixed and traversed in the same way. The values of the other $\boldsymbol{z}$-layers were obtained by sampling from a uniform distribution over $[-3; 3]$.

First, most models properly employ the latent space: the traversal shows almost no non-representative generated images, with a few exceptions for Mnist-VLAE and Mnist-VLAE-GAN. Noteworthy, these non-representative images mainly occur on the borders where the variance is high. The models were trained such that the latent space is standard normal, the latent space traversal, however, goes from -3 to 3, in areas with low probability density. In such areas, unrepresentative generations are no model failure.

Morpho-Mnist allows for another kind of analysis: By incorporating information of lower factors of variation, latent space trajectories for particular factors of variation can be calculated.



(a) VAE latent space exploration from $z_i = -3$ to $z_i = 3$ in both z dimensions



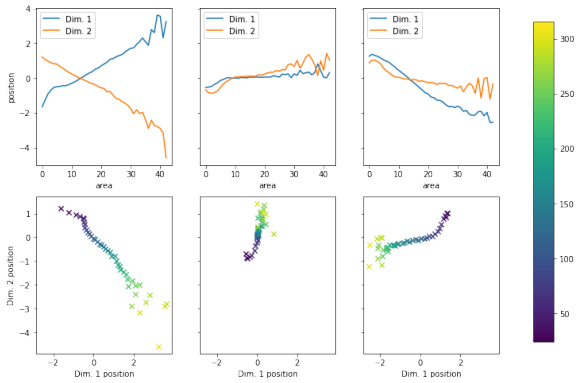
(b) VAE-GAN latent space exploration from $z_i = -3$ to $z_i = 3$ in both z dimensions

Figure 35: Latent space exploration for VAE models on MNIST. The original images have been inverted for the purpose of this figure.

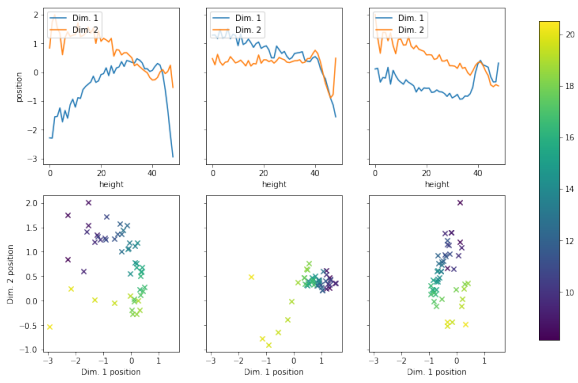
(a) VLAE latent space exploration from $z_i = -3$ to $z_i = -3$ in both dimensions for the respective z-dimension. The other z dimensions are sampled uniformly over $[-3; 3]$

(b) VLAE-GAN latent space exploration from $z_i = -3$ to $z_i = 3$ in both dimensions for the respective z-dimension. The other z dimensions are sampled uniformly over $[-3; 3]$

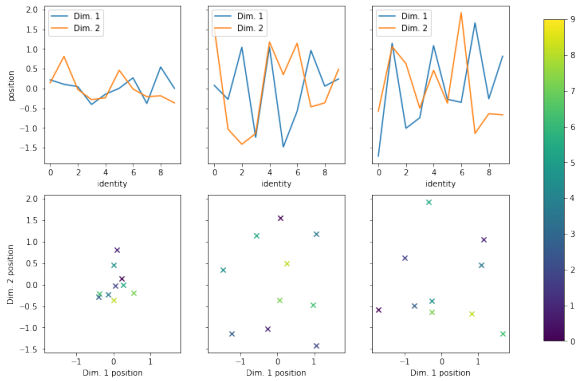
Figure 36: Latent space exploration for VLAE models on MNIST. The original images have been inverted for the purpose of this figure.



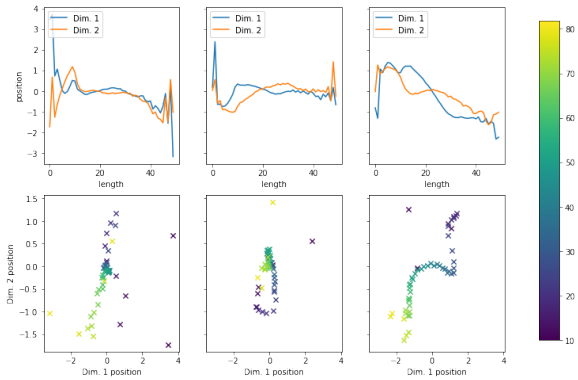
(a) area



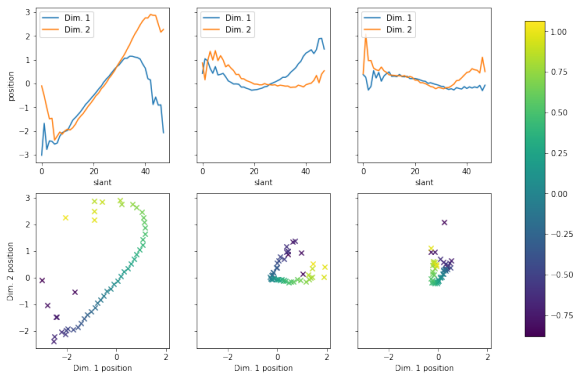
(b) height



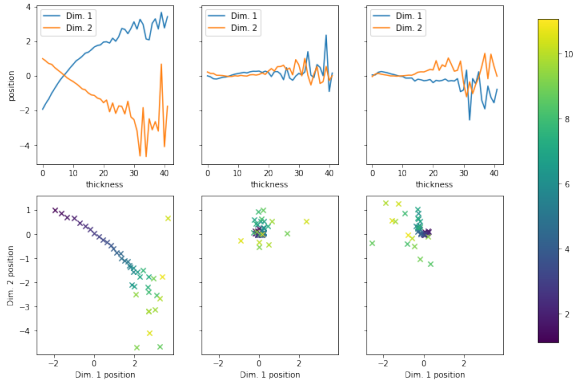
(c) identity



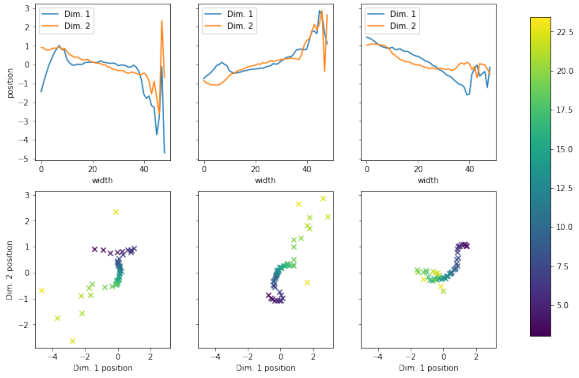
(d) length



(e) slant



(f) thickness



(g) width

Figure 37: Mean latent space values for MNIST-VLAE when fixing different factors of variation from Morpho-MNIST

Consider Figure 37[39]. Each sub-figure shows the latent space values for MNIST images, predicted by the MNIST-VLAE when fixing one Morpho-MNIST attribute and averaging over the others. The value range of each Morpho-MNIST attribute was divided into 50 evenly-sized bins, and the MNIST images were assigned to these bins based on the value of the corresponding attribute. Single bins can contain no values if no MNIST image has an attribute value in the corresponding range. The $\boldsymbol{z}$-predictions within one bin are averaged. Each column in a sub-figure in Figure 37 corresponds to one layer of VLAE (the first column corresponds to the first layer, etc.). The second row in each sub-figure shows the mean latent space position for each bin. The bins are ordered, increasing color values correspond to increasing attribute values as indicated by the respective color bar. The first row shows the latent space position for each dimension separately. The x-axis corresponds to the ordered bins.

If one layer does not learn a particular attribute, its value should not change much while the attribute values are varied. For example, this can be observed in Figure 37f on layers two and three. Even though the latent space values of layers two and three are non-stationary for high thickness values, this can be attributed to the few images of a high thickness (see Figure 15). Averaging over latent space positions if images with high thickness is less precise since only few images contribute to the mean value on the "high-thickness-bins". This can also be observed for layer one in Figure 37f. Apart from this explainable non-stationarity, only layer one seems to encode thickness, as it is the only layer showing a trend for this attribute. This is supported by Figure 36a, where thickness increases along the trajectory in Figure 37f. Another example is the slant (Figure 37e). The slant-trajectory is almost orthogonal to the thickness-trajectory (see Figure 36a).

However, many attributes do not seem to be learned in one layer. For example, the values of layers two and three are highly non-stationary for digit identity (Figure 37c). This indicates that digit identity is jointly learned in both layers. Figure 36a supports this. Traversals in both layers seem to influence digit identity, but there is still much variation within subregions. The same holds for other attributes (*area*, *height*, *length*, and *width*).

In conclusion, MNIST-VLAE succeeds in learning separable representations in some cases but also fails in other cases. A low variation of values within a layer when changing an individual factor of variation indicates that this layer does not learn this factor of variation. Although the layer could, inherently, learn an attribute and be overruled by the higher layers during reconstructions, this does not seem to be the case. The VLAE, in this regard, uses the latent space efficiently, as observed by jointly studying Figure 37 and Figure 36a.

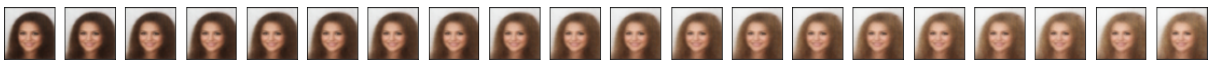
CelebA Figure 38 shows the latent space exploration of CelebA-VLAE-GAN model (see Section 3.4.4). The model was generated by evenly interpolating in $[-3; 3]$ in the respective layer and by sampling from a uniform distribution over $[-3; 3]$ for the other layers. Similar to MNIST, the model learns different factors of variation on different layers. The first layer mainly learns skin color, layer two hair color, and layer three pose and background color. Importantly, the model learns only a few factors of variation due to the small latent space dimensionality.

As discussed in Section 2.5.3, *feature consistency* is an important property of VAEs. It was empirically evaluated if the VAE models have the same property. Figure 39 shows the transition between black and blond hair, and female and male for a VAE trained on ImageNet.

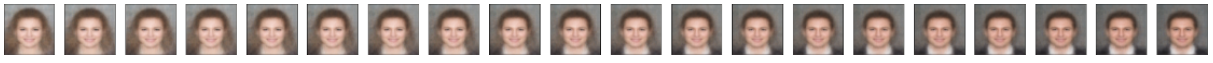
To generate the plots, the mean vectors of the different factors of variation (*male*, *black hair*, etc.) were obtained by predicting the posterior for corresponding images from CelebA. Then, for Figure 39a, a random latent vector $\boldsymbol{v}_1$ with "black hair" was chosen from the dataset. The transition was then generated by choosing different values $\alpha \in [0, 2]$ in the operation

[39] The corresponding figures for the other models can be found in Appendix E

Figure 38: Latent space exploration of CelebA-VLAE-GAN. Each of the three columns shows the exploration for the corresponding layer. Layer one mainly encodes skin color (gradient from darker to brighter skin color from top left to bottom right). Layer two mainly encodes hair color (gradient from darker to brighter hair color from left to right). Layer three mainly encodes pose and background color (top left and bottom right: right-oriented, top right: left oriented, center: more colorful background colors).



(a) Black to Blond Hair



(b) Female to Male

Figure 39: Interpolating between latent factors of variation in a VAE latent space, trained on CelebA

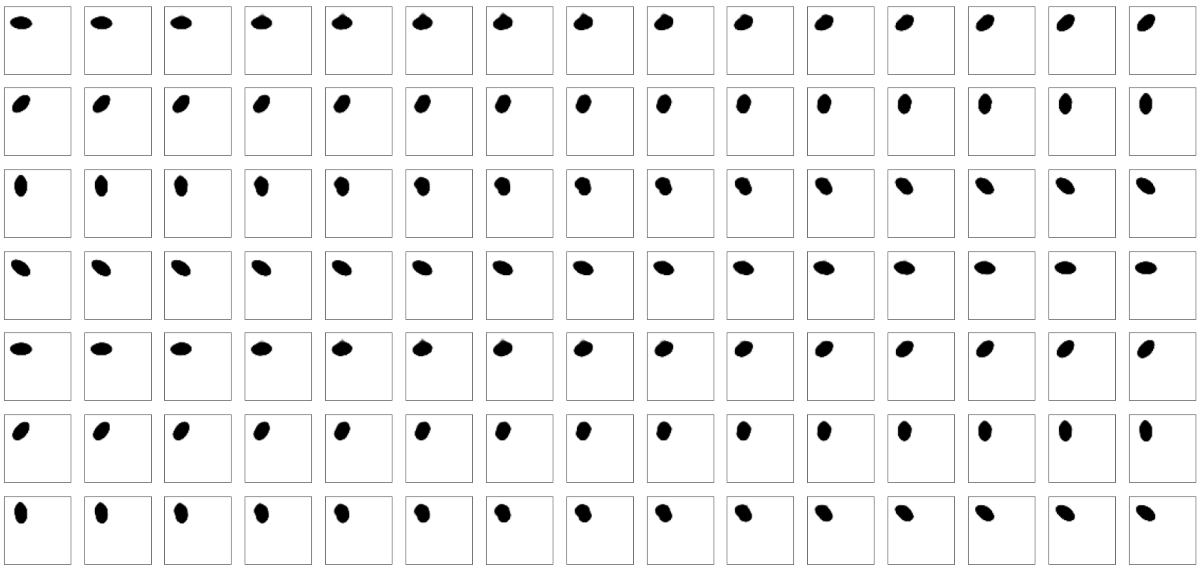
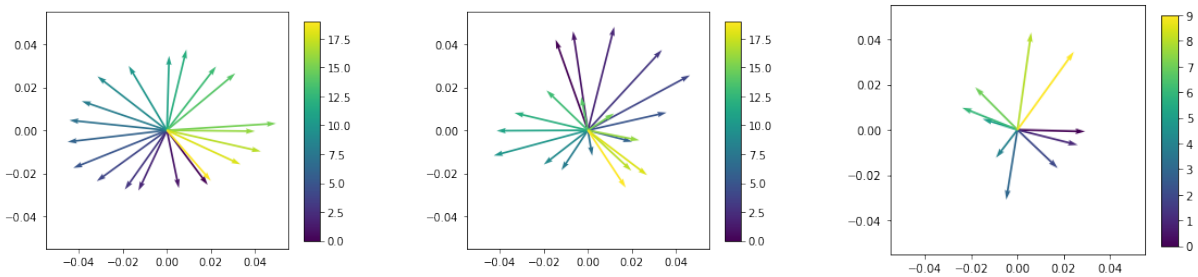


Figure 40: Latent spaces traversal between different rotation values for 10,000-VAE on the dsprites dataset



(a) *Ellipse* - only $\frac{1}{2}$ rotation is shown

(b) *Heart* - only $\frac{1}{2}$ rotation is shown

(c) *Square* - only $\frac{1}{4}$ rotation is shown

Figure 41: PCA-transformed latent space positions of different dsprites shapes with a fixed position, averaged over scales and a 10,000-VAE where only rotation is changed between objects. Increasing color values correspond to an increase in rotation.

$\boldsymbol{v}_1 - \boldsymbol{v}(\text{black hair}) + \alpha \boldsymbol{v}(\text{blond hair})$. For Figure 39b, a random image was generated by sampling $\boldsymbol{v}_2 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The transition was then generated by $\boldsymbol{v}_2 + \alpha \boldsymbol{v}(\text{blond hair})$ with $\alpha \in [0, 4]$.

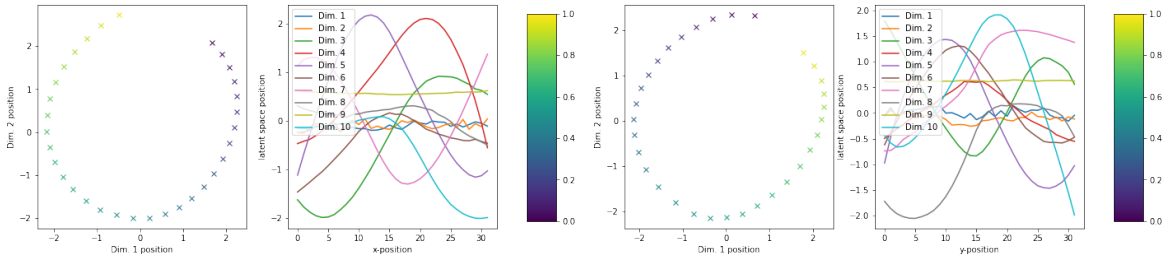
Even though larger values of α are required to obtain meaningful translations, the latent space has the discussed semantic property. The same holds for CelebA-VAE-GAN.

dsprites Section 4.3 discusses the latent space of dSprites-VAEs (see Section 3.4.3). One conclusion is that VAEs on dsprites do not learn a smooth transition of different scales. A transition between different shapes, however, is possible.

Figure 40 shows the latent space exploration between different rotation values of 10,000-VAE[40] with a reconstruction term weight of 10,000 on the dsprites dataset (the plots for the other models can be found in Appendix F). The 10,000-VAE learns a transition between different rotation values. This also holds for the other models (7,500-, 6,250-, 5000-, 3750-VAE, see Appendix F). However, how are the different rotation angles represented in the latent space?

Figure 41 shows the latent space position of an ellipse with a fixed position, averaged over

[40] "dSprites-VAE", see Section 3.4.1



(a) Traversal of the reduced latent space for different x-positions.

(b) Traversal of the reduced latent space for different y-positions.

Figure 42: Latent space of 10,000-VAE trained on dsprites. Different values are either for different x-positions or for different y-position. The other position is fixed to 1.0. It is averaged over all other parameters.

Figure 43: Linear interpolation between *top-left* and *bottom-right* 7,500-VAE latent space representations on dsprites leads to areas of low probability density.

different scales for different rotations. For this plot, the ten-dimensional latent space was reduced to a two-dimensional using principal component analysis (PCA) on the vector values. Only half (*Ellipse* and *Heart*) or one quarter (*Ellipse*) of the rotations are shown (i.e., rotations in $[0; \pi]$). For *Ellipse*, rotations in $[\pi; 2\pi]$ fill the circle a second time because an ellipse is isomorphic for a rotation by π. A square is isomorphic for rotations by multiples of $\frac{\pi}{2}$. For *Heart*, the model also fill the circle for a rotation by $< \pi$ even though this shape has no isomorphism for rotations $< 2\pi$. The reason for this behavior is unknown but it is assumed to be caused by the isomorphisms of the other shapes.

The behavior does not change for the other models (7,500-, 6,250-, 5000-, 3750-VAE) or different shapes. Rotations are learned naturally; A circle in the (reduced) latent space corresponds to a rotation of the object.

A similar behavior can be observed for the position.

The left sides of Figures 42a and 42b show the path in a reduced latent space[41]. Each arrow is the difference between two successive x-, or y-positions, mapped into the two-dimensional space. The path is curved, which is surprising since linear interpolations in the latent space are known to be successful for natural attributes such as hair color ([56] and CelebA in this section). Linear interpolation for the position, however, leads to regions of low probability density (see Figure 43). The right sides of Figures 42a and 42b show the values for the different dimensions in the non-reduced latent space. Some values almost resemble a sine-curve. This behavior is qualitatively similar for the other models (7,500-, 6,250-, 5000-, 3750-VAE).

Like rotation, for which this observation has been reported previously [13], VAEs seem to encode position in a periodic manner.

Regarding the human visual system, a high-level model of the ventral stream should not capture object position, scale, or rotation at all. Capturing orientation and position of a whole object is instead a property of the dorsal stream. However, by definition of the loss function[42], VAEs are trained to encode these properties. One way to make VAEs agnostic of these factors is to

[41] Plots for the other models (7,500-, 6,250-, 5,000, 3750-VAE, and VAE-GAN) can be found in Appendix E.)

[42] This holds for the pixel-wise and the adversarial loss.

center objects in the picture as in the CelebA dataset. However, even for CelebA, the models learn head rotation as one factor of variation.

There is no reason to assume that VAEs are a better model of the ventral than of the dorsal stream. Moreover, VAEs seem to learn positional factors of variation differently from non-positional ones. For factors of variation such as "hair color" or "gender" (for the CelebA dataset), a simple linear traversal in the latent space is sufficient to interpolate between these factors. Positional factors of variation, however, seem to be learned differently. Here, a linear interpolation is misleading because a highly curved interpolation would be required.

This difference in handling positional and non-positional attributes could be related to the two-stream hypothesis [25] where the *where* and *what* are also treated differently (see Section 2.1.3). In any case, it has to be considered in the work with VAEs.

For dSprites-VLAE model, a similar analysis allows to identify which layers encode which factor of variation[43]. Figure 44 shows the $\boldsymbol{z}$ values of different layers and dimensions for different factors of variation of dSprites-VLAE. The third layer seems to encode most of the factors of variation: x-, and y-position, scale, and shape. Although less strongly, the first and second layers encode orientation (Figures 44a and 44b) even though less strongly. Therefore, the representation is not separated; the different layers do not independently encode different factors of variation, even though these are, by definition, independent in the dsprites dataset. Independence is further discussed in Section 4.5 using the example of MNIST.

The first and second layers of dSprites-VLAE jointly learn orientation, similarly to dSprites-VAE. The scale and the x- and y-positions are mainly learned in the third layer. Again, the latent space trajectories for the x- and y-positions show a curved path, even though less circular compared to dSprites-VAE. dSprites-VLAE has three four-dimensional latent spaces. However, the reconstructions, especially in terms of shape-reconstruction, are less precise compared to the ten-dimensional-latent-space dSprites-VAE. dSprites-VLAEs, therefore, do not seem to use the hierarchical latent spaces efficiently under all circumstances.

dSprites-VAE-GAN and dSprites-VLAE-GAN behave similarly for the considerations in this section and are therefore not discussed.

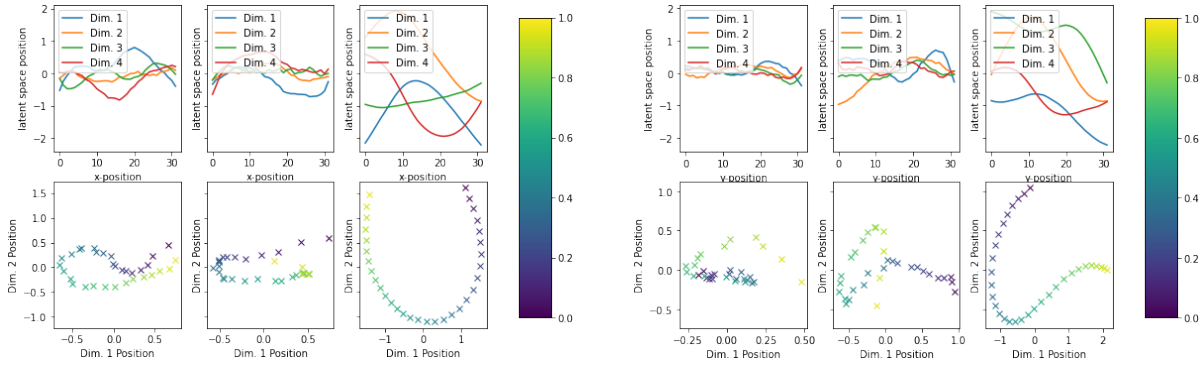
By analyzing the latent spaces of different models (VAE and VLAE models) on different datasets (dsprites and MNIST with Morpho-MNIST), it has been shown that VAE- and VLAE-models partially learn curved latent space trajectories for ordered latent space attributes (Figures 37, 42, and 44). It has empirically been validated that a simple linear interpolation between different orientation values in the latent space of a VAE trained on dsprites does not capture the learned "orientation-trajectory". The latent space is highly entangled for positional attributes, even if the reconstruction term is extremely low (see Appendix E). Hence, it is argued that linear interpolations sometimes do not capture the learned trajectory. Interpolating linearly can lead to wrong conclusions regarding the model performance.

4.5 Latent Space Separability on Generated Images

The VLAE learns embeddings on different levels. For MNIST, Zhao, Song, and Ermon [75] use three two-dimensional layers to learn image semantics of different granularity. They claim that their model can learn disentangled hierarchical features. Figure 36a shows reconstructions of this model when systematically exploring one dimension and randomly choosing the others. Evidently, the model can learn disentangled representations to some extent.

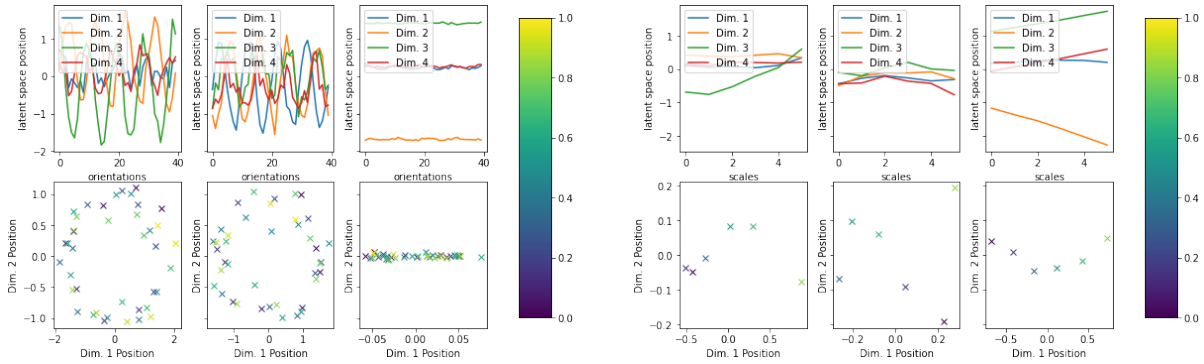
For instance, $\boldsymbol{z}_1$ seems to mainly encode the digit thickness, whereas $\boldsymbol{z}_3$ seems to encode digit identity. $\boldsymbol{z}_2$, however, also seems to influence the digit identity. It has been shown that these

[43] The plots for the corresponding dSprites-VLAE-GAN model can be found in Appendix E.



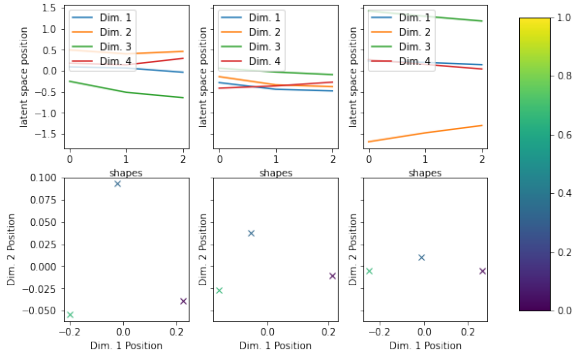
(a) Varying x-position

(b) Varying y-position



(c) Varying orientation

(d) Varying scale



(e) Varying shape

Figure 44: Values of different dimensions and layers in the VLAE latent space for different factor of variation values (first row in each subplot), and position in a PCA-reduced latent space (second row in each subplot). The model was trained on dsprites. The left column corresponds to the first embedding layer, the right one to the third. PCA was performed separately for each factor of variation and latent space layer. Different values correspond to different values for the respective factor of variation. For orientation and scale, the position is fixed to 0.0 in both directions, shape is fixed to *Square*. For x-and y-postition, the other position is fixed to 1.0 and shape is fixed to *Square*.

factors of variation are not learned independently in the latent space (see Section 4.4). However, a decoder could be able to ignore this redundancy and use only the non-redundant latent-space information. The following experiment investigates if decoders have this property (RQ6, see Table 1).

Algorithm 1 Generating Layer Representative Samples by Averaging Out Other Embedding Layers

1: **function** LayerRepresentativeSamples(numSamples,numApproximations)
2: $j \leftarrow 0$
3: $\mathcal{L} \leftarrow \varnothing$
4: **while** $i <$ numSamples **do**
5: $\boldsymbol{v} \leftarrow \boldsymbol{v} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
6: **for all** $j \in \{1, 2, 3\}$ **do**
7: $\boldsymbol{s}_j \leftarrow$ LayerRepresentativeSample($\boldsymbol{v}$, numApproximations, j)
8: **end for**
9: $\mathcal{L} \leftarrow \mathcal{L} \cup \{\{\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3\}\}$
10: **end while**
11: **return** $\mathcal{L}$
12: **end function**
13: **function** LayerRepresentativeSample(fixedDimensionValue, numApproximations, dimensionIndex)
14: $\mathcal{D} \leftarrow \{1, 2, 3\}$
15: $\alpha \leftarrow$ fixedDimensionValue
16: $\beta \leftarrow (D \setminus \text{dimensionIndex})_1$
17: $\gamma \leftarrow (D \setminus \text{dimensionIndex})_2$
18: $\boldsymbol{z}_\alpha \leftarrow \boldsymbol{a} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
19: $\mathcal{L} \leftarrow \varnothing$
20: $i \leftarrow 0$
21: **while** $i <$ numApproximations **do**
22: $\boldsymbol{z}_\beta^i \leftarrow \boldsymbol{b}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
23: $\boldsymbol{z}_\gamma^i \leftarrow \boldsymbol{c}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
24: $\mathcal{L} \leftarrow \mathcal{L} \cup \{$ VLAE-Decoder$(\boldsymbol{z}_\alpha, \boldsymbol{z}_\beta^j, \boldsymbol{z}_\gamma^j)$ $\}$
25: $i \leftarrow i + 1$
26: **end while**
27: **return** $\frac{1}{|\mathcal{L}|} \sum_j \mathcal{L}_j$
28: **end function**

Consider Algorithm 1. The function VLAE-Decoder calls the decoder of the VLAE, i.e., $p_\theta(\boldsymbol{x}|\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3)$. Calling the function LayerRepresentativeSamples returns an ordered set $\mathcal{L}$ of "layer representative samples." Each sample i contains three items, say $\boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \boldsymbol{x}_3^i$ that were created by fixing a value $\boldsymbol{v}$ in Line 5 of Algorithm 1. "Layer representative samples" means that for example $\boldsymbol{x}_1^i$ is approximately drawn from the marginal distribution

$$\boldsymbol{x}_1^i \sim p_\theta(\boldsymbol{v}|\boldsymbol{z}_1) = \int_{\boldsymbol{z}_2} \int_{\boldsymbol{z}_3} p_\theta(\boldsymbol{v}|\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3) d\boldsymbol{z}_2 d\boldsymbol{z}_3. \tag{47}$$

Now, if the embedding layers learn disentangled hierarchical representations, $p_\theta(\boldsymbol{x}|\boldsymbol{z}_1)$, $p_\theta(\boldsymbol{x}|\boldsymbol{z}_2)$, and $p_\theta(\boldsymbol{x}|\boldsymbol{z}_3)$ should be pairwise statistically independent:

$$p_\theta(\boldsymbol{x}|\boldsymbol{z}_i) \not\propto p_\theta(\boldsymbol{x}|\boldsymbol{z}_j) \quad \forall (i, j) : i \neq j. \tag{48}$$

However, this is not true. Choosing a value $\boldsymbol{z}_1 = \varphi$ such that $p_\theta(\boldsymbol{x}|\boldsymbol{z}_1 = \varphi)$ also leads to a high value of $p_\theta(\boldsymbol{x}|\boldsymbol{z}_2 = \varphi)$, leading to a violation of Equation 48.

4.5.1 Mnist

Consider Figure 45. It shows results of samples $(\boldsymbol{x}_1^1, \boldsymbol{x}_2^1), \ldots, (\boldsymbol{x}_1^{100}, \boldsymbol{x}_2^{100})$ that were generated by Algorithm 1. Thus, the parameter "numSamples" is chosen as 100, and the parameter "numApproximations" is chosen as 300. Each $\boldsymbol{x}_i^j$ is one generated MNIST image of size 28×28 pixels. Each box in Figure 45 corresponds to one of these pixels. The box index corresponds to the pixel in the MNIST image. The $\boldsymbol{x}$-values of dots in the same box then correspond to $\boldsymbol{x}_1^1|_{(3,1)}, \ldots, \boldsymbol{x}_1^{100}|_{(3,1)}$, i.e., the pixel intensities of one specific pixel (here: third row, first column (3,1)) over all 100 samples for $\boldsymbol{x}_1$, i.e., the sample generated by fixing $\boldsymbol{z}_1$. Analogously, the y-values of dots in the same box correspond to $\boldsymbol{x}_2^1|_{(3,1)}, \ldots, \boldsymbol{x}_2^{100}|_{(3,1)}$. Each dot corresponds to one fixed φ-value.

If changing the value of $\boldsymbol{z}_1$ is independent of changing the value of $\boldsymbol{z}_2$, the boxes should show no trend. This is true for outer boxes. Since they correspond to pixel values that always close to zero. Therefore, the values are in the bottom left corner, and no correlation can be observed. For center pixels, however, the plot shows something different. They show an interesting correlation pattern of negative and positive correlations.

Negative correlations for a pixel with index (i, j) indicate that $p_\theta(\boldsymbol{x}|_{(i,j)}|\boldsymbol{z}_1 = \varphi) \propto \frac{1}{p_\theta(\boldsymbol{x}|_{(i,j)}|\boldsymbol{z}_1 = \varphi)}$, i.e., choosing a value φ that leads to a high (i, j)-pixel intensity for the $\boldsymbol{z}_1$-representative sample leads to a low intensity of the same pixel of the corresponding $\boldsymbol{z}_1$-representative sample. This, however, is a violation of equation 48. The correlation only persists for individual pixels.

The behavior is similar for all VLAE models. All plots can be found in Appendix G.

Figure 46 shows a histogram of Pearson correlation coefficients for the pixel-wise intensities (compare Figure 45) The correlations are colored by whether the corrected p-values are < 0.05. Compared to the regular VAE-models, the GAN-models learn representations more independently in the different layers. This could have two reasons. First, the GAN-models could disregard the lower layers more strongly than the non-GAN models, effectively using only the third layer which is closest to the output layer. Second, the GAN-models could in fact learn more independent representations in terms of image generation. It is probably not the case that the GAN-models disregard lower layers, because they seem to incorporate lower-layer information for the generation of new images (compare Figures 36b and 75). Therefore, it is assumed that the discriminative loss leads the decoders to factoring out redundant information in the latent space layers to a higher degree.

4.5.2 dsprites

Consider Figure 47 showing the histograms of pixel-wise intensity correlations. The Figure supports the reasoning of the previous paragraph, i.e., that the GAN-generated images are more independent in terms of pixel-wise intensity correlation.

4.6 Pixel-wise Distribution of Generated Images

GANs (see Section 2.5.3) are trained by simultaneously training a *generator* to create new samples and a *discriminator* to discriminate between real and generated samples. VAEs are not forced in the same way to create indistinguishable samples. Instead, a reconstruction loss is used to force the reconstruction to be close to the real sample in terms of pixel-wise difference. Simultaneously, the KL-loss and the reparametrization trick force the model to place similar

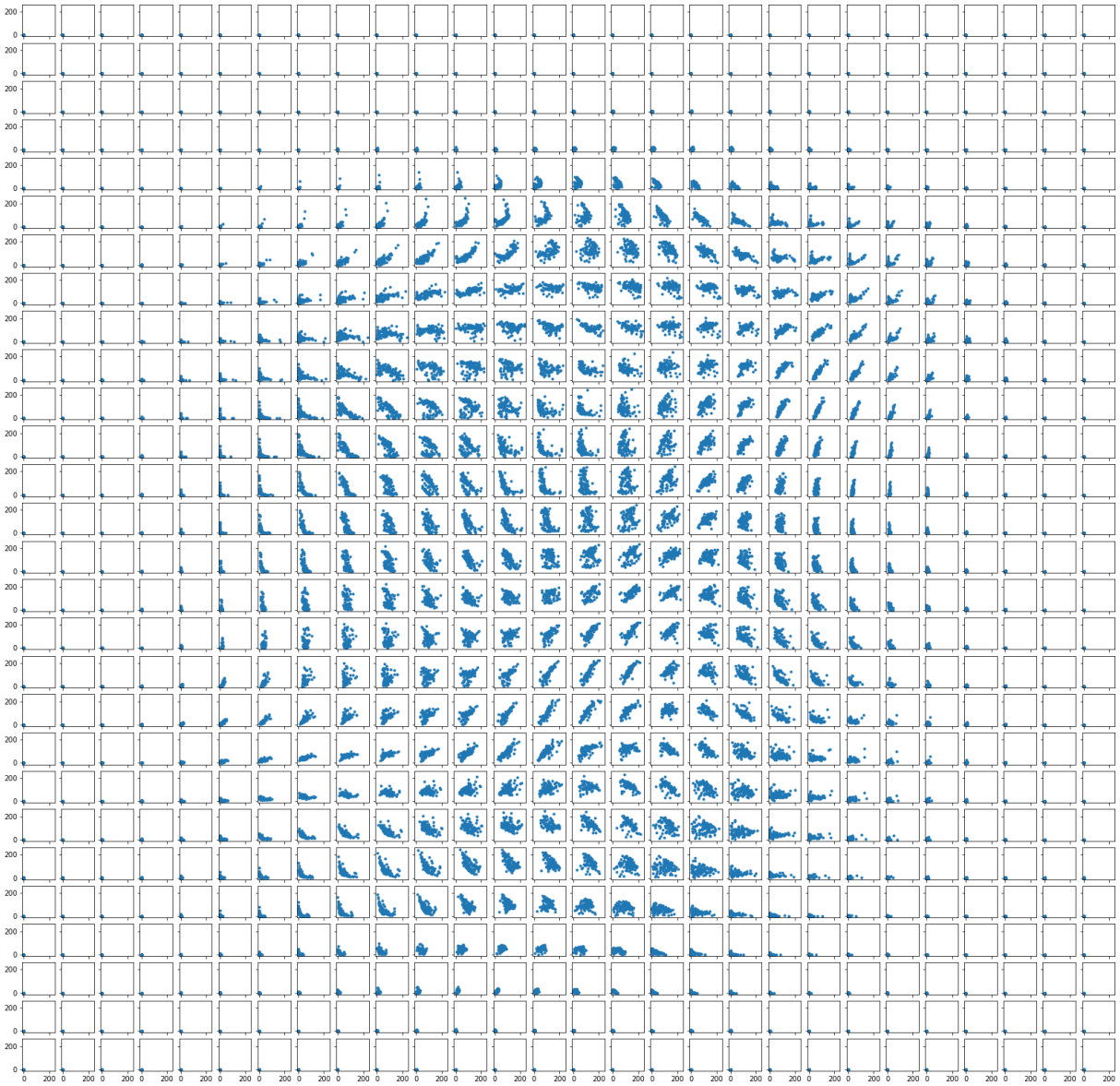
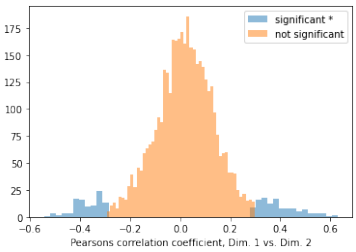
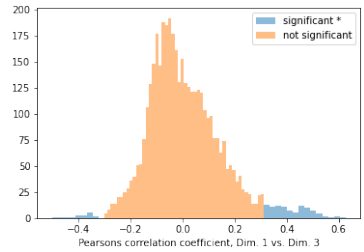


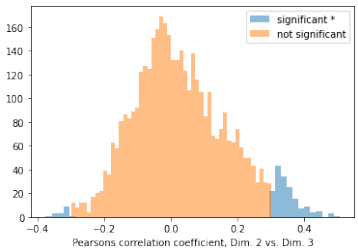
Figure 45: Correlation of pixel intensities when fixing $\boldsymbol{z}_1 = \boldsymbol{z}_2 = \varphi$ for MNIST-VLAE. Each box represents one of 28×28 MNIST pixels. The x-axis of each box encodes the mean pixel intensity of the $\boldsymbol{z}_1$-representative sample. The y-axis encodes the mean pixel intensity of the of the $\boldsymbol{z}_2$-representative sample. Dots within boxes belong to the same φ for both, $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$.



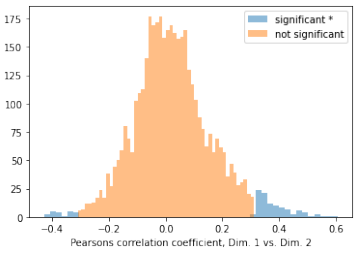
(a) MNIST-VLAE - Layer 1 vs. Layer 2



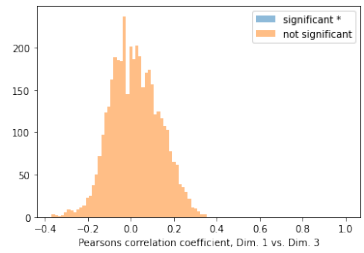
(b) MNIST-VLAE - Layer 1 vs. Layer 3



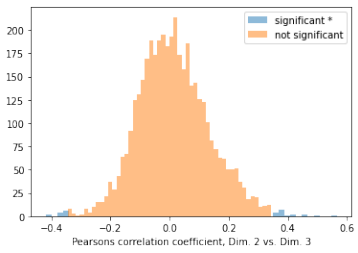
(c) MNIST-VLAE - Layer 2 vs. Layer 3



(d) MNIST-VLAE-GAN - Layer 1 vs. Layer 2

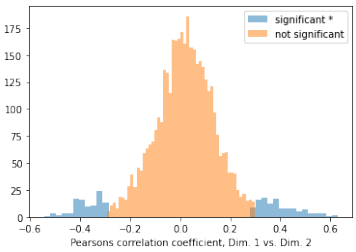


(e) MNIST-VLAE-GAN - Layer 1 vs. Layer 3

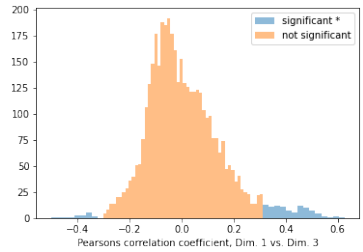


(f) MNIST-VLAE-GAN - Layer 2 vs. Layer 3

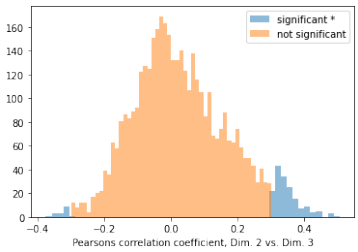
Figure 46: Histogram of correlations of pixel-wise intensities when fixing different pairs of dimensions for MNIST-VLAE and MNIST-VLAE-GAN. Each value in the histogram is the Pearson correlation coefficient for one pixel of a generated dsprites image (compare Figure 45 showing the correlations in the case of MNIST). The histogram is colored by whether the correlation is significant (H_0: The absolute correlation is equal to the absolute correlation of samples with zero correlation) at a 95% confidence level. The p-values have been corrected by the Benjamini/Hochberg method.



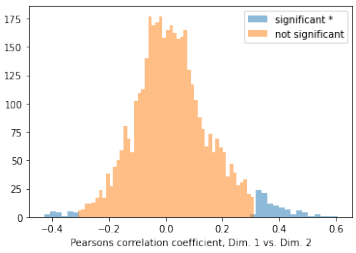
(a) dSprites-VLAE - Layer 1 vs. Layer 2



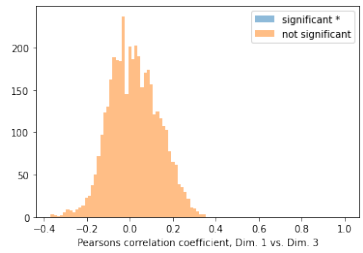
(b) dSprites-VLAE - Layer 1 vs. Layer 3



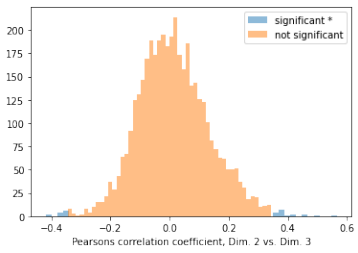
(c) dSprites-VLAE - Layer 2 vs. Layer 3



(d) dSprites-VLAE-GAN - Layer 1 vs. Layer 2



(e) dSprites-VLAE-GAN - Layer 1 vs. Layer 3



(f) dSprites-VLAE-GAN - Layer 2 vs. Layer 3

Figure 47: Histogram of correlations of pixel-wise intensities when fixing different pairs of dimensions for dSprites-VLAE and dSprites-VLAE-GAN. Each value in the histogram is the Pearson correlation coefficient for one pixel of a generated dsprites image (compare Figure 45 showing the correlations in the case of MNIST). The histogram is colored by whether the correlation is significant (H_0: The absolute correlation is equal to the absolute correlation of samples with zero correlation) at a 95% confidence level. The p-values have be corrected by the Benjamini/Hochberg method.

Model	p-value (mean)	p-value (sd)	p-value (skewness)	p-value (kurtosis)
VAE	< 0.001	< 0.001	< 0.001	< 0.001
VLAE	0.439	< 0.001	< 0.001	< 0.001
VAE-GAN	< 0.001	< 0.001	< 0.001	< 0.001
VLAE-GAN	< 0.001	< 0.001	0.023	0.227

Table 5: p-values of a Mann-Whitney U test. Each cell tests the hypothesis that the respective moments for the respective model are equal to the values for the MNIST test images. For each cell, the sample size was $2 \cdot 10,000$.

samples close to another in a continuous Gaussian embedding space. Therefore, drawing from the Gaussian embedding space should allow us to generate new samples similar to real samples. However, the question remains how indistinguishable these generated samples are from actual samples and whether VAE-models resemble the data distribution (RQ7 and RQ8, see Table 1).

Different statistical analyses were performed to address this question, revealing that generated samples can be correctly distinguished from actual samples.

4.6.1 Mnist

The following procedure was applied to generate samples for different models. One part of the VAE loss function (and of the other models) is the KL-term, forcing the models to match a standard multivariate normal distribution. The learned distribution, however, is not perfectly Gaussian because of the other loss function terms. Therefore, the models' encoders first was used to predict the mean values in z-space for the 10,000 validation images of the MNIST dataset. Subsequently, for each dimension, kernel density estimation (KDE) was performed to estimate the probability density function (PDF) of the latent space for MNIST-VAE and MNIST-VAE-GAN. Since the learned distribution, by definition, is covariance-free, the KDE was performed for each dimension separately. The estimated PDF was then used to generate 1,000 new images to perform the statistical analyses. Figure 48 shows the estimated KDE for MNIST-VLAE-GAN, plots for the other models can be found in Appendix H.1.

MNIST-VLAE and MNIST-VLAE-GAN cannot sample independently from the different latent spaces (see Section 4.5). For these models, the predicted mean values were directly used to reconstruct 10,000 images.

The MNIST test set of 10,000 images was compared to a 1,000 generated samples from MNIST-VAE, MNIST-VLAE, MNIST-VAE-GAN, and MNIST-VLAE-GAN according to the estimated posterior. First, the mean pixel values, i.e., the mean over all 28×28 pixel values, were compared (see Figure 49). The plot overlays the histograms of mean pixel values for the five conditions: MNIST, MNIST-VAE, MNIST-VAE-GAN, MNIST-VLAE, and MNIST-VLAE-GAN. The other plots in Figure 49 were created accordingly but for higher moments of the pixel value distributions.

Consider Figure 50 showing the same distribution but for the standard normal prior. The pixel-wise statistics are quite different compared to Figure 49 and resemble the true distribution less precisely. The aforementioned procedure is crucial when evaluating a model. Sampling from the prior leads to worse model performance in resembling the data distribution.

Figure 49 leads to the assumption that all models learn the pixel distributions to some extent. The overlap of the histograms is high for all models. Table 5 shows the results of a two-sided Mann-Whitney U test for the samples of moments of the pixel distributions. The results lead to two conclusions: (1) The VLAE-models capture the statistics of the pixel distribution better

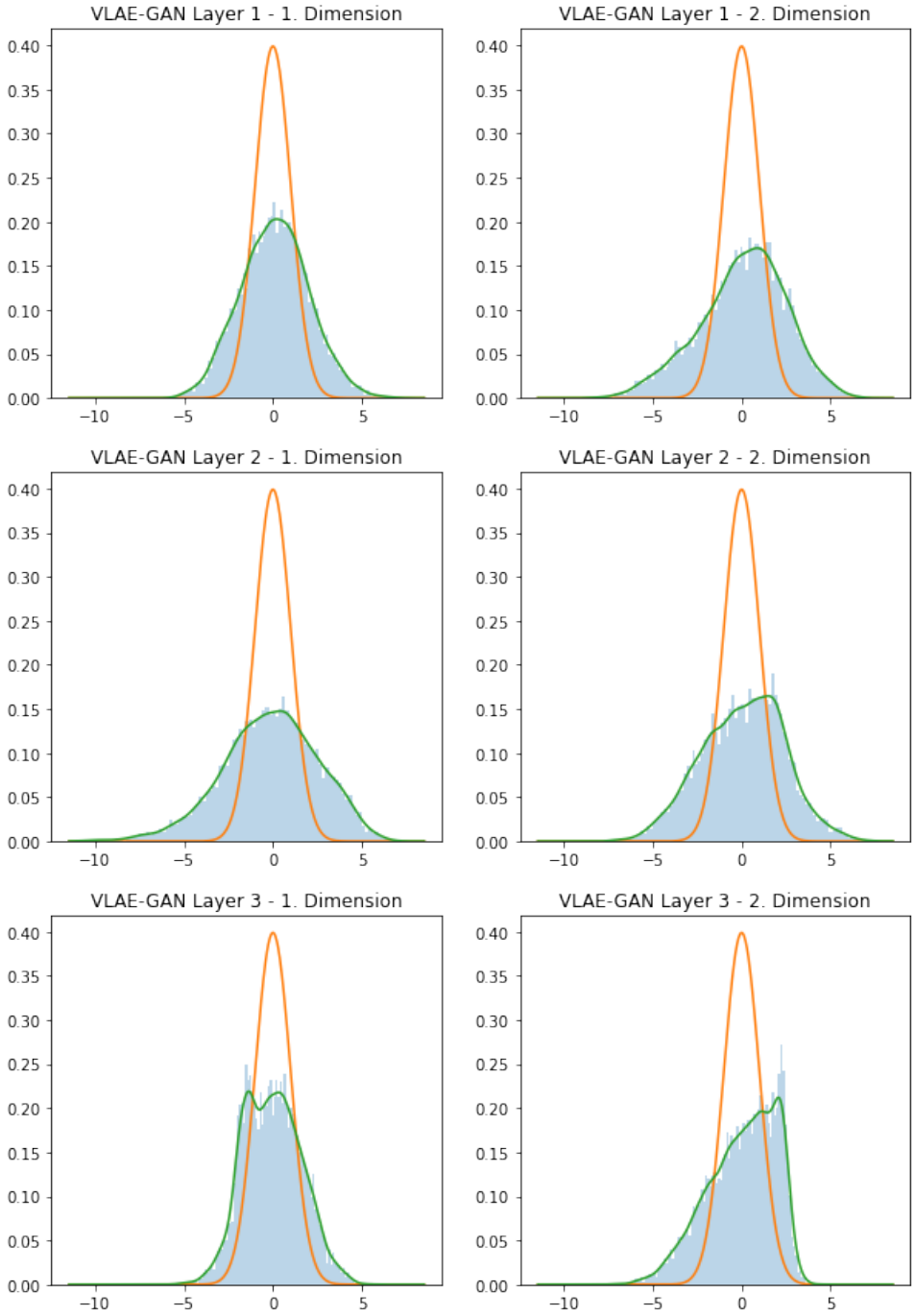
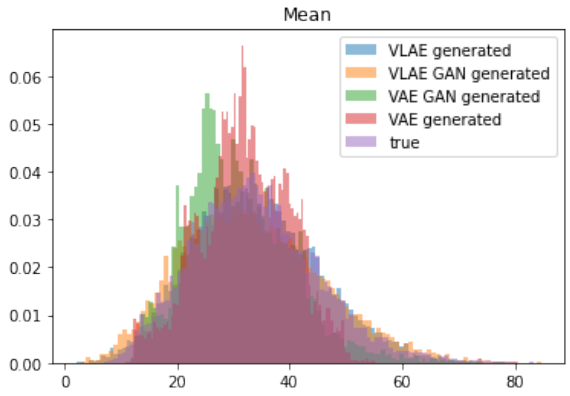
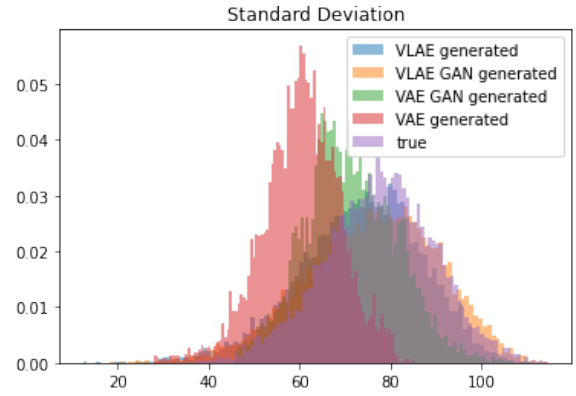


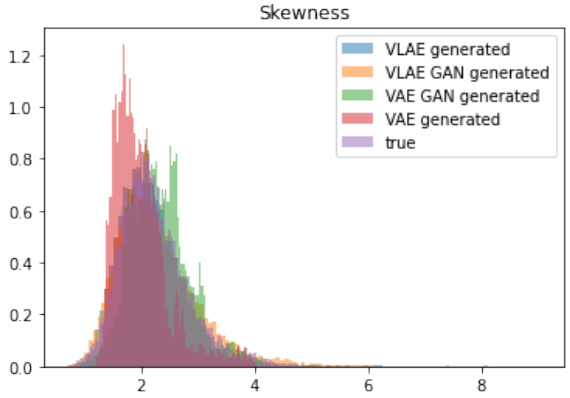
Figure 48: Histogram of mean z-values for different layers and dimensions of MNIST-VLAE-GAN (blue), the result of the KDE (green), and a standard normal distribution (orange). Additional plots can be found in Appendix H.



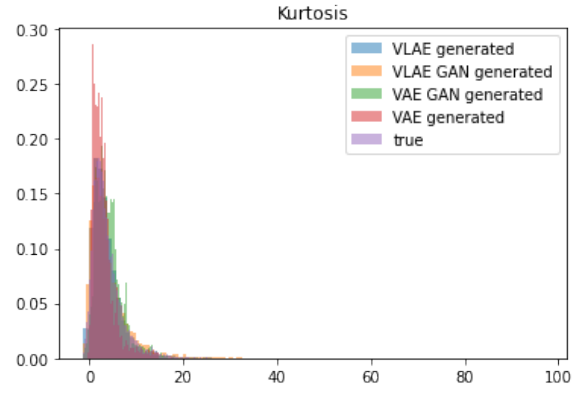
(a) Distribution of mean pixel intensities for different models



(b) Distribution of pixel intensity standard deviations for different models

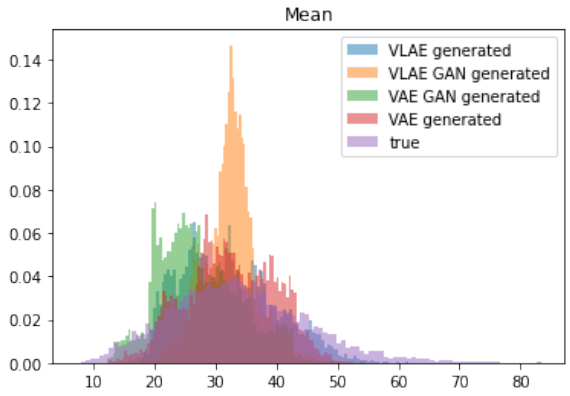


(c) Distribution of pixel intensity skewness for different models

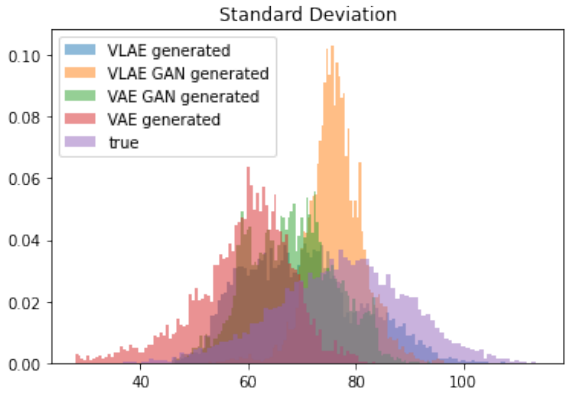


(d) Distribution of pixel intensity kurtosis for different models

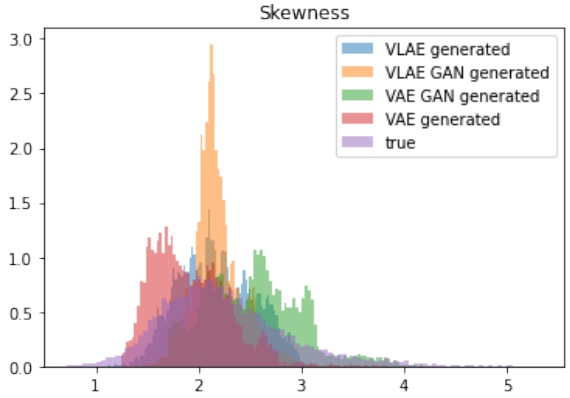
Figure 49: Pixel-wise distributions of different models and moments for the MNIST validation data set. Each sub-figure shows the distribution of the corresponding (normalized) moments. Each moment is calculated for the distribution of each pixel for multiple images. For MNIST, $28 \cdot 28$ moments are calulated.



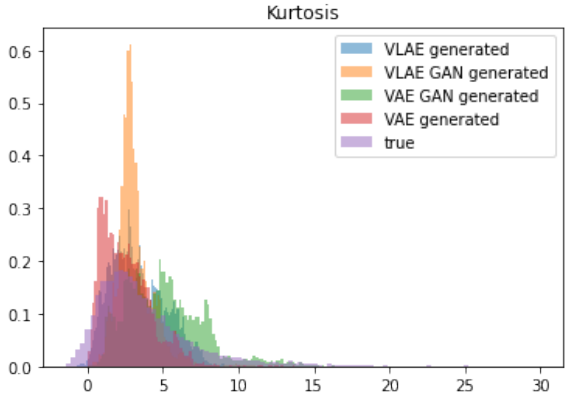
(a) Distribution of mean pixel intensities for different models



(b) Distribution of pixel intensity standard deviations for different models

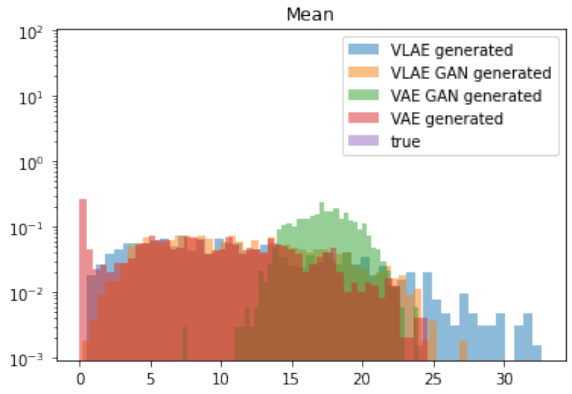


(c) Distribution of pixel intensity skewness for different models

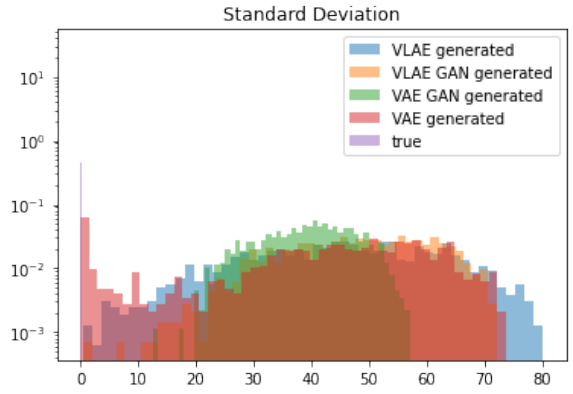


(d) Distribution of pixel intensity kurtosis for different models

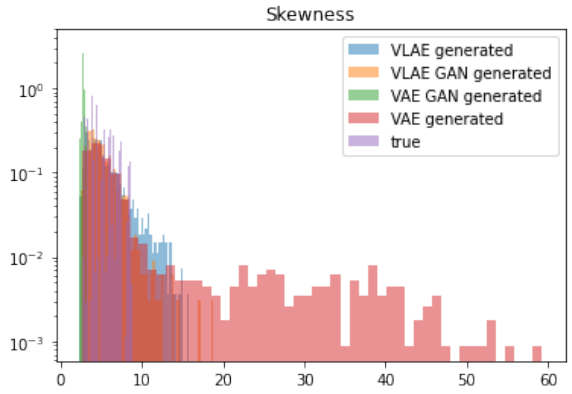
Figure 50: Pixel-wise distributions of different models and moments for the MNIST validation data set with a standard normal posterior. Each sub-figure shows the distribution of the corresponding (normalized) moments. Each moment is calculated for the distribution of each pixel for multiple images. For MNIST, $28 \cdot 28$ moments are calulated.



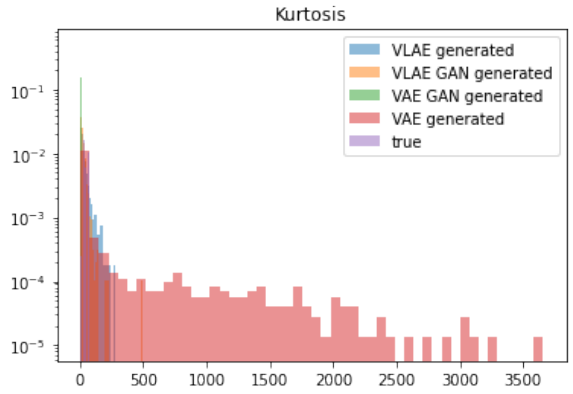
(a) Distribution of mean pixel intensities for different models



(b) Distribution of pixel intensity standard deviations for different models



(c) Distribution of pixel intensity skewness for different models



(d) Distribution of pixel intensity kurtosis for different models

Figure 51: Pixel-wise distributions of different models and moments for the dsprites validation data set. Each sub-figure shows the distribution of the corresponding (normalized) moments. Each moment is calculated for the distribution of each pixel for multiple images. For dSprites, $64 \cdot 64$ moments are calulated.

compared to the VAE-models. (2) None of the models captures the true pixel distribution (rejection of H_0: "The distributions are the same.").

To verify that none of the models generates indistinguishable samples, a discriminator network was trained to distinguish generated samples from true MNIST test images[44]. The discriminator network shows an accuracy of 1.0 for distinguishing for all models, i.e., it is perfectly able to predict generated from true samples for all models.

Comparing the pixel-wise statistics between generated (according to the KDE procedure explained above) and reconstructed samples furthermore showed that there is a significant difference ($p < 0.05$, two-sided Mann-Whitney U test) even though the difference between the $\boldsymbol{z}$ distribution of encoded validation images and $\boldsymbol{z}$s sampled from the estimated posterior is not significant ($p < 0.05$, two-sided Mann-Whitney U test). The reason for this is assumed to lie in samples for which the encoder predicts very low values $\log \sigma^2$ that have been observed during training. Enforcing a lower bound for $\log \sigma^2$ in the encoder could prevent this.

Model	p-value (mean)	p-value (sd)	p-value (skewness)	p-value (kurtosis)
VAE	< 0.001	< 0.001	< 0.001	< 0.001
VLAE	< 0.001	< 0.001	0.091	0.005
VAE-GAN	< 0.001	< 0.001	< 0.001	< 0.001
VLAE-GAN	< 0.001	< 0.001	0.005	0.061

Table 6: p-values of a Mann-Whitney U test. Each cell tests the hypothesis that the respective moments for the respective model are equal to the values for the dsprites test images. For each cell, the sample size was $2 \cdot 1,000$.

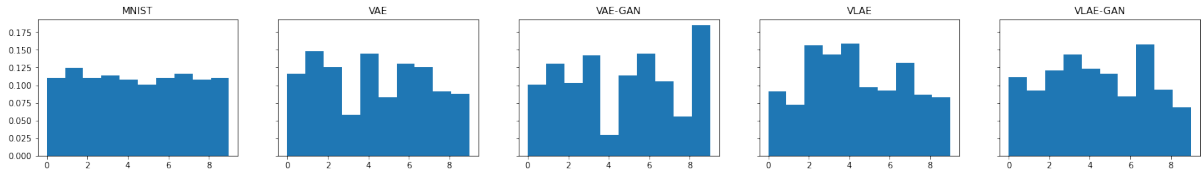


Figure 52: Distribution of generated class labels for different models.

4.6.2 dsprites

Figure 51 shows the pixel-wise distribution for the dsprites dataset. The VLAE-models (dSprites-VLAE and dSprites-VLAE-GAN) employ a four-dimensional, the VAE (dSprites-VAE and dSprites-VAE-GAN) a ten-dimensional latent space. Again, the VLAE-models better capture the true distribution. However, all distributions differ significantly from the true distribution (see Table 6). The KDE distributions can be found in Appendix H.2.

4.7 Class-Distribution of Generated Images

Do the VAE and VLAE models generate each number with the same probability as a numbers' probability in the training set as required by latent space disentanglement (see Section 2.6)? To answer this question (RQ2, see Table 1), the following procedure was applied.

First, let each model generate a large number of images by drawing the latent space variable(s) $\boldsymbol{z}$ from their estimated posterior (see Section 4.6 for details). Second, let a classifier classify each generated image. This procedure was applied to MNIST only. It is the only labeled dataset where the VAEs and VLAEs models have shown a promising performance.

Figure 52 shows the class-distribution of generated images. Even though it is sampled from the (estimated) posterior, the class distribution is quite uneven. This aligns with Figures 35 and 36, showing that the model maps some digits to very small subspaces. Why this happens, however, remains unclear. If there is less variation for a particular digit, i.e., the digit three is written very similarly in all cases, then all models should map threes to small subspaces. However, MNIST-VAE reproduces few threes whereas MNIST-VAE-GAN produces few fours. Overall, the VLAE models generate a more even distribution of generated digits.

Figure 53 shows the distribution of morphological attributes for generated digits. The Figure is created accordingly to Figure 15. The bottom row in Figure 15 is equivalent to the top row in Figure 53.

For most attributes, the VLAE models learn a smooth distribution that approximately resembles the true distribution. However, they fail in generating the same proportion of digits with a small

[44]The configuration of the discriminator network can be found in Appendix I. The network was trained using binary cross entropy for one epoch using the Adam optimizer with a learning rate of 0.01.

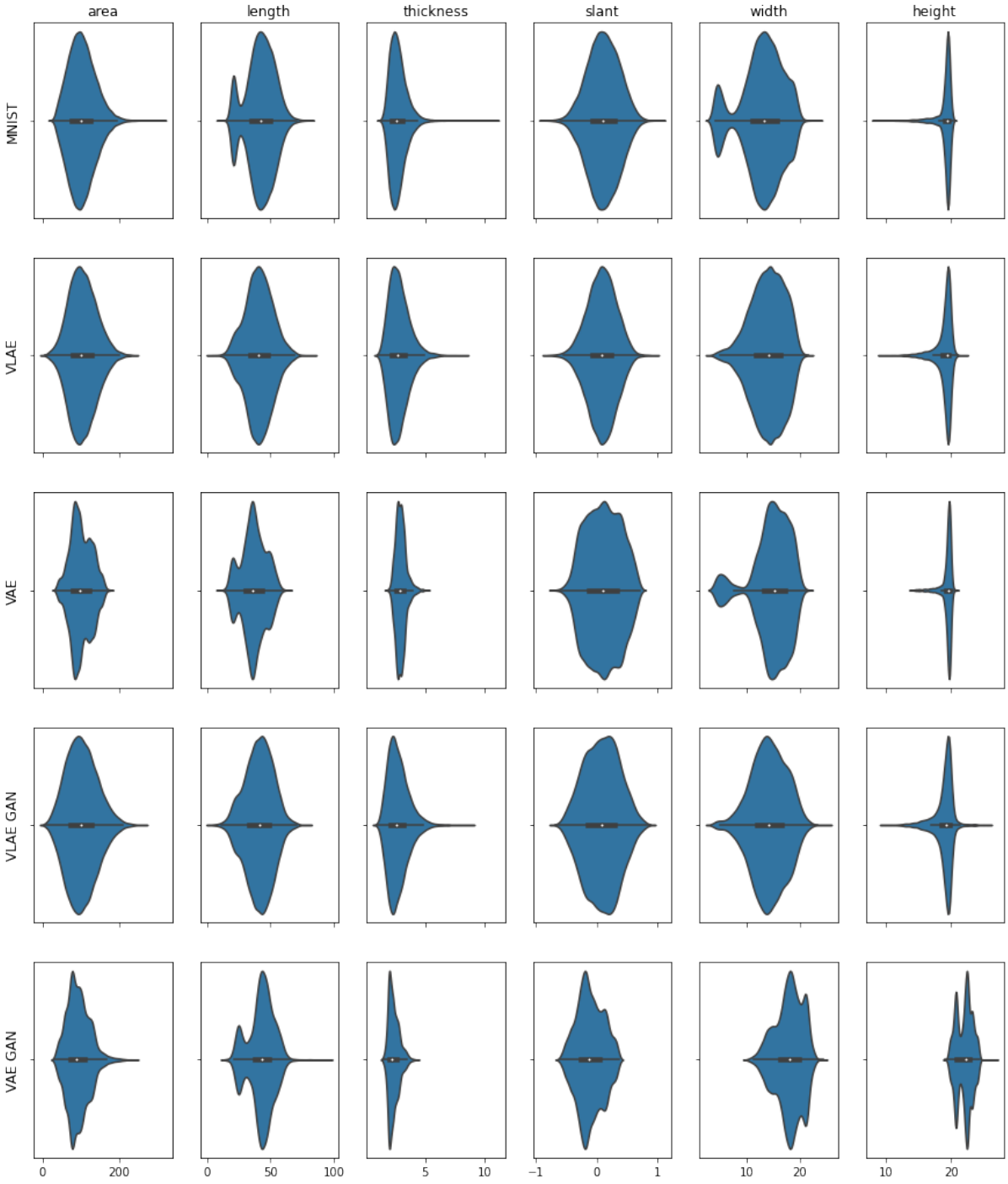
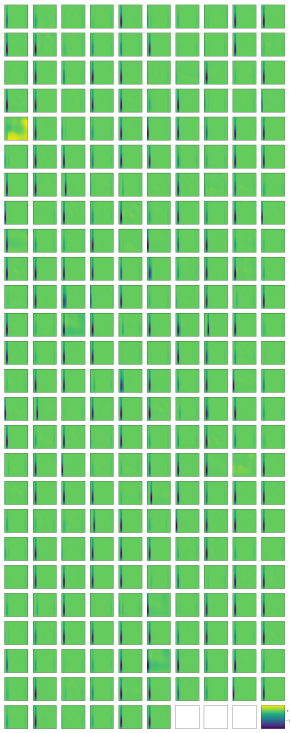
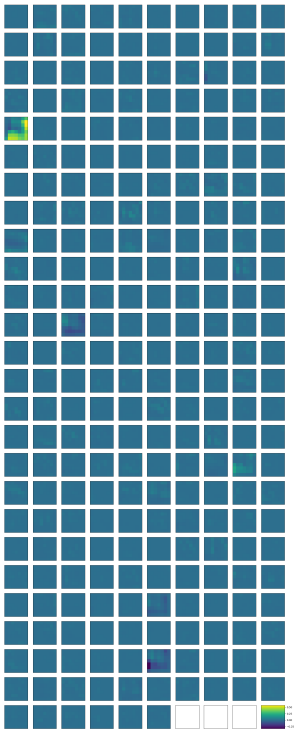


Figure 53: Distribution of Morpho-MNIST attributes for different models. The violin plots estimate the underlying distribution with KDE. Each violin plot shows the median (white point in the center). The thick black bar gives the range from first to the third quartile, then thin bars give the minimum and maximum excluding outliers.

(a) The original image.



(b) The feature maps after LeakyReLU 5



(c) The feature maps after max pooling of LeakyReLU 5

Figure 54: The original image and the feature maps after passing the image through the network until after the specified layer. The stripe artifacts can be observed in many feature maps in Figure 54b. They vanish after max-pooling (Figure 54c).

width (i.e., the digit one). The VAE model (but not VAE-GAN) is more successful in that regard. However, the VAE distributions are less smooth compared to the VLAE models.

4.8 Feature Map Stripes

One observation made during the analysis of the networks was the emergence of striped artifacts in the networks' feature maps (Figure 54). These stripes were observed in the AlexNetVAE on ImageNet. The stripes are either always horizontal or vertical for one network type. If they are vertical, they can appear on the left or right sides for the same network. If they are horizontal, they appear either on the top or on the bottom of the same network. The exact reason the networks show this behavior was not found, but it is assumed to be a combination of the following considerations.

Kernel Size The stripes were only observed for large kernel sizes. AlexNet Classifier and AlexNet-VAE use 11×11 kernels in the first layer. Using such large kernels with zero-padding (see next paragraph) causes the convolution operation to incorporate many zero-terms. For feature map pixels at borders and especially in corners, only a few non-zero values are contributing to the convolution. This results in overall low feature map values for these pixels.

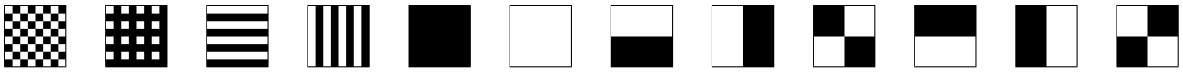


Figure 55: The test images used to analyze the networks behavior.

Padding The stripes occur due to the zero-padding in the network and the resulting contrast observed by the convolutional filters. Take Figure 54. Here, the stripes appear most evident on the bottom left of the image. The stripes indicate less active regions in the feature map: The feature map has an activity of around zero everywhere except for the location of the stripes. Here, the activity is strongly negative.

A comparison with the original image (Figure 54a) shows that for the left side of the image, the contrast is highest on the bottom if the image is zero-padded[45]. The contrast is high on the bottom of the image, the right side, and the right side of the top of the image. However, for this network, the stripes seem to occur for a sharp shift of black on the left to white on the right.

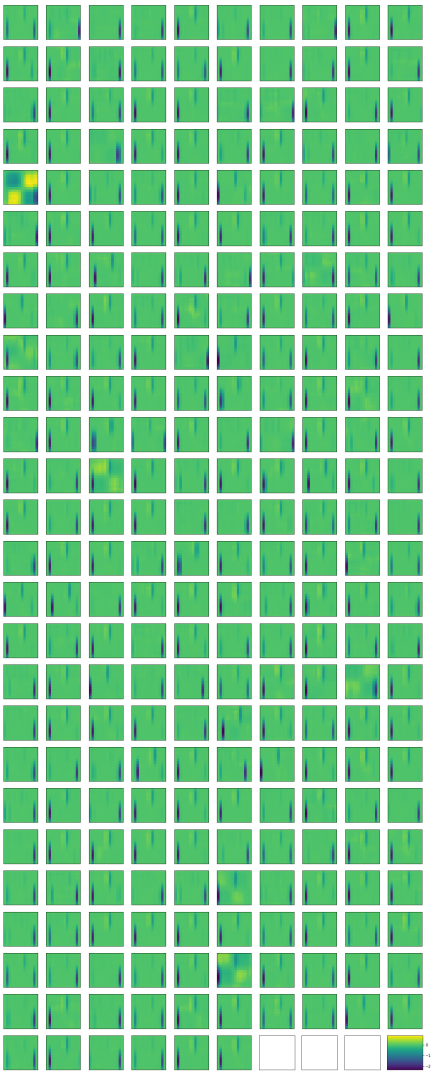
To better understand the behavior, the network was applied to a set of artificial test images (Figure 55).

Figures 55i and 55l turn out to be most insightful. Figure 56 shows the feature map after the last LeakyReLU (LeakyReLU 5) activation of the network given Figure 55i (Figure 56a) as well as the feature map after max-pooling of this feature map (Figure 56b). Figure 56a allows for multiple observations. Firstly, the feature map in the first column of the fifth row resembles the input stimulus itself. The fact that this feature map shows most of the activity (especially after max-pooling, see Figure 56b) has been observed for natural images as well, however not the resemblance of the input stimulus. Except for this feature map, stripes emerge either on the bottom of the left side or the bottom of the right side of the feature map. The sharp black-white contrast at the bottom of the left side. The bottom of the right side is more complicated. Here the test image was black, and the "contrast" is a black-black contrast - or no contrast. However, this is only true for the first feature map[46]. The following feature maps, again, are zero-padded. However, due to the bias term in the convolutions, these might be non-zero in the bottom-right and top-left square of the image, thus leading to a contrast. This explains why the network can be sensitive towards these black-black contrasts in the input image. If the bias term in the convolutions is removed, the black-black contrast sensitivity vanishes because the network is not anymore able to add a constant to the black pixel values on the bottom-right or top-left of the image. For this purpose, the batch normalization has to be removed, too, since it uses a bias term. This, however, does not qualitatively change the networks' behavior on real images.

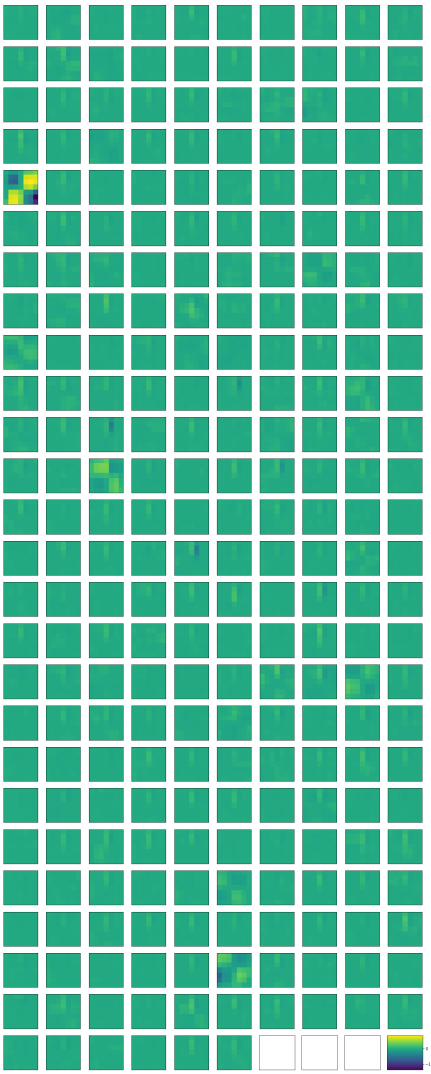
Network Depth As explained above, the emergence of feature maps is an amplifying process. Low-developed feature map stripes in lower layers lead to highly developed feature map stripes in higher layers. This effect was, to the best of the authors' belief, not reported previously. It seems to have no significant influence on network performance. However, it could, depending on the implementation, lead to a loss of precision as feature maps contain very different float values. Floating-point representations then need a larger exponent and have less storage for the mantissa.

Dataset The stripes were observed for the ImageNet dataset but not for CelebA. For CelebA, AlexNetVAE was able to reproduce and generate good images. For ImageNet, however, the

[45] Zero-padding can be understood as adding black pixels around the image.
[46] The first feature maps are not shown here.



(a) The original image.



(b) The feature maps after LeakyReLU 5

Figure 56: The feature maps with respect to test image 551.

network produced and generated blurry images that are only remotely related to the training data. It is therefore assumed that training on ImageNet leads to a high reconstruction error and, as a result of this, to large gradients for the whole training time. Together with the previous considerations, such high gradients are assumed to lead to the feature map stripes. They probably lead to more extreme weight configurations than for non-failure modes (e.g., CelebA).

4.9 Pixelwise vs. Adversarial Loss

This thesis employed two different methods to train the decoder to generate natural-looking images, namely the pixel-wise or the generative loss. Furthermore, the encoder of the generative models was trained such that the hidden-layer activity of the discriminator is similar for generated and real images (in addition to the KL-term).

Advantages, disadvantages, and use-cases for both loss functions are discussed in the following section.

First of all, both loss functions do not seem to be biologically plausible as they are not founded in Hebbian learning. For the encoder loss, it is argued that it is trained to elicit a "neural response" in the discriminator, similar to the one for real images. This "activity matching" seems to be somewhat related to Hebbian learning as it explicitly considers the amount of activity on the level of single neurons. However, discriminator and encoder itself are trained using backpropagation.

Larsen et al. [41] state that the pixel-wise loss can lead to very high values even for small translations (see Section 2.5.3). Seemingly like a disadvantage, this is only true for images with a high frequency and a notable variance of pixel values. A black-and-white image consisting of alternating black and white rows and the same image shifted by one pixel in the y-direction would result in a maximum pixel-wise loss as the two images are orthogonal. Yet, natural images are usually less susceptible to a sizeable pixel-wise loss for small translations because there are fewer regions of high contrast. In the context of natural images, the pixel-wise loss function, amongst other, enables the model to detect if an object is placed in another corner of the image as the loss would increase for such high translations. Nevertheless, if the aim is to generate more realistic images, the generative loss should be used since the pixel-wise loss leads to blurry reconstructions.

Another consideration is training stability. Due to the adversarial loss, training GANs has many challenges (mode collapse, error oscillation, see Section 2.5.1), often resulting in failure modes. Training regular VAEs, in contrast, is far more stable.

The choice of the loss function might also play an essential role in comparing a models' representation with human IT representations. Compared to the pixel-wise loss, the generative loss allows for a more holistic image assessment. This might play an important role in the emergence of a models' latent representation, i.e., might lead to a more realistic latent representation in terms of human IT activations. This aspect should be investigated in future work.

4.10 Model Limitations

All networks investigated in the course of this thesis are CNNs. Network design has to be chosen for each network type (see Section 3.4). Some of these hyperparameters are context-dependent. A network operating on MNIST, for example, receives inputs of size 28×28 pixels. Such networks do not have to be as deep as a network operating on ImageNet, as fewer layers are sufficient for the network to holistically capture the image in its entirety. Other hyperparameters are chosen based on previous research. Some configurations are known to be better for a specific

task than others (see Section 3.4). Matching layers to areas of the visual cortex, therefore, can be challenging.

Another consideration in the context of computational neuroscience is the biological plausibility. This thesis aims at answering how far VAEs are reasonable models of the human visual cortex. To do this, the model structure must allow comparisons with empirical data for the parts under investigation.

Specifically, lower model layers are compared with earlier, higher model layers with later regions in the visual cortex. Furthermore, the models are chosen such that similar inputs translate to similar encodings. The biological plausibility of the encoding should be further investigated in future research. However, due to practical reasons such as the availability of training data or computational resources, the model disregards the human bodys' actualities, a more refined model should incorporate.

Other than the models used in this thesis, the human eye receives a stream of visual stimuli. Perceiving movements of animate objects could play an important role in distinguishing between animate and inanimate objects. The dissimilarity between semantic representations of animate and inanimate objects has been shown in Khaligh-Razavi and Kriegeskorte [36]. Recurrent models such as LSTMs could extend the models used in this thesis to sequences of images.

This thesis only considered image data to form semantic representations, even though the human mind perceives the world through more senses. Future research could aim at investigating if and by how far semantic representations can be improved by using additional sources of information, such as sound.

4.11 Top-Down Connections

As discussed in Section 2.1.3, the lateral geniculate nucleus (LGN) and quaternary visual cortex (V4) receive input from top-down connections. The top-down connections into V4 are assumed to enable attentional mechanisms [58].

Similarly, the Ladder Variational Autoencoder (LVAE) model (see Section 2.5.3) employs top-down connections. In that regard, the LVAE is biologically more plausible compared to the original VAE models. Sønderby et al. [68] claim that the top-down pass helps improving the low-level feature representation because it enables the model to incorporate the higher-level context. Even though Zhao, Song, and Ermon [75] discuss that LVAEs do not employ the latent spaces as efficient as the VLAE, top-down connections enabling attentional mechanisms could play an important role designing more biologically plausible VAE models.

Future work should investigate if such top-down connections allow visual regions operating on a lower semantic level to incorporate the high-level context to disambiguate the lower-level representations.

4.12 Possible applications of VLAE-GAN model

VanRullen and Reddy [71] learn a mapping between test subjects' fMRI responses (when shown images from the CelebA dataset) and the latent space of a VAE. They discuss which brain regions have most influence in the mapping, showing that the occipital lobe contributes most and the temporal and frontoparietal cortex the least.

The VLAE and VLAE-GAN used in the course of this thesis use three latent spaces compared to one latent space for a regular VAE(-GAN). Furthermore it is designed such that the latent spaces encode features on different granularity levels.

Conducting an experiment that is similar to VanRullen and Reddy [71], but maps information from ealier regions of the visual cortex onto lower layers of the model, and higher regions onto higher layers, could help answering whether the lower-level representations are biologically plausible. However, a prerequisite for such a model is to find a VAE-like model explaining cortical activity.

4.13 Comparison to the Inferior Temporal Cortex

Khaligh-Razavi and Kriegeskorte [36] show that higher layers of CNNs trained in a supervised manner explain cortical activity. They also show that this is not true for a variety of unsupervised models. However, VAEs-models are not discussed. It cannot be ruled out that VAEs, even though being unrelated to lower regions of the visual system (see Section 4.1), explain cortical activity in the higher layers or the latent space. Future work should compare IT fMRI data to higher layers of VAEs to investigate this.

4.14 Semantic Representations

Other than unsupervised models, supervised models are known to explain cortical activity [36, 10]. Regarding semantic representations, hidden layer activations of supervised CNNs are a promising candidate.

Nevertheless, semantic representations should at least fulfill the requirements discussed in Section 2.3. Hidden layer activations of supervised CNNs have already shown to fulfill the requirement of biological plausibility to some extent [36]. Furthermore, the representational dissimilarity matrix (RDM) study indicates that similar stimuli activate similar subpopulations of "neurons" in the network.

However, it remains unclear whether these subpopulations consist of neighboring units. This is presumably not the case as the network has no intent to group units that are active for a particular stimulus, close to another.

Variational Autoencoders (VAEs), in contrast, have the property to map similar stimuli to neighboring areas of the latent space. A VAE as part of a supervised CNN[47], trained to reconstruct hidden layer activations, could therefore be a good candidate model for semantic representations. It is suspected to obtain the "explain-cortical-activity" property while also grouping similar stimuli onto neighboring areas in the latent space.

This approach, however, would still not employ sparse representations.

4.15 Sequential Data

All models that were trained in the course of this thesis were non-sequential. Even though supervised models trained on static images show relatedness to the visual system [36, 10, 40], the same does not seem to hold for unsupervised models. However, some unsupervised models trained on sequential data have shown to learn Gabor wavelets [4, 54]. This could indicate that unsupervised models need sequential data to form semantic representations. This line of reasoning is further supported by the fact that humans sequentially perceive the world.

A VAE or VLAE-model extended to sequential data could bring new insights into the role of CNNs as models of the visual system.

[47] Potentially even a sequential one, see Section 4.15

5 Conclusion

This thesis discussed if Variational Autoencoders (VAEs) are candidate models of the visual cortex, potentially allowing to obtain semantic representations of the input.

VAEs and Variational Ladder Autoencoders (VLAEs) neither show Gabor wavelets in early layers, nor do they employ sparseness for inner activities. In their current form, neither VAEs nor VLAEs seem to be a good model of the visual cortex. Modifications of the VAE model structure such as top-down connections or recurrent layers allowing for input sequences could shed new light on biological relatedness. Even though VAEs and VLAEs are not related to the biological example in the lower layers, a representational dissimilarity matrix (RDM) comparison between inferior temporal cortex (IT) and the high-level representations of VAEs should be conducted in the future. If VAE-models fail in explaining cortical activity, it should be investigated why supervised models succeed in this regard. One possible explanation is that the brain mainly learns in a supervised manner and that supervision is always required to build realistic models of the brain. This assumption seems to be too naïve: First, the models discussed in this thesis are too unrelated to the brain to allow such a conclusion. The role of top-down connections as well as the role of sequential data has not been investigated thoroughly. Furthermore, there are unsupervised models leading to Gabor wavelets [53, 4]. This is another hint that unsupervised models in general might be suitable models but need more refinement. VAE-models learn dense input representations. Even though this is benefical in terms of image generation, this representation might be too disconnected from the biological example. The higher-layer activity of supervised convolutional neural networks (CNNs) could serve as a biologically more plausible input representation. VAE-encodings of these representations might combine the advantages of the two models.

This work proposes new means of analyzing latent space entanglement in VAEs and independence of VLAE-generated items. It has been shown that VLAEs do not learn independent factors of variation independently in different layers. Furthermore, many factors of variation are learned across multiple layers. This contradicts the findings of Zhao, Song, and Ermon [75]. Additionally, studying only model-generated samples can be misguiding because decoders could be able to factor out redundant information in the latent spaces. However, the latent distributions are also not independent in terms of generated images as shown by the proposed method of analysis. VAEs and VLAEs often fail in retrieving the data distribution. Sampling from the posterior distribution leads to class distributions different from the input data. Therefore, latent space entanglement seems to be less controllable for VLAEs than for VAEs.

Considering lower-, and higher-level factors of variation, it has been shown that VAEs and VLAEs learn super-, and sub-clusters of different factors of variation. Furthermore, VAEs with a well-balanced loss function can encode and learn a transition between categorical factors of variation. However, poorly defined VAEs can even fail in learning continuous factors of variation if not enough samples are present in the training set.

Employing the adversarial loss function instead of the pixel-wise loss does not seem to lead to improvements in terms of biological plausibility or latent space entanglement. Yet, decoders of such models apparently do better factor out redundant information in the latent spaces. In conclusion, it seems that the pixel-wise loss functions has fewer disadvantages but more advantages than the adversarial loss.

References

[1] Andrew J. Anderson et al. "Of Words, Eyes and Brains: Correlating Image-Based Distributional Semantic Models with Neural Representations of Concepts". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1960–1970. URL: https://www.aclweb.org/anthology/D13-1202.

[2] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. "Latent space oddity: On the curvature of deep generative models". English. In: *Proceedings of International Conference on Learning Representations*. 6th International Conference on Learning Representations, ICLR 2018 ; Conference date: 30-04-2018 Through 03-05-2018. 2018.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[4] Pietro Berkes and Laurenz Wiskott. "Slow feature analysis yields a rich repertoire of complex cell properties". In: *Journal of Vision* 5.6 (July 2005), pp. 9–9. ISSN: 1534-7362. DOI: 10.1167/5.6.9. eprint: https://arvojournals.org/arvo/content_public/journal/jov/933512/jov-5-6-9.pdf. URL: https://doi.org/10.1167/5.6.9.

[5] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=B1xsqj09Fm.

[7] Gerben van den Broeke. "What auto-encoders could learn from brains". PhD thesis. Masters thesis, Aalto University, Finland, 2016.

[8] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing: Core Algorithms*. 1st ed. Springer Publishing Company, Incorporated, 2009. ISBN: 1848001940.

[9] Christopher P Burgess et al. "Understanding disentangling in β-VAE". In: *Learning Disentangled Representations: From Perception to Control Workshop* (2017).

[10] Charles F. Cadieu et al. "Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition". In: *PLOS Computational Biology* 10.12 (Dec. 2014), pp. 1–18. DOI: 10.1371/journal.pcbi.1003963. URL: https://doi.org/10.1371/journal.pcbi.1003963.

[11] D Castro et al. "Morpho-Mnist: Quantitative assessment and diagnostics for representation learning". In: *Journal of Machine Learning Research* 20 (2019).

[12] Tong Che et al. "Mode regularized generative adversarial networks". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (2017).

[13] Ricky T. Q. Chen et al. "Isolating Sources of Disentanglement in Variational Autoencoders". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 2610–2620. URL: http://papers.nips.cc/paper/7527-isolating-sources-of-disentanglement-in-variational-autoencoders.pdf.

[14] Xi Chen et al. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 2172–2180. URL: http://papers.nips.cc/paper/6399-infogan-interpretable-representation-learning-by-information-maximizing-generative-adversarial-nets.pdf.

[15] Davide Chicco, Peter Sadowski, and Pierre Baldi. "Deep Autoencoder Neural Networks for Gene Ontology Annotation Predictions". In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. BCB 14. Newport Beach, California: Association for Computing Machinery, 2014, pp. 533540. ISBN: 9781450328944. DOI: 10.1145/2649387.2649442. URL: https://doi.org/10.1145/2649387.2649442.

[16] J. Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[17] Robert Desimone and John Duncan. “Neural Mechanisms of Selective Visual Attention”. In: *Annual Review of Neuroscience* 18.1 (1995). PMID: 7605061, pp. 193–222. DOI: 10.1146/annurev.ne.18.030195.001205. eprint: https://doi.org/10.1146/annurev.ne.18.030195.001205. URL: https://doi.org/10.1146/annurev.ne.18.030195.001205.

[18] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *arXiv preprint arXiv:1603.07285* (2016).

[19] Michael Eickenberg et al. “Seeing it all: Convolutional network layers map the function of the human visual system”. In: *NeuroImage* 152 (2017), pp. 184 –194. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2016.10.001. URL: http://www.sciencedirect.com/science/article/pii/S1053811916305481.

[20] Scott L. Fairhall and Alfonso Caramazza. “Brain Regions That Represent Amodal Conceptual Knowledge”. In: *Journal of Neuroscience* 33.25 (2013), pp. 10552–10558. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.0051-13.2013. eprint: https://www.jneurosci.org/content/33/25/10552.full.pdf. URL: https://www.jneurosci.org/content/33/25/10552.

[21] Peter Foldiak. “Sparse coding in the primate cortex”. In: *The handbook of brain theory and neural networks* (2003).

[22] Jeremy Freeman et al. “A functional and perceptual signature of the second visual area in primates”. In: *Nature Neuroscience* 16.7 (2013), pp. 974–981. ISSN: 1546-1726. DOI: 10.1038/nn.3402. URL: https://doi.org/10.1038/nn.3402.

[23] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition”. In: *Competition and Cooperation in Neural Nets*. Ed. by Shun-ichi Amari and Michael A. Arbib. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 267–285. ISBN: 978-3-642-46466-9.

[24] E. Georganas et al. “Anatomy of High-Performance Deep Learning Convolutions on SIMD Architectures”. In: *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2018, pp. 830–841.

[25] Melvyn A Goodale, A David Milner, et al. “Separate visual pathways for perception and action”. In: *Trends in neurosciences* 15.1 (1992), pp. 20–25.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[27] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[28] Hyungrok Ham, Tae Joon Jun, and Daeyoung Kim. “Unbalanced GANs: Pre-training the Generator of Generative Adversarial Network using Variational Autoencoder”. In: *arXiv preprint arXiv:2002.02112* (2020).

[29] Irina Higgins et al. “beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework.” In: *Proceedings of the 5th International Conference on Learning Representations (ICLR)* 2.5 (2017), p. 6.

[30] Irina Higgins et al. “Towards a definition of disentangled representations”. In: *arXiv preprint arXiv:1812.02230* (2018).

[31] Xianxu Hou et al. “Deep feature consistent variational autoencoder”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 1133–1141.

[32] D. H. Hubel and T. N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. eng. In: *The Journal of physiology* 160.1 (1962).

14449617[pmid], pp. 106–154. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1962.sp006837. URL: https://pubmed.ncbi.nlm.nih.gov/14449617.

[33] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 694–711. ISBN: 978-3-319-46475-6.

[34] J. P. Jones and L. A. Palmer. "An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex". In: *Journal of Neurophysiology* 58.6 (1987). PMID: 3437332, pp. 1233–1258. DOI: 10.1152/jn.1987.58.6.1233. eprint: https://doi.org/10.1152/jn.1987.58.6.1233. URL: https://doi.org/10.1152/jn.1987.58.6.1233.

[35] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4401–4410.

[36] Seyed-Mahdi Khaligh-Razavi and Nikolaus Kriegeskorte. "Deep Supervised, but Not Unsupervised, Models May Explain IT Cortical Representation". In: *PLOS Computational Biology* 10.11 (Nov. 2014), pp. 1–29. DOI: 10.1371/journal.pcbi.1003915. URL: https://doi.org/10.1371/journal.pcbi.1003915.

[37] Hyunjik Kim and Andriy Mnih. "Disentangling by Factorising". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 2649–2658. URL: http://proceedings.mlr.press/v80/kim18b.html.

[38] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8237. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[39] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6114.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[41] Anders Boesen Lindbo Larsen et al. "Autoencoding beyond pixels using a learned similarity metric". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1558–1566. URL: http://proceedings.mlr.press/v48/larsen16.html.

[42] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541. eprint: https://doi.org/10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541.

[43] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[44] *Lexico*. 2020. URL: www.lexico.com.

[45] Grace Lindsay. "Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future". In: *Journal of Cognitive Neuroscience* 0.0 (0). PMID: 32027584, pp. 1–15. DOI: 10.1162/jocn_a_01544. eprint: https://doi.org/10.1162/jocn_a_01544. URL: https://doi.org/10.1162/jocn_a_01544.

[46] Z. Liu et al. “Deep Learning Face Attributes in the Wild”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3730–3738.

[47] Nikos K. Logothetis, Jon Pauls, and Tomaso Poggio. “Shape representation in the inferior temporal cortex of monkeys”. In: *Current Biology* 5.5 (1995), pp. 552 –563. ISSN: 0960-9822. DOI: https://doi.org/10.1016/S0960-9822(95)00108-4. URL: http://www.sciencedirect.com/science/article/pii/S0960982295001084.

[48] James Lucas et al. “Understanding posterior collapse in generative latent variable models”. In: *International Conference on Learning Representations, Workshop Paper* (2019).

[49] S. Mack et al. *Principles of Neural Science, Fifth Edition*. Principles of Neural Science. McGraw-Hill Education, 2013. ISBN: 9780071390118. URL: https://books.google.de/books?id=s64z-LdAlsEC.

[50] Alireza Makhzani et al. “Adversarial Autoencoders”. In: *International Conference on Learning Representations*. 2016. URL: http://arxiv.org/abs/1511.05644.

[51] Loic Matthey et al. *dSprites: Disentanglement testing Sprites dataset*. 2017. URL: https://github.com/deepmind/dsprites-dataset/.

[52] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2013. URL: http://arxiv.org/abs/1301.3781.

[53] Bruno A. Olshausen and David J. Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. In: *Nature* 381.6583 (1996), pp. 607–609. ISSN: 1476-4687. DOI: 10.1038/381607a0. URL: https://doi.org/10.1038/381607a0.

[54] R. B. Palm. “Prediction as a candidate for learning deep hierarchical models of data”. In: (2012). Supervised by Associate Professor Ole Winther, owi@imm.dtu.dk, DTU Informatics, and Morten Mørup, mm@imm.dtu.dk, DTU Informatics. URL: http://www.imm.dtu.dk/English.aspx.

[55] Stanislav Pidhorskyi, Donald A. Adjeroh, and Gianfranco Doretto. “Adversarial Latent Autoencoders”. In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[56] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1511.06434.

[57] Antonio J. Rodríguez-Sánchez, Mazyar Fallah, and Ale Leonardis. “Editorial: Hierarchical Object Representations in the Visual Cortex and Computer Vision”. In: *Frontiers in Computational Neuroscience* 9 (2015), p. 142. ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00142. URL: https://www.frontiersin.org/article/10.3389/fncom.2015.00142.

[58] AnnaăW. Roe et al. “Toward a Unified Theory of Visual Area V4”. In: *Neuron* 74.1 (2012), pp. 12 –29. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2012.03.011. URL: http://www.sciencedirect.com/science/article/pii/S0896627312002747.

[59] Yu-Ping Ruan, Zhen-Hua Ling, and Yu Hu. “Exploring Semantic Representation in Brain Activity Using Word Embeddings”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 669–679. DOI: 10.18653/v1/D16-1064. URL: https://www.aclweb.org/anthology/D16-1064.

[60] Alan B. Rubens and D. Frank Benson. “Associative Visual Agnosia”. In: *Archives of Neurology* 24.4 (Apr. 1971), pp. 305–316. ISSN: 0003-9942. DOI: 10.1001/archneur.1971.00480340037003. eprint: https://jamanetwork.com/journals/jamaneurology/articlepdf/570416/archneur_24_4_003.pdf. URL: https://doi.org/10.1001/archneur.1971.00480340037003.

[61] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. URL: https://doi.org/10.1007/s11263-015-0816-y.

[62] H. Shao, A. Kumar, and P. Fletcher. “The Riemannian Geometry of Deep Generative Models”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Los Alamitos, CA, USA: IEEE Computer Society, 2018, pp. 428–4288. DOI: 10.1109/CVPRW.2018.00071. URL: https://doi.ieeecomputersociety.org/10.1109/CVPRW.2018.00071.

[63] Rui Shu et al. “Weakly Supervised Disentanglement with Guarantees”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=HJgSwyBKvr.

[64] J.T. Springenberg et al. “Striving for Simplicity: The All Convolutional Net”. In: *ICLR (workshop track)*. 2015. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a.

[65] Larry Squire et al. *Fundamental Neuroscience*. Academic Press, 2012.

[66] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[67] Christian Szegedy et al. “Going Deeper With Convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[68] Casper Kaae Sønderby et al. “Ladder Variational Autoencoders”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., 2016, pp. 3738–3746. URL: http://papers.nips.cc/paper/6275-ladder-variational-autoencoders.pdf.

[69] T. Trappenberg. *Fundamentals of Computational Neuroscience*. OUP Oxford, 2010. ISBN: 9780199568413. URL: https://books.google.de/books?id=1xSLktiRAX4C.

[70] Michael Tschannen, Olivier Bachem, and Mario Lucic. “Recent advances in autoencoder-based representation learning”. In: *Third workshop on Bayesian Deep Learning (NeurIPS 2018)*. 2018. URL: http://www.nari.ee.ethz.ch/pubs/p/autoenc2018.

[71] Rufin VanRullen and Leila Reddy. “Reconstructing faces from fMRI patterns using deep generative neural networks”. In: *Communications Biology* 2.1 (2019), p. 193. ISSN: 2399-3642. DOI: 10.1038/s42003-019-0438-y. URL: https://doi.org/10.1038/s42003-019-0438-y.

[72] Jacob Walker et al. “An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 835–851. ISBN: 978-3-319-46478-7.

[73] Haiguang Wen et al. “Neural Encoding and Decoding with Deep Learning for Dynamic Natural Vision”. In: *Cerebral Cortex* 28.12 (Oct. 2017), pp. 4136–4160. ISSN: 1047-3211. DOI: 10.1093/cercor/bhx268. eprint: https://academic.oup.com/cercor/article-pdf/28/12/4136/26338870/bhx268.pdf. URL: https://doi.org/10.1093/cercor/bhx268.

[74] Takashi Yoshida and Kenichi Ohki. “Natural images are reliably represented by sparse and variable populations of neurons in visual cortex”. In: *Nature Communications* 11.1 (2020), p. 872. ISSN: 2041-1723. DOI: 10.1038/s41467-020-14645-x. URL: https://doi.org/10.1038/s41467-020-14645-x.

[75] Shengjia Zhao, Jiaming Song, and Stefano Ermon. “Learning Hierarchical Features from Deep Generative Models”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 4091–4099. URL: http://proceedings.mlr.press/v70/zhao17c.html.

[76] Shengjia Zhao, Jiaming Song, and Stefano Ermon. “Towards deeper understanding of variational autoencoding models”. In: *arXiv preprint arXiv:1702.08658* (2017).

A Network Architectures

A.1 VAE-models

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 28, 28, 3)	0	
encoder_conv_0 (Conv2D)	(None, 14, 14, 32)	896	encoder_input[0][0]
batch_normalization_1 (BatchNor	(None, 14, 14, 32)	128	encoder_conv_0[0][0]
re_lu_1 (ReLU)	(None, 14, 14, 32)	0	batch_normalization_1[0][0]
encoder_conv_1 (Conv2D)	(None, 7, 7, 64)	18496	re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 7, 7, 64)	256	encoder_conv_1[0][0]
re_lu_2 (ReLU)	(None, 7, 7, 64)	0	batch_normalization_2[0][0]
encoder_conv_2 (Conv2D)	(None, 7, 7, 128)	73856	re_lu_2[0][0]
batch_normalization_3 (BatchNor	(None, 7, 7, 128)	512	encoder_conv_2[0][0]
re_lu_3 (ReLU)	(None, 7, 7, 128)	0	batch_normalization_3[0][0]
encoder_conv_3 (Conv2D)	(None, 7, 7, 256)	295168	re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 7, 7, 256)	1024	encoder_conv_3[0][0]
re_lu_4 (ReLU)	(None, 7, 7, 256)	0	batch_normalization_4[0][0]
flatten_1 (Flatten)	(None, 12544)	0	re_lu_4[0][0]
mu (Dense)	(None, 2)	25090	flatten_1[0][0]
log_var (Dense)	(None, 2)	25090	flatten_1[0][0]
encoder_output (Lambda) log_var[0][0]	(None, 2)	0	mu[0][0]

Total params: 440,516
Trainable params: 439,556
Non-trainable params: 960

Listing 1: Mnist-VAE Encoder

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 2)	0
dense_1 (Dense)	(None, 12544)	37632
reshape_1 (Reshape)	(None, 7, 7, 256)	0
decoder_conv_t_0 (Conv2DTran	(None, 7, 7, 256)	1048832
batch_normalization_5 (Batch	(None, 7, 7, 256)	1024
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 256)	0
decoder_conv_t_1 (Conv2DTran	(None, 7, 7, 128)	524416
batch_normalization_6 (Batch	(None, 7, 7, 128)	512
leaky_re_lu_2 (LeakyReLU)	(None, 7, 7, 128)	0
decoder_conv_t_2 (Conv2DTran	(None, 14, 14, 64)	131136
batch_normalization_7 (Batch	(None, 14, 14, 64)	256
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
decoder_conv_t_3 (Conv2DTran	(None, 28, 28, 3)	3075
activation_1 (Activation)	(None, 28, 28, 3)	0

Total params: 1,746,883
Trainable params: 1,745,987
Non-trainable params: 896

Listing 2: Mnist-VAE Decoder

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 64, 64, 1)	0	
encoder_conv_0 (Conv2D)	(None, 32, 32, 32)	544	encoder_input[0][0]
batch_normalization_1 (BatchNor	(None, 32, 32, 32)	128	encoder_conv_0[0][0]
re_lu_1 (ReLU)	(None, 32, 32, 32)	0	batch_normalization_1[0][0]
encoder_conv_1 (Conv2D)	(None, 16, 16, 64)	32832	re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 16, 16, 64)	256	encoder_conv_1[0][0]
re_lu_2 (ReLU)	(None, 16, 16, 64)	0	batch_normalization_2[0][0]
encoder_conv_2 (Conv2D)	(None, 8, 8, 128)	131200	re_lu_2[0][0]
batch_normalization_3 (BatchNor	(None, 8, 8, 128)	512	encoder_conv_2[0][0]
re_lu_3 (ReLU)	(None, 8, 8, 128)	0	batch_normalization_3[0][0]
encoder_conv_3 (Conv2D)	(None, 4, 4, 256)	524544	re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 4, 4, 256)	1024	encoder_conv_3[0][0]

re_lu_4 (ReLU)	(None, 4, 4, 256)	0	batch_normalization_4[0][0]
encoder_conv_4 (Conv2D)	(None, 4, 4, 512)	2097664	re_lu_4[0][0]
batch_normalization_5 (BatchNor	(None, 4, 4, 512)	2048	encoder_conv_4[0][0]
re_lu_5 (ReLU)	(None, 4, 4, 512)	0	batch_normalization_5[0][0]
flatten_1 (Flatten)	(None, 8192)	0	re_lu_5[0][0]
mu (Dense)	(None, 10)	81930	flatten_1[0][0]
log_var (Dense)	(None, 10)	81930	flatten_1[0][0]
encoder_output (Lambda)	(None, 10)	0	mu[0][0] log_var[0][0]

Total params: 2,954,612
Trainable params: 2,952,628
Non-trainable params: 1,984

Listing 3: (dSprites, 7,500, 6,250, 5,000, 3,750)-VAE Encoder

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 10)	0
dense_1 (Dense)	(None, 8192)	90112
reshape_1 (Reshape)	(None, 4, 4, 512)	0
decoder_conv_t_0 (Conv2DTran	(None, 8, 8, 512)	4194816
batch_normalization_6 (Batch	(None, 8, 8, 512)	2048
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 512)	0
decoder_conv_t_1 (Conv2DTran	(None, 16, 16, 256)	2097408
batch_normalization_7 (Batch	(None, 16, 16, 256)	1024
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 256)	0
decoder_conv_t_2 (Conv2DTran	(None, 32, 32, 128)	524416
batch_normalization_8 (Batch	(None, 32, 32, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 32, 32, 128)	0
decoder_conv_t_3 (Conv2DTran	(None, 64, 64, 1)	2049
activation_1 (Activation)	(None, 64, 64, 1)	0

Total params: 6,912,385
Trainable params: 6,910,593
Non-trainable params: 1,792

Listing 4: (dSprites, 7,500, 6,250, 5,000, 3,750)-VAE Decoder

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 128, 128, 3)	0	
encoder_conv_0 (Conv2D)	(None, 64, 64, 32)	1568	encoder_input[0][0]
batch_normalization_1 (BatchNor	(None, 64, 64, 32)	128	encoder_conv_0[0][0]
re_lu_1 (ReLU)	(None, 64, 64, 32)	0	batch_normalization_1[0][0]
encoder_conv_1 (Conv2D)	(None, 32, 32, 64)	32832	re_lu_1[0][0]
batch_normalization_2 (BatchNor	(None, 32, 32, 64)	256	encoder_conv_1[0][0]
re_lu_2 (ReLU)	(None, 32, 32, 64)	0	batch_normalization_2[0][0]
encoder_conv_2 (Conv2D)	(None, 16, 16, 128)	131200	re_lu_2[0][0]
batch_normalization_3 (BatchNor	(None, 16, 16, 128)	512	encoder_conv_2[0][0]
re_lu_3 (ReLU)	(None, 16, 16, 128)	0	batch_normalization_3[0][0]
encoder_conv_3 (Conv2D)	(None, 8, 8, 256)	524544	re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 8, 8, 256)	1024	encoder_conv_3[0][0]
re_lu_4 (ReLU)	(None, 8, 8, 256)	0	batch_normalization_4[0][0]
encoder_conv_4 (Conv2D)	(None, 8, 8, 512)	2097664	re_lu_4[0][0]
batch_normalization_5 (BatchNor	(None, 8, 8, 512)	2048	encoder_conv_4[0][0]
re_lu_5 (ReLU)	(None, 8, 8, 512)	0	batch_normalization_5[0][0]
flatten_1 (Flatten)	(None, 32768)	0	re_lu_5[0][0]
mu (Dense)	(None, 8)	262152	flatten_1[0][0]
log_var (Dense)	(None, 8)	262152	flatten_1[0][0]
encoder_output (Lambda)	(None, 8)	0	mu[0][0] log_var[0][0]

Total params: 3,316,080
Trainable params: 3,314,096
Non-trainable params: 1,984

Listing 5: CelebA-VAE Encoder

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 8)	0
dense_1 (Dense)	(None, 32768)	294912
reshape_1 (Reshape)	(None, 8, 8, 512)	0
decoder_conv_t_0 (Conv2DTran	(None, 16, 16, 512)	4194816
batch_normalization_6 (Batch	(None, 16, 16, 512)	2048
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 512)	0
decoder_conv_t_1 (Conv2DTran	(None, 32, 32, 256)	2097408
batch_normalization_7 (Batch	(None, 32, 32, 256)	1024
leaky_re_lu_2 (LeakyReLU)	(None, 32, 32, 256)	0
decoder_conv_t_2 (Conv2DTran	(None, 64, 64, 128)	524416
batch_normalization_8 (Batch	(None, 64, 64, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 64, 64, 128)	0
decoder_conv_t_3 (Conv2DTran	(None, 128, 128, 3)	6147
activation_1 (Activation)	(None, 128, 128, 3)	0

Total params: 7,121,283
Trainable params: 7,119,491
Non-trainable params: 1,792

Listing 6: CelebA-VAE Decoder

A.2 VLAE-models

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 28, 28, 1)	0	
inference_0_conv2d_0 (Conv2D)	(None, 14, 14, 64)	1664	input_1[0][0]
inference_0_relu_0 (ReLU)	(None, 14, 14, 64)	0	inference_0_conv2d_0[0][0]
inference_1_conv2d_0 (Conv2D)	(None, 7, 7, 64)	36928	inference_0_relu_0[0][0]
inference_1_relu_0 (ReLU)	(None, 7, 7, 64)	0	inference_1_conv2d_0[0][0]
ladder_2_conv2d_0 (Conv2D)	(None, 7, 7, 64)	36928	inference_1_relu_0[0][0]
ladder_2_relu_0 (ReLU)	(None, 7, 7, 64)	0	ladder_2_conv2d_0[0][0]
ladder_0_conv2d_0 (Conv2D)	(None, 14, 14, 64)	1664	input_1[0][0]
ladder_1_conv2d_0 (Conv2D)	(None, 7, 7, 64)	102464	inference_0_relu_0[0][0]
ladder_2_conv2d_1 (Conv2D)	(None, 7, 7, 64)	36928	ladder_2_relu_0[0][0]
ladder_0_relu_0 (ReLU)	(None, 14, 14, 64)	0	ladder_0_conv2d_0[0][0]
ladder_1_relu_0 (ReLU)	(None, 7, 7, 64)	0	ladder_1_conv2d_0[0][0]
ladder_2_relu_1 (ReLU)	(None, 7, 7, 64)	0	ladder_2_conv2d_1[0][0]
ladder_0_flatten (Flatten)	(None, 12544)	0	ladder_0_relu_0[0][0]
ladder_1_flatten (Flatten)	(None, 3136)	0	ladder_1_relu_0[0][0]
ladder_2_flatten (Flatten)	(None, 3136)	0	ladder_2_relu_1[0][0]
mu_1 (Dense)	(None, 2)	25090	ladder_0_flatten[0][0]
log_var_1 (Dense)	(None, 2)	25090	ladder_0_flatten[0][0]
mu_2 (Dense)	(None, 2)	6274	ladder_1_flatten[0][0]
log_var_2 (Dense)	(None, 2)	6274	ladder_1_flatten[0][0]
mu_3 (Dense)	(None, 2)	6274	ladder_2_flatten[0][0]
log_var_3 (Dense)	(None, 2)	6274	ladder_2_flatten[0][0]
z_1_latent (Lambda) log_var_1[0][0]	(None, 2)	0	mu_1[0][0]
z_2_latent (Lambda) log_var_2[0][0]	(None, 2)	0	mu_2[0][0]
z_3_latent (Lambda) log_var_3[0][0]	(None, 2)	0	mu_3[0][0]

Total params: 291,852
Trainable params: 291,852
Non-trainable params: 0

Listing 7: MNIST-VLAE-factor-1 Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 2)	0	
generative_2_dense_0 (Dense)	(None, 1024)	3072	z_3[0][0]
generative_2_relu_0 (ReLU)	(None, 1024)	0	generative_2_dense_0[0][0]
generative_2_dense_1 (Dense)	(None, 1024)	1049600	generative_2_relu_0[0][0]

generative_2_relu_1 (ReLU)	(None, 1024)	0	generative_2_dense_1[0][0]
z_2 (InputLayer)	(None, 2)	0	
concatenate_2_and_1 (Concatenat z_2[0][0]	(None, 1026)	0	generative_2_relu_1[0][0]
generative_1_dense_0 (Dense)	(None, 1024)	1051648	concatenate_2_and_1[0][0]
generative_1_relu_0 (ReLU)	(None, 1024)	0	generative_1_dense_0[0][0]
generative_1_dense_1 (Dense)	(None, 1024)	1049600	generative_1_relu_0[0][0]
generative_1_relu_1 (ReLU)	(None, 1024)	0	generative_1_dense_1[0][0]
z_1 (InputLayer)	(None, 2)	0	
concatenate_1_and_0 (Concatenat z_1[0][0]	(None, 1026)	0	generative_1_relu_1[0][0]
generative_0_dense_0 (Dense)	(None, 3136)	3220672	concatenate_1_and_0[0][0]
generative_0_relu_0 (ReLU)	(None, 3136)	0	generative_0_dense_0[0][0]
generative_0_reshape_0 (Reshape	(None, 7, 7, 64)	0	generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0	(None, 14, 14, 64)	102464	generative_0_reshape_0[0][0]
generative_0_leaky_relu_transpo	(None, 14, 14, 64)	0	generative_0_conv2d_transpose_0[0
generative_0_conv2d_transpose_1	(None, 28, 28, 64)	36928	generative_0_leaky_relu_transpose
generative_0_leaky_relu_transpo	(None, 28, 28, 64)	0	generative_0_conv2d_transpose_1[0
generative_0_conv2d_transpose_2	(None, 28, 28, 1)	1601	generative_0_leaky_relu_transpose
generative_0_sigmoid_0 (Activat	(None, 28, 28, 1)	0	generative_0_conv2d_transpose_2[0

Total params: 6,515,585
Trainable params: 6,515,585
Non-trainable params: 0

Listing 8: MNIST-VLAE-factor-1 Decoder

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 28, 28, 1)	0	
inference_0_conv2d_0 (Conv2D)	(None, 14, 14, 32)	832	input_1[0][0]
inference_0_relu_0 (ReLU)	(None, 14, 14, 32)	0	inference_0_conv2d_0[0][0]
inference_1_conv2d_0 (Conv2D)	(None, 7, 7, 32)	9248	inference_0_relu_0[0][0]
inference_1_relu_0 (ReLU)	(None, 7, 7, 32)	0	inference_1_conv2d_0[0][0]
ladder_2_conv2d_0 (Conv2D)	(None, 7, 7, 32)	9248	inference_1_relu_0[0][0]
ladder_2_relu_0 (ReLU)	(None, 7, 7, 32)	0	ladder_2_conv2d_0[0][0]
ladder_0_conv2d_0 (Conv2D)	(None, 14, 14, 32)	832	input_1[0][0]
ladder_1_conv2d_0 (Conv2D)	(None, 7, 7, 32)	25632	inference_0_relu_0[0][0]
ladder_2_conv2d_1 (Conv2D)	(None, 7, 7, 32)	9248	ladder_2_relu_0[0][0]
ladder_0_relu_0 (ReLU)	(None, 14, 14, 32)	0	ladder_0_conv2d_0[0][0]
ladder_1_relu_0 (ReLU)	(None, 7, 7, 32)	0	ladder_1_conv2d_0[0][0]
ladder_2_relu_1 (ReLU)	(None, 7, 7, 32)	0	ladder_2_conv2d_1[0][0]
ladder_0_flatten (Flatten)	(None, 6272)	0	ladder_0_relu_0[0][0]
ladder_1_flatten (Flatten)	(None, 1568)	0	ladder_1_relu_0[0][0]
ladder_2_flatten (Flatten)	(None, 1568)	0	ladder_2_relu_1[0][0]
mu_1 (Dense)	(None, 2)	12546	ladder_0_flatten[0][0]
log_var_1 (Dense)	(None, 2)	12546	ladder_0_flatten[0][0]
mu_2 (Dense)	(None, 2)	3138	ladder_1_flatten[0][0]
log_var_2 (Dense)	(None, 2)	3138	ladder_1_flatten[0][0]
mu_3 (Dense)	(None, 2)	3138	ladder_2_flatten[0][0]
log_var_3 (Dense)	(None, 2)	3138	ladder_2_flatten[0][0]
z_1_latent (Lambda) log_var_1[0][0]	(None, 2)	0	mu_1[0][0]
z_2_latent (Lambda) log_var_2[0][0]	(None, 2)	0	mu_2[0][0]
z_3_latent (Lambda) log_var_3[0][0]	(None, 2)	0	mu_3[0][0]

Total params: 92,684
Trainable params: 92,684
Non-trainable params: 0

Listing 9: MNIST-VLAE-factor-2 Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 2)	0	
generative_2_dense_0 (Dense)	(None, 512)	1536	z_3[0][0]
generative_2_relu_0 (ReLU)	(None, 512)	0	generative_2_dense_0[0][0]

```
generative_2_dense_1 (Dense)     (None, 512)          262656     generative_2_relu_0[0][0]
generative_2_relu_1 (ReLU)       (None, 512)          0          generative_2_dense_1[0][0]
z_2 (InputLayer)                 (None, 2)            0
concatenate_2_and_1 (Concatenat (None, 514)          0          generative_2_relu_1[0][0]
z_2[0][0]
generative_1_dense_0 (Dense)     (None, 512)          263680     concatenate_2_and_1[0][0]
generative_1_relu_0 (ReLU)       (None, 512)          0          generative_1_dense_0[0][0]
generative_1_dense_1 (Dense)     (None, 512)          262656     generative_1_relu_0[0][0]
generative_1_relu_1 (ReLU)       (None, 512)          0          generative_1_dense_1[0][0]
z_1 (InputLayer)                 (None, 2)            0
concatenate_1_and_0 (Concatenat (None, 514)          0          generative_1_relu_1[0][0]
z_1[0][0]
generative_0_dense_0 (Dense)     (None, 1568)         807520     concatenate_1_and_0[0][0]
generative_0_relu_0 (ReLU)       (None, 1568)         0          generative_0_dense_0[0][0]
generative_0_reshape_0 (Reshape (None, 7, 7, 32)     0          generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0 (None, 14, 14, 32)   25632      generative_0_reshape_0[0][0]
generative_0_leaky_relu_transpo (None, 14, 14, 32)   0          generative_0_conv2d_transpose_0[0
generative_0_conv2d_transpose_1 (None, 28, 28, 32)   9248       generative_0_leaky_relu_transpose
generative_0_leaky_relu_transpo (None, 28, 28, 32)   0          generative_0_conv2d_transpose_1[0
generative_0_conv2d_transpose_2 (None, 28, 28, 1)    801        generative_0_leaky_relu_transpose
generative_0_sigmoid_0 (Activat (None, 28, 28, 1)    0          generative_0_conv2d_transpose_2[0
Total params: 1,633,729
Trainable params: 1,633,729
Non-trainable params: 0
```

Listing 10: MNIST-VLAE-factor-2 Decoder

```
Layer (type)                     Output Shape         Param #    Connected to
input_1 (InputLayer)             (None, 28, 28, 1)    0
inference_0_conv2d_0 (Conv2D)    (None, 14, 14, 22)   572        input_1[0][0]
inference_0_relu_0 (ReLU)        (None, 14, 14, 22)   0          inference_0_conv2d_0[0][0]
inference_1_conv2d_0 (Conv2D)    (None, 7, 7, 22)     4378       inference_0_relu_0[0][0]
inference_1_relu_0 (ReLU)        (None, 7, 7, 22)     0          inference_1_conv2d_0[0][0]
ladder_2_conv2d_0 (Conv2D)       (None, 7, 7, 22)     4378       inference_1_relu_0[0][0]
ladder_2_relu_0 (ReLU)           (None, 7, 7, 22)     0          ladder_2_conv2d_0[0][0]
ladder_0_conv2d_0 (Conv2D)       (None, 14, 14, 22)   572        input_1[0][0]
ladder_1_conv2d_0 (Conv2D)       (None, 7, 7, 22)     12122      inference_0_relu_0[0][0]
ladder_2_conv2d_1 (Conv2D)       (None, 7, 7, 22)     4378       ladder_2_relu_0[0][0]
ladder_0_relu_0 (ReLU)           (None, 14, 14, 22)   0          ladder_0_conv2d_0[0][0]
ladder_1_relu_0 (ReLU)           (None, 7, 7, 22)     0          ladder_1_conv2d_0[0][0]
ladder_2_relu_1 (ReLU)           (None, 7, 7, 22)     0          ladder_2_conv2d_1[0][0]
ladder_0_flatten (Flatten)       (None, 4312)         0          ladder_0_relu_0[0][0]
ladder_1_flatten (Flatten)       (None, 1078)         0          ladder_1_relu_0[0][0]
ladder_2_flatten (Flatten)       (None, 1078)         0          ladder_2_relu_1[0][0]
mu_1 (Dense)                     (None, 2)            8626       ladder_0_flatten[0][0]
log_var_1 (Dense)                (None, 2)            8626       ladder_0_flatten[0][0]
mu_2 (Dense)                     (None, 2)            2158       ladder_1_flatten[0][0]
log_var_2 (Dense)                (None, 2)            2158       ladder_1_flatten[0][0]
mu_3 (Dense)                     (None, 2)            2158       ladder_2_flatten[0][0]
log_var_3 (Dense)                (None, 2)            2158       ladder_2_flatten[0][0]
z_1_latent (Lambda)              (None, 2)            0          mu_1[0][0]
log_var_1[0][0]
z_2_latent (Lambda)              (None, 2)            0          mu_2[0][0]
log_var_2[0][0]
z_3_latent (Lambda)              (None, 2)            0          mu_3[0][0]
log_var_3[0][0]
Total params: 52,284
Trainable params: 52,284
Non-trainable params: 0
```

Listing 11: MNIST-VLAE-factor-3 Encoder

```
Layer (type)                     Output Shape         Param #    Connected to
z_3 (InputLayer)                 (None, 2)            0
generative_2_dense_0 (Dense)     (None, 342)          1026       z_3[0][0]
```

```
generative_2_relu_0 (ReLU)       (None, 342)           0          generative_2_dense_0[0][0]
generative_2_dense_1 (Dense)     (None, 342)           117306     generative_2_relu_0[0][0]
generative_2_relu_1 (ReLU)       (None, 342)           0          generative_2_dense_1[0][0]
z_2 (InputLayer)                 (None, 2)             0
concatenate_2_and_1 (Concatenat (None, 344)           0          generative_2_relu_1[0][0]
z_2[0][0]
generative_1_dense_0 (Dense)     (None, 342)           117990     concatenate_2_and_1[0][0]
generative_1_relu_0 (ReLU)       (None, 342)           0          generative_1_dense_0[0][0]
generative_1_dense_1 (Dense)     (None, 342)           117306     generative_1_relu_0[0][0]
generative_1_relu_1 (ReLU)       (None, 342)           0          generative_1_dense_1[0][0]
z_1 (InputLayer)                 (None, 2)             0
concatenate_1_and_0 (Concatenat (None, 344)           0          generative_1_relu_1[0][0]
z_1[0][0]
generative_0_dense_0 (Dense)     (None, 1078)          371910     concatenate_1_and_0[0][0]
generative_0_relu_0 (ReLU)       (None, 1078)          0          generative_0_dense_0[0][0]
generative_0_reshape_0 (Reshape (None, 7, 7, 22)      0          generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0 (None, 14, 14, 22)    12122      generative_0_reshape_0[0][0]
generative_0_leaky_relu_transpo (None, 14, 14, 22)    0          generative_0_conv2d_transpose_0[0
generative_0_conv2d_transpose_1 (None, 28, 28, 22)    4378       generative_0_leaky_relu_transpose
generative_0_leaky_relu_transpo (None, 28, 28, 22)    0          generative_0_conv2d_transpose_1[0
generative_0_conv2d_transpose_2 (None, 28, 28, 1)     551        generative_0_leaky_relu_transpose
generative_0_sigmoid_0 (Activat (None, 28, 28, 1)     0          generative_0_conv2d_transpose_2[0
Total params: 742,589
Trainable params: 742,589
Non-trainable params: 0
```

Listing 12: MNIST-VLAE-factor-3 Decoder

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 64, 64, 1)	0	
inference_0_conv2d_0 (Conv2D)	(None, 32, 32, 32)	544	input_1[0][0]
inference_0_dropout_0 (Dropout)	(None, 32, 32, 32)	0	inference_0_conv2d_0[0][0]
inference_0_batch_norm_0 (Batch	(None, 32, 32, 32)	128	inference_0_dropout_0[0][0]
inference_0_relu_0 (ReLU)	(None, 32, 32, 32)	0	inference_0_batch_norm_0[0][0]
inference_0_conv2d_1 (Conv2D)	(None, 32, 32, 32)	16416	inference_0_relu_0[0][0]
inference_0_dropout_1 (Dropout)	(None, 32, 32, 32)	0	inference_0_conv2d_1[0][0]
inference_0_batch_norm_1 (Batch	(None, 32, 32, 32)	128	inference_0_dropout_1[0][0]
inference_0_relu_1 (ReLU)	(None, 32, 32, 32)	0	inference_0_batch_norm_1[0][0]
inference_1_conv2d_0 (Conv2D)	(None, 16, 16, 64)	32832	inference_0_relu_1[0][0]
inference_1_dropout_0 (Dropout)	(None, 16, 16, 64)	0	inference_1_conv2d_0[0][0]
inference_1_batch_norm_0 (Batch	(None, 16, 16, 64)	256	inference_1_dropout_0[0][0]
inference_1_relu_0 (ReLU)	(None, 16, 16, 64)	0	inference_1_batch_norm_0[0][0]
inference_1_conv2d_1 (Conv2D)	(None, 16, 16, 64)	65600	inference_1_relu_0[0][0]
inference_1_dropout_1 (Dropout)	(None, 16, 16, 64)	0	inference_1_conv2d_1[0][0]
inference_1_batch_norm_1 (Batch	(None, 16, 16, 64)	256	inference_1_dropout_1[0][0]
inference_1_relu_1 (ReLU)	(None, 16, 16, 64)	0	inference_1_batch_norm_1[0][0]
ladder_2_conv2d_0 (Conv2D)	(None, 8, 8, 256)	262400	inference_1_relu_1[0][0]
ladder_2_dropout_0 (Dropout)	(None, 8, 8, 256)	0	ladder_2_conv2d_0[0][0]
ladder_2_batch_norm_0 (BatchNor	(None, 8, 8, 256)	1024	ladder_2_dropout_0[0][0]
ladder_2_relu_0 (ReLU)	(None, 8, 8, 256)	0	ladder_2_batch_norm_0[0][0]
ladder_0_conv2d_0 (Conv2D)	(None, 32, 32, 128)	2176	input_1[0][0]
ladder_1_conv2d_0 (Conv2D)	(None, 16, 16, 256)	131328	inference_0_relu_1[0][0]
ladder_2_conv2d_1 (Conv2D)	(None, 4, 4, 512)	2097664	ladder_2_relu_0[0][0]
ladder_0_dropout_0 (Dropout)	(None, 32, 32, 128)	0	ladder_0_conv2d_0[0][0]
ladder_1_dropout_0 (Dropout)	(None, 16, 16, 256)	0	ladder_1_conv2d_0[0][0]
ladder_2_dropout_1 (Dropout)	(None, 4, 4, 512)	0	ladder_2_conv2d_1[0][0]
ladder_0_batch_norm_0 (BatchNor	(None, 32, 32, 128)	512	ladder_0_dropout_0[0][0]
ladder_1_batch_norm_0 (BatchNor	(None, 16, 16, 256)	1024	ladder_1_dropout_0[0][0]
ladder_2_batch_norm_1 (BatchNor	(None, 4, 4, 512)	2048	ladder_2_dropout_1[0][0]
ladder_0_relu_0 (ReLU)	(None, 32, 32, 128)	0	ladder_0_batch_norm_0[0][0]
ladder_1_relu_0 (ReLU)	(None, 16, 16, 256)	0	ladder_1_batch_norm_0[0][0]
ladder_2_relu_1 (ReLU)	(None, 4, 4, 512)	0	ladder_2_batch_norm_1[0][0]
ladder_0_conv2d_1 (Conv2D)	(None, 32, 32, 128)	262272	ladder_0_relu_0[0][0]

ladder_1_conv2d_1 (Conv2D)	(None, 16, 16, 256)	1048832	ladder_1_relu_0[0][0]
ladder_2_conv2d_2 (Conv2D)	(None, 2, 2, 1024)	8389632	ladder_2_relu_1[0][0]
ladder_0_dropout_1 (Dropout)	(None, 32, 32, 128)	0	ladder_0_conv2d_1[0][0]
ladder_1_dropout_1 (Dropout)	(None, 16, 16, 256)	0	ladder_1_conv2d_1[0][0]
ladder_2_dropout_2 (Dropout)	(None, 2, 2, 1024)	0	ladder_2_conv2d_2[0][0]
ladder_0_batch_norm_1 (BatchNor	(None, 32, 32, 128)	512	ladder_0_dropout_1[0][0]
ladder_1_batch_norm_1 (BatchNor	(None, 16, 16, 256)	1024	ladder_1_dropout_1[0][0]
ladder_2_batch_norm_2 (BatchNor	(None, 2, 2, 1024)	4096	ladder_2_dropout_2[0][0]
ladder_0_relu_1 (ReLU)	(None, 32, 32, 128)	0	ladder_0_batch_norm_1[0][0]
ladder_1_relu_1 (ReLU)	(None, 16, 16, 256)	0	ladder_1_batch_norm_1[0][0]
ladder_2_relu_2 (ReLU)	(None, 2, 2, 1024)	0	ladder_2_batch_norm_2[0][0]
ladder_0_flatten (Flatten)	(None, 131072)	0	ladder_0_relu_1[0][0]
ladder_1_flatten (Flatten)	(None, 65536)	0	ladder_1_relu_1[0][0]
ladder_2_flatten (Flatten)	(None, 4096)	0	ladder_2_relu_2[0][0]
ladder_0_dropout (Dropout)	(None, 131072)	0	ladder_0_flatten[0][0]
ladder_1_dropout (Dropout)	(None, 65536)	0	ladder_1_flatten[0][0]
ladder_2_dropout (Dropout)	(None, 4096)	0	ladder_2_flatten[0][0]
mu_1 (Dense)	(None, 4)	524292	ladder_0_dropout[0][0]
log_var_1 (Dense)	(None, 4)	524292	ladder_0_dropout[0][0]
mu_2 (Dense)	(None, 4)	262148	ladder_1_dropout[0][0]
log_var_2 (Dense)	(None, 4)	262148	ladder_1_dropout[0][0]
mu_3 (Dense)	(None, 4)	16388	ladder_2_dropout[0][0]
log_var_3 (Dense)	(None, 4)	16388	ladder_2_dropout[0][0]
z_1_latent (Lambda)	(None, 4)	0	mu_1[0][0] log_var_1[0][0]
z_2_latent (Lambda)	(None, 4)	0	mu_2[0][0] log_var_2[0][0]
z_3_latent (Lambda)	(None, 4)	0	mu_3[0][0] log_var_3[0][0]

Total params: 13,926,360
Trainable params: 13,920,856
Non-trainable params: 5,504

Listing 13: dSprites-VLAE Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 4)	0	
generative_2_dense_0 (Dense)	(None, 1024)	5120	z_3[0][0]
generative_2_dropout_0 (Dropout	(None, 1024)	0	generative_2_dense_0[0][0]
generative_2_batch_norm_0 (Batc	(None, 1024)	4096	generative_2_dropout_0[0][0]
generative_2_relu_0 (ReLU)	(None, 1024)	0	generative_2_batch_norm_0[0][0]
generative_2_dense_1 (Dense)	(None, 1024)	1049600	generative_2_relu_0[0][0]
generative_2_dropout_1 (Dropout	(None, 1024)	0	generative_2_dense_1[0][0]
generative_2_batch_norm_1 (Batc	(None, 1024)	4096	generative_2_dropout_1[0][0]
generative_2_relu_1 (ReLU)	(None, 1024)	0	generative_2_batch_norm_1[0][0]
z_2 (InputLayer)	(None, 4)	0	
concatenate_2_and_1 (Concatenat	(None, 1028)	0	generative_2_relu_1[0][0] z_2[0][0]
generative_1_dense_0 (Dense)	(None, 1024)	1053696	concatenate_2_and_1[0][0]
generative_1_dropout_0 (Dropout	(None, 1024)	0	generative_1_dense_0[0][0]
generative_1_batch_norm_0 (Batc	(None, 1024)	4096	generative_1_dropout_0[0][0]
generative_1_relu_0 (ReLU)	(None, 1024)	0	generative_1_batch_norm_0[0][0]
generative_1_dense_1 (Dense)	(None, 1024)	1049600	generative_1_relu_0[0][0]
generative_1_dropout_1 (Dropout	(None, 1024)	0	generative_1_dense_1[0][0]
generative_1_batch_norm_1 (Batc	(None, 1024)	4096	generative_1_dropout_1[0][0]
generative_1_relu_1 (ReLU)	(None, 1024)	0	generative_1_batch_norm_1[0][0]
z_1 (InputLayer)	(None, 4)	0	
concatenate_1_and_0 (Concatenat	(None, 1028)	0	generative_1_relu_1[0][0] z_1[0][0]
generative_0_dense_0 (Dense)	(None, 4096)	4214784	concatenate_1_and_0[0][0]
generative_0_dropout_0 (Dropout	(None, 4096)	0	generative_0_dense_0[0][0]
generative_0_batch_norm_0 (Batc	(None, 4096)	16384	generative_0_dropout_0[0][0]
generative_0_relu_0 (ReLU)	(None, 4096)	0	generative_0_batch_norm_0[0][0]
generative_0_reshape_0 (Reshape	(None, 2, 2, 1024)	0	generative_0_relu_0[0][0]

generative_0_conv2d_transpose_0 (None, 4, 4, 1024)	16778240	generative_0_reshape_0[0][0]
generative_0_dropout_1 (Dropout (None, 4, 4, 1024)	0	generative_0_conv2d_transpose_0[0
generative_0_batch_norm_1 (Batc (None, 4, 4, 1024)	4096	generative_0_dropout_1[0][0]
generative_0_leaky_relu_transpo (None, 4, 4, 1024)	0	generative_0_batch_norm_1[0][0]
generative_0_conv2d_transpose_1 (None, 8, 8, 512)	8389120	generative_0_leaky_relu_transpose
generative_0_dropout_2 (Dropout (None, 8, 8, 512)	0	generative_0_conv2d_transpose_1[0
generative_0_batch_norm_2 (Batc (None, 8, 8, 512)	2048	generative_0_dropout_2[0][0]
generative_0_leaky_relu_transpo (None, 8, 8, 512)	0	generative_0_batch_norm_2[0][0]
generative_0_conv2d_transpose_2 (None, 16, 16, 256)	2097408	generative_0_leaky_relu_transpose
generative_0_dropout_3 (Dropout (None, 16, 16, 256)	0	generative_0_conv2d_transpose_2[0
generative_0_batch_norm_3 (Batc (None, 16, 16, 256)	1024	generative_0_dropout_3[0][0]
generative_0_leaky_relu_transpo (None, 16, 16, 256)	0	generative_0_batch_norm_3[0][0]
generative_0_conv2d_transpose_3 (None, 32, 32, 128)	524416	generative_0_leaky_relu_transpose
generative_0_dropout_4 (Dropout (None, 32, 32, 128)	0	generative_0_conv2d_transpose_3[0
generative_0_batch_norm_4 (Batc (None, 32, 32, 128)	512	generative_0_dropout_4[0][0]
generative_0_leaky_relu_transpo (None, 32, 32, 128)	0	generative_0_batch_norm_4[0][0]
generative_0_conv2d_transpose_4 (None, 64, 64, 1)	2049	generative_0_leaky_relu_transpose
generative_0_dropout_5 (Dropout (None, 64, 64, 1)	0	generative_0_conv2d_transpose_4[0
generative_0_sigmoid_0 (Activat (None, 64, 64, 1)	0	generative_0_dropout_5[0][0]

Total params: 35,204,481
Trainable params: 35,184,257
Non-trainable params: 20,224

Listing 14: dSprites-VLAE Decoder

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 128, 128, 3)	0	
inference_0_conv2d_0 (Conv2D)	(None, 64, 64, 32)	1568	input_1[0][0]
inference_0_dropout_0 (Dropout)	(None, 64, 64, 32)	0	inference_0_conv2d_0[0][0]
inference_0_batch_norm_0 (Batch	(None, 64, 64, 32)	128	inference_0_dropout_0[0][0]
inference_0_relu_0 (ReLU)	(None, 64, 64, 32)	0	inference_0_batch_norm_0[0][0]
inference_0_conv2d_1 (Conv2D)	(None, 64, 64, 32)	16416	inference_0_relu_0[0][0]
inference_0_dropout_1 (Dropout)	(None, 64, 64, 32)	0	inference_0_conv2d_1[0][0]
inference_0_batch_norm_1 (Batch	(None, 64, 64, 32)	128	inference_0_dropout_1[0][0]
inference_0_relu_1 (ReLU)	(None, 64, 64, 32)	0	inference_0_batch_norm_1[0][0]
inference_1_conv2d_0 (Conv2D)	(None, 32, 32, 64)	32832	inference_0_relu_1[0][0]
inference_1_dropout_0 (Dropout)	(None, 32, 32, 64)	0	inference_1_conv2d_0[0][0]
inference_1_batch_norm_0 (Batch	(None, 32, 32, 64)	256	inference_1_dropout_0[0][0]
inference_1_relu_0 (ReLU)	(None, 32, 32, 64)	0	inference_1_batch_norm_0[0][0]
inference_1_conv2d_1 (Conv2D)	(None, 32, 32, 64)	65600	inference_1_relu_0[0][0]
inference_1_dropout_1 (Dropout)	(None, 32, 32, 64)	0	inference_1_conv2d_1[0][0]
inference_1_batch_norm_1 (Batch	(None, 32, 32, 64)	256	inference_1_dropout_1[0][0]
inference_1_relu_1 (ReLU)	(None, 32, 32, 64)	0	inference_1_batch_norm_1[0][0]
ladder_2_conv2d_0 (Conv2D)	(None, 16, 16, 256)	262400	inference_1_relu_1[0][0]
ladder_2_dropout_0 (Dropout)	(None, 16, 16, 256)	0	ladder_2_conv2d_0[0][0]
ladder_2_batch_norm_0 (BatchNor	(None, 16, 16, 256)	1024	ladder_2_dropout_0[0][0]
ladder_2_relu_0 (ReLU)	(None, 16, 16, 256)	0	ladder_2_batch_norm_0[0][0]
ladder_2_conv2d_1 (Conv2D)	(None, 8, 8, 512)	2097664	ladder_2_relu_0[0][0]
ladder_2_dropout_1 (Dropout)	(None, 8, 8, 512)	0	ladder_2_conv2d_1[0][0]
ladder_2_batch_norm_1 (BatchNor	(None, 8, 8, 512)	2048	ladder_2_dropout_1[0][0]
ladder_2_relu_1 (ReLU)	(None, 8, 8, 512)	0	ladder_2_batch_norm_1[0][0]
ladder_0_conv2d_0 (Conv2D)	(None, 64, 64, 128)	6272	input_1[0][0]
ladder_1_conv2d_0 (Conv2D)	(None, 32, 32, 256)	131328	inference_0_relu_1[0][0]
ladder_2_conv2d_2 (Conv2D)	(None, 8, 8, 512)	4194816	ladder_2_relu_1[0][0]
ladder_0_dropout_0 (Dropout)	(None, 64, 64, 128)	0	ladder_0_conv2d_0[0][0]
ladder_1_dropout_0 (Dropout)	(None, 32, 32, 256)	0	ladder_1_conv2d_0[0][0]
ladder_2_dropout_2 (Dropout)	(None, 8, 8, 512)	0	ladder_2_conv2d_2[0][0]
ladder_0_batch_norm_0 (BatchNor	(None, 64, 64, 128)	512	ladder_0_dropout_0[0][0]
ladder_1_batch_norm_0 (BatchNor	(None, 32, 32, 256)	1024	ladder_1_dropout_0[0][0]
ladder_2_batch_norm_2 (BatchNor	(None, 8, 8, 512)	2048	ladder_2_dropout_2[0][0]
ladder_0_relu_0 (ReLU)	(None, 64, 64, 128)	0	ladder_0_batch_norm_0[0][0]
ladder_1_relu_0 (ReLU)	(None, 32, 32, 256)	0	ladder_1_batch_norm_0[0][0]
ladder_2_relu_2 (ReLU)	(None, 8, 8, 512)	0	ladder_2_batch_norm_2[0][0]

ladder_0_conv2d_1 (Conv2D)	(None, 64, 64, 128)	262272	ladder_0_relu_0[0][0]
ladder_1_conv2d_1 (Conv2D)	(None, 32, 32, 256)	1048832	ladder_1_relu_0[0][0]
ladder_2_conv2d_3 (Conv2D)	(None, 8, 8, 512)	4194816	ladder_2_relu_2[0][0]
ladder_0_dropout_1 (Dropout)	(None, 64, 64, 128)	0	ladder_0_conv2d_1[0][0]
ladder_1_dropout_1 (Dropout)	(None, 32, 32, 256)	0	ladder_1_conv2d_1[0][0]
ladder_2_dropout_3 (Dropout)	(None, 8, 8, 512)	0	ladder_2_conv2d_3[0][0]
ladder_0_batch_norm_1 (BatchNor	(None, 64, 64, 128)	512	ladder_0_dropout_1[0][0]
ladder_1_batch_norm_1 (BatchNor	(None, 32, 32, 256)	1024	ladder_1_dropout_1[0][0]
ladder_2_batch_norm_3 (BatchNor	(None, 8, 8, 512)	2048	ladder_2_dropout_3[0][0]
ladder_0_relu_1 (ReLU)	(None, 64, 64, 128)	0	ladder_0_batch_norm_1[0][0]
ladder_1_relu_1 (ReLU)	(None, 32, 32, 256)	0	ladder_1_batch_norm_1[0][0]
ladder_2_relu_3 (ReLU)	(None, 8, 8, 512)	0	ladder_2_batch_norm_3[0][0]
ladder_0_flatten (Flatten)	(None, 524288)	0	ladder_0_relu_1[0][0]
ladder_1_flatten (Flatten)	(None, 262144)	0	ladder_1_relu_1[0][0]
ladder_2_flatten (Flatten)	(None, 32768)	0	ladder_2_relu_3[0][0]
ladder_0_dropout (Dropout)	(None, 524288)	0	ladder_0_flatten[0][0]
ladder_1_dropout (Dropout)	(None, 262144)	0	ladder_1_flatten[0][0]
ladder_2_dropout (Dropout)	(None, 32768)	0	ladder_2_flatten[0][0]
mu_1 (Dense)	(None, 2)	1048578	ladder_0_dropout[0][0]
log_var_1 (Dense)	(None, 2)	1048578	ladder_0_dropout[0][0]
mu_2 (Dense)	(None, 2)	524290	ladder_1_dropout[0][0]
log_var_2 (Dense)	(None, 2)	524290	ladder_1_dropout[0][0]
mu_3 (Dense)	(None, 2)	65538	ladder_2_dropout[0][0]
log_var_3 (Dense)	(None, 2)	65538	ladder_2_dropout[0][0]
z_1_latent (Lambda)	(None, 2)	0	mu_1[0][0] log_var_1[0][0]
z_2_latent (Lambda)	(None, 2)	0	mu_2[0][0] log_var_2[0][0]
z_3_latent (Lambda)	(None, 2)	0	mu_3[0][0] log_var_3[0][0]

Total params: 15,602,636
Trainable params: 15,597,132
Non-trainable params: 5,504

Listing 15: CelebA-VLAE Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 2)	0	
generative_2_dense_0 (Dense)	(None, 256)	768	z_3[0][0]
generative_2_dropout_0 (Dropout	(None, 256)	0	generative_2_dense_0[0][0]
generative_2_batch_norm_0 (Batc	(None, 256)	1024	generative_2_dropout_0[0][0]
generative_2_relu_0 (ReLU)	(None, 256)	0	generative_2_batch_norm_0[0][0]
generative_2_dense_1 (Dense)	(None, 512)	131584	generative_2_relu_0[0][0]
generative_2_dropout_1 (Dropout	(None, 512)	0	generative_2_dense_1[0][0]
generative_2_batch_norm_1 (Batc	(None, 512)	2048	generative_2_dropout_1[0][0]
generative_2_relu_1 (ReLU)	(None, 512)	0	generative_2_batch_norm_1[0][0]
z_2 (InputLayer)	(None, 2)	0	
concatenate_2_and_1 (Concatenat	(None, 514)	0	generative_2_relu_1[0][0] z_2[0][0]
generative_1_dense_0 (Dense)	(None, 512)	263680	concatenate_2_and_1[0][0]
generative_1_dropout_0 (Dropout	(None, 512)	0	generative_1_dense_0[0][0]
generative_1_batch_norm_0 (Batc	(None, 512)	2048	generative_1_dropout_0[0][0]
generative_1_relu_0 (ReLU)	(None, 512)	0	generative_1_batch_norm_0[0][0]
generative_1_dense_1 (Dense)	(None, 1024)	525312	generative_1_relu_0[0][0]
generative_1_dropout_1 (Dropout	(None, 1024)	0	generative_1_dense_1[0][0]
generative_1_batch_norm_1 (Batc	(None, 1024)	4096	generative_1_dropout_1[0][0]
generative_1_relu_1 (ReLU)	(None, 1024)	0	generative_1_batch_norm_1[0][0]
z_1 (InputLayer)	(None, 2)	0	
concatenate_1_and_0 (Concatenat	(None, 1026)	0	generative_1_relu_1[0][0] z_1[0][0]
generative_0_dense_0 (Dense)	(None, 32768)	33652736	concatenate_1_and_0[0][0]
generative_0_dropout_0 (Dropout	(None, 32768)	0	generative_0_dense_0[0][0]
generative_0_batch_norm_0 (Batc	(None, 32768)	131072	generative_0_dropout_0[0][0]
generative_0_relu_0 (ReLU)	(None, 32768)	0	generative_0_batch_norm_0[0][0]

generative_0_reshape_0 (Reshape	(None, 8, 8, 512)	0	generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0	(None, 8, 8, 1024)	8389632	generative_0_reshape_0[0][0]
generative_0_dropout_1 (Dropout	(None, 8, 8, 1024)	0	generative_0_conv2d_transpose_0[0
generative_0_batch_norm_1 (Batc	(None, 8, 8, 1024)	4096	generative_0_dropout_1[0][0]
generative_0_leaky_relu_transpo	(None, 8, 8, 1024)	0	generative_0_batch_norm_1[0][0]
generative_0_conv2d_transpose_1	(None, 16, 16, 512)	8389120	generative_0_leaky_relu_transpose
generative_0_dropout_2 (Dropout	(None, 16, 16, 512)	0	generative_0_conv2d_transpose_1[0
generative_0_batch_norm_2 (Batc	(None, 16, 16, 512)	2048	generative_0_dropout_2[0][0]
generative_0_leaky_relu_transpo	(None, 16, 16, 512)	0	generative_0_batch_norm_2[0][0]
generative_0_conv2d_transpose_2	(None, 32, 32, 256)	2097408	generative_0_leaky_relu_transpose
generative_0_dropout_3 (Dropout	(None, 32, 32, 256)	0	generative_0_conv2d_transpose_2[0
generative_0_batch_norm_3 (Batc	(None, 32, 32, 256)	1024	generative_0_dropout_3[0][0]
generative_0_leaky_relu_transpo	(None, 32, 32, 256)	0	generative_0_batch_norm_3[0][0]
generative_0_conv2d_transpose_3	(None, 64, 64, 128)	524416	generative_0_leaky_relu_transpose
generative_0_dropout_4 (Dropout	(None, 64, 64, 128)	0	generative_0_conv2d_transpose_3[0
generative_0_batch_norm_4 (Batc	(None, 64, 64, 128)	512	generative_0_dropout_4[0][0]
generative_0_leaky_relu_transpo	(None, 64, 64, 128)	0	generative_0_batch_norm_4[0][0]
generative_0_conv2d_transpose_4	(None, 128, 128, 64)	131136	generative_0_leaky_relu_transpose
generative_0_dropout_5 (Dropout	(None, 128, 128, 64)	0	generative_0_conv2d_transpose_4[0
generative_0_batch_norm_5 (Batc	(None, 128, 128, 64)	256	generative_0_dropout_5[0][0]
generative_0_leaky_relu_transpo	(None, 128, 128, 64)	0	generative_0_batch_norm_5[0][0]
generative_0_conv2d_transpose_5	(None, 128, 128, 3)	3075	generative_0_leaky_relu_transpose
generative_0_dropout_6 (Dropout	(None, 128, 128, 3)	0	generative_0_conv2d_transpose_5[0
generative_0_sigmoid_0 (Activat	(None, 128, 128, 3)	0	generative_0_dropout_6[0][0]

Total params: 54,257,091
Trainable params: 54,182,979
Non-trainable params: 74,112

Listing 16: CelebA-VLAE Decoder

A.3 VAE-GAN-models

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 28, 28, 1)	0	
encoder_conv_0 (Conv2D)	(None, 14, 14, 32)	320	encoder_input[0][0]
batch_normalization_2 (BatchNor	(None, 14, 14, 32)	128	encoder_conv_0[0][0]
re_lu_1 (ReLU)	(None, 14, 14, 32)	0	batch_normalization_2[0][0]
encoder_conv_1 (Conv2D)	(None, 7, 7, 64)	18496	re_lu_1[0][0]
batch_normalization_3 (BatchNor	(None, 7, 7, 64)	256	encoder_conv_1[0][0]
re_lu_2 (ReLU)	(None, 7, 7, 64)	0	batch_normalization_3[0][0]
encoder_conv_2 (Conv2D)	(None, 7, 7, 128)	73856	re_lu_2[0][0]
batch_normalization_4 (BatchNor	(None, 7, 7, 128)	512	encoder_conv_2[0][0]
re_lu_3 (ReLU)	(None, 7, 7, 128)	0	batch_normalization_4[0][0]
encoder_conv_3 (Conv2D)	(None, 7, 7, 256)	295168	re_lu_3[0][0]
batch_normalization_5 (BatchNor	(None, 7, 7, 256)	1024	encoder_conv_3[0][0]
re_lu_4 (ReLU)	(None, 7, 7, 256)	0	batch_normalization_5[0][0]
flatten_2 (Flatten)	(None, 12544)	0	re_lu_4[0][0]
mu (Dense)	(None, 2)	25090	flatten_2[0][0]
log_var (Dense)	(None, 2)	25090	flatten_2[0][0]
encoder_output (Lambda)	(None, 2)	0	mu[0][0] log_var[0][0]

Total params: 439,940
Trainable params: 438,980
Non-trainable params: 960

Listing 17: Mnist-VAE-GAN Encoder

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 2)	0
dense_2 (Dense)	(None, 12544)	37632
reshape_1 (Reshape)	(None, 7, 7, 256)	0
decoder_conv_t_0 (Conv2DTran	(None, 7, 7, 256)	1048576
batch_normalization_6 (Batch	(None, 7, 7, 256)	1024

```
re_lu_5 (ReLU)                  (None, 7, 7, 256)         0
decoder_conv_t_1 (Conv2DTran    (None, 7, 7, 128)         524288
batch_normalization_7 (Batch    (None, 7, 7, 128)         512
re_lu_6 (ReLU)                  (None, 7, 7, 128)         0
decoder_conv_t_2 (Conv2DTran    (None, 14, 14, 64)        131072
batch_normalization_8 (Batch    (None, 14, 14, 64)        256
re_lu_7 (ReLU)                  (None, 14, 14, 64)        0
decoder_conv_t_3 (Conv2DTran    (None, 28, 28, 1)         1024
activation_2 (Activation)       (None, 28, 28, 1)         0
=================================================================
Total params: 1,744,384
Trainable params: 1,743,488
Non-trainable params: 896
_________________________________________________________________
```

Listing 18: MNIST-VAE-GAN Decoder

```
Layer (type)                 Output Shape              Param #
=================================================================
discriminator_input (InputLa (None, 28, 28, 1)         0
conv2d_1 (Conv2D)            (None, 14, 14, 64)        1088
leaky_re_lu_1 (LeakyReLU)    (None, 14, 14, 64)        0
conv2d_2 (Conv2D)            (None, 7, 7, 128)         131200
batch_normalization_1 (Batch (None, 7, 7, 128)         512
conv2d_3 (Conv2D)            (None, 2, 2, 1)           129
flatten_1 (Flatten)          (None, 4)                 0
dense_1 (Dense)              (None, 1)                 5
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 132,934
Trainable params: 132,678
Non-trainable params: 256
_________________________________________________________________
```

Listing 19: MNIST-VAE-GAN Discriminator

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
encoder_input (InputLayer)   (None, 64, 64, 1)         0
encoder_conv_0 (Conv2D)      (None, 32, 32, 32)        544         encoder_input[0][0]
batch_normalization_3 (BatchNor (None, 32, 32, 32)     128         encoder_conv_0[0][0]
re_lu_1 (ReLU)               (None, 32, 32, 32)        0           batch_normalization_3[0][0]
encoder_conv_1 (Conv2D)      (None, 16, 16, 64)        32832       re_lu_1[0][0]
batch_normalization_4 (BatchNor (None, 16, 16, 64)     256         encoder_conv_1[0][0]
re_lu_2 (ReLU)               (None, 16, 16, 64)        0           batch_normalization_4[0][0]
encoder_conv_2 (Conv2D)      (None, 8, 8, 128)         131200      re_lu_2[0][0]
batch_normalization_5 (BatchNor (None, 8, 8, 128)      512         encoder_conv_2[0][0]
re_lu_3 (ReLU)               (None, 8, 8, 128)         0           batch_normalization_5[0][0]
encoder_conv_3 (Conv2D)      (None, 4, 4, 256)         524544      re_lu_3[0][0]
batch_normalization_6 (BatchNor (None, 4, 4, 256)      1024        encoder_conv_3[0][0]
re_lu_4 (ReLU)               (None, 4, 4, 256)         0           batch_normalization_6[0][0]
encoder_conv_4 (Conv2D)      (None, 4, 4, 512)         2097664     re_lu_4[0][0]
batch_normalization_7 (BatchNor (None, 4, 4, 512)      2048        encoder_conv_4[0][0]
re_lu_5 (ReLU)               (None, 4, 4, 512)         0           batch_normalization_7[0][0]
flatten_2 (Flatten)          (None, 8192)              0           re_lu_5[0][0]
mu (Dense)                   (None, 10)                81930       flatten_2[0][0]
log_var (Dense)              (None, 10)                81930       flatten_2[0][0]
encoder_output (Lambda)      (None, 10)                0           mu[0][0]
                                                                   log_var[0][0]
==================================================================================================
Total params: 2,954,612
Trainable params: 2,952,628
Non-trainable params: 1,984
__________________________________________________________________________________________________
```

Listing 20: dSprites-VAE-GAN Encoder

```
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   (None, 10)                0
dense_2 (Dense)              (None, 8192)              90112
reshape_1 (Reshape)          (None, 4, 4, 512)         0
_________________________________________________________________
```

```
decoder_conv_t_0 (Conv2DTran (None, 8, 8, 512)          4194304
batch_normalization_8 (Batch (None, 8, 8, 512)          2048
re_lu_6 (ReLU)               (None, 8, 8, 512)          0
decoder_conv_t_1 (Conv2DTran (None, 16, 16, 256)        2097152
batch_normalization_9 (Batch (None, 16, 16, 256)        1024
re_lu_7 (ReLU)               (None, 16, 16, 256)        0
decoder_conv_t_2 (Conv2DTran (None, 32, 32, 128)        524288
batch_normalization_10 (Batc (None, 32, 32, 128)        512
re_lu_8 (ReLU)               (None, 32, 32, 128)        0
decoder_conv_t_3 (Conv2DTran (None, 64, 64, 1)          2048
activation_2 (Activation)    (None, 64, 64, 1)          0
=================================================================
Total params: 6,911,488
Trainable params: 6,909,696
Non-trainable params: 1,792
_________________________________________________________________
```

Listing 21: dSprites-VAE-GAN Decoder

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
discriminator_input (InputLa (None, 64, 64, 1)         0
conv2d_1 (Conv2D)            (None, 32, 32, 64)        1088
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 64)        0
conv2d_2 (Conv2D)            (None, 16, 16, 128)       131200
batch_normalization_1 (Batch (None, 16, 16, 128)       512
conv2d_3 (Conv2D)            (None, 8, 8, 256)         524544
batch_normalization_2 (Batch (None, 8, 8, 256)         1024
conv2d_4 (Conv2D)            (None, 2, 2, 1)           257
flatten_1 (Flatten)          (None, 4)                 0
dense_1 (Dense)              (None, 1)                 5
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 658,630
Trainable params: 657,862
Non-trainable params: 768
_________________________________________________________________
```

Listing 22: dSprites-VAE-GAN Discriminator

```
_____________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=============================================================================================
encoder_input (InputLayer)   (None, 128, 128, 3)       0
_____________________________________________________________________________________________
encoder_conv_0 (Conv2D)      (None, 64, 64, 32)        1568        encoder_input[0][0]
_____________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 64, 64, 32)     128         encoder_conv_0[0][0]
_____________________________________________________________________________________________
re_lu_1 (ReLU)               (None, 64, 64, 32)        0           batch_normalization_4[0][0]
_____________________________________________________________________________________________
encoder_conv_1 (Conv2D)      (None, 32, 32, 64)        32832       re_lu_1[0][0]
_____________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 32, 32, 64)     256         encoder_conv_1[0][0]
_____________________________________________________________________________________________
re_lu_2 (ReLU)               (None, 32, 32, 64)        0           batch_normalization_5[0][0]
_____________________________________________________________________________________________
encoder_conv_2 (Conv2D)      (None, 16, 16, 128)       131200      re_lu_2[0][0]
_____________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 16, 16, 128)    512         encoder_conv_2[0][0]
_____________________________________________________________________________________________
re_lu_3 (ReLU)               (None, 16, 16, 128)       0           batch_normalization_6[0][0]
_____________________________________________________________________________________________
encoder_conv_3 (Conv2D)      (None, 8, 8, 256)         524544      re_lu_3[0][0]
_____________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 8, 8, 256)      1024        encoder_conv_3[0][0]
_____________________________________________________________________________________________
re_lu_4 (ReLU)               (None, 8, 8, 256)         0           batch_normalization_7[0][0]
_____________________________________________________________________________________________
encoder_conv_4 (Conv2D)      (None, 8, 8, 512)         2097664     re_lu_4[0][0]
_____________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 8, 8, 512)      2048        encoder_conv_4[0][0]
_____________________________________________________________________________________________
re_lu_5 (ReLU)               (None, 8, 8, 512)         0           batch_normalization_8[0][0]
_____________________________________________________________________________________________
flatten_2 (Flatten)          (None, 32768)             0           re_lu_5[0][0]
_____________________________________________________________________________________________
mu (Dense)                   (None, 8)                 262152      flatten_2[0][0]
_____________________________________________________________________________________________
log_var (Dense)              (None, 8)                 262152      flatten_2[0][0]
_____________________________________________________________________________________________
encoder_output (Lambda)      (None, 8)                 0           mu[0][0]
                                                                   log_var[0][0]
=============================================================================================
Total params: 3,316,080
Trainable params: 3,314,096
Non-trainable params: 1,984
_____________________________________________________________________________________________
```

Listing 23: CelebA-VAE-GAN Encoder

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
decoder_input (InputLayer)   (None, 8)                0
dense_2 (Dense)              (None, 32768)             294912
reshape_1 (Reshape)          (None, 8, 8, 512)         0
decoder_conv_t_0 (Conv2DTran (None, 16, 16, 512)       4194304
batch_normalization_9 (Batch (None, 16, 16, 512)       2048
re_lu_6 (ReLU)               (None, 16, 16, 512)       0
decoder_conv_t_1 (Conv2DTran (None, 32, 32, 256)       2097152
batch_normalization_10 (Batc (None, 32, 32, 256)       1024
re_lu_7 (ReLU)               (None, 32, 32, 256)       0
decoder_conv_t_2 (Conv2DTran (None, 64, 64, 128)       524288
batch_normalization_11 (Batc (None, 64, 64, 128)       512
re_lu_8 (ReLU)               (None, 64, 64, 128)       0
decoder_conv_t_3 (Conv2DTran (None, 128, 128, 3)       6144
activation_2 (Activation)    (None, 128, 128, 3)       0
Total params: 7,120,384
Trainable params: 7,118,592
Non-trainable params: 1,792
```

Listing 24: CelebA-VAE-GAN Decoder

```
Layer (type)                 Output Shape              Param #
discriminator_input (InputLa (None, 128, 128, 3)       0
conv2d_1 (Conv2D)            (None, 64, 64, 64)        3136
leaky_re_lu_1 (LeakyReLU)    (None, 64, 64, 64)        0
conv2d_2 (Conv2D)            (None, 32, 32, 128)       131200
batch_normalization_1 (Batch (None, 32, 32, 128)       512
conv2d_3 (Conv2D)            (None, 16, 16, 256)       524544
batch_normalization_2 (Batch (None, 16, 16, 256)       1024
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 256)       0
conv2d_4 (Conv2D)            (None, 8, 8, 512)         2097664
batch_normalization_3 (Batch (None, 8, 8, 512)         2048
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 512)         0
conv2d_5 (Conv2D)            (None, 2, 2, 1)           513
flatten_1 (Flatten)          (None, 4)                 0
dense_1 (Dense)              (None, 1)                 5
activation_1 (Activation)    (None, 1)                 0
Total params: 2,760,646
Trainable params: 2,758,854
Non-trainable params: 1,792
```

Listing 25: CelebA-VAE-GAN Discriminator

A.4 VLAE-GAN-models

```
Layer (type)                 Output Shape              Param #     Connected to
vlae_gan_encoder_input (InputLa (None, 28, 28, 1)      0
inference_0_conv2d_0 (Conv2D) (None, 14, 14, 64)       1664        vlae_gan_encoder_input[0][0]
inference_0_relu_0 (ReLU)    (None, 14, 14, 64)        0           inference_0_conv2d_0[0][0]
inference_1_conv2d_0 (Conv2D) (None, 7, 7, 128)        73856       inference_0_relu_0[0][0]
inference_1_relu_0 (ReLU)    (None, 7, 7, 128)         0           inference_1_conv2d_0[0][0]
ladder_2_conv2d_0 (Conv2D)   (None, 7, 7, 256)         295168      inference_1_relu_0[0][0]
ladder_2_relu_0 (ReLU)       (None, 7, 7, 256)         0           ladder_2_conv2d_0[0][0]
ladder_0_conv2d_0 (Conv2D)   (None, 14, 14, 64)        1664        vlae_gan_encoder_input[0][0]
ladder_1_conv2d_0 (Conv2D)   (None, 7, 7, 128)         204928      inference_0_relu_0[0][0]
ladder_2_conv2d_1 (Conv2D)   (None, 7, 7, 512)         1180160     ladder_2_relu_0[0][0]
ladder_0_relu_0 (ReLU)       (None, 14, 14, 64)        0           ladder_0_conv2d_0[0][0]
ladder_1_relu_0 (ReLU)       (None, 7, 7, 128)         0           ladder_1_conv2d_0[0][0]
ladder_2_relu_1 (ReLU)       (None, 7, 7, 512)         0           ladder_2_conv2d_1[0][0]
ladder_0_flatten (Flatten)   (None, 12544)             0           ladder_0_relu_0[0][0]
ladder_1_flatten (Flatten)   (None, 6272)              0           ladder_1_relu_0[0][0]
ladder_2_flatten (Flatten)   (None, 25088)             0           ladder_2_relu_1[0][0]
mu_1 (Dense)                 (None, 2)                 25090       ladder_0_flatten[0][0]
```

log_var_1 (Dense)	(None, 2)	25090	ladder_0_flatten[0][0]
mu_2 (Dense)	(None, 2)	12546	ladder_1_flatten[0][0]
log_var_2 (Dense)	(None, 2)	12546	ladder_1_flatten[0][0]
mu_3 (Dense)	(None, 2)	50178	ladder_2_flatten[0][0]
log_var_3 (Dense)	(None, 2)	50178	ladder_2_flatten[0][0]
z_1_latent (Lambda)	(None, 2)	0	mu_1[0][0] log_var_1[0][0]
z_2_latent (Lambda)	(None, 2)	0	mu_2[0][0] log_var_2[0][0]
z_3_latent (Lambda)	(None, 2)	0	mu_3[0][0] log_var_3[0][0]

Total params: 1,933,068
Trainable params: 1,933,068
Non-trainable params: 0

Listing 26: Mnist-VLAE-GAN Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 2)	0	
generative_2_dense_0 (Dense)	(None, 1024)	3072	z_3[0][0]
generative_2_relu_0 (ReLU)	(None, 1024)	0	generative_2_dense_0[0][0]
generative_2_dense_1 (Dense)	(None, 1024)	1049600	generative_2_relu_0[0][0]
generative_2_relu_1 (ReLU)	(None, 1024)	0	generative_2_dense_1[0][0]
z_2 (InputLayer)	(None, 2)	0	
concatenate_2_and_1 (Concatenat	(None, 1026)	0	generative_2_relu_1[0][0] z_2[0][0]
generative_1_dense_0 (Dense)	(None, 1024)	1051648	concatenate_2_and_1[0][0]
generative_1_relu_0 (ReLU)	(None, 1024)	0	generative_1_dense_0[0][0]
generative_1_dense_1 (Dense)	(None, 1024)	1049600	generative_1_relu_0[0][0]
generative_1_relu_1 (ReLU)	(None, 1024)	0	generative_1_dense_1[0][0]
z_1 (InputLayer)	(None, 2)	0	
concatenate_1_and_0 (Concatenat	(None, 1026)	0	generative_1_relu_1[0][0] z_1[0][0]
generative_0_dense_0 (Dense)	(None, 25088)	25765376	concatenate_1_and_0[0][0]
generative_0_relu_0 (ReLU)	(None, 25088)	0	generative_0_dense_0[0][0]
generative_0_reshape_0 (Reshape	(None, 7, 7, 512)	0	generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0	(None, 14, 14, 512)	4194304	generative_0_reshape_0[0][0]
generative_0_relu_1 (ReLU)	(None, 14, 14, 512)	0	generative_0_conv2d_transpose_0[0
generative_0_conv2d_transpose_1	(None, 28, 28, 256)	2097152	generative_0_relu_1[0][0]
generative_0_relu_2 (ReLU)	(None, 28, 28, 256)	0	generative_0_conv2d_transpose_1[0
generative_0_conv2d_transpose_2	(None, 28, 28, 1)	4096	generative_0_relu_2[0][0]
generative_0_activation_0 (Acti	(None, 28, 28, 1)	0	generative_0_conv2d_transpose_2[0

Total params: 35,214,848
Trainable params: 35,214,848
Non-trainable params: 0

Listing 27: Mnist-VLAE-GAN Decoder

Layer (type)	Output Shape	Param #
discriminator_input (InputLa	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	1088
leaky_re_lu_1 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	131200
batch_normalization_1 (Batch	(None, 7, 7, 128)	512
conv2d_3 (Conv2D)	(None, 2, 2, 1)	129
flatten_1 (Flatten)	(None, 4)	0
dense_1 (Dense)	(None, 1)	5
activation_1 (Activation)	(None, 1)	0

Total params: 132,934
Trainable params: 132,678
Non-trainable params: 256

Listing 28: Mnist-VLAE-GAN Discriminator

Layer (type)	Output Shape	Param #	Connected to

vlae_gan_encoder_input (InputLa	(None, 64, 64, 1)	0	
inference_0_conv2d_0 (Conv2D)	(None, 32, 32, 32)	544	vlae_gan_encoder_input[0][0]
inference_0_batch_norm_0 (Batch	(None, 32, 32, 32)	128	inference_0_conv2d_0[0][0]
inference_0_relu_0 (ReLU)	(None, 32, 32, 32)	0	inference_0_batch_norm_0[0][0]
inference_0_conv2d_1 (Conv2D)	(None, 32, 32, 32)	16416	inference_0_relu_0[0][0]
inference_0_batch_norm_1 (Batch	(None, 32, 32, 32)	128	inference_0_conv2d_1[0][0]
inference_0_relu_1 (ReLU)	(None, 32, 32, 32)	0	inference_0_batch_norm_1[0][0]
inference_1_conv2d_0 (Conv2D)	(None, 16, 16, 64)	32832	inference_0_relu_1[0][0]
inference_1_batch_norm_0 (Batch	(None, 16, 16, 64)	256	inference_1_conv2d_0[0][0]
inference_1_relu_0 (ReLU)	(None, 16, 16, 64)	0	inference_1_batch_norm_0[0][0]
inference_1_conv2d_1 (Conv2D)	(None, 16, 16, 64)	65600	inference_1_relu_0[0][0]
inference_1_batch_norm_1 (Batch	(None, 16, 16, 64)	256	inference_1_conv2d_1[0][0]
inference_1_relu_1 (ReLU)	(None, 16, 16, 64)	0	inference_1_batch_norm_1[0][0]
ladder_2_conv2d_0 (Conv2D)	(None, 8, 8, 256)	262400	inference_1_relu_1[0][0]
ladder_2_batch_norm_0 (BatchNor	(None, 8, 8, 256)	1024	ladder_2_conv2d_0[0][0]
ladder_2_relu_0 (ReLU)	(None, 8, 8, 256)	0	ladder_2_batch_norm_0[0][0]
ladder_0_conv2d_0 (Conv2D)	(None, 32, 32, 128)	2176	vlae_gan_encoder_input[0][0]
ladder_1_conv2d_0 (Conv2D)	(None, 16, 16, 256)	131328	inference_0_relu_1[0][0]
ladder_2_conv2d_1 (Conv2D)	(None, 4, 4, 512)	2097664	ladder_2_relu_0[0][0]
ladder_0_batch_norm_0 (BatchNor	(None, 32, 32, 128)	512	ladder_0_conv2d_0[0][0]
ladder_1_batch_norm_0 (BatchNor	(None, 16, 16, 256)	1024	ladder_1_conv2d_0[0][0]
ladder_2_batch_norm_1 (BatchNor	(None, 4, 4, 512)	2048	ladder_2_conv2d_1[0][0]
ladder_0_relu_0 (ReLU)	(None, 32, 32, 128)	0	ladder_0_batch_norm_0[0][0]
ladder_1_relu_0 (ReLU)	(None, 16, 16, 256)	0	ladder_1_batch_norm_0[0][0]
ladder_2_relu_1 (ReLU)	(None, 4, 4, 512)	0	ladder_2_batch_norm_1[0][0]
ladder_0_conv2d_1 (Conv2D)	(None, 32, 32, 128)	262272	ladder_0_relu_0[0][0]
ladder_1_conv2d_1 (Conv2D)	(None, 16, 16, 256)	1048832	ladder_1_relu_0[0][0]
ladder_2_conv2d_2 (Conv2D)	(None, 2, 2, 1024)	8389632	ladder_2_relu_1[0][0]
ladder_0_batch_norm_1 (BatchNor	(None, 32, 32, 128)	512	ladder_0_conv2d_1[0][0]
ladder_1_batch_norm_1 (BatchNor	(None, 16, 16, 256)	1024	ladder_1_conv2d_1[0][0]
ladder_2_batch_norm_2 (BatchNor	(None, 2, 2, 1024)	4096	ladder_2_conv2d_2[0][0]
ladder_0_relu_1 (ReLU)	(None, 32, 32, 128)	0	ladder_0_batch_norm_1[0][0]
ladder_1_relu_1 (ReLU)	(None, 16, 16, 256)	0	ladder_1_batch_norm_1[0][0]
ladder_2_relu_2 (ReLU)	(None, 2, 2, 1024)	0	ladder_2_batch_norm_2[0][0]
ladder_0_flatten (Flatten)	(None, 131072)	0	ladder_0_relu_1[0][0]
ladder_1_flatten (Flatten)	(None, 65536)	0	ladder_1_relu_1[0][0]
ladder_2_flatten (Flatten)	(None, 4096)	0	ladder_2_relu_2[0][0]
mu_1 (Dense)	(None, 4)	524292	ladder_0_flatten[0][0]
log_var_1 (Dense)	(None, 4)	524292	ladder_0_flatten[0][0]
mu_2 (Dense)	(None, 4)	262148	ladder_1_flatten[0][0]
log_var_2 (Dense)	(None, 4)	262148	ladder_1_flatten[0][0]
mu_3 (Dense)	(None, 4)	16388	ladder_2_flatten[0][0]
log_var_3 (Dense)	(None, 4)	16388	ladder_2_flatten[0][0]
z_1_latent (Lambda)	(None, 4)	0	mu_1[0][0] log_var_1[0][0]
z_2_latent (Lambda)	(None, 4)	0	mu_2[0][0] log_var_2[0][0]
z_3_latent (Lambda)	(None, 4)	0	mu_3[0][0] log_var_3[0][0]

Total params: 13,926,360
Trainable params: 13,920,856
Non-trainable params: 5,504

Listing 29: dSprites-VLAE-GAN Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 4)	0	
generative_2_dense_0 (Dense)	(None, 1024)	5120	z_3[0][0]
generative_2_batch_norm_0 (Batc	(None, 1024)	4096	generative_2_dense_0[0][0]
generative_2_relu_0 (ReLU)	(None, 1024)	0	generative_2_batch_norm_0[0][0]
generative_2_dense_1 (Dense)	(None, 1024)	1049600	generative_2_relu_0[0][0]
generative_2_batch_norm_1 (Batc	(None, 1024)	4096	generative_2_dense_1[0][0]
generative_2_relu_1 (ReLU)	(None, 1024)	0	generative_2_batch_norm_1[0][0]

```
z_2 (InputLayer)                 (None, 4)             0
concatenate_2_and_1 (Concatenat (None, 1028)          0           generative_2_relu_1[0][0]
                                                                  z_2[0][0]
generative_1_dense_0 (Dense)     (None, 1024)          1053696     concatenate_2_and_1[0][0]
generative_1_batch_norm_0 (Batc (None, 1024)          4096        generative_1_dense_0[0][0]
generative_1_relu_0 (ReLU)       (None, 1024)          0           generative_1_batch_norm_0[0][0]
generative_1_dense_1 (Dense)     (None, 1024)          1049600     generative_1_relu_0[0][0]
generative_1_batch_norm_1 (Batc (None, 1024)          4096        generative_1_dense_1[0][0]
generative_1_relu_1 (ReLU)       (None, 1024)          0           generative_1_batch_norm_1[0][0]
z_1 (InputLayer)                 (None, 4)             0
concatenate_1_and_0 (Concatenat (None, 1028)          0           generative_1_relu_1[0][0]
                                                                  z_1[0][0]
generative_0_dense_0 (Dense)     (None, 4096)          4214784     concatenate_1_and_0[0][0]
generative_0_batch_norm_0 (Batc (None, 4096)          16384       generative_0_dense_0[0][0]
generative_0_relu_0 (ReLU)       (None, 4096)          0           generative_0_batch_norm_0[0][0]
generative_0_reshape_0 (Reshape (None, 2, 2, 1024)    0           generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0 (None, 4, 4, 1024)    16777216    generative_0_reshape_0[0][0]
generative_0_batch_norm_1 (Batc (None, 4, 4, 1024)    4096        generative_0_conv2d_transpose_0[0
generative_0_relu_1 (ReLU)       (None, 4, 4, 1024)    0           generative_0_batch_norm_1[0][0]
generative_0_conv2d_transpose_1 (None, 8, 8, 512)     8388608     generative_0_relu_1[0][0]
generative_0_batch_norm_2 (Batc (None, 8, 8, 512)     2048        generative_0_conv2d_transpose_1[0
generative_0_relu_2 (ReLU)       (None, 8, 8, 512)     0           generative_0_batch_norm_2[0][0]
generative_0_conv2d_transpose_2 (None, 16, 16, 256)   2097152     generative_0_relu_2[0][0]
generative_0_batch_norm_3 (Batc (None, 16, 16, 256)   1024        generative_0_conv2d_transpose_2[0
generative_0_relu_3 (ReLU)       (None, 16, 16, 256)   0           generative_0_batch_norm_3[0][0]
generative_0_conv2d_transpose_3 (None, 32, 32, 128)   524288      generative_0_relu_3[0][0]
generative_0_batch_norm_4 (Batc (None, 32, 32, 128)   512         generative_0_conv2d_transpose_3[0
generative_0_relu_4 (ReLU)       (None, 32, 32, 128)   0           generative_0_batch_norm_4[0][0]
generative_0_conv2d_transpose_4 (None, 64, 64, 1)     2048        generative_0_relu_4[0][0]
generative_0_activation_0 (Acti (None, 64, 64, 1)     0           generative_0_conv2d_transpose_4[0
Total params: 35,202,560
Trainable params: 35,182,336
Non-trainable params: 20,224
```

Listing 30: dSprites-VLAE-GAN Decoder

```
Layer (type)                 Output Shape              Param #
discriminator_input (InputLa (None, 64, 64, 1)         0
conv2d_1 (Conv2D)            (None, 32, 32, 64)        1088
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 64)        0
conv2d_2 (Conv2D)            (None, 16, 16, 128)       131200
batch_normalization_1 (Batch (None, 16, 16, 128)       512
conv2d_3 (Conv2D)            (None, 8, 8, 256)         524544
batch_normalization_2 (Batch (None, 8, 8, 256)         1024
conv2d_4 (Conv2D)            (None, 2, 2, 1)           257
flatten_1 (Flatten)          (None, 4)                 0
dense_1 (Dense)              (None, 1)                 5
activation_1 (Activation)    (None, 1)                 0
Total params: 658,630
Trainable params: 657,862
Non-trainable params: 768
```

Listing 31: dSprites-VLAE-GAN Discriminator

```
Layer (type)                 Output Shape              Param #     Connected to
vlae_gan_encoder_input (InputLa (None, 128, 128, 3)    0
inference_0_conv2d_0 (Conv2D)    (None, 64, 64, 32)    1568        vlae_gan_encoder_input[0][0]
inference_0_batch_norm_0 (Batch (None, 64, 64, 32)     128         inference_0_conv2d_0[0][0]
inference_0_relu_0 (ReLU)        (None, 64, 64, 32)    0           inference_0_batch_norm_0[0][0]
inference_0_conv2d_1 (Conv2D)    (None, 64, 64, 32)    16416       inference_0_relu_0[0][0]
inference_0_batch_norm_1 (Batch (None, 64, 64, 32)     128         inference_0_conv2d_1[0][0]
inference_0_relu_1 (ReLU)        (None, 64, 64, 32)    0           inference_0_batch_norm_1[0][0]
inference_1_conv2d_0 (Conv2D)    (None, 32, 32, 64)    32832       inference_0_relu_1[0][0]
inference_1_batch_norm_0 (Batch (None, 32, 32, 64)     256         inference_1_conv2d_0[0][0]
```

inference_1_relu_0 (ReLU)	(None, 32, 32, 64)	0	inference_1_batch_norm_0[0][0]
inference_1_conv2d_1 (Conv2D)	(None, 32, 32, 64)	65600	inference_1_relu_0[0][0]
inference_1_batch_norm_1 (Batch	(None, 32, 32, 64)	256	inference_1_conv2d_1[0][0]
inference_1_relu_1 (ReLU)	(None, 32, 32, 64)	0	inference_1_batch_norm_1[0][0]
ladder_2_conv2d_0 (Conv2D)	(None, 16, 16, 256)	262400	inference_1_relu_1[0][0]
ladder_2_batch_norm_0 (BatchNor	(None, 16, 16, 256)	1024	ladder_2_conv2d_0[0][0]
ladder_2_relu_0 (ReLU)	(None, 16, 16, 256)	0	ladder_2_batch_norm_0[0][0]
ladder_2_conv2d_1 (Conv2D)	(None, 8, 8, 512)	2097664	ladder_2_relu_0[0][0]
ladder_2_batch_norm_1 (BatchNor	(None, 8, 8, 512)	2048	ladder_2_conv2d_1[0][0]
ladder_2_relu_1 (ReLU)	(None, 8, 8, 512)	0	ladder_2_batch_norm_1[0][0]
ladder_0_conv2d_0 (Conv2D)	(None, 64, 64, 128)	6272	vlae_gan_encoder_input[0][0]
ladder_1_conv2d_0 (Conv2D)	(None, 32, 32, 256)	131328	inference_0_relu_1[0][0]
ladder_2_conv2d_2 (Conv2D)	(None, 8, 8, 512)	4194816	ladder_2_relu_1[0][0]
ladder_0_batch_norm_0 (BatchNor	(None, 64, 64, 128)	512	ladder_0_conv2d_0[0][0]
ladder_1_batch_norm_0 (BatchNor	(None, 32, 32, 256)	1024	ladder_1_conv2d_0[0][0]
ladder_2_batch_norm_2 (BatchNor	(None, 8, 8, 512)	2048	ladder_2_conv2d_2[0][0]
ladder_0_relu_0 (ReLU)	(None, 64, 64, 128)	0	ladder_0_batch_norm_0[0][0]
ladder_1_relu_0 (ReLU)	(None, 32, 32, 256)	0	ladder_1_batch_norm_0[0][0]
ladder_2_relu_2 (ReLU)	(None, 8, 8, 512)	0	ladder_2_batch_norm_2[0][0]
ladder_0_conv2d_1 (Conv2D)	(None, 64, 64, 128)	262272	ladder_0_relu_0[0][0]
ladder_1_conv2d_1 (Conv2D)	(None, 32, 32, 256)	1048832	ladder_1_relu_0[0][0]
ladder_2_conv2d_3 (Conv2D)	(None, 8, 8, 512)	4194816	ladder_2_relu_2[0][0]
ladder_0_batch_norm_1 (BatchNor	(None, 64, 64, 128)	512	ladder_0_conv2d_1[0][0]
ladder_1_batch_norm_1 (BatchNor	(None, 32, 32, 256)	1024	ladder_1_conv2d_1[0][0]
ladder_2_batch_norm_3 (BatchNor	(None, 8, 8, 512)	2048	ladder_2_conv2d_3[0][0]
ladder_0_relu_1 (ReLU)	(None, 64, 64, 128)	0	ladder_0_batch_norm_1[0][0]
ladder_1_relu_1 (ReLU)	(None, 32, 32, 256)	0	ladder_1_batch_norm_1[0][0]
ladder_2_relu_3 (ReLU)	(None, 8, 8, 512)	0	ladder_2_batch_norm_3[0][0]
ladder_0_flatten (Flatten)	(None, 524288)	0	ladder_0_relu_1[0][0]
ladder_1_flatten (Flatten)	(None, 262144)	0	ladder_1_relu_1[0][0]
ladder_2_flatten (Flatten)	(None, 32768)	0	ladder_2_relu_3[0][0]
mu_1 (Dense)	(None, 2)	1048578	ladder_0_flatten[0][0]
log_var_1 (Dense)	(None, 2)	1048578	ladder_0_flatten[0][0]
mu_2 (Dense)	(None, 2)	524290	ladder_1_flatten[0][0]
log_var_2 (Dense)	(None, 2)	524290	ladder_1_flatten[0][0]
mu_3 (Dense)	(None, 2)	65538	ladder_2_flatten[0][0]
log_var_3 (Dense)	(None, 2)	65538	ladder_2_flatten[0][0]
z_1_latent (Lambda)	(None, 2)	0	mu_1[0][0] log_var_1[0][0]
z_2_latent (Lambda)	(None, 2)	0	mu_2[0][0] log_var_2[0][0]
z_3_latent (Lambda)	(None, 2)	0	mu_3[0][0] log_var_3[0][0]

Total params: 15,602,636
Trainable params: 15,597,132
Non-trainable params: 5,504

Listing 32: CelebA-VLAE-GAN Encoder

Layer (type)	Output Shape	Param #	Connected to
z_3 (InputLayer)	(None, 2)	0	
generative_2_dense_0 (Dense)	(None, 256)	768	z_3[0][0]
generative_2_batch_norm_0 (Batc	(None, 256)	1024	generative_2_dense_0[0][0]
generative_2_relu_0 (ReLU)	(None, 256)	0	generative_2_batch_norm_0[0][0]
generative_2_dense_1 (Dense)	(None, 512)	131584	generative_2_relu_0[0][0]
generative_2_batch_norm_1 (Batc	(None, 512)	2048	generative_2_dense_1[0][0]
generative_2_relu_1 (ReLU)	(None, 512)	0	generative_2_batch_norm_1[0][0]
z_2 (InputLayer)	(None, 2)	0	
concatenate_2_and_1 (Concatenat	(None, 514)	0	generative_2_relu_1[0][0] z_2[0][0]
generative_1_dense_0 (Dense)	(None, 512)	263680	concatenate_2_and_1[0][0]
generative_1_batch_norm_0 (Batc	(None, 512)	2048	generative_1_dense_0[0][0]
generative_1_relu_0 (ReLU)	(None, 512)	0	generative_1_batch_norm_0[0][0]
generative_1_dense_1 (Dense)	(None, 1024)	525312	generative_1_relu_0[0][0]

generative_1_batch_norm_1 (Batc	(None, 1024)	4096	generative_1_dense_1[0][0]
generative_1_relu_1 (ReLU)	(None, 1024)	0	generative_1_batch_norm_1[0][0]
z_1 (InputLayer)	(None, 2)	0	
concatenate_1_and_0 (Concatenat	(None, 1026)	0	generative_1_relu_1[0][0] z_1[0][0]
generative_0_dense_0 (Dense)	(None, 32768)	33652736	concatenate_1_and_0[0][0]
generative_0_batch_norm_0 (Batc	(None, 32768)	131072	generative_0_dense_0[0][0]
generative_0_relu_0 (ReLU)	(None, 32768)	0	generative_0_batch_norm_0[0][0]
generative_0_reshape_0 (Reshape	(None, 8, 8, 512)	0	generative_0_relu_0[0][0]
generative_0_conv2d_transpose_0	(None, 8, 8, 1024)	8388608	generative_0_reshape_0[0][0]
generative_0_batch_norm_1 (Batc	(None, 8, 8, 1024)	4096	generative_0_conv2d_transpose_0[0
generative_0_relu_1 (ReLU)	(None, 8, 8, 1024)	0	generative_0_batch_norm_1[0][0]
generative_0_conv2d_transpose_1	(None, 16, 16, 512)	8388608	generative_0_relu_1[0][0]
generative_0_batch_norm_2 (Batc	(None, 16, 16, 512)	2048	generative_0_conv2d_transpose_1[0
generative_0_relu_2 (ReLU)	(None, 16, 16, 512)	0	generative_0_batch_norm_2[0][0]
generative_0_conv2d_transpose_2	(None, 32, 32, 256)	2097152	generative_0_relu_2[0][0]
generative_0_batch_norm_3 (Batc	(None, 32, 32, 256)	1024	generative_0_conv2d_transpose_2[0
generative_0_relu_3 (ReLU)	(None, 32, 32, 256)	0	generative_0_batch_norm_3[0][0]
generative_0_conv2d_transpose_3	(None, 64, 64, 128)	524288	generative_0_relu_3[0][0]
generative_0_batch_norm_4 (Batc	(None, 64, 64, 128)	512	generative_0_conv2d_transpose_3[0
generative_0_relu_4 (ReLU)	(None, 64, 64, 128)	0	generative_0_batch_norm_4[0][0]
generative_0_conv2d_transpose_4	(None, 128, 128, 64)	131072	generative_0_relu_4[0][0]
generative_0_batch_norm_5 (Batc	(None, 128, 128, 64)	256	generative_0_conv2d_transpose_4[0
generative_0_relu_5 (ReLU)	(None, 128, 128, 64)	0	generative_0_batch_norm_5[0][0]
generative_0_conv2d_transpose_5	(None, 128, 128, 3)	3072	generative_0_relu_5[0][0]
generative_0_activation_0 (Acti	(None, 128, 128, 3)	0	generative_0_conv2d_transpose_5[0

Total params: 54,255,104
Trainable params: 54,180,992
Non-trainable params: 74,112

Listing 33: CelebA-VLAE-GAN Decoder

Layer (type)	Output Shape	Param #
discriminator_input (InputLa	(None, 128, 128, 3)	0
conv2d_1 (Conv2D)	(None, 64, 64, 64)	3136
leaky_re_lu_1 (LeakyReLU)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	131200
batch_normalization_1 (Batch	(None, 32, 32, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 256)	524544
batch_normalization_2 (Batch	(None, 16, 16, 256)	1024
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 256)	0
conv2d_4 (Conv2D)	(None, 8, 8, 512)	2097664
batch_normalization_3 (Batch	(None, 8, 8, 512)	2048
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 512)	0
conv2d_5 (Conv2D)	(None, 2, 2, 1)	513
flatten_1 (Flatten)	(None, 4)	0
dense_1 (Dense)	(None, 1)	5
activation_1 (Activation)	(None, 1)	0

Total params: 2,760,646
Trainable params: 2,758,854
Non-trainable params: 1,792

Listing 34: CelebA-VLAE-GAN Discriminator

A.5 AlexNet Classifier

Layer (type)	Output Shape	Param #
model_input (InputLayer)	(None, 224, 224, 3)	0
conv2d_1 (Conv2D)	(None, 56, 56, 96)	34944
batch_normalization_1 (Batch	(None, 56, 56, 96)	384
re_lu_1 (ReLU)	(None, 56, 56, 96)	0
max_pooling2d_1 (MaxPooling2	(None, 28, 28, 96)	0

conv2d_2 (Conv2D)	(None, 28, 28, 256)	614656
batch_normalization_2 (Batch	(None, 28, 28, 256)	1024
re_lu_2 (ReLU)	(None, 28, 28, 256)	0
max_pooling2d_2 (MaxPooling2	(None, 14, 14, 256)	0
conv2d_3 (Conv2D)	(None, 14, 14, 384)	885120
batch_normalization_3 (Batch	(None, 14, 14, 384)	1536
re_lu_3 (ReLU)	(None, 14, 14, 384)	0
conv2d_4 (Conv2D)	(None, 14, 14, 384)	1327488
batch_normalization_4 (Batch	(None, 14, 14, 384)	1536
re_lu_4 (ReLU)	(None, 14, 14, 384)	0
conv2d_5 (Conv2D)	(None, 14, 14, 256)	884992
batch_normalization_5 (Batch	(None, 14, 14, 256)	1024
re_lu_5 (ReLU)	(None, 14, 14, 256)	0
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 256)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 4096)	51384320
re_lu_6 (ReLU)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
re_lu_7 (ReLU)	(None, 4096)	0
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 1000)	4097000
softmax_1 (Softmax)	(None, 1000)	0

Total params: 76,015,336
Trainable params: 76,012,584
Non-trainable params: 2,752

Listing 35: Alexnet Classifier

A.6 AlexNet-VAE

Layer (type)	Output Shape	Param #	Connected to
model_input (InputLayer)	(None, 224, 224, 3)	0	
conv2d_1 (Conv2D)	(None, 56, 56, 96)	34944	model_input[0][0]
batch_normalization_1 (BatchNor	(None, 56, 56, 96)	384	conv2d_1[0][0]
re_lu_1 (ReLU)	(None, 56, 56, 96)	0	batch_normalization_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 28, 28, 96)	0	re_lu_1[0][0]
conv2d_2 (Conv2D)	(None, 28, 28, 256)	614656	max_pooling2d_1[0][0]
batch_normalization_2 (BatchNor	(None, 28, 28, 256)	1024	conv2d_2[0][0]
re_lu_2 (ReLU)	(None, 28, 28, 256)	0	batch_normalization_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 256)	0	re_lu_2[0][0]
conv2d_3 (Conv2D)	(None, 14, 14, 384)	885120	max_pooling2d_2[0][0]
batch_normalization_3 (BatchNor	(None, 14, 14, 384)	1536	conv2d_3[0][0]
re_lu_3 (ReLU)	(None, 14, 14, 384)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 14, 14, 384)	1327488	re_lu_3[0][0]
batch_normalization_4 (BatchNor	(None, 14, 14, 384)	1536	conv2d_4[0][0]
re_lu_4 (ReLU)	(None, 14, 14, 384)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 14, 14, 256)	884992	re_lu_4[0][0]
batch_normalization_5 (BatchNor	(None, 14, 14, 256)	1024	conv2d_5[0][0]
re_lu_5 (ReLU)	(None, 14, 14, 256)	0	batch_normalization_5[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 256)	0	re_lu_5[0][0]
flatten_1 (Flatten)	(None, 12544)	0	max_pooling2d_3[0][0]
dense_1 (Dense)	(None, 4096)	51384320	flatten_1[0][0]
re_lu_6 (ReLU)	(None, 4096)	0	dense_1[0][0]
dense_2 (Dense)	(None, 4096)	16781312	re_lu_6[0][0]
re_lu_7 (ReLU)	(None, 4096)	0	dense_2[0][0]
mu (Dense)	(None, 2000)	8194000	re_lu_7[0][0]
log_var (Dense)	(None, 2000)	8194000	re_lu_7[0][0]
encoder_output (Lambda)	(None, 2000)	0	mu[0][0] log_var[0][0]

Total params: 88,306,336
Trainable params: 88,303,584
Non-trainable params: 2,752

Listing 36: AlexNet-VAE Encoder

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	(None, 2000)	0
dense_3 (Dense)	(None, 4096)	8196096
re_lu_8 (ReLU)	(None, 4096)	0
dense_4 (Dense)	(None, 12544)	51392768
re_lu_9 (ReLU)	(None, 12544)	0
dense_5 (Dense)	(None, 12544)	157364480
re_lu_10 (ReLU)	(None, 12544)	0
reshape_1 (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 384)	885120
batch_normalization_6 (Batch	(None, 14, 14, 384)	1536
re_lu_11 (ReLU)	(None, 14, 14, 384)	0
conv2d_transpose_2 (Conv2DTr	(None, 14, 14, 384)	1327488
batch_normalization_7 (Batch	(None, 14, 14, 384)	1536
re_lu_12 (ReLU)	(None, 14, 14, 384)	0
conv2d_transpose_3 (Conv2DTr	(None, 28, 28, 256)	884992
batch_normalization_8 (Batch	(None, 28, 28, 256)	1024
re_lu_13 (ReLU)	(None, 28, 28, 256)	0
conv2d_transpose_4 (Conv2DTr	(None, 56, 56, 96)	614496
batch_normalization_9 (Batch	(None, 56, 56, 96)	384
re_lu_14 (ReLU)	(None, 56, 56, 96)	0
conv2d_transpose_5 (Conv2DTr	(None, 224, 224, 3)	34851
batch_normalization_10 (Batc	(None, 224, 224, 3)	12
activation_1 (Activation)	(None, 224, 224, 3)	0

Total params: 220,704,783
Trainable params: 220,702,537
Non-trainable params: 2,246

Listing 37: AlexNet-VAE Decoder

B Additional Plots For Section 4.2

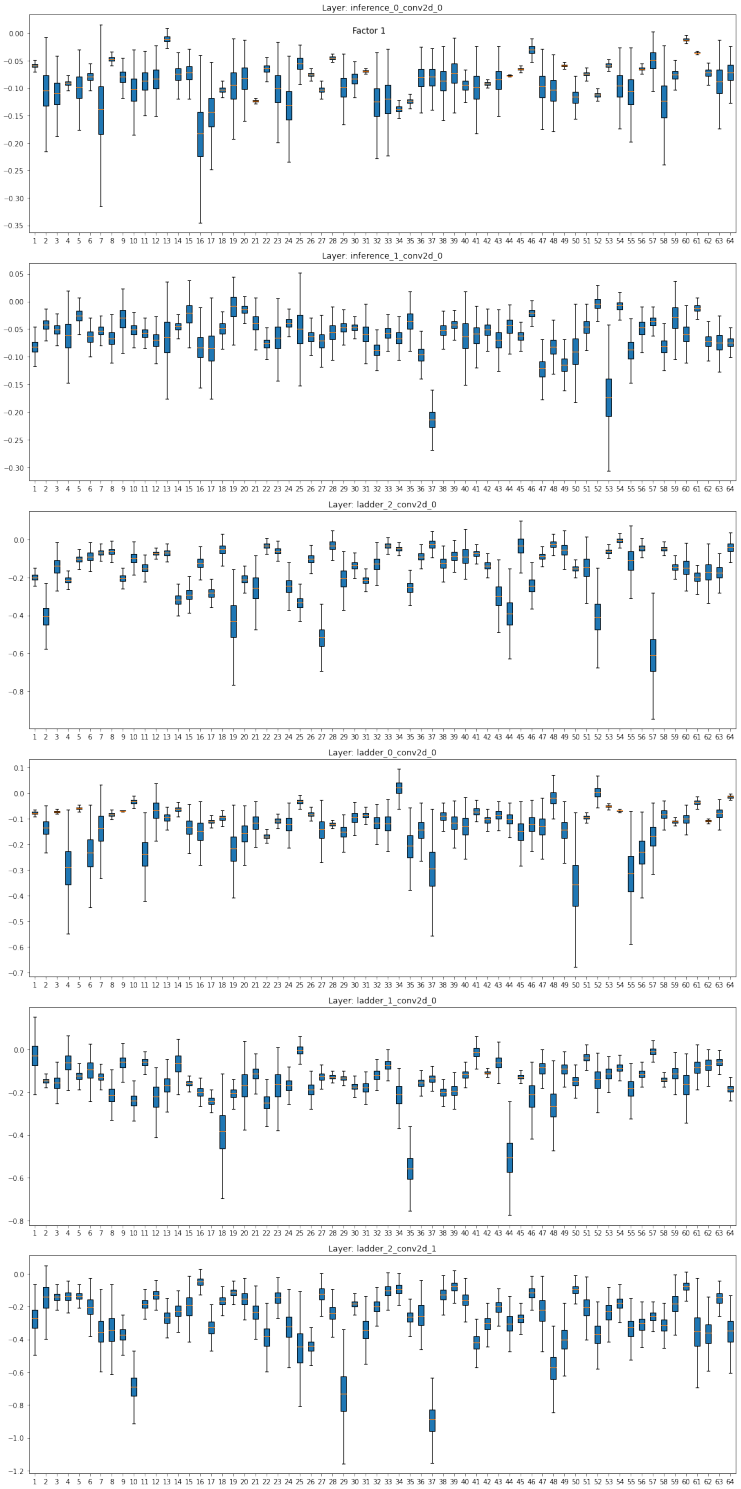


Figure 57: Feature map activities for MNIST-VLAE-factor-1. Each boxplot corresponds to the distribution of the mean value of one feature map after the convolution.

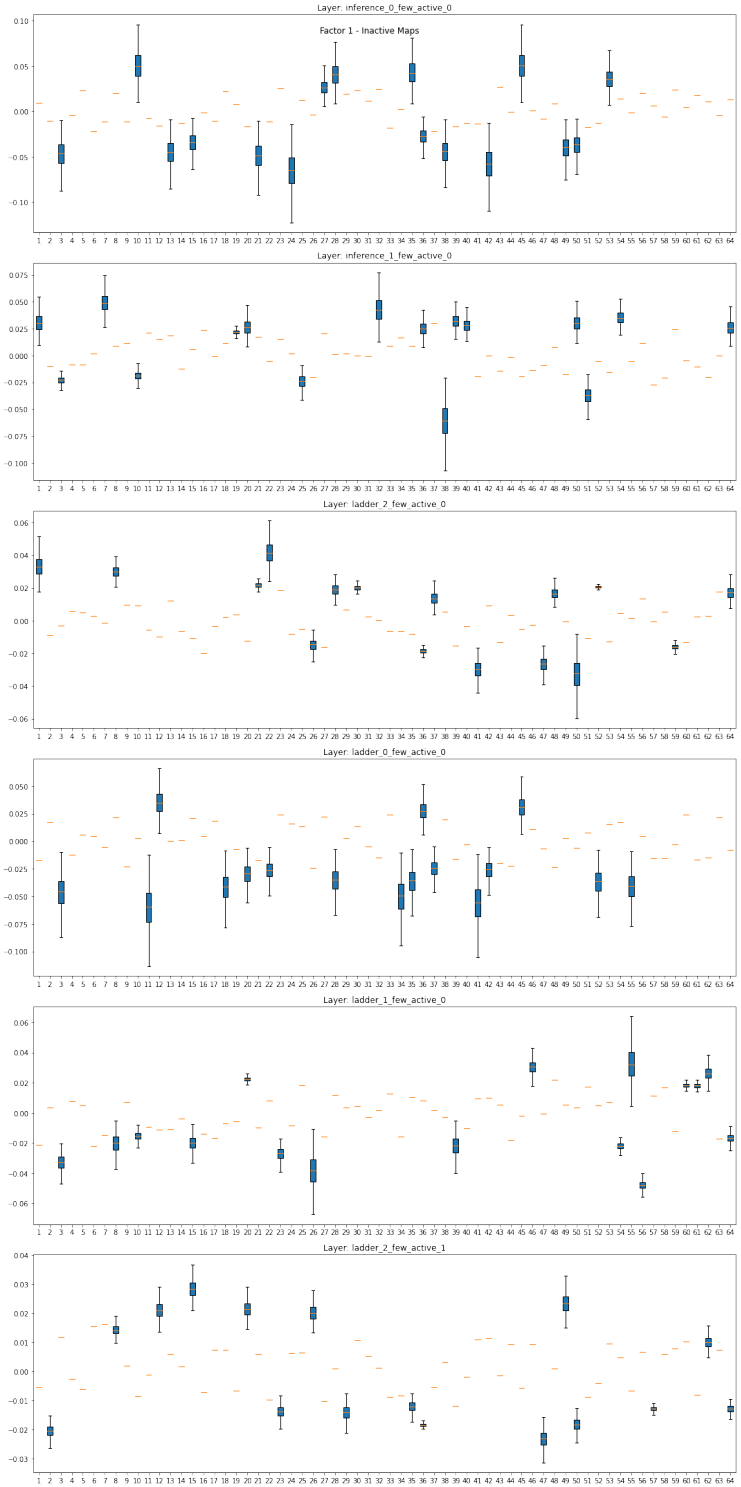


Figure 58: Activities of only the $\lceil \frac{n}{4} \rceil$ most active feature maps in MNIST-VLAE-factor-1. Each boxplot corresponds to the distribution of the mean value of one feature map after the convolution. Less active feature maps are set to their mean values.

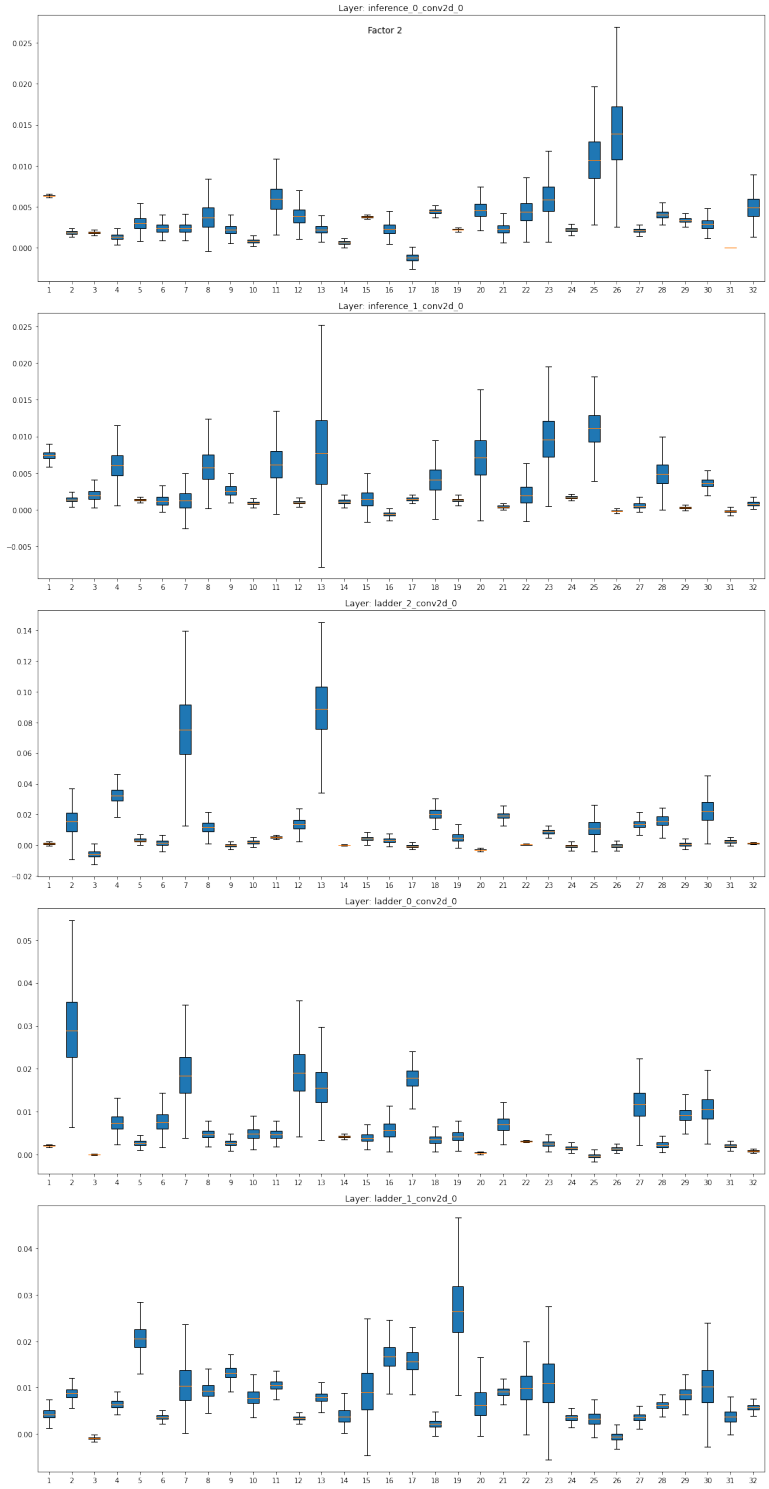


Figure 59: Feature map activities for MNIST-VLAE-factor-2. Each boxplot corresponds to the distribution of the mean value of one feature map after the convolution.

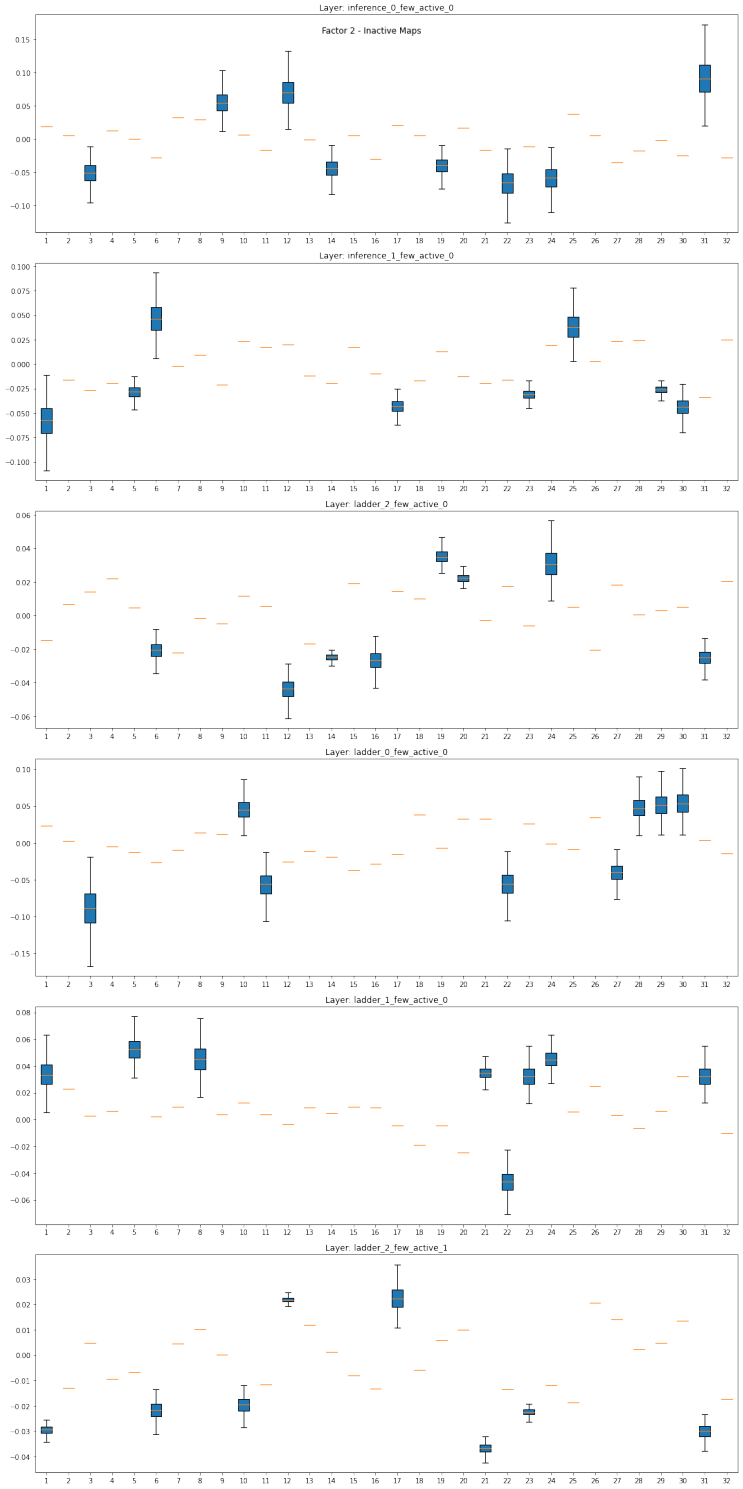
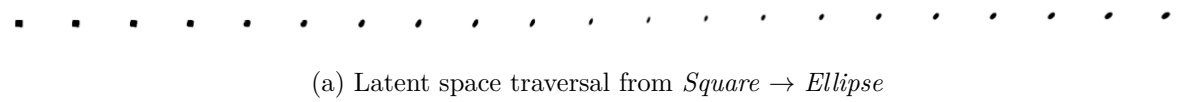


Figure 60: Activities of only the $\lceil \frac{n}{4} \rceil$ most active feature maps in MNIST-VLAE-factor-2. Each boxplot corresponds to the distribution of the mean value of one feature map after the convolution. Less active feature maps are set to their mean values.

C Additional Plots For Section 4.3



(a) Latent space traversal from *Square* → *Ellipse*

(b) Latent space traversal from *Ellipse* → *Heart*

(c) Latent space traversal from *Heart* → *Square*

Figure 61: Latent space traversal between latent space representations of images with certain shapes for 7,500-VAE. Color-values were inverted for this plot.

(a) Latent space traversal from *Square* → *Ellipse*

(b) Latent space traversal from *Ellipse* → *Heart*

(c) Latent space traversal from *Heart* → *Square*

Figure 62: Latent space traversal between latent space representations of images with certain shapes for 5,000-VAE. Color-values were inverted for this plot.

D Feature Extraction Network - Section 4.3

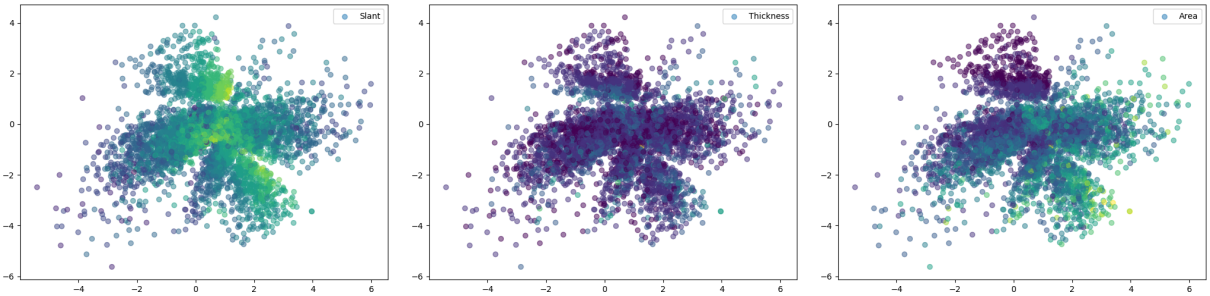
Layer (type)	Output Shape	Param #
discriminator_input (InputLa	(None, 64, 64, 1)	0
conv2d_91 (Conv2D)	(None, 32, 32, 64)	1664
batch_normalization_114 (Bat	(None, 32, 32, 64)	256
leaky_re_lu_55 (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_92 (Conv2D)	(None, 16, 16, 128)	204928
batch_normalization_115 (Bat	(None, 16, 16, 128)	512
leaky_re_lu_56 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_93 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_116 (Bat	(None, 8, 8, 256)	1024
leaky_re_lu_57 (LeakyReLU)	(None, 8, 8, 256)	0
conv2d_94 (Conv2D)	(None, 8, 8, 512)	1180160
batch_normalization_117 (Bat	(None, 8, 8, 512)	2048
leaky_re_lu_58 (LeakyReLU)	(None, 8, 8, 512)	0
conv2d_95 (Conv2D)	(None, 3, 3, 3)	1539
flatten_31 (Flatten)	(None, 27)	0
dense_31 (Dense)	(None, 3)	84
activation_31 (Activation)	(None, 3)	0

Total params: 1,687,383
Trainable params: 1,685,463
Non-trainable params: 1,920

Listing 38: Feature extraction network for perceptual path length (PPL) computation.

E Additional Plots for Section 4.4.1

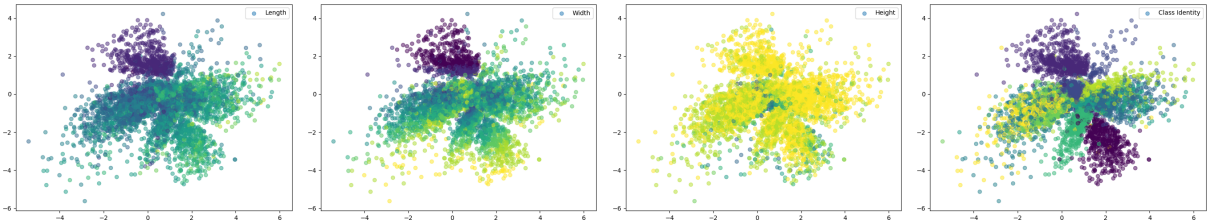
E.1 Mnist



(a) Latent space colored by digit slant

(b) Latent space colored by digit thickness

(c) Latent space colored by digit area



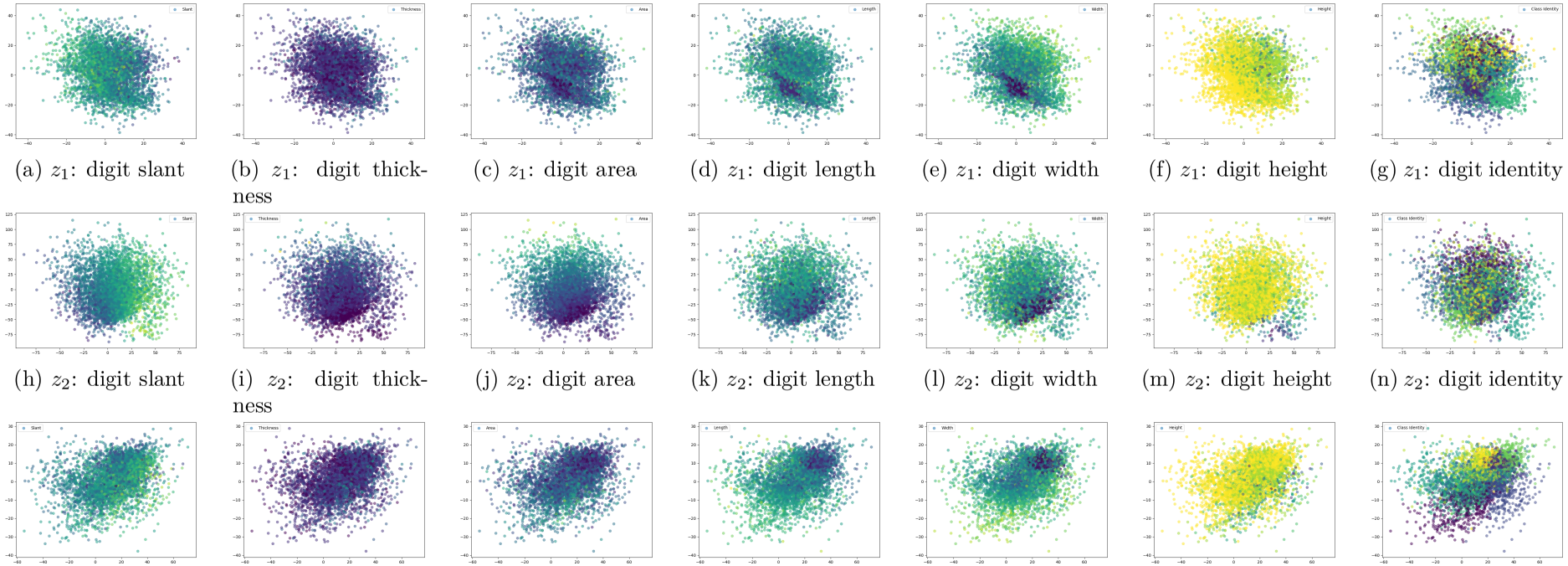
(d) Latent space colored by digit length

(e) Latent space colored by digit width

(f) Latent space colored by digit height

(g) Latent space colored by digit identity

Figure 63: Latent space colored by different means of MNIST-VAE-generative adversarial network (GAN)



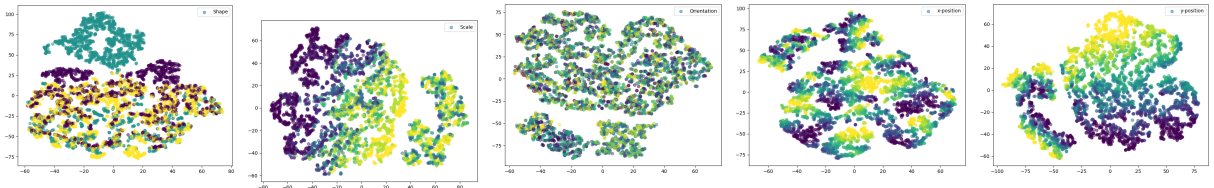
(a) z_1: digit slant (b) z_1: digit thickness (c) z_1: digit area (d) z_1: digit length (e) z_1: digit width (f) z_1: digit height (g) z_1: digit identity

(h) z_2: digit slant (i) z_2: digit thickness (j) z_2: digit area (k) z_2: digit length (l) z_2: digit width (m) z_2: digit height (n) z_2: digit identity

(o) z_3: digit slant (p) z_3: digit thickness (q) z_3: digit area (r) z_3: digit length (s) z_3: digit width (t) z_3: digit height (u) z_3: digit identity

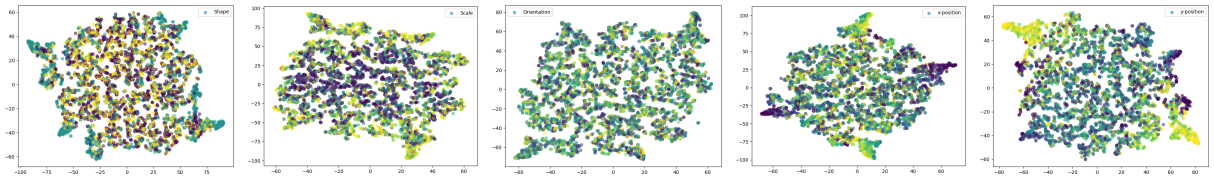
Figure 64: Latent space colored by different means of MNIST-VLAE-GAN

E.2 dSprites

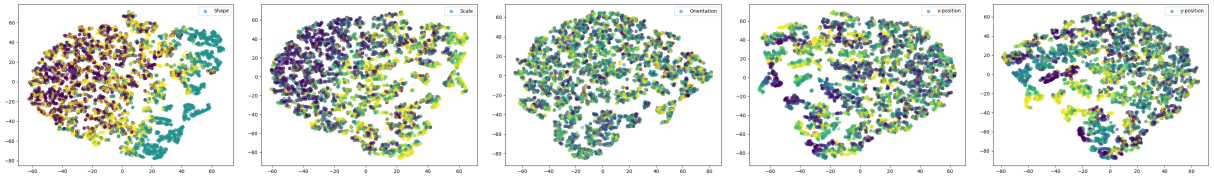


(a) Latent space colored by object shape (b) Latent space colored by object scale (c) Latent space colored by object orientation (d) Latent space colored by object x-position (e) Latent space colored by object y-position

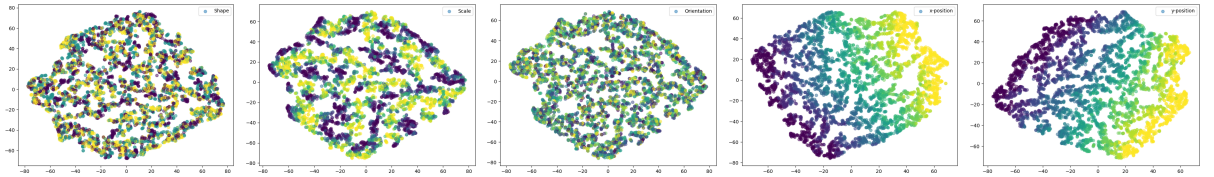
Figure 65: t-distributed stochastic neighbor embedding (t-SNE)-reduced latent space embeddings colored by different means of dSprites-VAE-GAN



(a) Latent space z_1 colored by object shape (b) Latent space z_1 colored by object scale (c) Latent space z_1 colored by object orientation (d) Latent space z_1 colored by object x-position (e) Latent space z_1 colored by object y-position



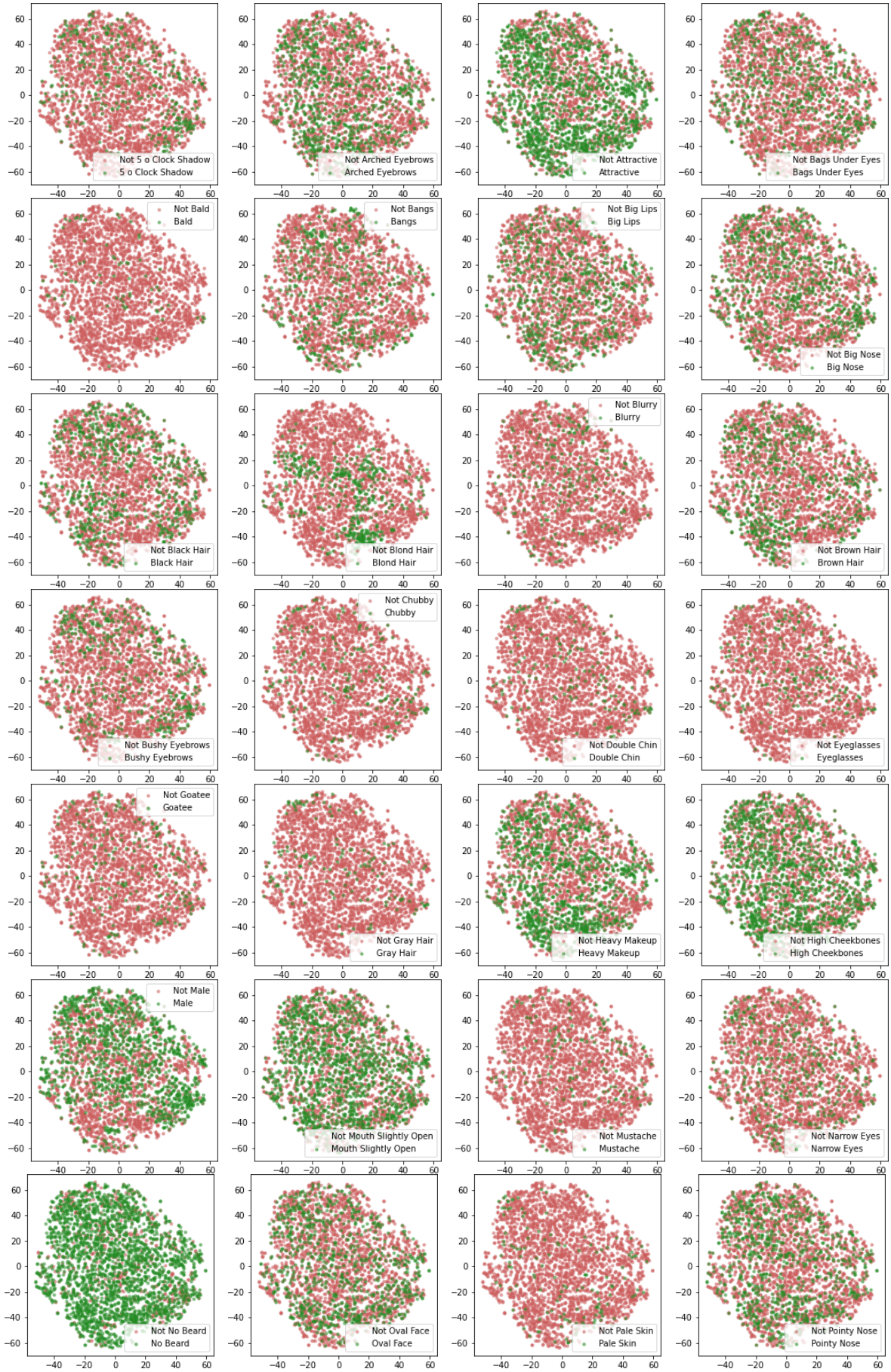
(f) Latent space z_2 colored by object shape (g) Latent space z_2 colored by object scale (h) Latent space z_2 colored by object orientation (i) Latent space z_2 colored by object x-position (j) Latent space z_2 colored by object y-position



(k) Latent space z_3 colored by object shape (l) Latent space z_3 colored by object scale (m) Latent space z_3 colored by object orientation (n) Latent space z_3 colored by object x-position (o) Latent space z_3 colored by object y-position

Figure 66: Latent space of dSprites-VLAE-GAN colored by different means

E.3 CelebA


Not 5 o Clock Shadow
5 o Clock Shadow
Not Arched Eyebrows
Arched Eyebrows
Not Attractive
Attractive
Not Bags Under Eyes
Bags Under Eyes
Not Bald
Bald
Not Bangs
Bangs
Not Big Lips
Big Lips
Not Big Nose
Big Nose
Not Blurry
Blurry
Not Black Hair
Black Hair
Not Blond Hair
Blond Hair
Not Brown Hair
Brown Hair
Not Chubby
Chubby
Not Bushy Eyebrows
Bushy Eyebrows
Not Double Chin
Double Chin
Not Eyeglasses
Eyeglasses
Not Goatee
Goatee
Not Gray Hair
Gray Hair
Not Heavy Makeup
Heavy Makeup
Not High Cheekbones
High Cheekbones
Not Male
Male
Not Mouth Slightly Open
Mouth Slightly Open
Not Mustache
Mustache
Not Narrow Eyes
Narrow Eyes
Not No Beard
No Beard
Not Oval Face
Oval Face
Not Pale Skin
Pale Skin
Not Pointy Nose
Pointy Nose

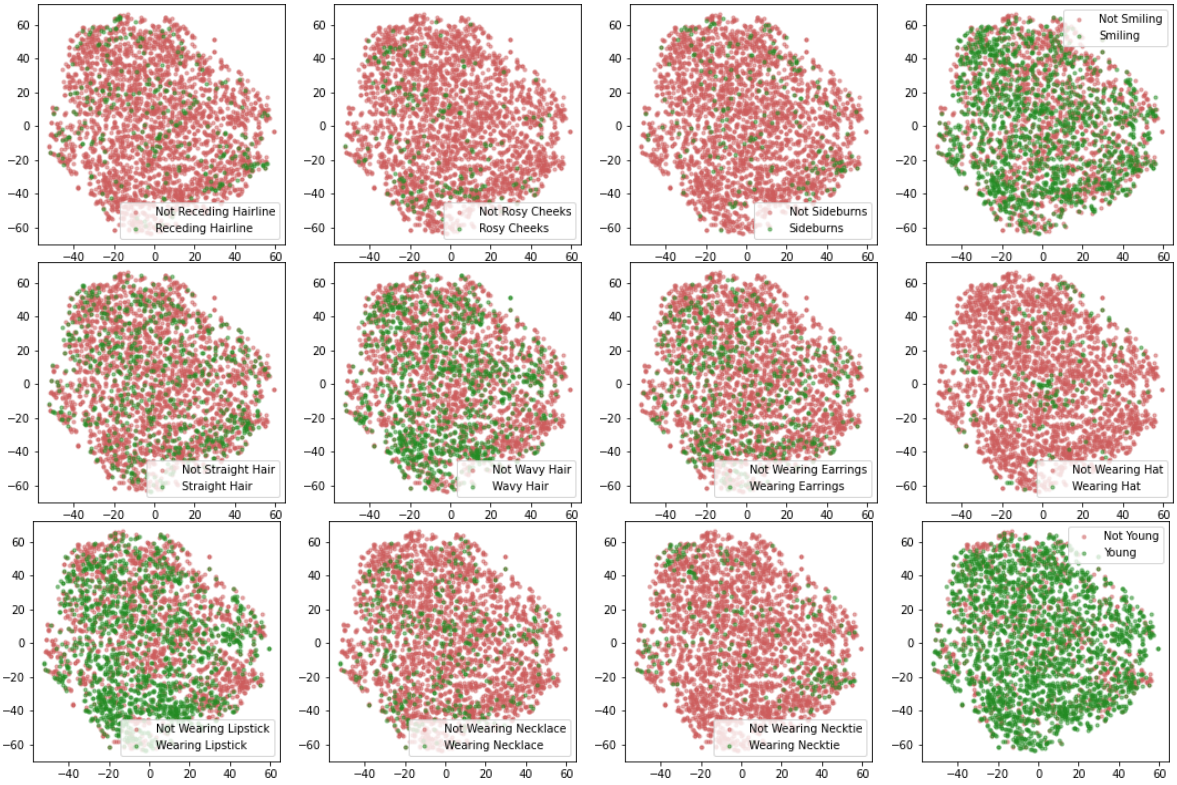


Figure 67: t-SNE-reduced Latent Space of CelebA-VAE, colored by Factors of Variation (present or not present)


Not 5 o Clock Shadow
5 o Clock Shadow
Not Arched Eyebrows
Arched Eyebrows
Not Attractive
Attractive
Not Bags Under Eyes
Bags Under Eyes
Not Bald
Bald
Not Bangs
Bangs
Not Big Lips
Big Lips
Not Big Nose
Big Nose
Not Black Hair
Black Hair
Not Blond Hair
Blond Hair
Not Blurry
Blurry
Not Brown Hair
Brown Hair
Not Bushy Eyebrows
Bushy Eyebrows
Not Chubby
Chubby
Not Double Chin
Double Chin
Not Eyeglasses
Eyeglasses
Not Goatee
Goatee
Not Gray Hair
Gray Hair
Not Heavy Makeup
Heavy Makeup
Not High Cheekbones
High Cheekbones
Not Male
Male
Not Mouth Slightly Open
Mouth Slightly Open
Not Mustache
Mustache
Not Narrow Eyes
Narrow Eyes
Not No Beard
No Beard
Not Oval Face
Oval Face
Not Pale Skin
Pale Skin
Not Pointy Nose
Pointy Nose

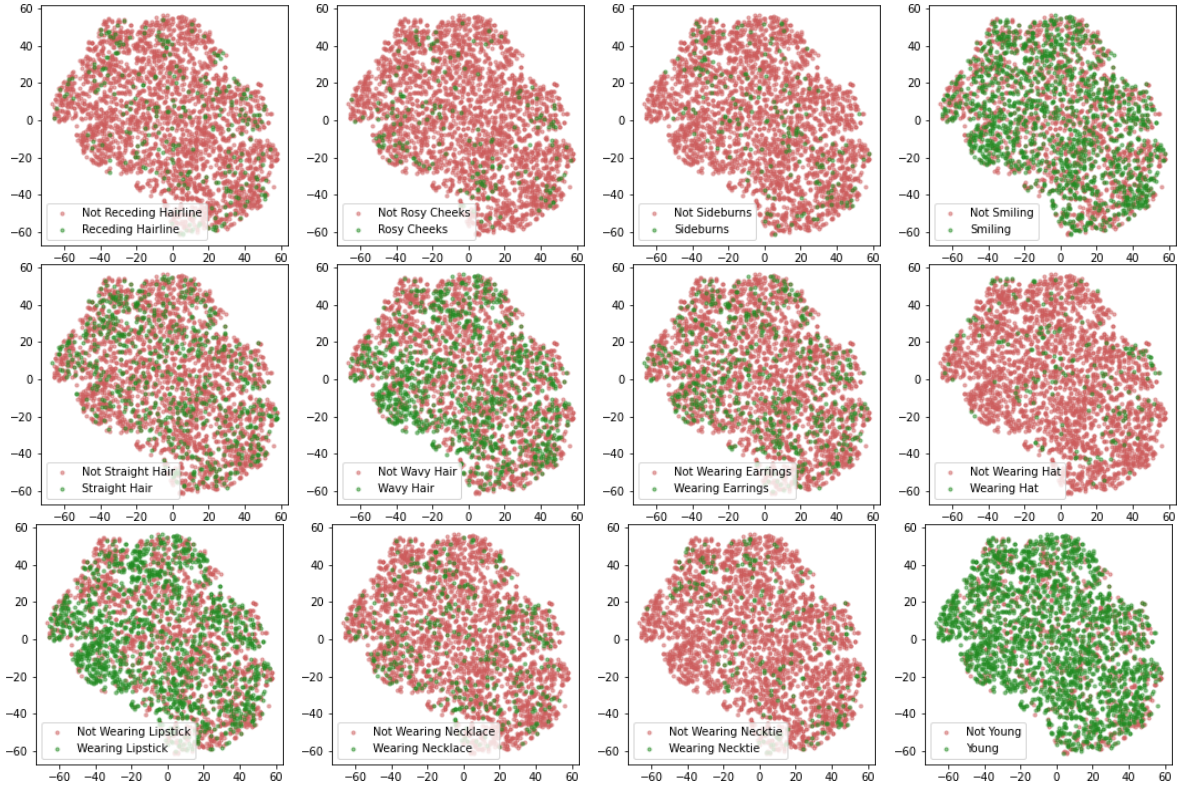
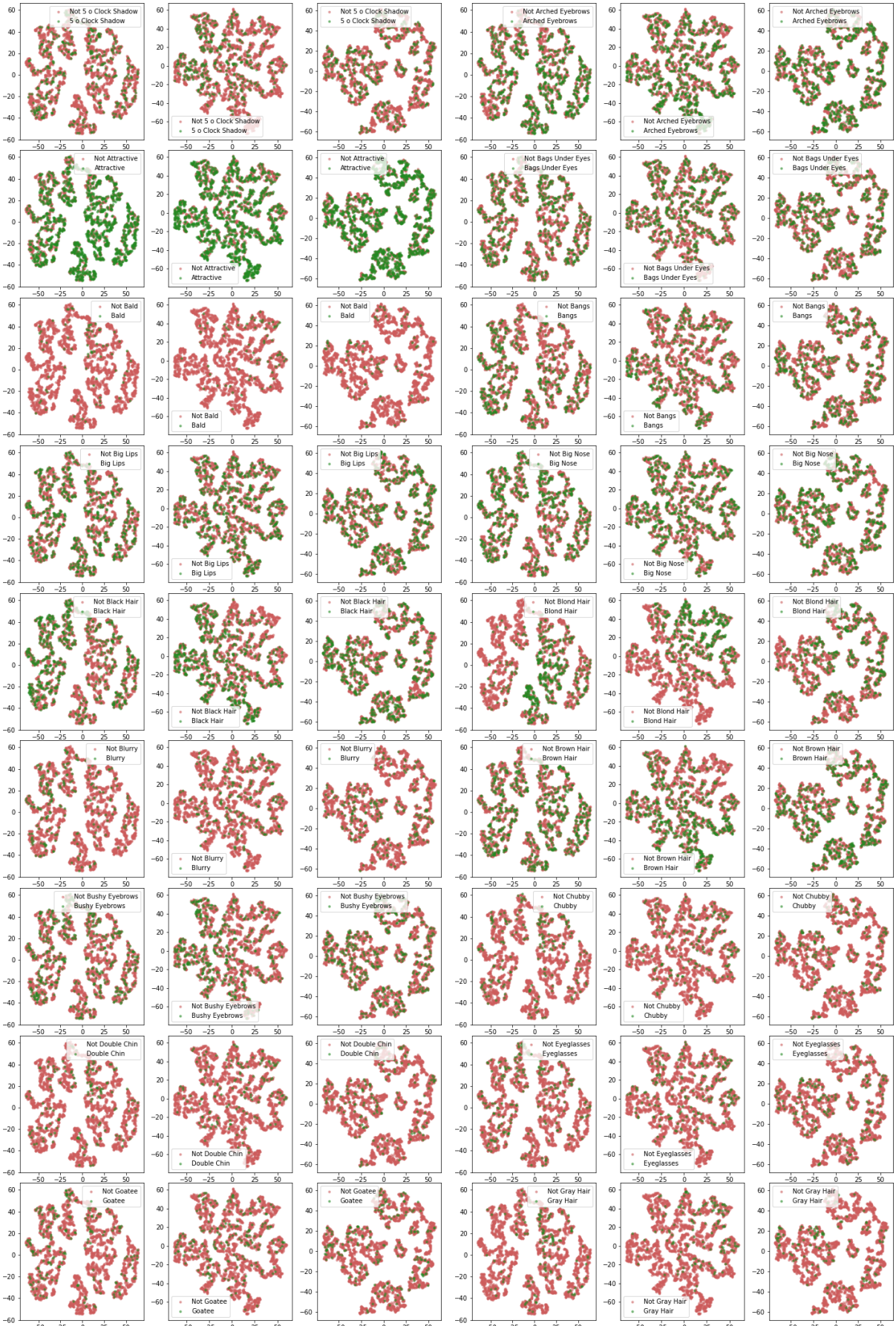
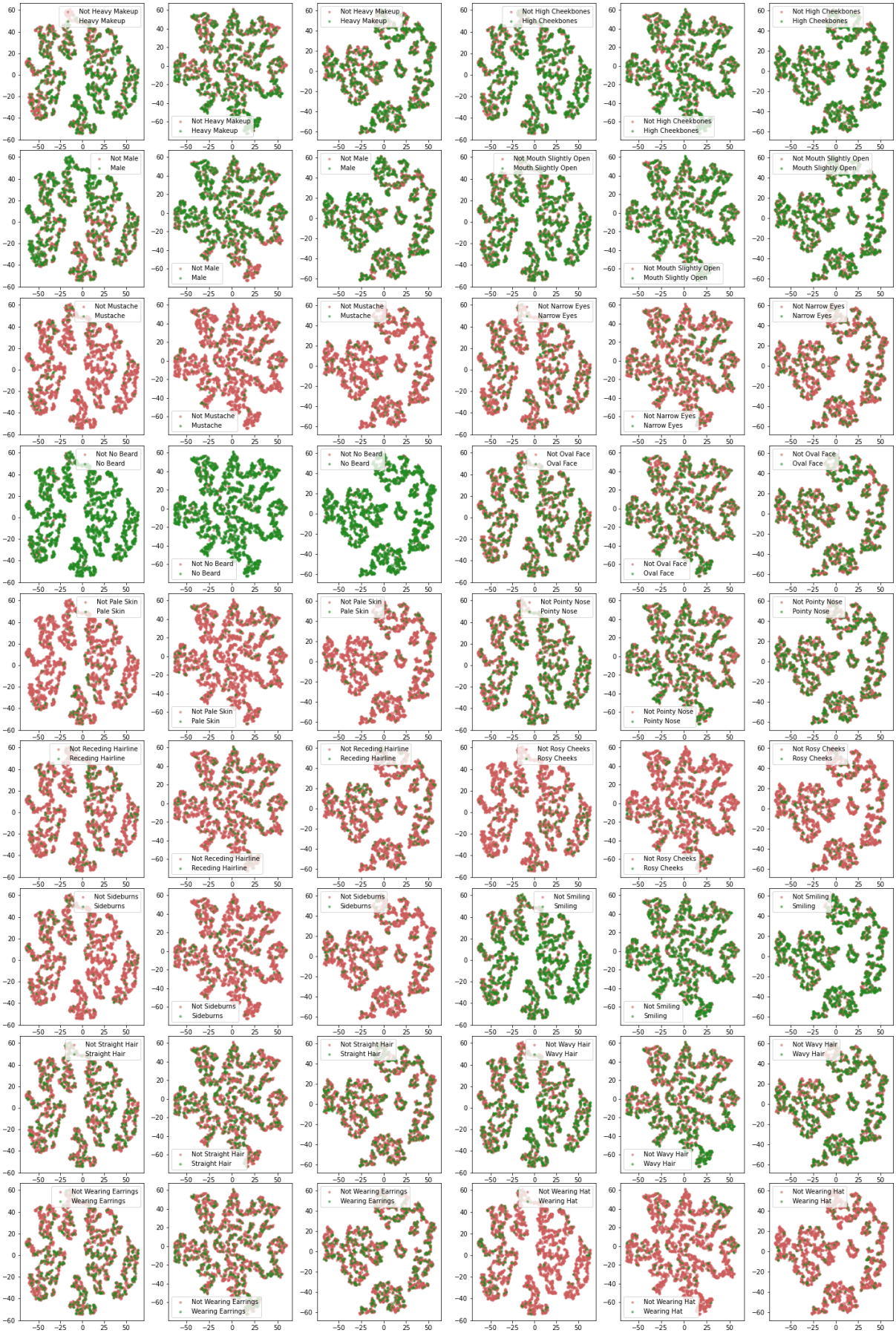


Figure 68: t-SNE-reduced Latent Space of CelebA-VAE-GAN, colored by Factors of Variation (present or not present)


Not 5 o Clock Shadow
5 o Clock Shadow
Not Arched Eyebrows
Arched Eyebrows
Not Attractive
Attractive
Not Bags Under Eyes
Bags Under Eyes
Not Bald
Bald
Not Bangs
Bangs
Not Big Lips
Big Lips
Not Big Nose
Big Nose
Not Black Hair
Black Hair
Not Blond Hair
Blond Hair
Not Blurry
Blurry
Not Brown Hair
Brown Hair
Not Bushy Eyebrows
Bushy Eyebrows
Not Chubby
Chubby
Not Double Chin
Double Chin
Not Eyeglasses
Eyeglasses
Not Goatee
Goatee
Not Gray Hair
Gray Hair


Not Heavy Makeup
Heavy Makeup
Not High Cheekbones
High Cheekbones
Not Male
Male
Not Mouth Slightly Open
Mouth Slightly Open
Not Mustache
Mustache
Not Narrow Eyes
Narrow Eyes
Not No Beard
No Beard
Not Oval Face
Oval Face
Not Pale Skin
Pale Skin
Not Pointy Nose
Pointy Nose
Not Receding Hairline
Receding Hairline
Not Rosy Cheeks
Rosy Cheeks
Not Sideburns
Sideburns
Not Smiling
Smiling
Not Straight Hair
Straight Hair
Not Wavy Hair
Wavy Hair
Not Wearing Earrings
Wearing Earrings
Not Wearing Hat
Wearing Hat

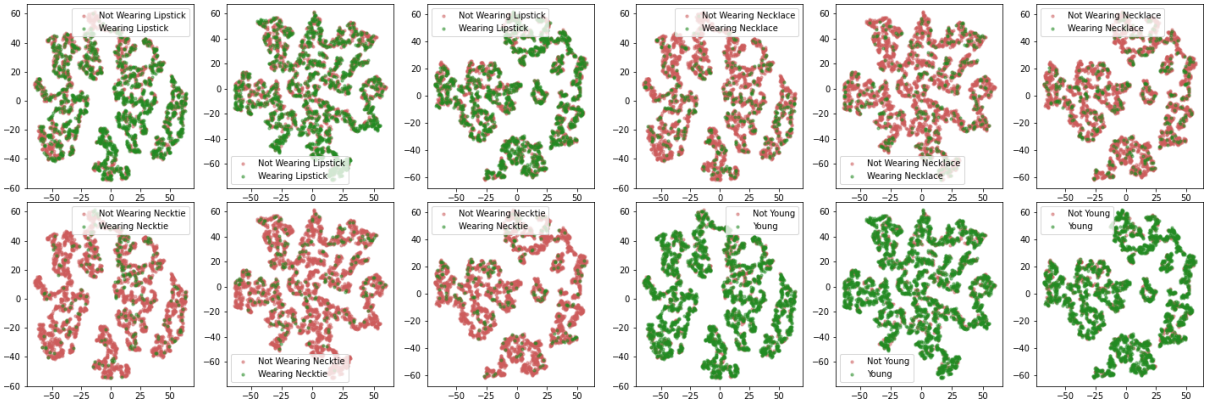
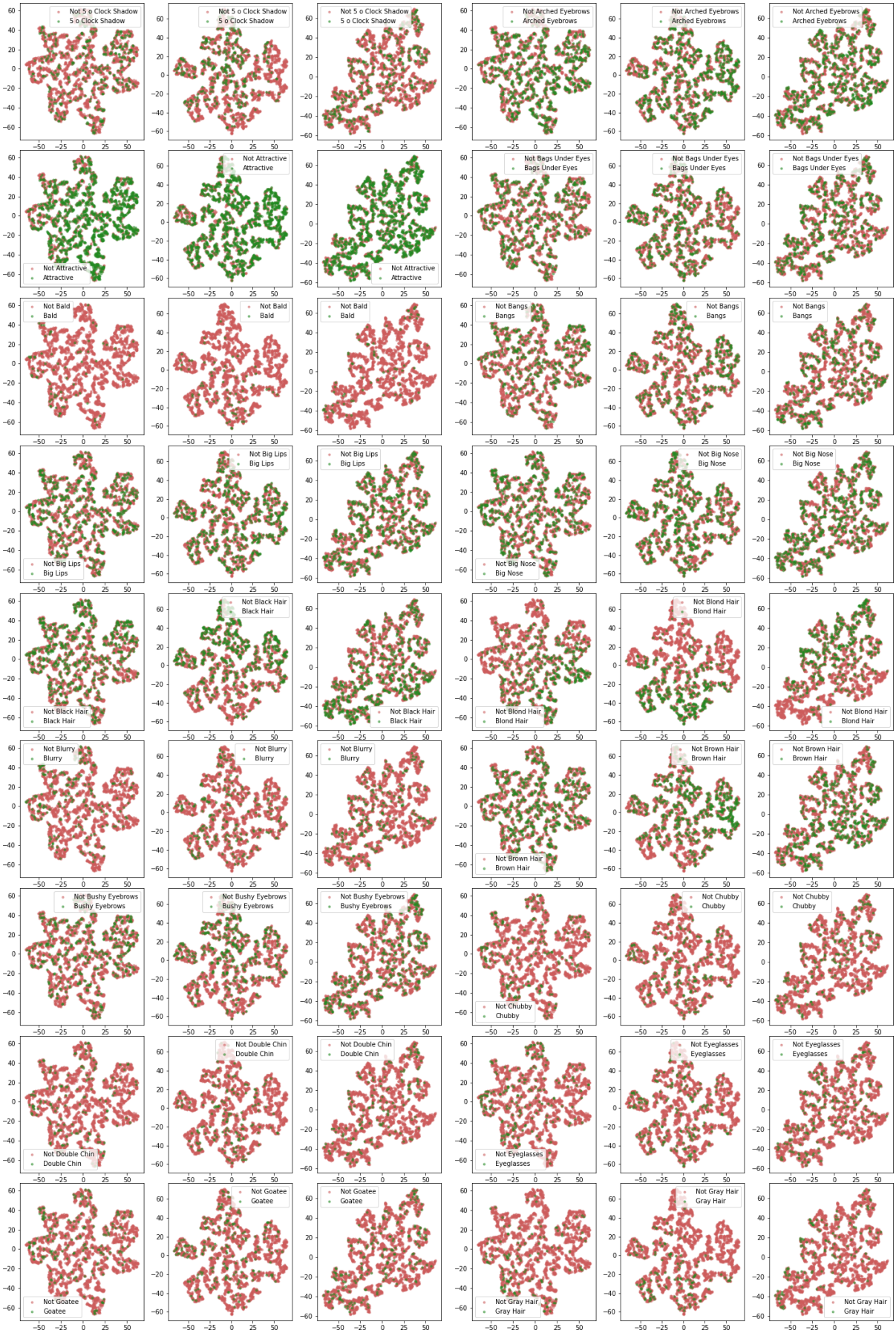
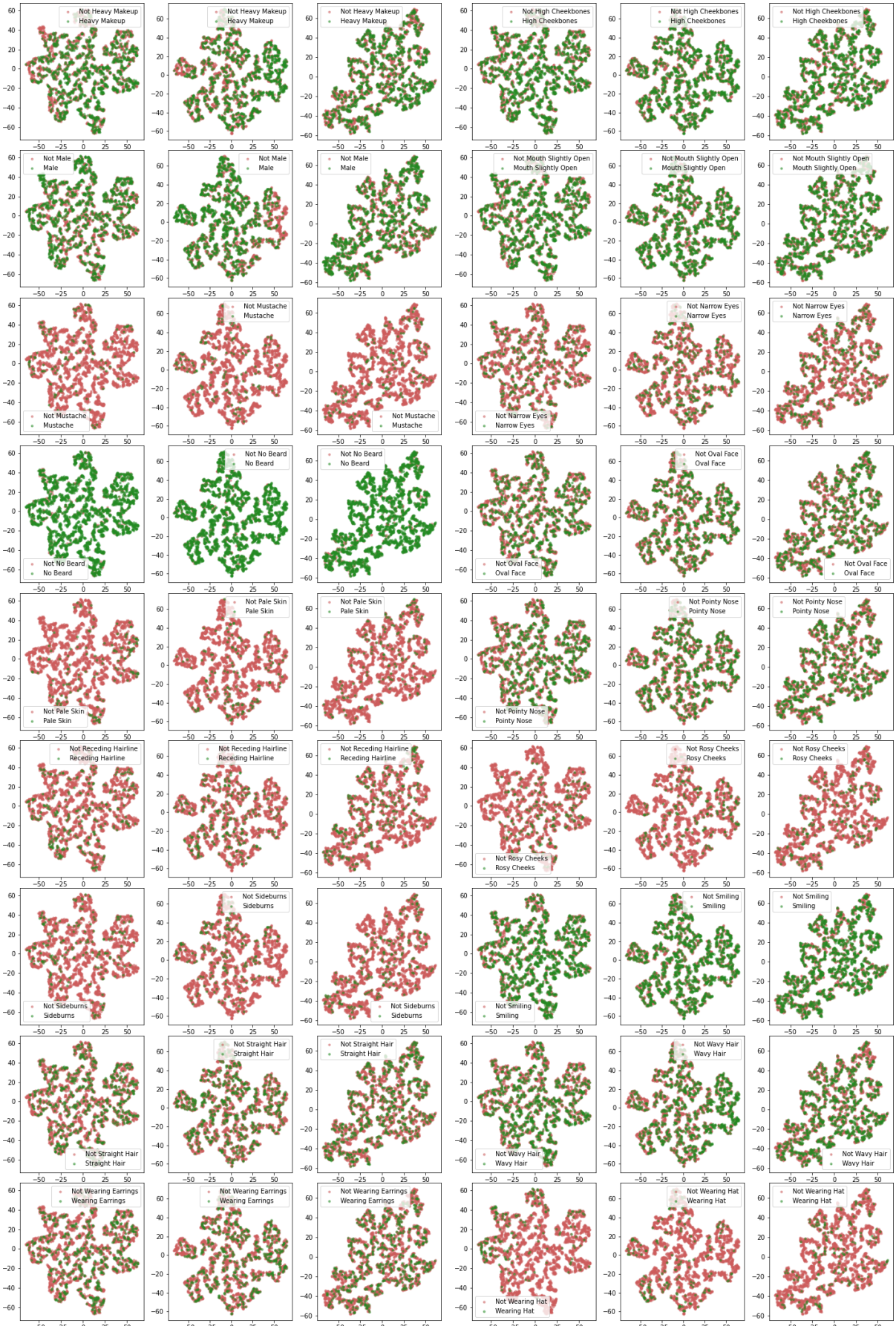


Figure 69: t-SNE-reduced Latent Space of CelebA-VLAE, colored by Factors of Variation (present or not present). Different columns correspond to the different layers.


Not 5 o Clock Shadow
5 o Clock Shadow
Not Arched Eyebrows
Arched Eyebrows
Not Attractive
Attractive
Not Bags Under Eyes
Bags Under Eyes
Not Bald
Bald
Not Bangs
Bangs
Not Big Lips
Big Lips
Not Big Nose
Big Nose
Not Black Hair
Black Hair
Not Blond Hair
Blond Hair
Not Blurry
Blurry
Not Brown Hair
Brown Hair
Not Bushy Eyebrows
Bushy Eyebrows
Not Chubby
Chubby
Not Double Chin
Double Chin
Not Eyeglasses
Eyeglasses
Not Goatee
Goatee
Not Gray Hair
Gray Hair


Not Heavy Makeup
Heavy Makeup
Not High Cheekbones
High Cheekbones
Not Male
Male
Not Mouth Slightly Open
Mouth Slightly Open
Not Mustache
Mustache
Not Narrow Eyes
Narrow Eyes
Not No Beard
No Beard
Not Oval Face
Oval Face
Not Pale Skin
Pale Skin
Not Pointy Nose
Pointy Nose
Not Receding Hairline
Receding Hairline
Not Rosy Cheeks
Rosy Cheeks
Not Sideburns
Sideburns
Not Smiling
Smiling
Not Straight Hair
Straight Hair
Not Wavy Hair
Wavy Hair
Not Wearing Earrings
Wearing Earrings
Not Wearing Hat
Wearing Hat

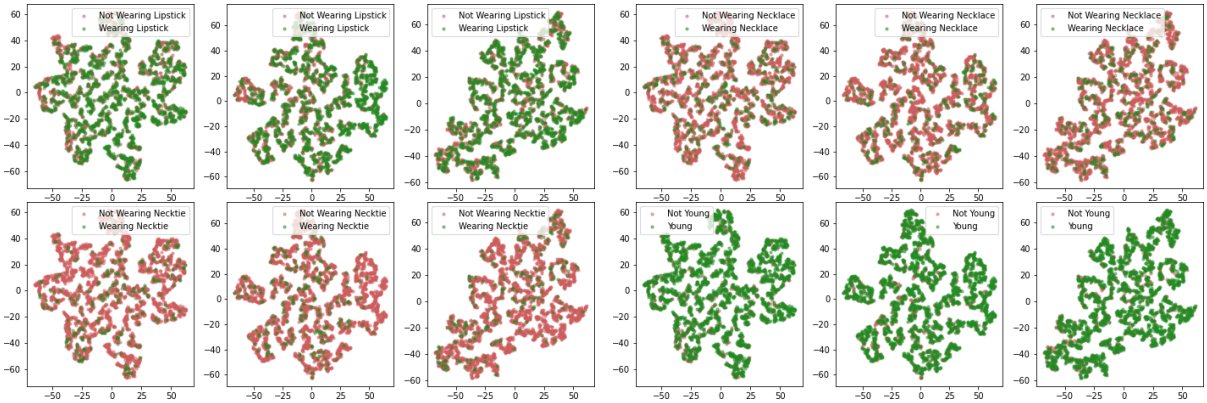


Figure 70: t-SNE-reduced Latent Space of CelebA-VLAE-GAN, colored by Factors of Variation (present or not present). Different columns correspond to the different layers.

F Additional Plots for Section 4.4.2

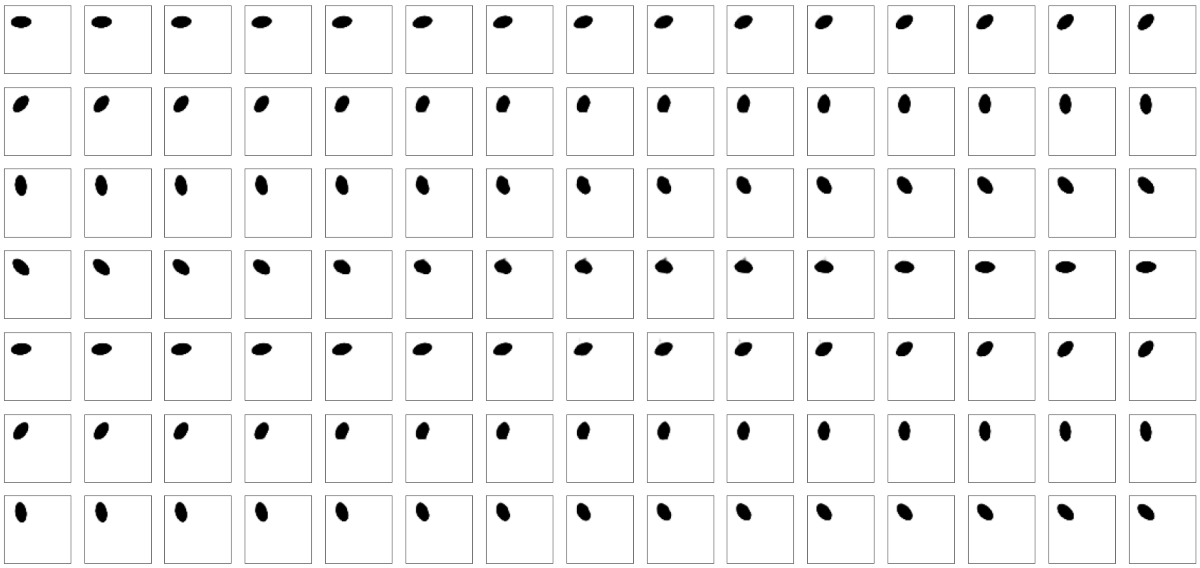


Figure 71: Latent spaces traversal between different rotation values for 7,500-VAE on the dSprites dataset

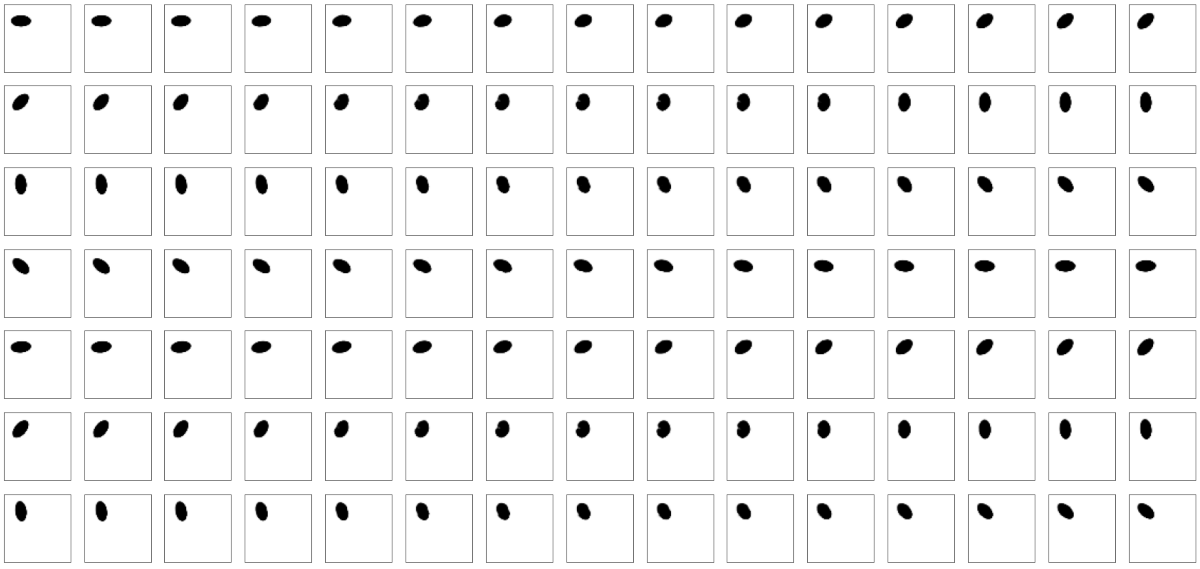


Figure 72: Latent spaces traversal between different rotation values for 6,250-VAE on the dSprites dataset

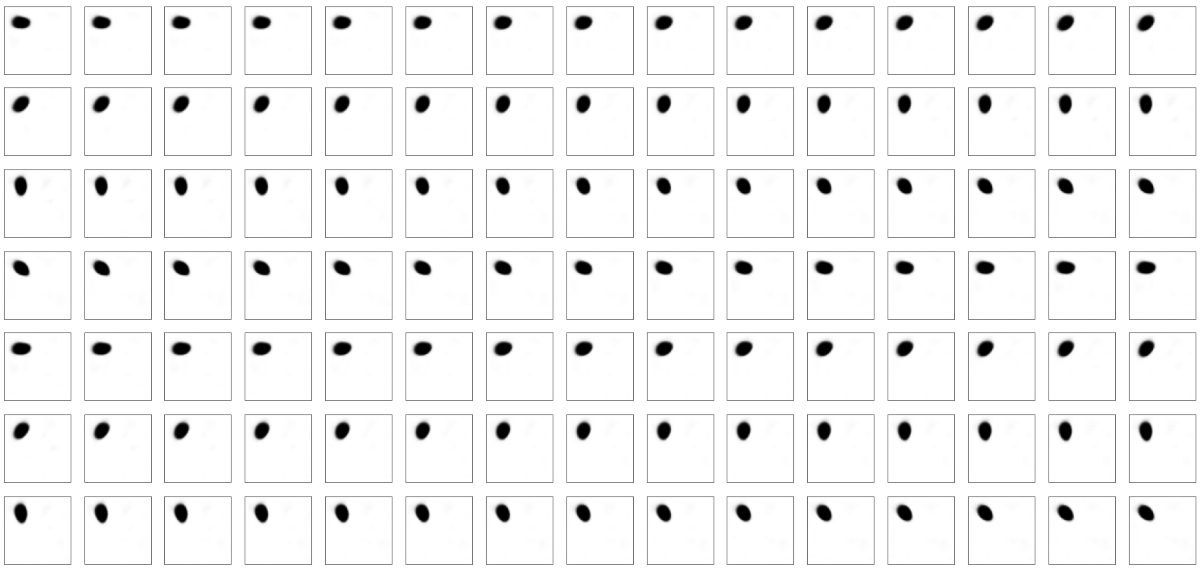


Figure 73: Latent spaces traversal between different rotation values for 5,000-VAE on the dsprites dataset

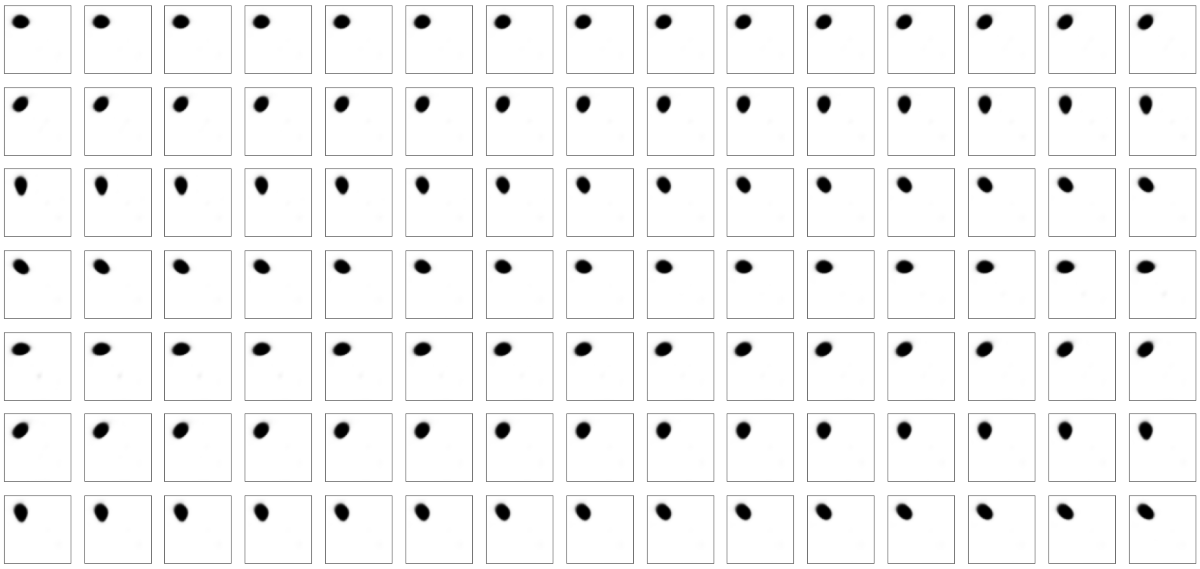
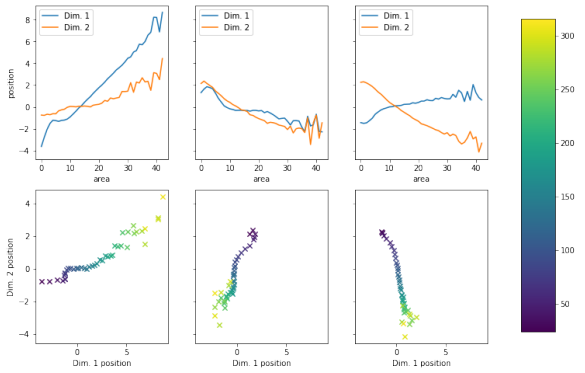
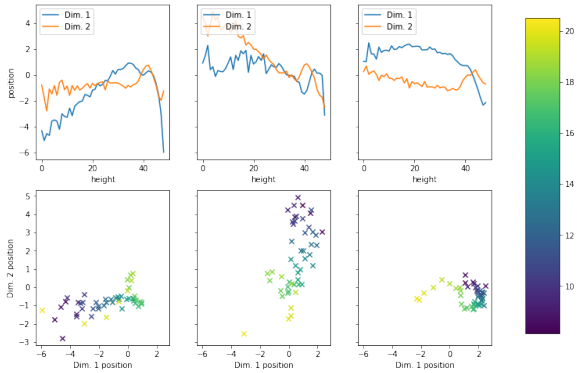


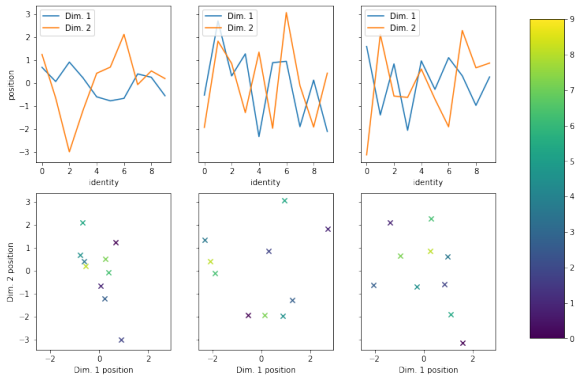
Figure 74: Latent spaces traversal between different rotation values for 3,750-VAE on the dsprites dataset



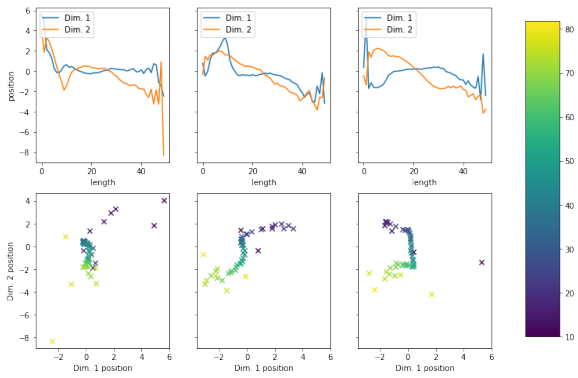
(a) area



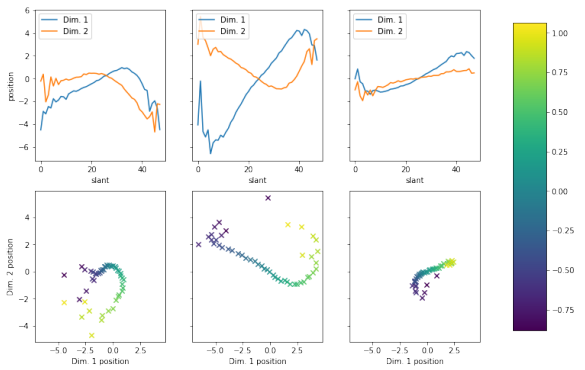
(b) height



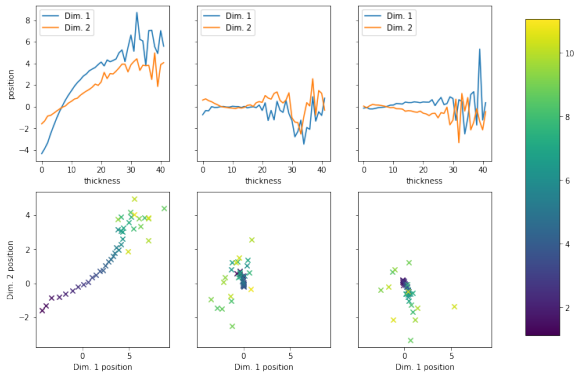
(c) identity



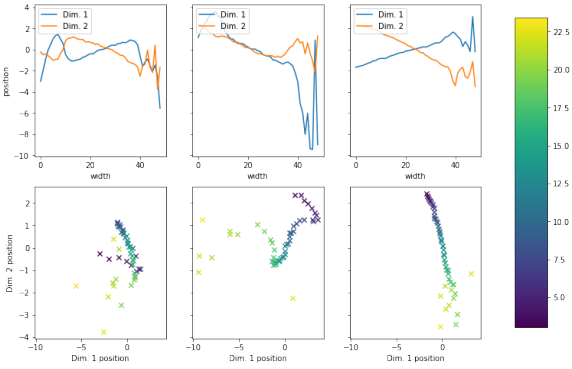
(d) length



(e) slant



(f) thickness



(g) width

Figure 75: Mean latent space values for MNIST-VLAE-GAN when fixing different factors of variation from Morpho-MNIST

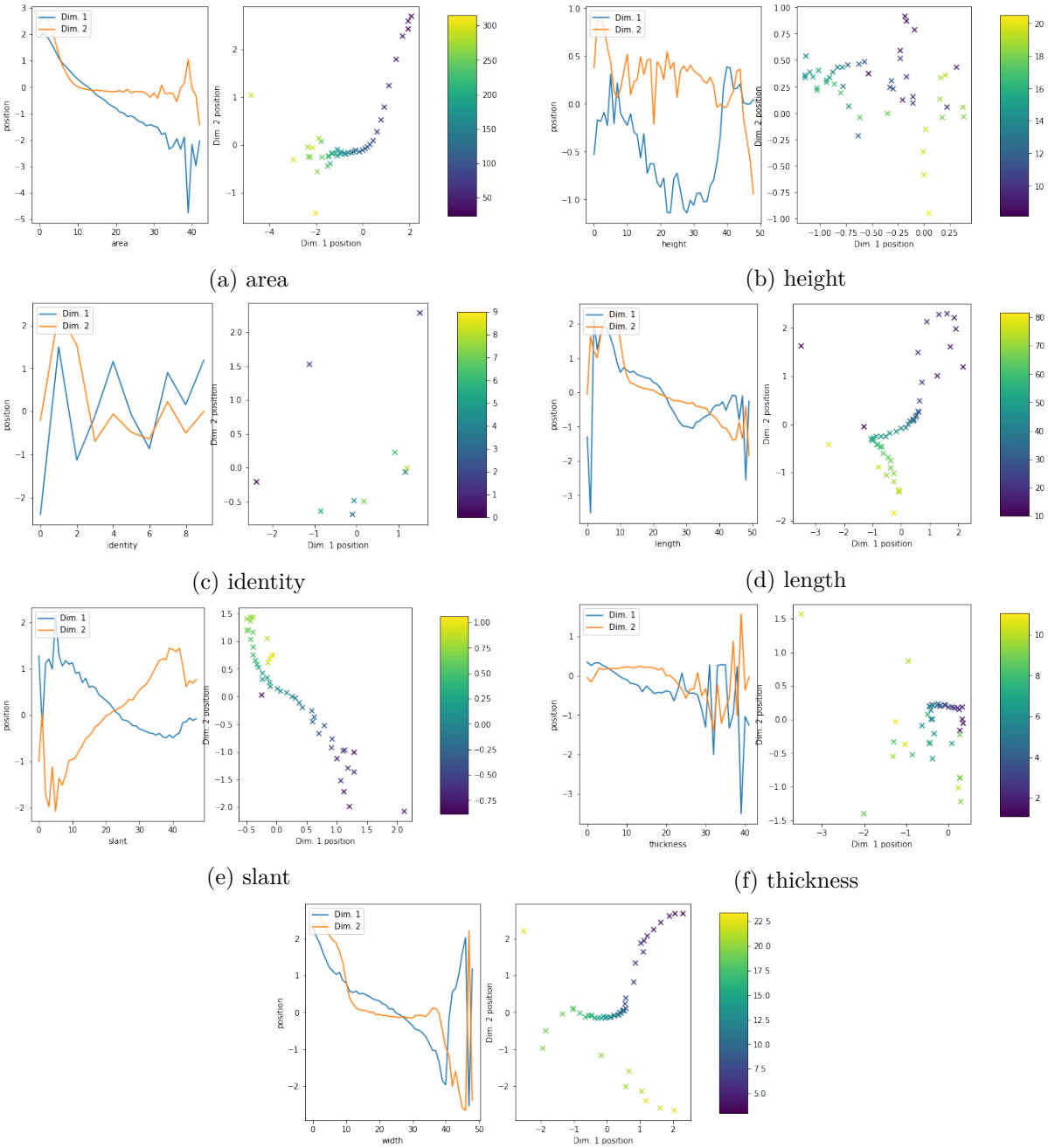


Figure 76: Mean latent space values for MNIST-VAE when fixing different factors of variation from Morpho-MNIST

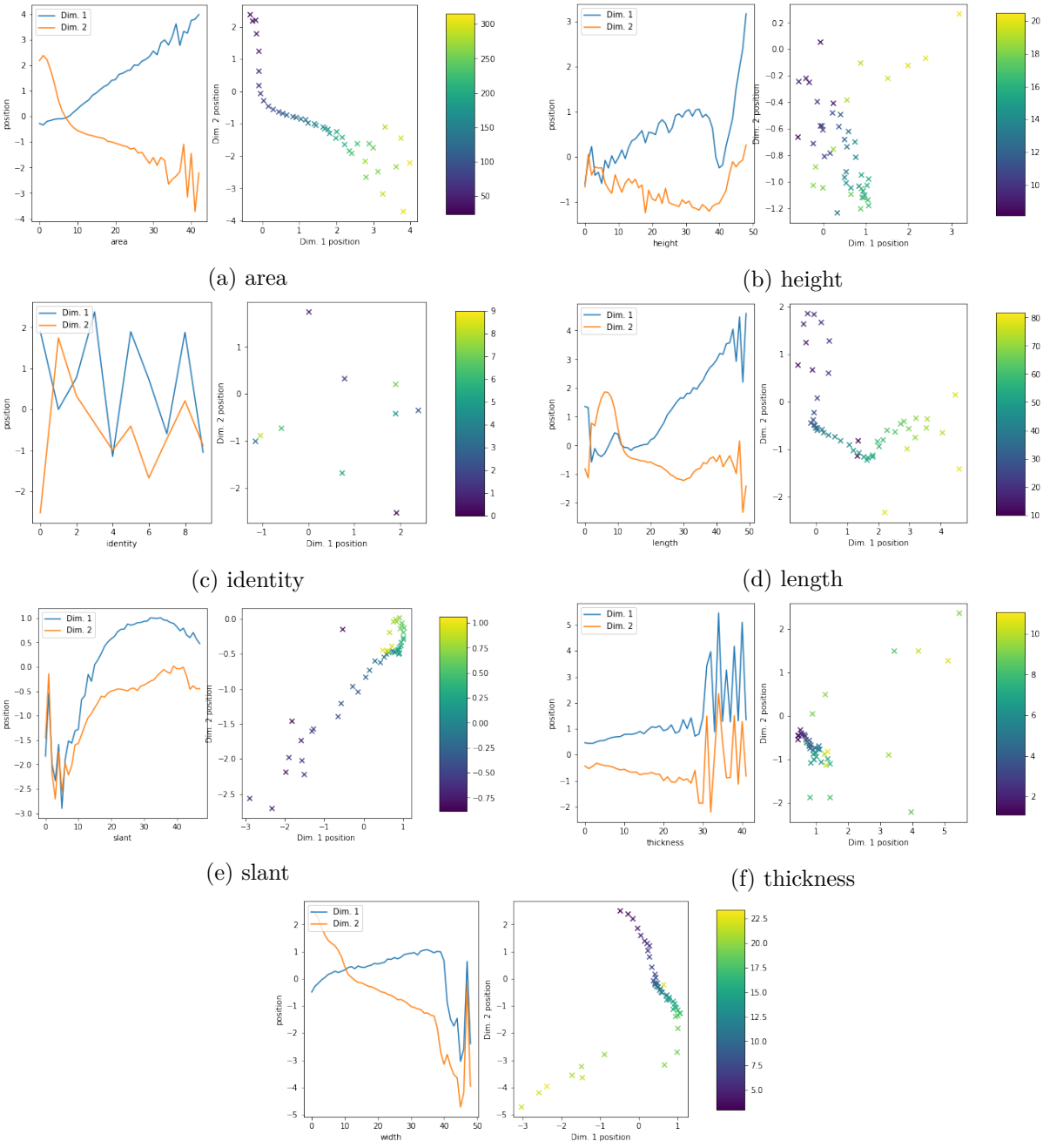
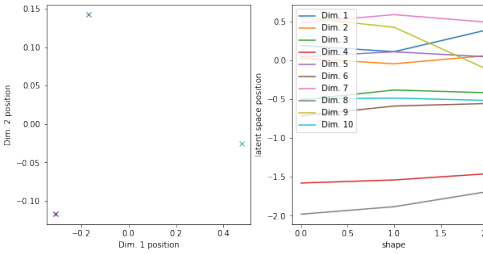
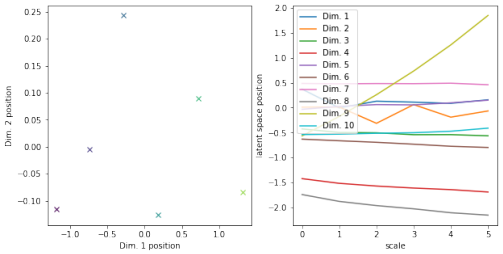


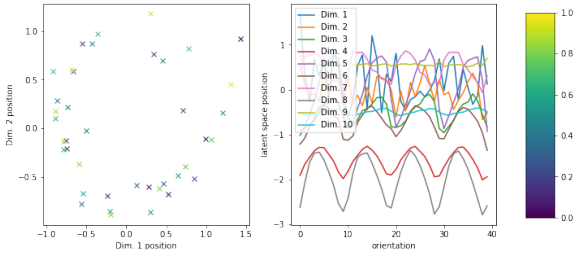
Figure 77: Mean latent space values for MNIST-VAE-GAN when fixing different factors of variation from Morpho-MNIST



(a) Traversal of the reduced latent space for different shapes.

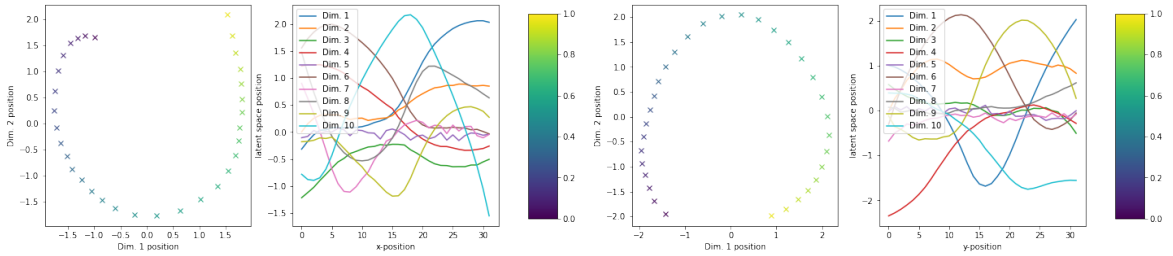


(b) Traversal of the reduced latent space for different scales.



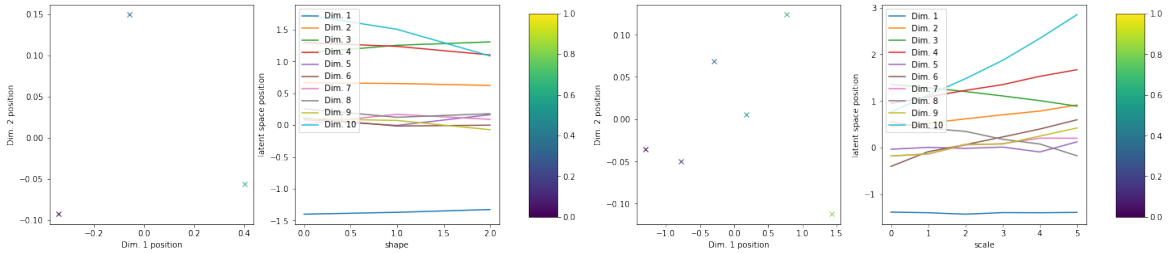
(c) Traversal of the reduced latent space for different orientations.

Figure 78: Latent space of 10,000-VAE. Different values are either for different x-positions or for different y-position. The other position is fixed to 1.0. It is averaged over all other parameters.



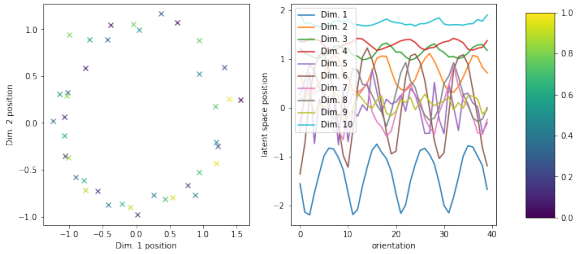
(a) Traversal of the reduced latent space for different x-positions.

(b) Traversal of the reduced latent space for different y-positions.



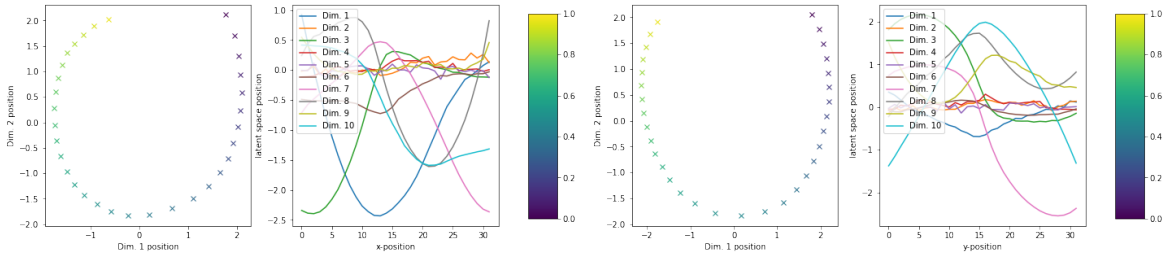
(c) Traversal of the reduced latent space for different shapes.

(d) Traversal of the reduced latent space for different scales.



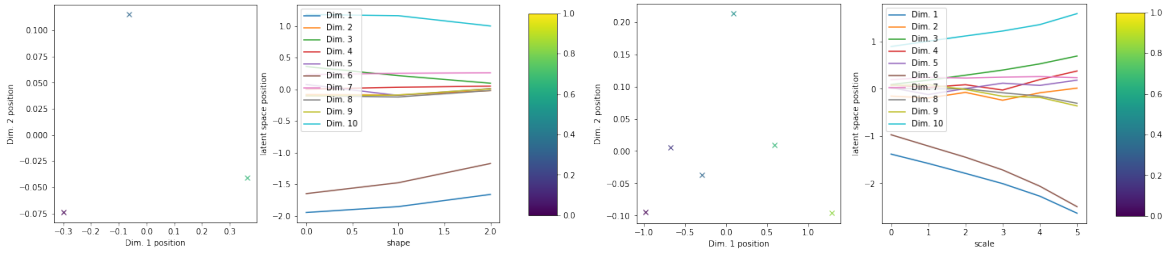
(e) Traversal of the reduced latent space for different orientations.

Figure 79: Latent space of 7,500-VAE. Different values are either for different x-positions or for different y-position. The other position is fixed to 1.0. It is averaged over all other parameters.



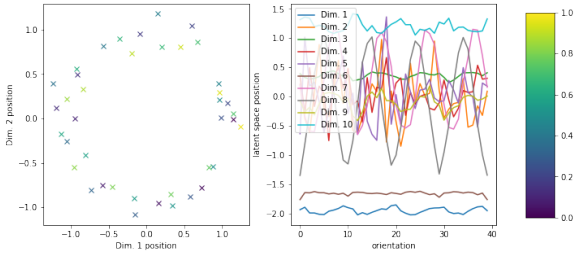
(a) Traversal of the reduced latent space for different x-positions.

(b) Traversal of the reduced latent space for different y-positions.



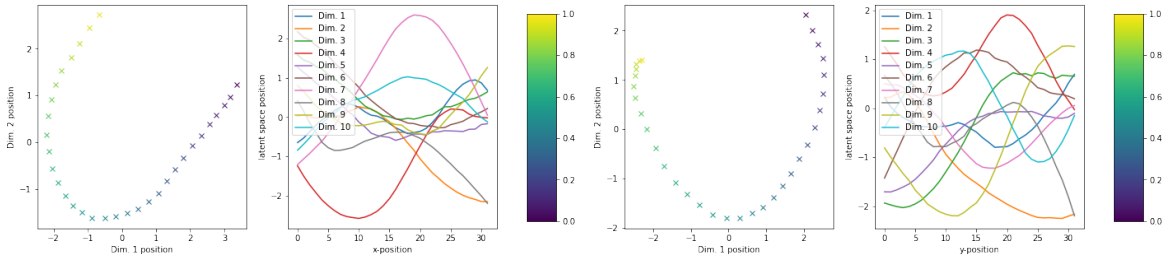
(c) Traversal of the reduced latent space for different shapes.

(d) Traversal of the reduced latent space for different scales.



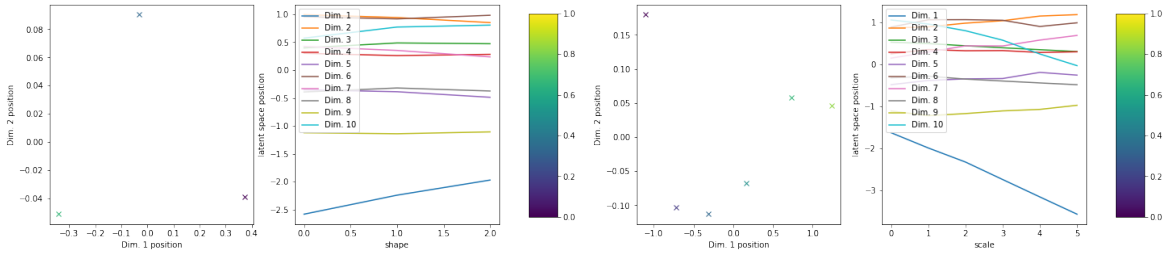
(e) Traversal of the reduced latent space for different orientations.

Figure 80: Latent space of 6,250-VAE. Different values correspond to different values for the respective factor of variation. For orientation and scale, the position is fixed to 0.0 in both directions, shape is fixed to *Square*. For x-and y-postition, the other position is fixed to 1.0 and shape is fixed to *Square*.



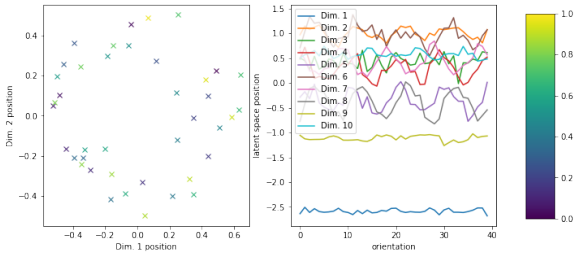
(a) Traversal of the reduced latent space for different x-positions.

(b) Traversal of the reduced latent space for different y-positions.



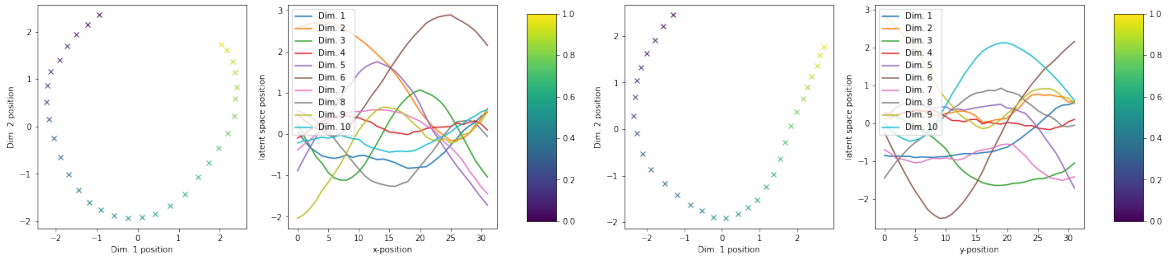
(c) Traversal of the reduced latent space for different shapes.

(d) Traversal of the reduced latent space for different scales.



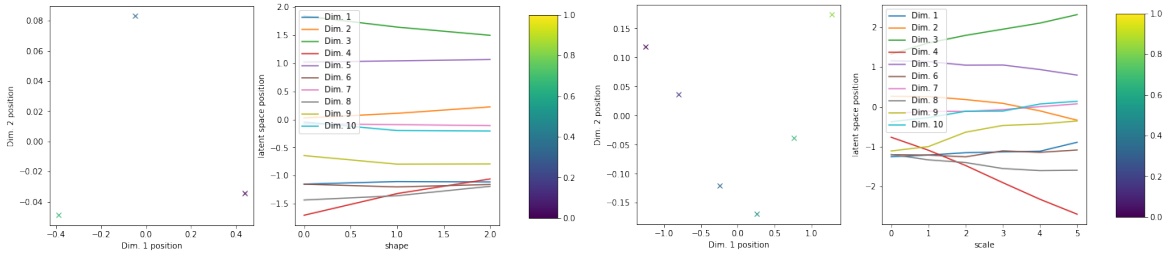
(e) Traversal of the reduced latent space for different orientations.

Figure 81: Latent space of 5,000-VAE. Different values correspond to different values for the respective factor of variation. For orientation and scale, the position is fixed to 0.0 in both directions, shape is fixed to *Square*. For x-and y-postition, the other position is fixed to 1.0 and shape is fixed to *Square*.



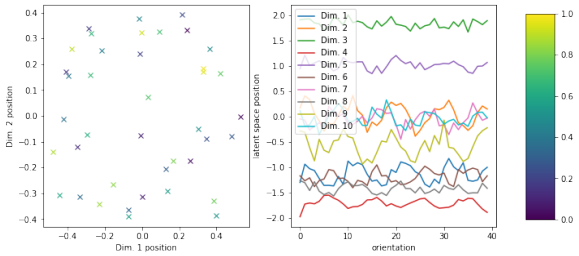
(a) Traversal of the reduced latent space for different x-positions.

(b) Traversal of the reduced latent space for different y-positions.



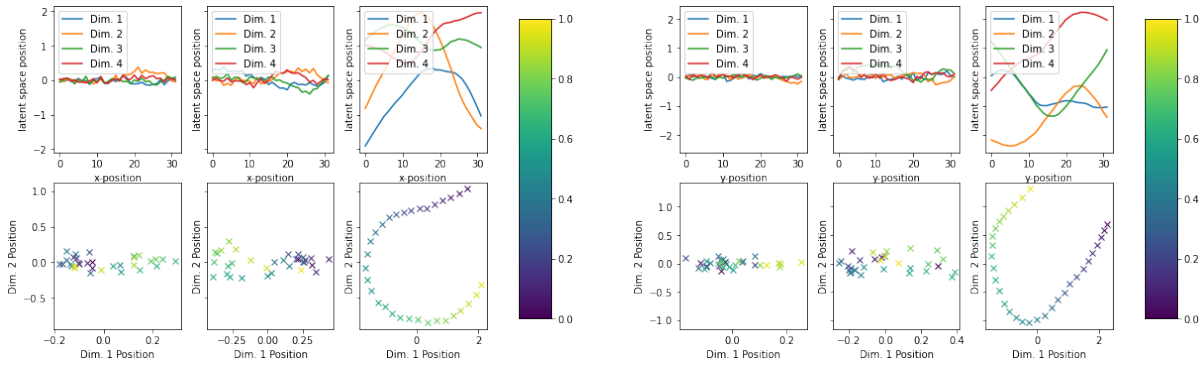
(c) Traversal of the reduced latent space for different shapes.

(d) Traversal of the reduced latent space for different scales.



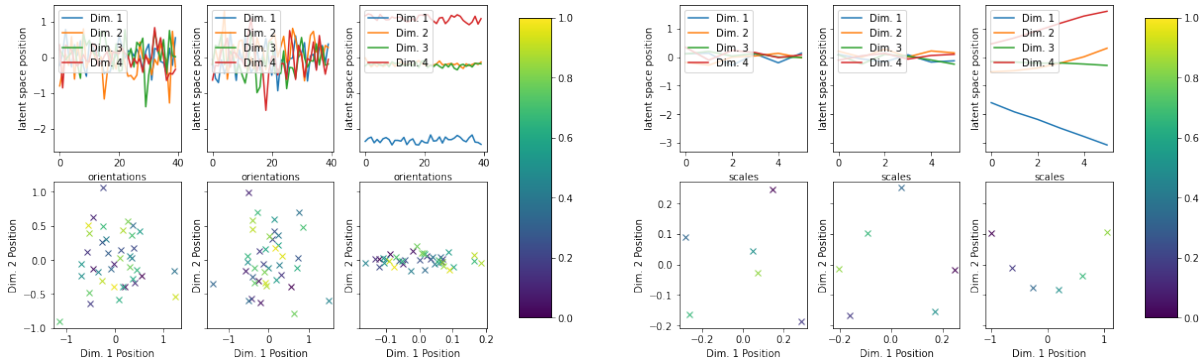
(e) Traversal of the reduced latent space for different orientations.

Figure 82: Latent space of 3,750-VAE. Different values correspond to different values for the respective factor of variation. For orientation and scale, the position is fixed to 0.0 in both directions, shape is fixed to *Square*. For x-and y-postition, the other position is fixed to 1.0 and shape is fixed to *Square*.



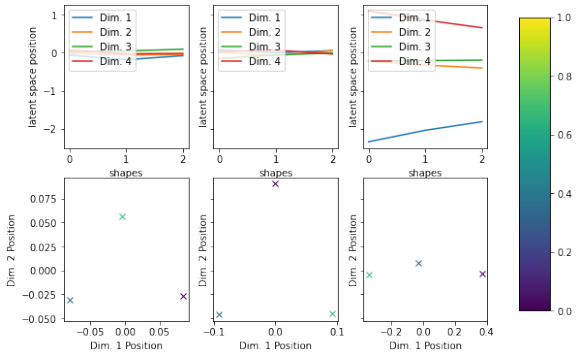
(a) Varying x-position

(b) Varying y-position



(c) Varying orientation

(d) Varying scale



(e) Varying shape

Figure 83: Values of different dimensions and layers in the dSprites-VLAE-GAN latent space for different factor of variation values (first row in each subplot), and position in a principal component analysis (PCA)-reduced latent space (second row in each subplot). The left column corresponds to the first embedding layer, the right one to the third. PCA was performed separately for each factor of variation and latent space layer. Different values correspond to different values for the respective factor of variation. For orientation and scale, the position is fixed to 0.0 in both directions, shape is fixed to *Square*. For x-and y-postition, the other position is fixed to 1.0 and shape is fixed to *Square*.

G Additional Plots for Section 4.5

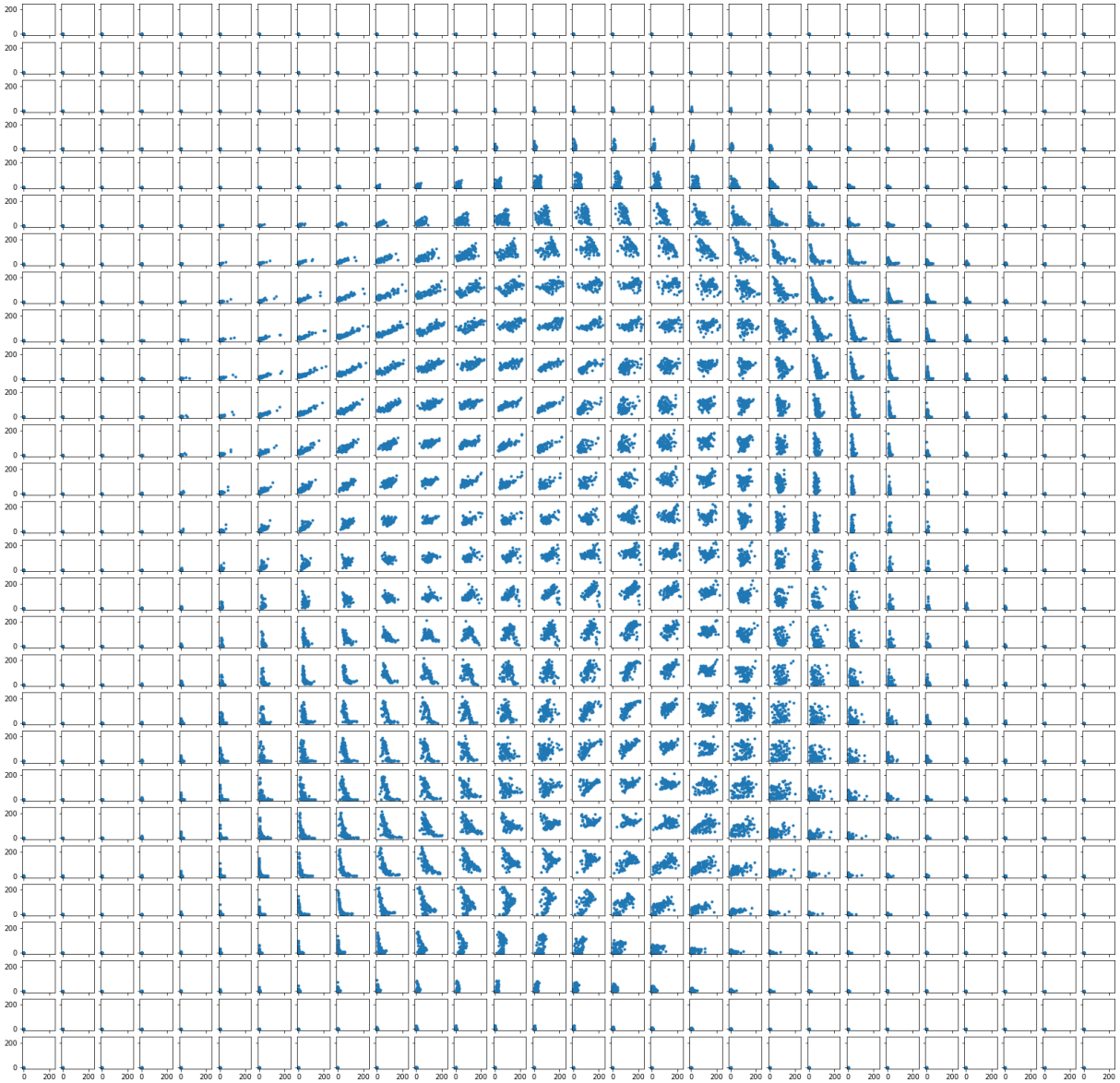


Figure 84: Proportionality of pixel intensities when fixing $\boldsymbol{z}_1 = \boldsymbol{z}_3 = \varphi$ for MNIST-VLAE. Created accordingly to Figure 45.

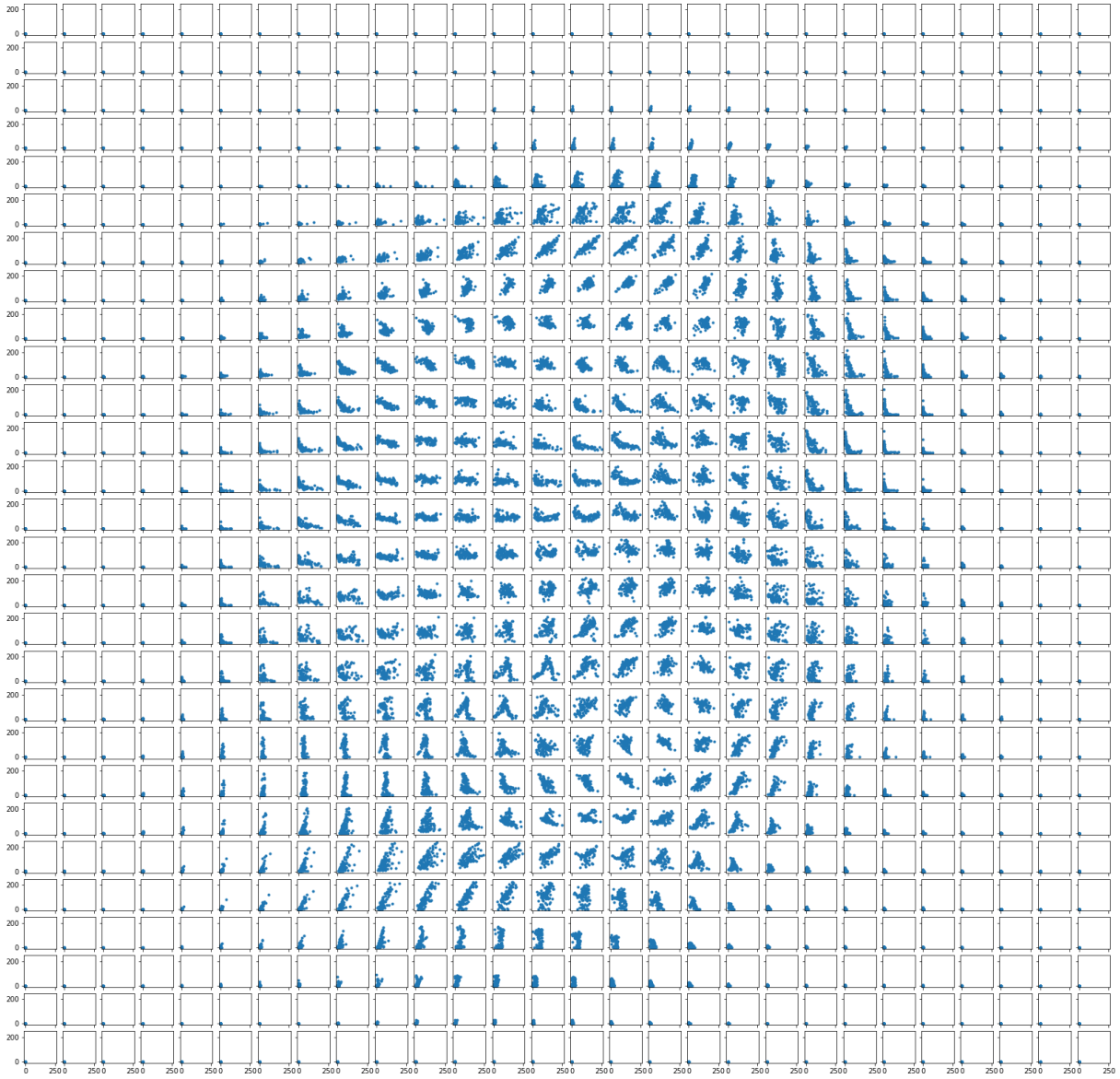


Figure 85: Correlation of pixel intensities when fixing $\boldsymbol{z}_2 = \boldsymbol{z}_3 = \varphi$ for MNIST-VLAE. Created accordingly to Figure 45.

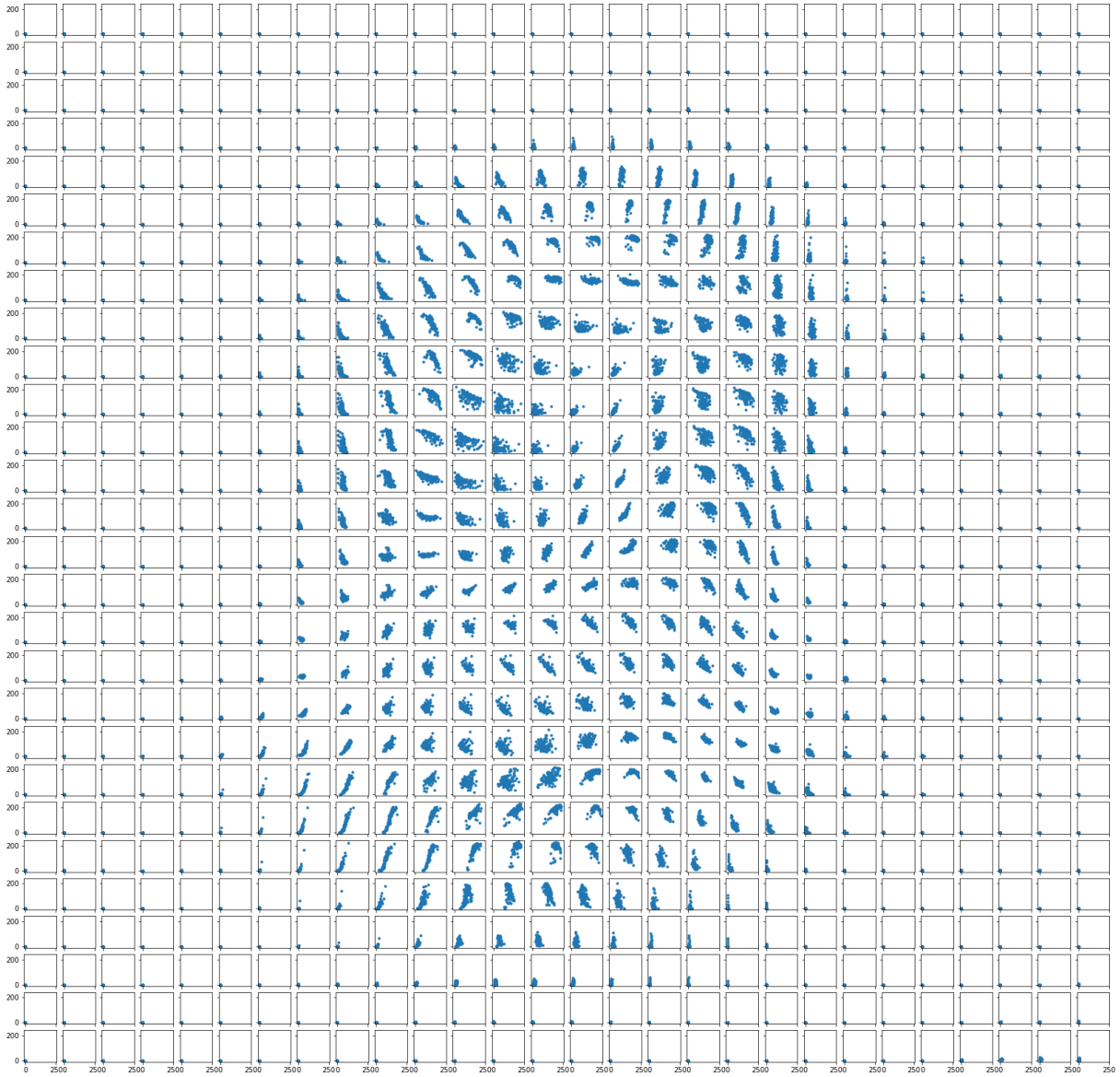


Figure 86: Correlation of pixel intensities when fixing $\boldsymbol{z}_1 = \boldsymbol{z}_2 = \varphi$ for MNIST-VLAE-GAN. Created accordingly to Figure 45.

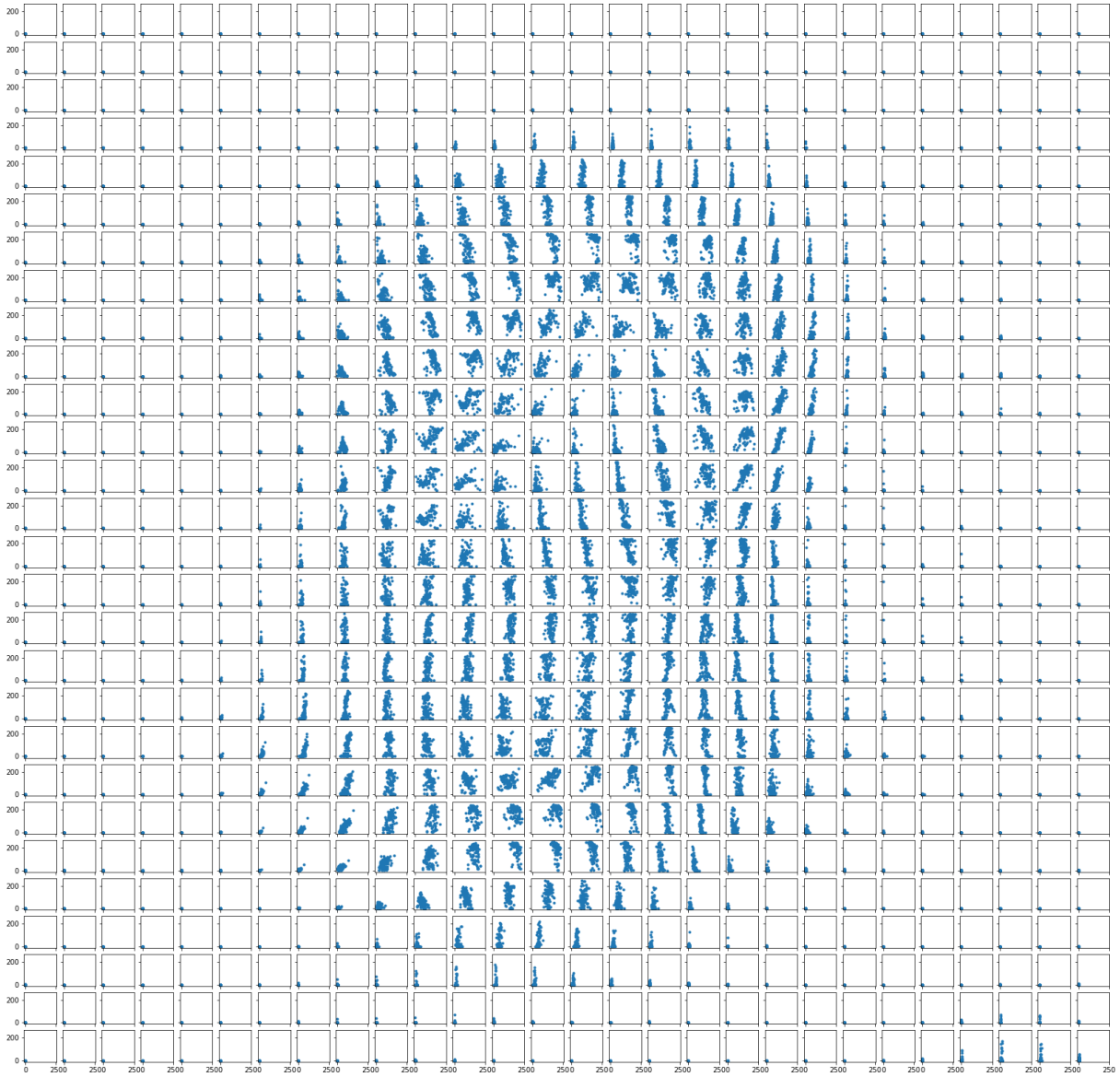


Figure 87: Correlation of pixel intensities when fixing $\boldsymbol{z}_1 = \boldsymbol{z}_3 = \varphi$ for MNIST-VLAE-GAN. Created accordingly to Figure 45.

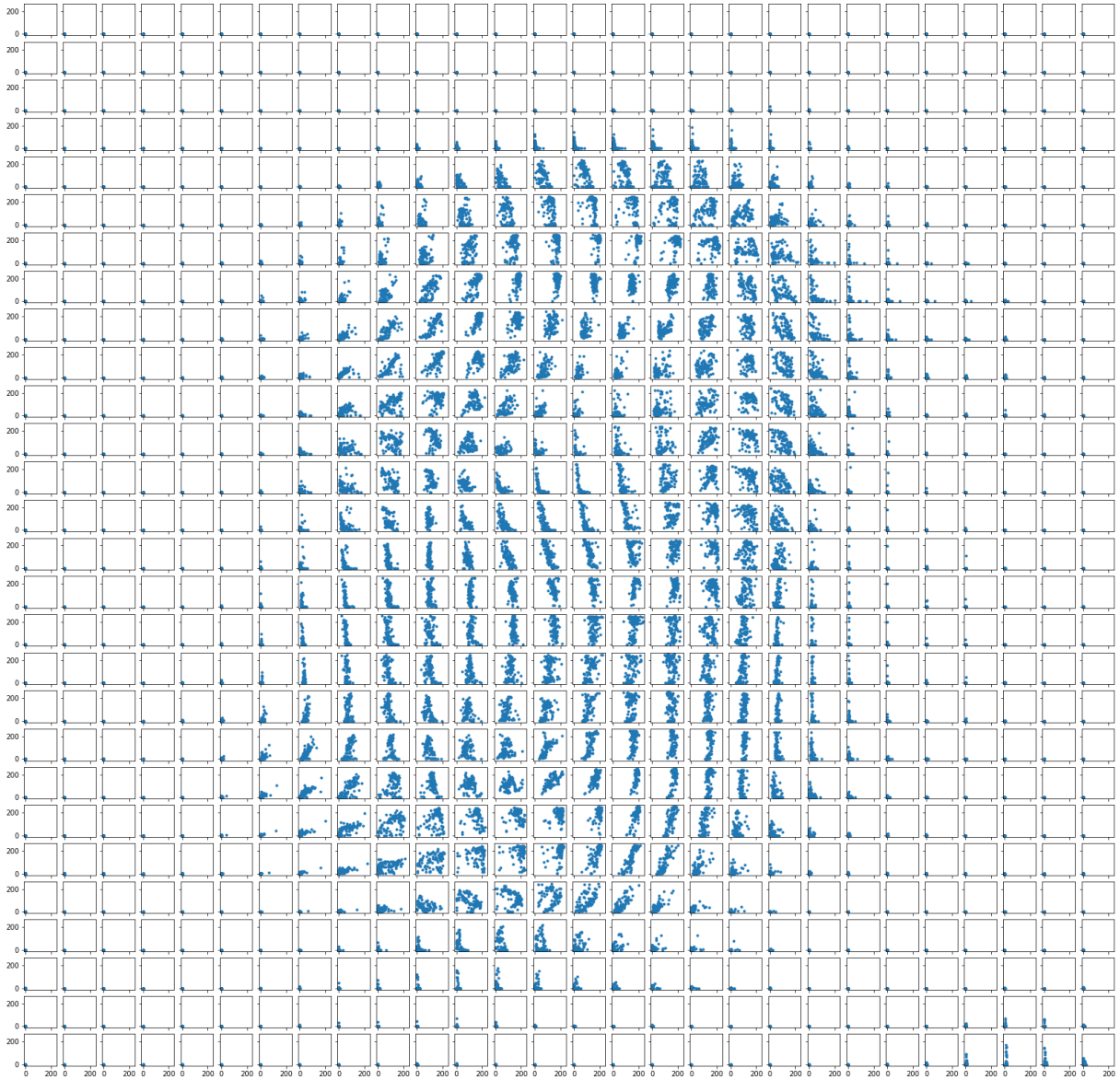


Figure 88: Correlation of pixel intensities when fixing $\boldsymbol{z}_2 = \boldsymbol{z}_3 = \varphi$ for MNIST-VLAE-GAN. Created accordingly to Figure 45.

H Additional Plots for Section 4.6

H.1 Mnist

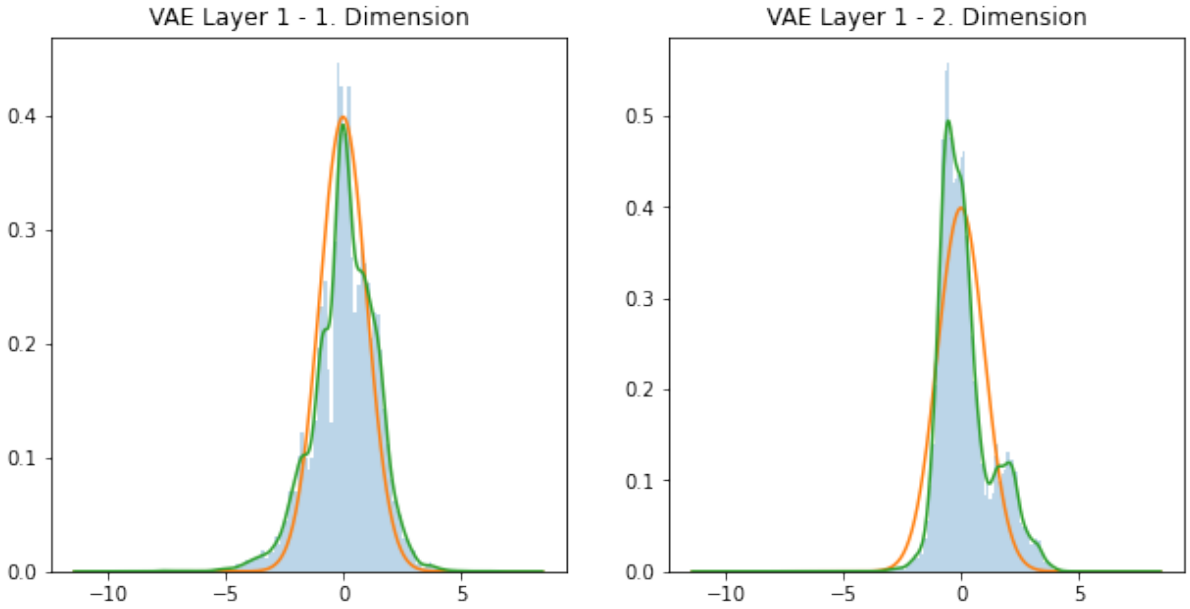


Figure 89: Latent space distribution for different layers and dimensions of MNIST-VAE (blue), the result of the kernel density estimation (KDE) (green), and a standard normal distribution (orange).

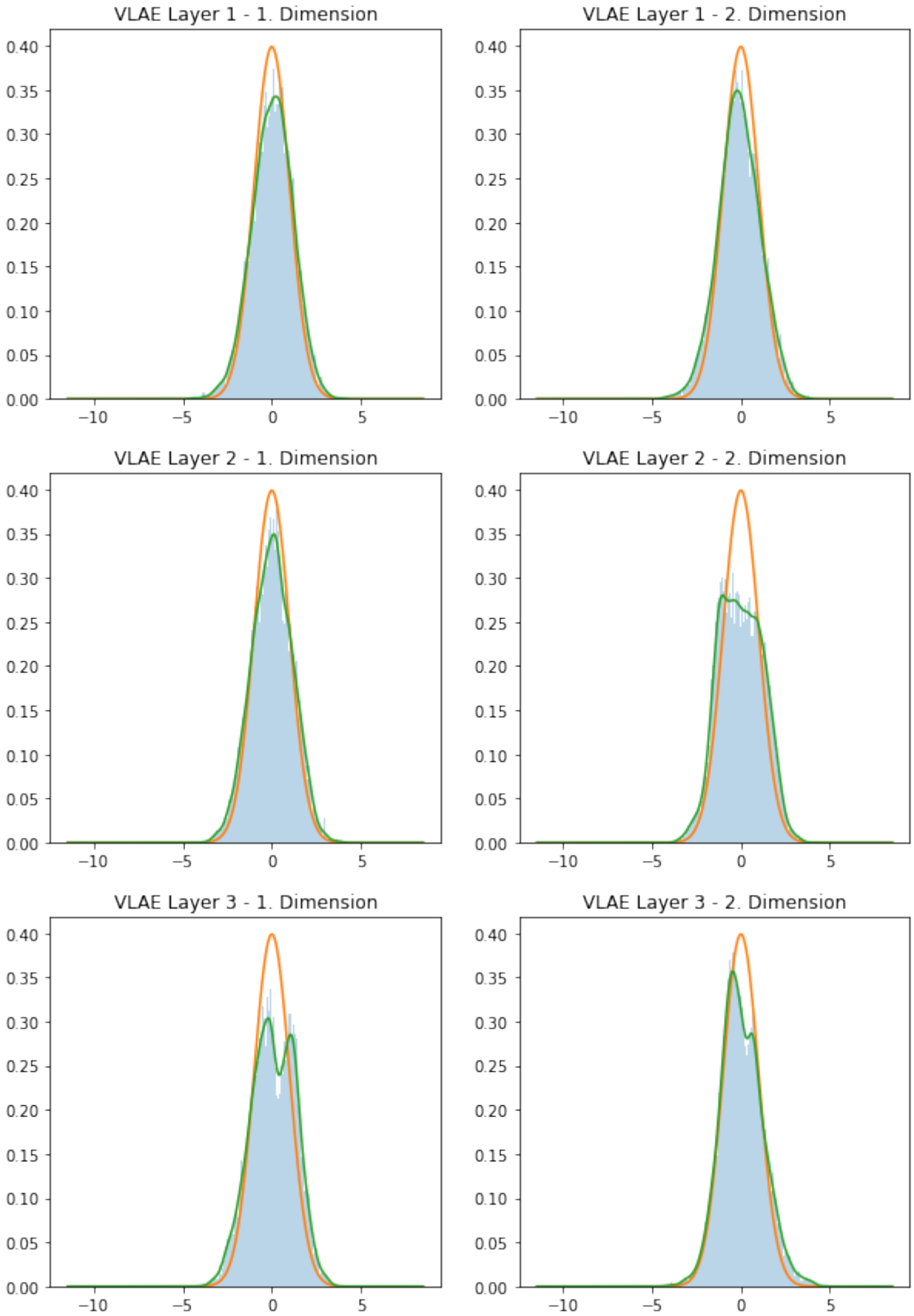


Figure 90: Latent space distribution for different layers and dimensions of MNIST-VLAE (blue), the result of the KDE (green), and a standard normal distribution (orange).

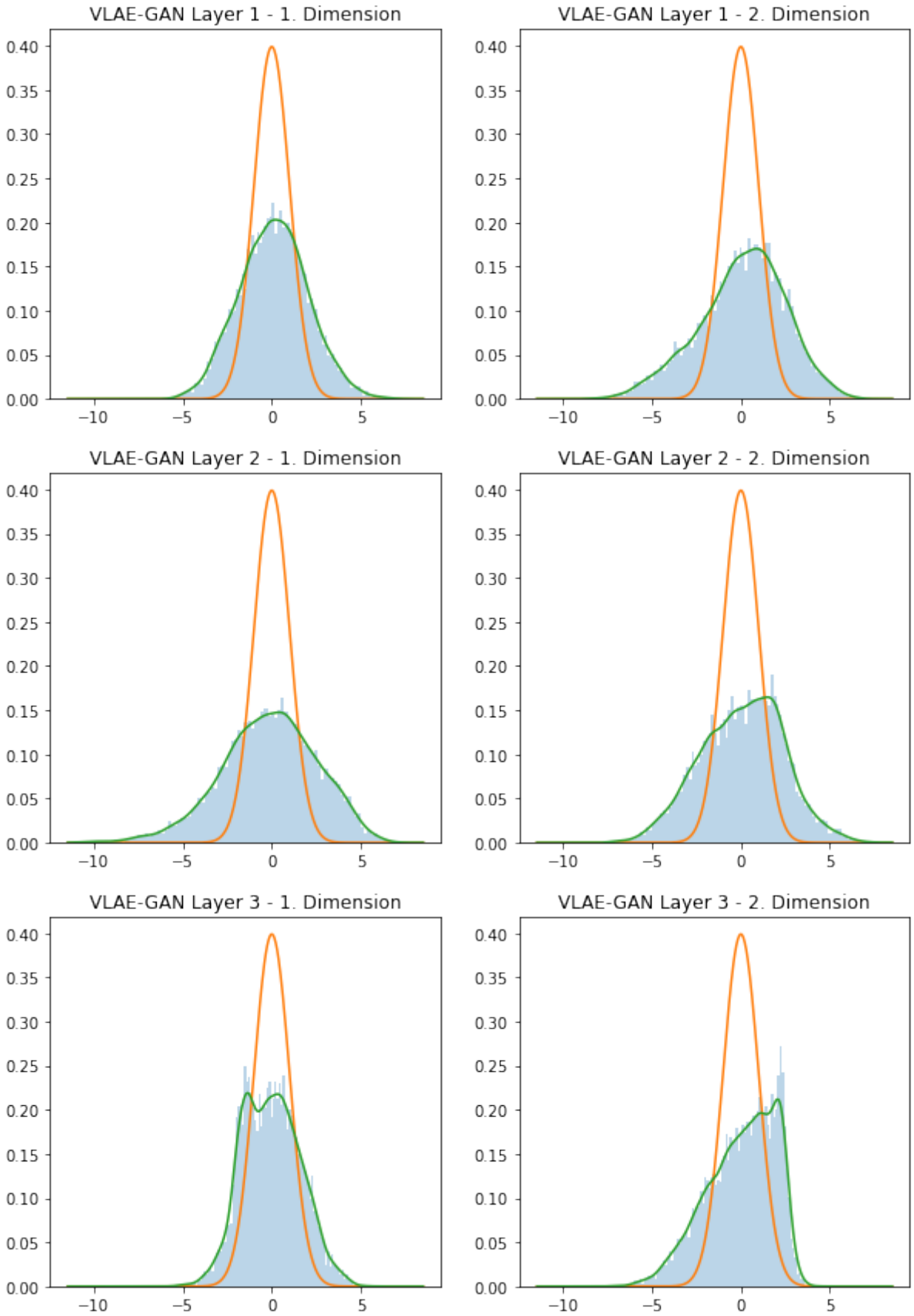


Figure 91: Latent space distribution for different layers and dimensions of Mnist-VLAE-GAN (blue), the result of the KDE (green), and a standard normal distribution (orange).

H.2 dsprites

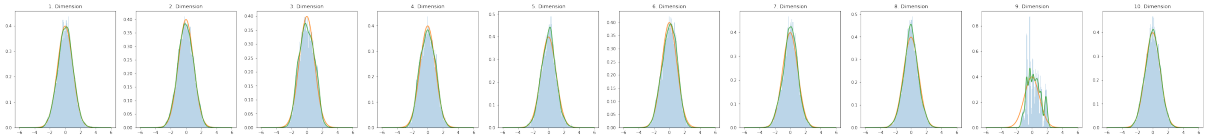


Figure 92: Latent space distribution for different layers and dimensions of dsprites-VAE (blue), the result of the KDE (green), and a standard normal distribution (orange).

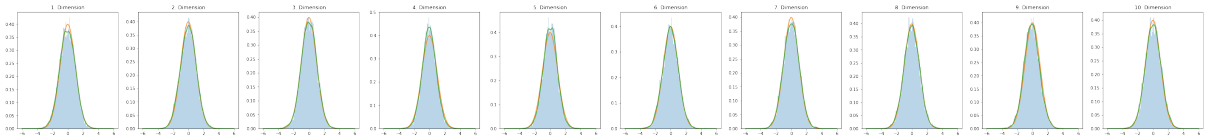


Figure 93: Latent space distribution for different layers and dimensions of dsprites-VAE-GAN (blue), the result of the KDE (green), and a standard normal distribution (orange).

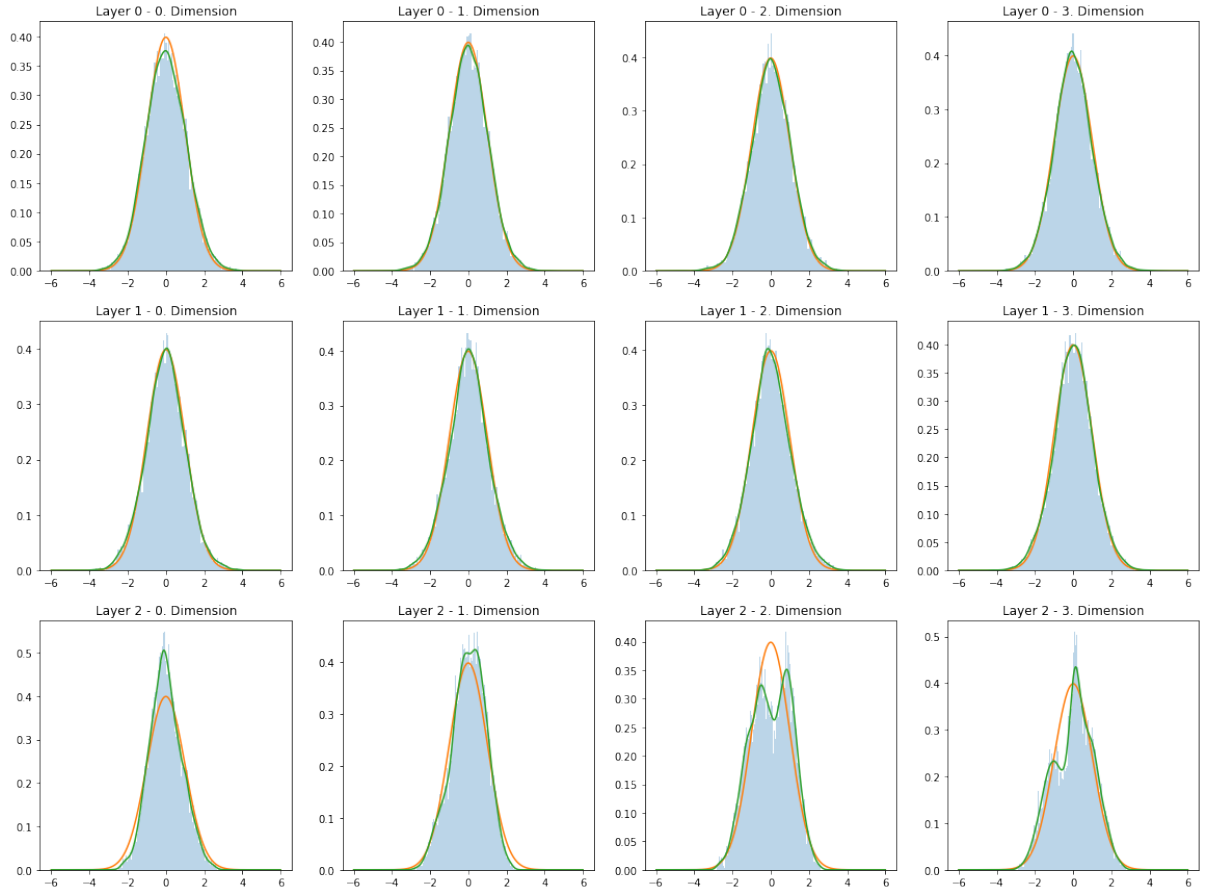


Figure 94: Latent space distribution for different layers and dimensions of dsprites-VLAE (blue), the result of the KDE (green), and a standard normal distribution (orange).

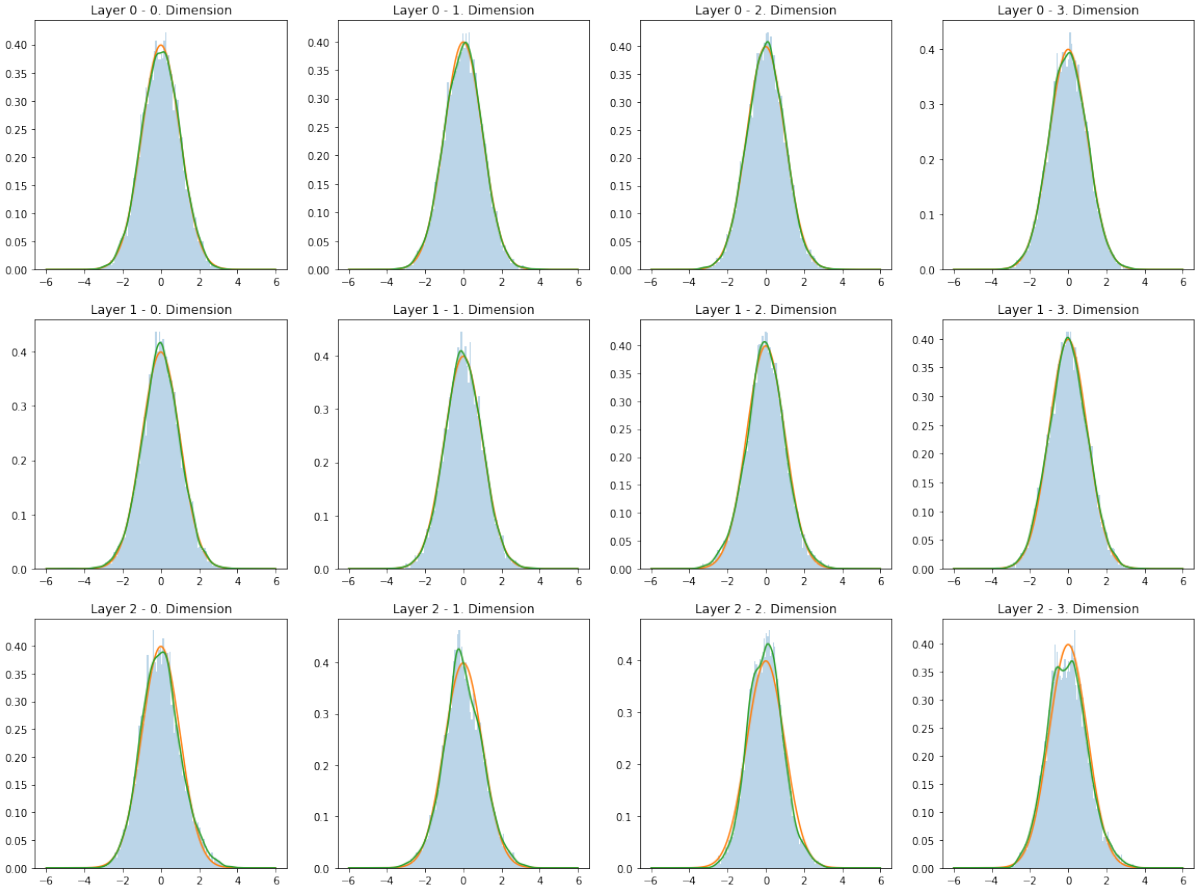


Figure 95: Latent space distribution for different layers and dimensions of dsprites-VLAE-GAN (blue), the result of the KDE (green), and a standard normal distribution (orange).

I Discriminator Network - Section 4.6.1

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 28, 28, 1)	0
conv2d_4 (Conv2D)	(None, 24, 24, 20)	520
leaky_re_lu_4 (LeakyReLU)	(None, 24, 24, 20)	0
batch_normalization_23 (Batc	(None, 24, 24, 20)	80
conv2d_5 (Conv2D)	(None, 22, 22, 20)	3620
leaky_re_lu_5 (LeakyReLU)	(None, 22, 22, 20)	0
batch_normalization_24 (Batc	(None, 22, 22, 20)	80
flatten_5 (Flatten)	(None, 9680)	0
dense_15 (Dense)	(None, 100)	968100
leaky_re_lu_6 (LeakyReLU)	(None, 100)	0
batch_normalization_25 (Batc	(None, 100)	400
dense_16 (Dense)	(None, 1)	101

Total params: 972,901
Trainable params: 972,621
Non-trainable params: 280

Listing 39: The discriminator network used to distinguish generated VAE/VLAE samples from true MNIST images.

Acronyms

AdaIN adaptive instance normalization

ALAE adversarial latent autoencoder

CNN convolutional neural network

CNS central nervous system

ELBO evidence lower bound

GAN generative adversarial network

IT inferior temporal cortex

ILSVRC2017 Large Scale Visual Recognition Challenge 2017

LeakyReLU leaky rectified linear unit

LGN lateral geniculate nucleus

KDE kernel density estimation

KL-divergence Kullback-Leibler divergence

KL Kullback-Leibler

MSE mean squared error

NLP natural language processing

PCA principal component analysis

PDF probability density function

PPL perceptual path length

ReLU rectified linear unit

RDM representational dissimilarity matrix

SVM support vector machine

TEO temporo-occipital area

t-SNE t-distributed stochastic neighbor embedding

VAE Variational Autoencoder

VLAE Variational Ladder Autoencoder

LVAE Ladder Variational Autoencoder

V1 primary visual cortex

V2 secondary visual cortex

V4 quaternary visual cortex

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für eine andere Prüfung eingereichte Arbeit.

Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Duisburg, 21.7.2020

Ort, Datum

Unterschrift