UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Kaarel Tark

Disentanglement of features in variational autoencoders

Master's Thesis (15 ECTS)

Supervisor: Meelis Kull, PhD

Tartu 2022

Disentanglement of features in variational autoencoders

Abstract:
Machine learning models, especially neural networks, have shown excellent performance in classifying different images. The features these models learn are often complex and hard to interpret. Learning disentangled features from images is a way to tackle explainability and create features with semantic meaning. A learned feature is disentangled if it represents only a single property of an object. For example, if we had an image of a chair, we would assume that one feature changes its size, but nothing else. Another feature changes the chair leg shape and nothing else. Beta variational autoencoders (β-VAE) have shown promising performance in learning disentangled features from images without supervision. If there is enough data, the model can learn the features without needing large amounts of labelled data. After learning features, we can use a smaller amount of labelled data to train an additional model on top of the learned features (few-shot learning). The experiments of β-VAE architectures have been with simple images with known generative factors. Usually, all generative factors are independent, and the architecture assumes that there is a small number of them. Recently a new dataset has been published where some features are dependent ($\mathrm{Boxhead}$ dataset). The experiments with existing architectures showed relatively poor performance on β-VAE based architectures to capture those features. Based on exploratory analysis of β-VAE architecture based models, we propose a new architecture to improve the result. For evaluation, we introduce new metrics in addition to the commonly used ones. Our results showed no substantial performance difference between our proposed and β-VAE architectures. Based on the results of the main experiments, we conduct additional exploratory experiments on a dataset where the object does not rotate.

Keywords: machine learning, variational autoencoder, unsupervised learning, image processing, disentanglement

CERCS:P170 Computer science, numerical analysis, systems, control; P176 Artificial intelligence; T111 Imaging, image processing.

Tunnuste lahtiharutamine kasutades variatsioonilisi autoenkoodereid

Lühikokkuvõte:
Masinõppe mudelid, eriti närvivõrgud, on näidanud häid tulemusi erinevate piltide klassifitseerimisel. Nende mudelite õpitud omadused on sageli keerulised ja raskesti tõlgendatavad. Piltidelt interpreteeritavate ja maailma hästi kirjeldavate lahtiharutatud tunnuste õppimine on viis seletatavuse parandamiseks. Õpitud tunnust võib nimetada lahtiharutatuks, kui ta muudab ainult ühte omadust. Kui meil on pilt toolist, siis üks tunnus muudab selle suurust, kuid mitte midagi muud. Teine tunnus muudab tooli jala kuju, kuid mitte midagi muud. Beetavariatsioonilised autokodeerijad (β-VAE) on näidanud paljulubavaid tulemusi lahtiharutatud tunnuste leidmiseks. Kui andmeid on piisavalt, saab mudel ise õppida andmestikku kirjeldavaid tunnuseid. Pärast tunnuste õppimist saab kasutada väikest hulka märgendatud andmeid, et treenida täiendav mudel tegema mõnda ennustust. β-VAE arhitektuuride katsed on tehtud üldjuhul lihtsate tehislikult genereeritud piltidega, mille genereerimisel kasutatud tunnused on teada. Tavaliselt on kõik generatiivsed tegurid sõltumatud ja ka katsetes eeldatakse, et tunnuste arv on väike. Hiljuti avaldati uus andmestik, kus leidub tunnuseid mis ei ole sõltumatud (*Boxheadi* andmestik). Eksperimendid olemasolevate arhitektuuridega näitasid, et β-VAE arhitektuuril põhinevad mudelid ei tuvasta hästi väikeseid sõltuvaid tunnuseid. Käesoleva töö esimeses osas uurime β-VAE arhitektuuril põhinevate mudelite käitumist *Boxheadi* andmestikul. Esialgse uurimuse põhjal pakume välja uue arhitektuuri, et parandada sõltuvate tunnuste tuvastamist. Meie poolt pakutud arhitektuuri headuse hindamiseks võrdleme tema tulemusi β-VAE arhitektuuril põhinevate mudelitega. Lisaks enimkasutatavatele mõõdikutele defineerime kaks uut mõõdikut hindamaks tunnuste headust. Meie tulemused ei näidanud olulist erinevust sõltuvate tunnuste tuvastamisel meie poolt välja pakutud arhitektuuri ja β-VAE vahel. Põhikatsete tulemuste põhjal viime läbi täiendavad katsed andmestikul, kus objekt ei pöörle.

Võtmesõnad: masinõpe, variatsiooniline autoenkooder, juhendamata õpe, pilditöötlus, lahtiharutatus

CERCS:P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine; P176 Tehisintellekt; T111 Pilditehnika.

Contents

1 Introduction

Recently, variational autoencoders have shown promising neuroscience results [Higgins et al., 2021]. Also, there has been much focus on machine learning interpretability for neural network models to give more insights into why the machine made a specific decision. Having understandable input and learned features from data is one component of interpretability. If we think about an image, we can decompose it into a set of semantically meaningful features: light conditions, the direction of the light, the hue of the light, the position of an object, and object properties like colour, shape, and texture. β-VAEs family models try to identify these features from images. A feature learned in an unsupervised way with a semantic meaning (a single feature defines a single condition change) enables the creation of new applications [Higgins et al., 2018b]. It also reduces the need to label huge amounts of data before applying downstream classification models [Schönfeld et al., 2018].

Using only data to identify its generative factors can be a complex task. It might be that features are not separable as the features could be so mixed in data that there are many ways they could have been generated. For example, two features define the x and y coordinates of a data point in 2D space. If both features were generated from Gaussian distributions with the same parameters, then they would form a circular point cloud. When we would try to capture the initial generative factors, we would not know in which direction the initial axis were defined. There would be unlimited ways the axis could have been when the data was generated.

The experiments on images have often been done on very simple datasets ([Matthey et al., 2017], [Deng, 2012], [Burgess, Chris and Kim, Hyunjik, 2018]). They design the architecture to capture a similar number of generative factors as there is known to be in the data. Also, the generative factors in the datasets are all statistically independent. It is unlikely that continuous, factorised, fixed-dimensional representations are the optimal choice for modelling many real-world generative processes [Slavin Ross and Doshi-Velez, 2021]. Recently a new dataset, $\mathrm{Boxhead}$ [Chen et al., 2021], has been published where generative factors of features on the images are not all statistically independent. Their work has shown that the β-VAE-like architectures do not separate well features that cover only a small area of pixels and depend on other features.

The goal of this thesis is to: investigate β-VAE behaviour with a dataset with dependent features; propose an architecture that would improve capturing dependent features that cover only a few pixels; conduct the experiments of the proposed architecture, and compare the results to β-VAE architecture.

We first conduct a set of exploratory experiments, based on which we motivate our neural network architecture decisions and then compare our performance to similar β-VAE architecture and discuss the results. For evaluation, we introduce new metrics in addition to the commonly used ones due to limitations of using them in cases where we allow our model to learn possibly hundreds of features. Based on the results of the

main experiments, we conduct additional exploratory experiments on a dataset where the object does not rotate.

The rest of this thesis is structured as follows: background is discussed in Section 2; the setup and metrics of the experiment used are described in Section 3; the exploratory analysis and our architecture motivation are described in Section 4; the proposed architecture with experiments results are discussed in Section 5; we interpret the results in Section 6; and, in Section 7 conclude and cover future work.

2 Background

Our proposed method relies strongly on the β-variational autoencoder. We give a basic overview of the related concepts that support understanding of the thesis. First, we will introduce representation learning with the main focus on autoencoders. Secondly, we will discuss what kind of properties has a disentangled data representation, followed by defining generative models - both are important properties of the variational autoencoder. After which, we will introduce variational autoencoders and the role of neural networks in them. Finally, we introduce the dataset $\mathrm{Boxhead}$, which will be used for our experiments.

2.1 Representation learning

Representation learning is a set of methods that extract features from input to another space. There are many representation techniques, such as Independent Component Analysis (ICA) [Hyvärinen and Oja, 2000], Principal Component Analysis (PCA) [Hotelling, 1933] and Linear Discriminant Analysis (LDA) [Mika et al., 1999]. PCA is an unsupervised method to reduce the dimensionality of the data by aligning space to capture the most variance from the data. LDA is a supervised method to increase class separation and decrease variation inside the class, and ICA is a method to decompose information into independent features. With the emergence of neural networks, autoencoders [Rumelhart et al., 1985] were developed. With the support of back-propagation, the network learns to recreate the input from features that have lower dimensionality than the initial input. Due to the limited number of parameters in each layer, the network is motivated to learn the most relevant representation for the specific task. The autoencoder usually contains an encoder, a decoder, and a bottleneck (see Figure 1). The bottleneck $\mathbf{z}$ limits the information passing from the encoder $\mathbf{z} = q_\phi(\mathbf{x})$ (ϕ defines encoder parameters) to the decoder. The bottleneck is a compressed representation of the input, often with some information loss compared to the input. To generate a reconstruction of the input, learned features $\mathbf{z}$ (also referred to as latent variables) are passed through decoder $\hat{\mathbf{x}} = f_\theta(\mathbf{z})$ (θ defines decoder parameters). The amount of information possible to keep in the architecture (capacity) depends on the size of the encoder, decoder, and the number of latent variables in the bottleneck. A neural network without activation functions (non-linear layers) is similar to PCA, which does linear transformations [Rolínek et al., 2018]. Autoencoders are trained iteratively by pushing the input x through the architecture and calculating the difference between the decoder output $\hat{\mathbf{x}}$ and input $\mathbf{x}$ (loss function $loss = ||\mathbf{x} - \hat{\mathbf{x}}||_2$). By iterating through a set of images and adjusting network weights using back-propagation, it learns in the latent variables $\mathbf{z}$ the information most needed to decode the input image.

Autoencoders mainly focus on the reconstruction of the images. They do not give any meaning to the latent variables except that they represent the most crucial info to reconstruct the original image. Variational autoencoders (discussed in Chapter 2.4) introduce additional loss constraints to make learned features more meaningful.

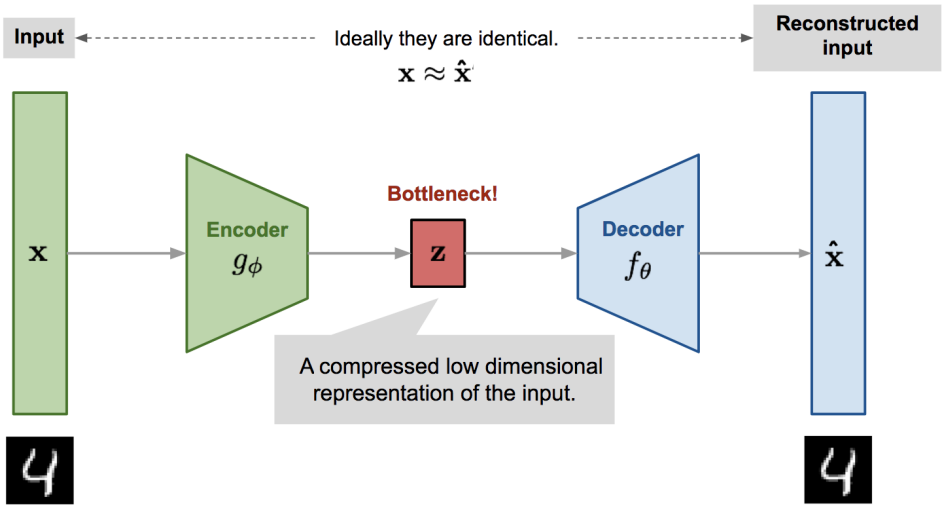

Figure 1. Input x encoded, z latent variables (image from [Weng, Lilian, 2018]).

2.2 Disentangled representation

Neural network architectures often include inductive bias to give good results on cases it has not seen during training. An alternative approach is to learn from the input data a representation that would describe its generative structure [Higgins et al., 2018a].

Learning a disentangled representation means learning a vector of different factors that generated the currently observed state [Bengio et al., 2012][Higgins et al., 2018a]. If we would change the value of the feature, then only a single aspect of the world state would change. If we imagined a basketball in an empty room, this would mean learning features about the ball (size, color, rotation, texture), position in the room, color of the room walls, and lighting conditions as separate features. We could then change ball position, and the other world state parameters would not change. It is similar to symmetry in physics: the symmetry of an object is a transformation that leaves specific properties of the object invariant [Higgins et al., 2018a].

2.3 Generative models

Discriminative machine learning models estimate the probability of label y based on a single data point $p(y|x)$. It defines the probability of each point belonging to a specific class (see Figure 2). Generative models model the distribution over all data points $p(x)$ (probability density function). Conditional Generative models estimate the probability of data given a label $p(x|y)$. Bayesian rule naturally binds the probabilities together: $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$.

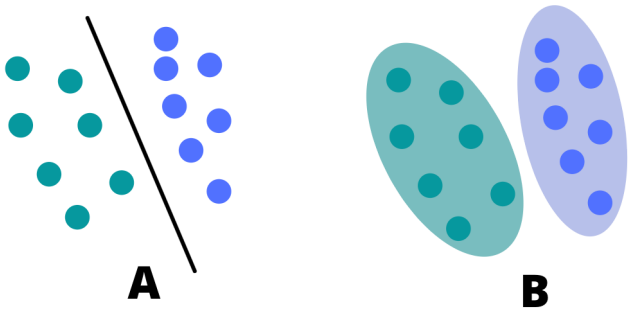

Figure 2. (A) - discriminative approach estimates the data points' probability of belonging to a class. (B) - probabilistic approach estimates the probability distribution of data.

2.4 Variational autoencoders

Our introduction of variational autoencoders is impacted strongly by Joseph Rocca's article in Towards Data Science portal [Rocca, 2019]. Autoencoders are unsupervised machine learning models that, via an encoder, compress input signals to a lower dimensionality space and, via decoder, return the signal to the initial space in a deterministic manner. Variational autoencoders are autoencoders, but they learn distributions with specific properties instead of deterministic values. Similarly to PCA, variational autoencoders also encourage orthogonality in the lower dimensional latent space [Rolínek et al., 2018]. Latent variables learned by VAE are continuous, and by changing latent variables values, we can generate new meaningful images out of them.

Variational autoencoders assume some generative factors $\mathbf{z} \in Z$ from which the data $\mathbf{x} \in X$ has been generated (see Figure 3). Vector $\mathbf{z}$ is called a latent variable. It is not directly observable and is a representation of $\mathbf{x}$ in a lower-dimensional space compared to input. From the value of $\mathbf{x}$, we can infer $\mathbf{z}$.

Figure 3. Examples $\mathbf{x}$ are generated from $\mathbf{z}$, and from $\mathbf{x}$, we can infer $\mathbf{z}$.

Before we continue, we need to define some base terms:

- $p(\mathbf{z})$ - prior distribution of the latent variables.
- $p_\theta(\mathbf{x}|\mathbf{z})$ - the likelihood of $\mathbf{x}$ given the latent vector.
- $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$ - a joint distribution of our data and latent vector (multiplication of likelihood and prior).
- $p_\theta(\mathbf{x})$ - distribution of data when generated from the decoder (marginal distribution over all possible z).
- $p_\theta(\mathbf{z}|\mathbf{x})$ - posterior distribution defining how probable is a latent variable for a given input.
- θ - set of true parameters for our probability distributions.

We will introduce the main parts of VAE derivation required to understand the relation to neural networks. For full details, please refer to the initial VAE paper [Kingma and Welling, 2014].

The goal is to find a good set of latent variables $\mathbf{z}$, which generate $\mathbf{x}$. Applying the Bayes theorem, we can define it as:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p_\theta(\mathbf{z}|\mathbf{x})p(\mathbf{z})}{\int p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})dz}. \tag{1}$$

Computationally there is a problem with the denominator as the marginal probability (evidence) is intractable, and we would need to evaluate all possible $\mathbf{z}$ values.

Instead of directly calculating $p_\theta(\mathbf{z}|\mathbf{x})$, it is estimated using variational inference, which approximates the posterior with a family of distributions $q_\phi(\mathbf{z}|\mathbf{x})$. If we estimate posterior using Gaussian distribution, we estimate its mean and variance for each latent variable. In order to understand how good is our estimation, we can use Kullback-Leibler divergence [Kullback and Leibler, 1951] to quantify the information loss (how many bits of information are lost when using one distribution to present the other).

$$\begin{aligned} &D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \mid\mid p_\theta(\mathbf{z} \mid \mathbf{x})) \\ &\quad = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{z} \mid \mathbf{x})\right) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}, \mathbf{z})\right) + \log p_\theta(\mathbf{x}) \end{aligned} \tag{2}$$

It is also impossible to calculate Eq. (2) directly due to evidence $p_\theta(\mathbf{x})$ being intractable (also in the denominator for Eq. (1)). In order to approximate the posterior distribution, we will use Evidence Lower Bound (ELBO).

$$ELBO = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}, \mathbf{z})\right) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{z} \mid \mathbf{x})\right) \tag{3}$$

KL-divergence is always greater or equal to zero. If we maximise ELBO, then we minimise KL-divergence at the same time. ELBO can be derived into a computationally simpler format where with some assumptions, we can calculate it

$$\begin{aligned} ELBO =& \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x},\mathbf{z})\right) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{z}\mid\mathbf{x})\right) \\ =& \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x},\mathbf{z})\right) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_\theta(\mathbf{x})}\right) \\ =& \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x},\mathbf{z})\right) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p(\mathbf{z})\right) \\ & - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}\mid\mathbf{z})\right) + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x})\right) \\ =& \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}\mid\mathbf{z})\right) - D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}\mid\mathbf{x}) \,||\, p(\mathbf{z})\right). \end{aligned} \tag{4}$$

Previous Eq. (4) is used as the loss function for VAE.

$$L_{\mathrm{VAE}}(\theta,\phi) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\log p_\theta(\mathbf{x}\mid\mathbf{z}) + D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}\mid\mathbf{x}) \,||\, p(\mathbf{z})\right), \tag{5}$$

where θ and ϕ define the parameters (weights and biases) of decoder and encoder in neural networks. In order to do the estimation a couple of assumptions are made:

$$\begin{aligned} p(\mathbf{z}) &= \mathcal{N}(0, I), \text{ where } I \text{ is identity matrix,} \\ p_\theta(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(f(\mathbf{z}), cI), \text{ where } f \in F \text{ and } c > 0, \\ q_\phi(\mathbf{z}|\mathbf{x}) &\equiv \mathcal{N}(g(\mathbf{x}), h(\mathbf{x})) \quad g \in G, h \in H. \end{aligned} \tag{6}$$

In Eq. (6), the prior is a standard Gaussian distribution (mean = 0, covariance is an identity matrix), and likelihood is Gaussian distribution with a mean of deterministic function f of $\mathbf{z}$ and a constant times the identity matrix. F defines functions family for all possible θ values. Estimated posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is a Gaussian distribution defined by two functions, g and h, defining the distribution mean and covariance matrix. G and H define function family for all possible ϕ values.

In Eq. (5), the first term is negative log-likelihood, which motivates the decoder to recreate the same image as received as input to the encoder. For example, if the input image has a white pixel and the decoder returns a low probability on it being white, then we get a high reconstruction loss for this pixel [Altosaar, Jaan , 2016]. The KL-regularisation term adds compression/regularisation to the loss. It measures the distance between estimated and prior distribution. The network seeks a balance between being able to recreate the same image as given in input and having low compression regularisation loss. The loss is calculated for each sample in the dataset separately. Total loss is the sum of individual losses.

Encoder architecture is given in Figure 4. The network's encoder output is $q_\phi(\mathbf{z}\mid\mathbf{x})$. It is assumed to be a multidimensional Gaussian distribution with a diagonal covariance matrix where variables are independent. With the assumption, the architecture estimates mean with function g and variance with function h.

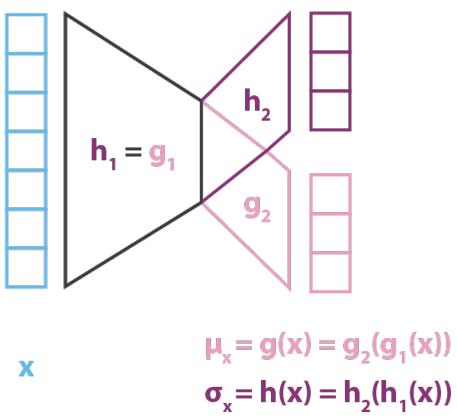

Figure 4. Encoder representation as a neural network. Functions g and h share part of the architecture (image from [Rocca, 2019]).

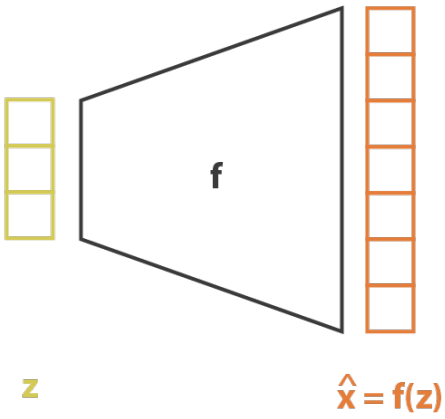

Figure 5. Decoder representation as a neural network. Functions f receives latent variables and outputs generated object (image from [Rocca, 2019])

Decoder architecture is given in Figure 5. The input is $\mathbf{z}$, and the output is probability distribution mean values for each pixel in the image.

The learned latent variables are Gaussian distributions with a specified mean and variance. For the architecture to be able to train, this is a problem, as sampling from a random variable is not differentiable, and we cannot use back-propagation to adjust the model parameters. For this, the reparametrisation trick is introduced. Due to the properties of Gaussian distribution, we can extract the stochastic part of sampling to a separate path.

Extracting stochasticity to $\boldsymbol{\epsilon}$ gives the model possibility to learn in the encoder the mean and variance because gradient-based learning derivatives with respect to weights would not include a sampling procedure.

$$\mathbf{z} \sim q_\phi\left(\mathbf{z} \mid \mathbf{x}_i\right) = \mathcal{N}\left(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2 \boldsymbol{I}\right) \tag{7}$$

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \text{ and } \odot \text{ is element-wise multiplication.} \tag{8}$$

The overall structure of the variational autoencoder is in Figure 6. From input $\mathbf{x}$, it estimates the mean and variance of latent variables, then samples latent variables $\mathbf{z}$ using the reparametrisation trick. In order to receive the representation of the input, it processes the latent variables through the decoder and receives the reconstruction of input (with information loss).

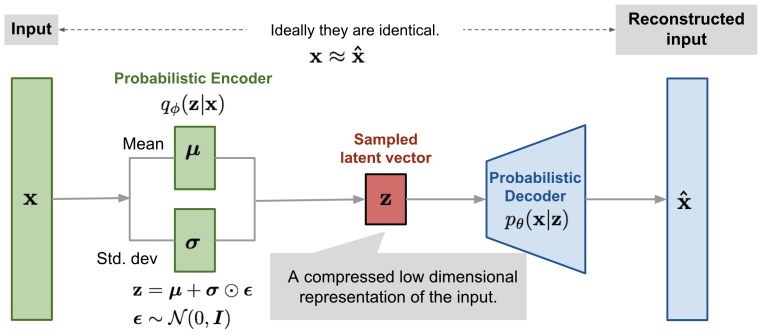

Figure 6. The architecture of variational autoencoder. Using a probabilistic encoder map input $\mathbf{x}$ to a list of Gaussian distribution mean and variance parameters. Sample $\mathbf{z}$ using reparametrisation and keeping stochasticity in $\boldsymbol{\epsilon}$. Finally, send the sampled $\mathbf{z}$ through the decoder to get reconstruction (image from [Weng, Lilian, 2018]).

β-VAE is an enhancement of VAE, which focuses on learning a disentangled representation [Higgins et al., 2017]. They introduce an additional hyperparameter β, which controls the trade-off between reconstruction and compression ($\beta = 1$ corresponds to VAE). The higher the β value, the more it forces the model to encode the info more effectively and introduces additional disentanglement. Higher β values capture the independent variables from the generative factors more effectively than VAE. The loss function of β-VAE is defined as:

$$L_{\text{BETA}}(\theta, \phi, \beta) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x} \mid \mathbf{z}) + \beta \cdot D_{\text{KL}} \left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right). \tag{9}$$

2.5 Dataset - Boxhead

Disentanglement is often learned from datasets which fulfil the criteria that generative factors are independent. Often dSprites [Matthey et al., 2017] or a similar dataset is used where the generative factor values are known and predefined. Similarly, the Boxhead dataset has known properties, but in addition to independent generative factors, it explicitly generates dependent features. 16 example images (each image 64x64 pixels) are shown in Figure 7. It contains of a coloured box in a 3D environment where the properties of the object and environment change for each image. Each image in the datasets has ten generative factors (see Figure 8). We group

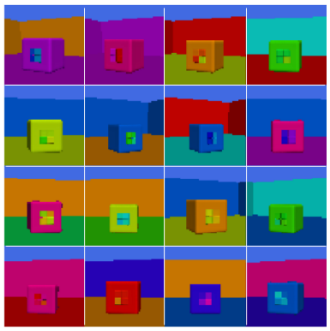

Figure 7. 16 examples of Boxhead dataset samples (image from [Chen et al., 2021]).

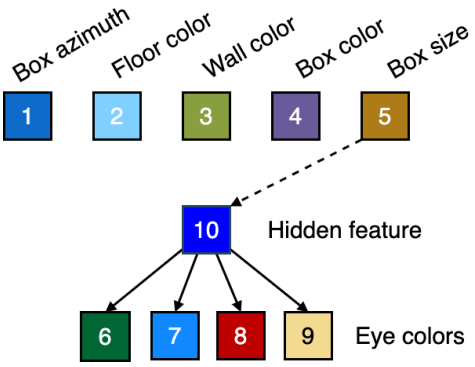

Figure 8. The generative process of Boxhead dataset samples. The generation of micro-level factors is not independent. All eye colors are dependent on the hidden feature value, and depending on the dataset, the hidden value can also be dependent on the box size.

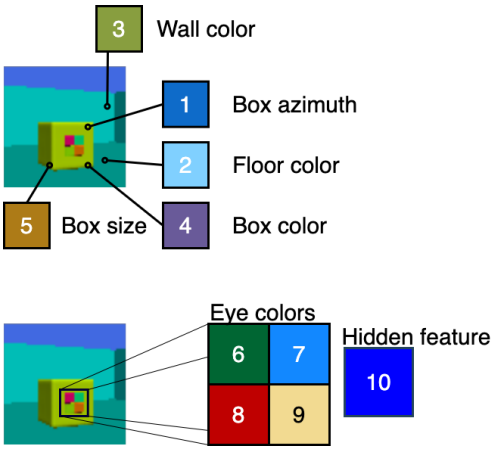

Figure 9. Factors 1-5 are macro-level factors. Factors 6-10 are micro-level factors.

the generative factors similarly, as in the Boxhead paper [Chen et al., 2021]. In Figure 9 top part of the image contains **macro-level factors** (factors 1-5), and the lower image shows **micro-level factors** (factors 6-9 and 10). Note that the generative factor (number 10) is not directly observable from the image.

We used three different variations of the dataset (boxhead_07, boxheadsimple and boxheadsimple2). Each of them had 51 328 images. We use the same names for datasets as in the generative scripts referenced in the related paper [Chen et al., 2021]. All Boxhead variants have the same generative functions for wall color, floor color, box color, box size and box azimuth.

Color feature values are mapped from 0 to 1 into RGB colors space (for example, value 0 is red, 1/6 is yellow, and 2/6 is green). Example RGB color wheel[1] scale is given in Figure 10, where the zero point is at red color (value 0.99 is mainly red and also 0.01 is mainly red). For color features mod 1 is applied as it scales the values to the required range.

Figure 10. RGB color wheel. 0-360 degrees corresponds to generated values 0-1.

Same generative process in all datasets is:

- wall color $\sim uniform(0, 1)$;

[1]Color wheel image from https://commons.wikimedia.org/wiki/File:RGB_color_wheel_360.svg (Last checked on May-16, 2022)

- floor color $\sim uniform(0, 1)$;
- box color $\sim uniform(0, 1)$;
- box size $\sim uniform(1, 1.25)$;
- box azimuth $\sim uniform(-\frac{\pi}{6}, \frac{\pi}{6})$.

The difference is in generative factors related to eyes and their hidden feature. For $\mathrm{boxhead\_07}$ dataset, the generative factor values sampling procedure is:

- hidden feature $\sim \mathcal{N}(box\_color, 0.2)$;
- top-left eye color $\sim uniform(-0.1 + hidden\_feature, 0.1 + hidden\_feature) \mod 1$;
- top-right eye color $\sim uniform(-0.1 + hidden\_feature, 0.1 + hidden\_feature) \mod 1$;
- bottom-left eye color $\sim uniform(-0.1 + hidden\_feature, 0.1 + hidden\_feature) \mod 1$;
- bottom-right eye color $\sim uniform(-0.1 + hidden\_feature, 0.1 + hidden\_feature) \mod 1$.

For $\mathrm{boxheadsimple}$ dataset the generative factor values sampling procedure is:

- hidden feature $\sim uniform(0, 1)$;
- top-left eye color $\sim \mathcal{N}(hidden\_feature, 0.1) \mod 1$;
- top-right eye color $\sim \mathcal{N}(hidden\_feature, 0.1) \mod 1$;
- bottom-left eye color $\sim \mathcal{N}(hidden\_feature, 0.1) \mod 1$;
- bottom-right eye color $\sim \mathcal{N}(hidden\_feature, 0.1) \mod 1$.

For $\mathrm{boxheadsimple2}$ dataset the generative factor values sampling procedure is:

- hidden feature $\sim uniform(0, 1)$;
- top-left eye color $\sim \mathcal{N}(hidden\_feature, 0.2) \mod 1$;
- top-right eye color $\sim \mathcal{N}(hidden\_feature, 0.2) \mod 1$;
- bottom-left eye color $\sim \mathcal{N}(hidden\_feature, 0.2) \mod 1$;
- bottom-right eye color $\sim \mathcal{N}(hidden\_feature, 0.2) \mod 1$.

3 Experiments setup and metrics

The current section introduces the experiment setup when performing the proposed architecture and baseline evaluations. It continues by introducing the metrics and visualisations used to evaluate the experiments.

3.1 Experiments setup

We evaluate our architecture on 3 $\mathrm{Boxhead}$ datasets and compare our results to β-VAE results. All datasets contain 51328 images, where the single image size is 64x64 pixels. In all experiments, 200 000 iterations were run with batch size 64. Training a single multi-layer model took 36 hours on a single NVIDIA A100 GPU (University of Tartu HPC cluster). Training a β-VAE took about 7 hours on Tesla T4 (Google Colab). The experiments have been implemented in Pytorch Lightning [Grid.ai, Inc, 2022] with metrics and test images reported to Weights and Biases [Weights and Biases, Inc, 2022]. The source code for experiments is available[2]. β-VAE implementation has been adapted from the existing Pytorch implementation [Lee, WonKwang and Metger, Tony, 2022].

3.2 Metrics

In the current section, we introduce the metric we will be using during the evaluation of our experiments.

3.2.1 Disentanglement and Completeness

We report the performance of the models with respect to DCI metrics [Eastwood and Williams, 2018], which models have been trained using 10% of the initial training data (sampled randomly). Implementation of the DCI metric was adapted from the Google Research disentanglement library [Locatello et al., 2019].

The DCI score is defined by entropy measure from information theory. If a single latent variable contributes equally to predicting all generative factors, then the disentanglement score would be 0. If it contributes only to a single generative factor, the score would be 1. For completeness, score entropy is calculated over a generative factor. If all latent variables equally support a generative factor prediction, then the completeness score would be 0. If there is only a single latent variable predicting the generative factor, then the score would be 1.

Due to possible limitations of DCI ([Sepliarskaia et al., 2021]), and due to using many latent variables, we calculate it based on two different methods: joint model DCI and per latent variable model DCI. The joint model DCI evaluates all latent variables together

[2]https://github.com/ktark/multilayerBVAE

and has been adapted in many experiments, but only in cases where latent variables are relatively similar to actual generative factors. In our case, the space of the latent variables was more extensive, and we let the architecture decide how many latent variables to use. Therefore, per latent variable model DCI metric, which evaluates each latent variable separately, was introduced.

Both metrics calculations use the same DCI disentanglement and completeness calculation algorithm but differ in the importance matrix calculation. We present first importance matrix calculation algorithms and then DCI metrics algorithms using the matrices as input. The process is as follows:

Calculation of the importance matrix method depends on the model used to predict generative factor values and if we want to calculate the matrix jointly over all latent variables or per latent variable.

1. Take a set of sample images X and their generative factor values Y.
2. Send each image x through the encoder and sample $\mathbf{z}$ from the posterior.
3. Train models to predict generative factors. The model can be trained by using all latent variables or by using a single latent variable.

   (**A**) **Joint method.** For each generative factor k ($k \in K$, where K is the list of generative factors), train models m_k to predict y_k (the true generative factor value) using values from all latent variables $\mathbf{z}$.

   (**B**) **Per latent variable method.** For reach generative factor and latent variable combination, train a model $m_{k,d}$ ($d \in D$, where D is the size of latent variable layer $\mathbf{z}$) to predict y_k using values of single latent variable $\mathbf{z}_d$ (per latent variable model).

4. Generate feature importance matrix $R_{D,K}$, where columns are generative features and rows are latent variables. The values used in the matrix depend on the type of model used.

Calculation of DCI completeness and disentanglement uses normalised importance matrix ($R_{D,K}$ as input for calculation. The process is as follows:

1. Calculate entropy for each latent variable ($d \in D$) over all generative factors

$$H_d^{latent} = -\sum_{k=0}^{K-1} R_{d,k} \log_K \left(R_{d,k}\right).$$

2. Calculate entropy for each generative factor ($k \in K$) over all latent variables

$$H_k^{factor} = -\sum_{d=0}^{D-1} R_{d,k} \log_D \left(R_{d,k}\right).$$

3. Calculate the importance of each latent variable over all generative factors

$$\mathsf{imp}_d = \frac{\sum_k R_{d,k}}{\sum_{d,k} R_{d,k}}.$$

4. Calculate the predictability for each generative factor over all latent variables

$$\mathsf{pred}_k = \frac{\sum_d R_{d,k}}{\sum_{d,k} R_{d,k}}.$$

5. Calculate the disentanglement score for each latent variable

$$\mathsf{disent}_d = \mathsf{imp}_d \left(1 - H_d^{latent}\right).$$

6. Calculate the completeness score for each generative factor

$$\mathsf{compl}_k = \mathsf{pred}_k \left(1 - H_k^{factor}\right).$$

7. DCI disentanglement score is the sum of all latent variables' disentanglement scores

$$\mathsf{disent} = \sum_{d=1}^{D} \mathsf{disent}_d.$$

8. DCI completeness score is the sum of all generative factors' completeness scores

$$\mathsf{compl} = \sum_{k=1}^{K} \mathsf{compl}_k.$$

We calculate the DCI completeness and disentanglement metrics using two different methods. They differ in the model used to predict the generative factor and how the importance matrix is calculated.

- For **joint model DCI** over all latent variables, we train Gradient Boost regressor models to predict each generative factor y_k (Joint method in calculation of the importance matrix process). As features, we use all latent variables $\mathbf{z}$. The feature importance vector absolute values returned by the model are used to populate the importance matrix.

- For **per latent variable model DCI**, we first generate polynomial $z_d + z_d^2 + z_d^3 + z_d^4$ features from each latent variable z_d. Then, we train linear regression models to predict each generative factor y_k using the polynomial features (Per latent variable method in the calculation of importance matrix process). Finally, for each model, we calculate R^2 score to predict y_k, which we use to populate the importance matrix.

We report DCI metrics separately for all generative factors and micro-level generative factors. For the micro-level generative factors metric, we limit the list of latent variables only to those that predict more micro-level generative factors than others. We keep a latent variable in the metric calculation if the sum of micro-level generative factors (factors 6-10, see Figure 9) is bigger than the sum of other factors (factors 1-5) values for that latent variable in the importance matrix.

3.2.2 Latent traversals

We used latent traversal images with the support of metrics during our evaluation. Latent traversal images can be created for any image. We have used a single image and mapped it through the encoder. Each latent variable was independently from others changed, and a new image was generated through the decoder. We selected the new latent variable values between -3 and +2.5 (-3, -2.5, -2, ..., +2.5). It enabled us to inspect each latent variable impact on the output visually.

The latent traversal single images generation process is as follows:

1. send image x through the encoder and get the posterior $\mathbf{z}$ value mean (no sampling);
2. change a single dimension value of $\mathbf{z}$;
3. To get a reconstruction image, send the changed $\mathbf{z}$ vector through the decoder.

Figure 11 shows an example of single latent variable traversal images. In the first row, each image is generated by changing the latent variable to be a value from -3 to +2.5. The current latent variable changes the background color, and we can see the original background colour encoded near value 1.

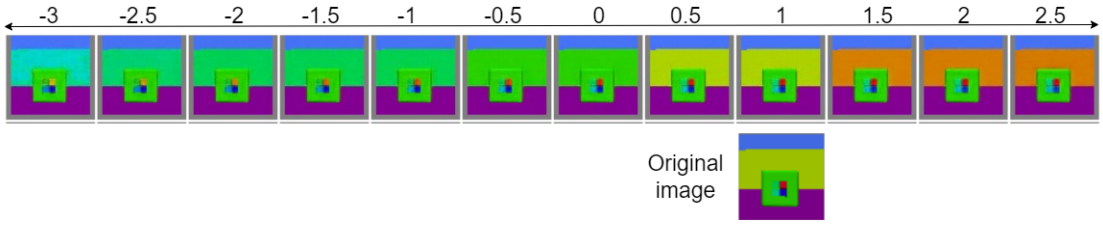

Figure 11. Latent traversal images are generated by changing a single latent parameter value (representing background color) and using a decoder to generate an image. The current latent variable value (note that it is a specific value, not range) is shown on a scale on top of the images. The original image mapped through the encoder is shown under value 1, which is closest to the original image.

Metrics shown on traversal images can be seen in Figure 12. First, with black background color, we have a latent variable identifier and then, on the next row, report the mean KL-divergence of all training set images in nats (similar to bits, but with logarithm base e). As latent variable layers are bottlenecks in our architecture, KL-regularisation shows how much data was encoded to the variable to create a reconstruction image [Burgess et al., 2019]. The more effective the architecture is in encoding the input closer to the prior, the lower the value would be. If it would always precisely match the prior distribution for all samples, then it would be a constant value, and this info would be in reconstruction added by the decoder. It would not need to go through the encoding process. In the visualisation additional metrics are shown as bar charts which illustrate the relation between latent variable values and generative factors.

- Spearman correlation absolute value between generative feature and current latent parameter mean value. It indicates if the values are monotonically related.
- For each latent parameter, a linear regressor on polynomial $z_d + z_d^2 + z_d^3 + z_d^4$ was trained (notated as Multinom. on traversal images) to predict each generative feature. In bar charts, we show the R^2 score value to predict a specific generative factor. The same values are calculated for **per latent variable model DCI** importance matrix R (Section 3.2.1).
- Gradient Boosting (GB) regression feature importance (normalised sum to 1). Finds the most important latent variables to predict that generative factor. The same values are calculated for **joint model DCI** importance matrix R (Section 3.2.1).

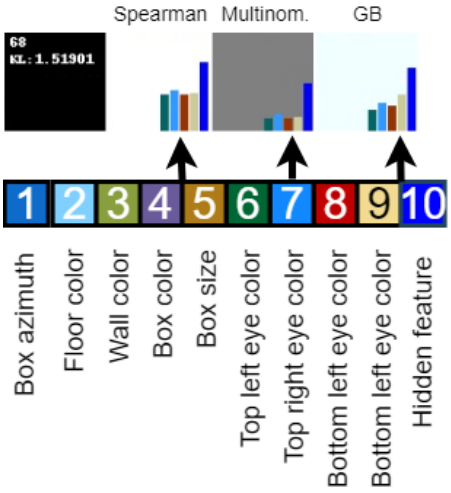

Figure 12. The black background reports the latent identifier on the first row and the average KL-divergence in nats on the second row (calculated based on the whole training set). The bar chart shows the metric value for each generative factor for that latent variable (each bar in range 0-1).

3.2.3 Best disentanglement score of generative factor

We introduce an additional metric to evaluate the best disentanglement achieved for each generative factor.

1. For each latent variable train a linear regressor on polynomial $z_d + z_d^2 + z_d^3 + z_d^4$ to predict each generative feature. Calculate the R^2 score to predict a specific generative factor.

2. For each latent variable, identify the generative factor it predicts the most and subtract the second-best R^2 score from it.

3. For each generative feature, find the maximum value of step 2, where this factor had the highest R^2 score.

We visualised the results as heatmaps for easier comparison between experiments.

4 Exploratory analysis: towards identifying micro-level dependent features

The current section discusses the initial exploratory experiments (failed attempts) conducted on various β -VAE architecture modifications on the Boxhead dataset. The insights gathered from this phase motivate the architecture proposed in the following chapter.

There have been a set of different hierarchical approaches to VAE, such as VLAE [Zhao et al., 2017], pro-VLAE [Sønderby et al., 2016], BIVA [Corti et al., 2019] and NVAE [Vahdat and Kautz, 2020]. The related work focuses on learning a single set of good latent variables from different layers of neural networks with some modifications. Our approach is similar but investigates learning multiple latent variables layers, where from each layer, we can also generate images. Each latent variables layer gets input only on the previous layer variables, and each layer has different compression/regularisation strength.

4.1 Investigating β - VAE on Boxhead dataset

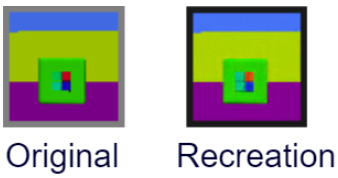

Figure 13. Example of the original input image and reconstruction image of β - VAE ($\beta = 1.5$).

Initial evaluation of the Boxhead dataset [Chen et al., 2021] showed that β - VAE does not identify micro-level factors as disentangled. It can recreate the initial image, as seen in Figure 13, but cannot disentangle micro-level factors. In Figure 14, we can see the latent variable impacting box size, where the latent variable value change impacts a single feature in the image. In Figure 15 the single latent variable value change impacts multiple micro-level factors. In order to be disentangled, they would need to impact only a single generative factor (for example, only the left-top eye color).

There is a trade-off between reconstruction and disentanglement [Burgess et al., 2019], controlled by β. The higher the β, the better disentanglement but worse reconstruction. Similarly, the lower the β, the less disentanglement but better reconstruction. We started from regular β -VAE and identified the required size of architecture that would give us good representation, as disentanglement is not possible without good reconstruction of

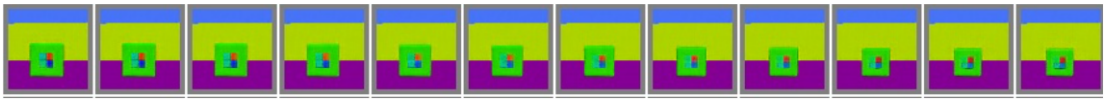

Figure 14. Example of the impact of one latent variable on the output image. The latent variable shown has learned to disentangle box size (bigger on the left, smaller on the right).

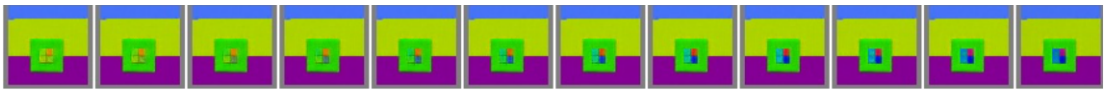

Figure 15. Example of the impact of one latent variable on the output image. The latent variable shown is a mixture of micro-level factors (information about how the images were generated is given in Section 3.2.2). The latent variable value change impacts all visible micro-level factors reconstruction.

the input. We decided to use encoder and decoder architectures from Boxhead paper [Chen et al., 2021], as they captured the required micro-level details on the image. After identifying the state of excellent reconstruction, we started searching β -VAE hyperparameter β value range where the micro-level features reconstruction would change. By visually inspecting the output images, we identified that $\beta = 0.5$ is the value, where we still have excellent reconstruction of the micro-level features. In higher values, we start losing reconstruction in the inner small box. $\beta = 5.0$ is the value where we do not have any details of the micro-level features visible.

In Figure 16, we can see that at $\beta = 0.5$, we get excellent reconstruction. At $\beta = 1.5$, we start to lose a bit of detail. At $\beta = 2.5$, we can still slightly see some color differences in the micro-level factors, and at $\beta = 5$, the details of micro-level factors are lost. We expected that the higher our beta, with enough details kept in reconstruction, the more disentanglement we should see in the micro-level factors in learned latent variables. The initial β ranges identification was required to understand if some β values would also disentangle micro-level details in the original β-VAE architecture. Also, it was needed in order to avoid seeking hyperparameter values in ranges which are not of our interest. As a result, we concluded that β-VAE could not disentangle micro-level details on Boxhead

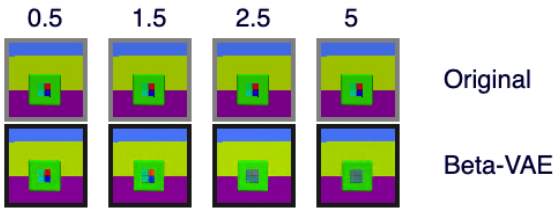

Figure 16. In the first row input image (original). In the second row β-VAE model reconstructions of the input image. At the top applied β value is shown.

dataset. The micro-level details cover too few pixels for the objective function to motivate the small details to be disentangled. Therefore, architectural or loss function updates are required.

4.2 Initial exploratory experiments

The current section will discuss the different architectural trials made. We will also describe the main conclusions and motivate our proposed architecture.

We tried an approach where we would have two latent feature levels. The motivation was to see if we stack multiple β-VAE architectures so that the first layer learns good reconstruction but does not well disentangle. The next layer would improve the disentanglement by combining and cleaning the latent variables learned in the previous layer. Figure 17 describes the main parts of the architecture. The two latent layers shared the base encoder, but both have their own decoders. A fixed structure encoder connects the two latent layers. A single latent variable exists in the second vector for every ten latent variables in the first layer. Between the latent layers, we had an additional encoder with fully connected layers and ReLU for capturing non-linearity.

We tuned β values on both layers to have a visually good reconstruction of micro-level details on both layers so that the second layer would have better disentanglement. This approach did not give the expected result. Latent variables learned on different layers often did not align. When the first layer latent layers seemed unused by the decoder, then the second layer latent, being a combination of the first layer latent variables, could decode some generative factor.

Different features also evolve in different training iterations. We hypothesised that it might be due to weight initialisation or learning being caught in a local minimum. We tried to change the learning process by controlling β during learning. Annealing hyperparameter values related to model capacity were also introduced in the β-VAE follow-up article [Burgess et al., 2019]. For the first 100 000 iterations, we had no KL-regularisation on the first layer and used capacity annealing on the second layer. After that, we run an additional 100 000 iterations with KL-regularisation on the first layer. As in the first trial, the features on different layers did not get aligned. The model did not force multiple small features from the previous layer to be combined into a single one in the last layer. Instead it compressed them into separate latent variables. It might have been due to our fixed architecture or due to $\mathrm{Boxhead}$ dataset having too few pixels in each micro-level feature.

A natural thing to try was to change the latent layers learning order. The changes compared to the initial trial are shown in Figure 18. We have 12 latent variables (12 mean and 12 variance values) in the first layer, and each latent variable is connected to 12 second layer latent variables (mean and variance values). The hypothesis was that we would first learn the macro-level features, and the second layer would split the mixed latent variables into separate disentangled ones. It did not improve the results. The

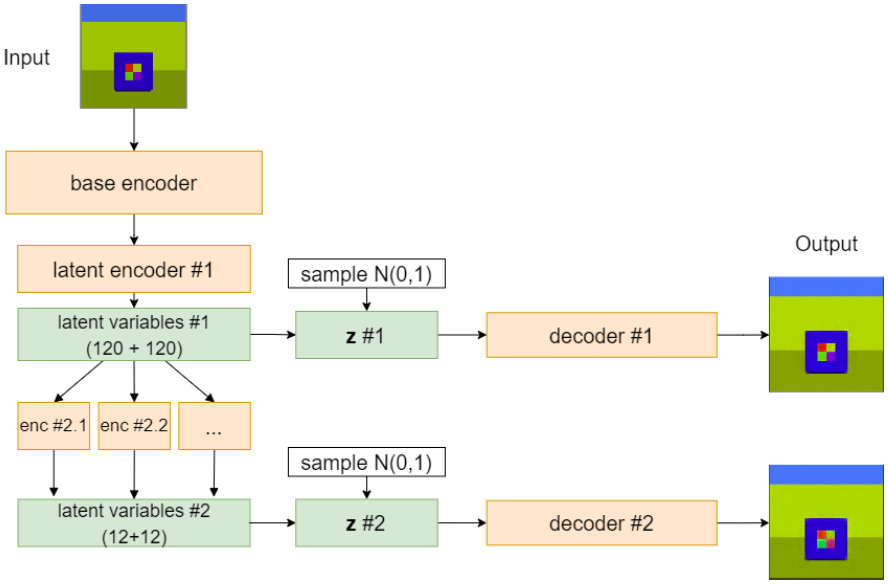

Figure 17. The architecture of the initial trial. Both latent variable layers share the base encoder but have their own decoders and sampling procedures. First and second layer latent variables are connected with fixed constructed encoders (on image notated as enc) where a set of latent variables (mean and variance) is connected to a single latent variable (mean and variance) in the other layer. First layer has 120 (120 mean + 120 variance values) latent variables and second layer has 12 (12 mean + 12 variance values).

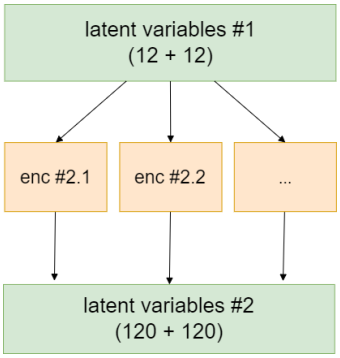

Figure 18. Trial to learn the first 12 latent variables on the first layer and then split each latent variable into 12 latent variables in the second layer. The current Figure shows only the differences compared to Figure 17.

first layer generative factors identified were as expected in the first layer, but the factors captured did not improve in the second layer. The second layer latents did not improve

because there was no other signal to the second layer except the info gathered from the first latent layer.

Based on the initial experiments, we concluded that our architecture had to take into account:

- We need to have good reconstruction to possibly see any disentanglement on Boxhead datasets on latent variables identifying micro-level generative factors.
- Controlling the learning process by annealing hyperparameters in loss makes learning complex and increases the hyperparameters search space - if possible, try to avoid it by using regularisation.
- Try not to use fixed signal paths, but allow the network to learn the relations between layers. The starting state of weights seems to have a substantial impact where the model learns the generative factors. Without making assumptions about the relations in the data, we cannot force them to learn specific features aligned on both layers. If we want to learn the hierarchy between layers, we should let the model learn the dependencies.

5 Proposed architecture

5.1 Model

The section defines the neural network architecture used during evaluation and discusses the expectations.

The architecture was strongly motivated by β - VAE architecture and tried to tackle the insights discussed in the previous chapter. Compared to the exploratory research, we stack five layers of latent variables (with additional encoders) to see if we may have better disentanglement on lower or higher layers. Figure 19 shows the general architecture, where every latent layer has its own latent parameters estimations, own sampling and own decoder. As discussed in the conclusions of our previous chapter, our goal was to get excellent reconstruction on the first layer. We used a large base encoder and decoders with enough capacity to learn required features (Based encoder and decoder adapted from Boxhead evaluation [Chen et al., 2021]). We also increased the number of latent variables to 500 in the first layer so that the architecture could split the features as granular as it requires and would not need to put together features due to the small latent space.

The latent layers were related to each other through an encoder described in Table 1. The input signal of each layer was received only from latent variables of the previous layer (the mean and variance). The latent layers are connected through an additional encoder that combines all current layer variables and allows learning non-linear (Leaky

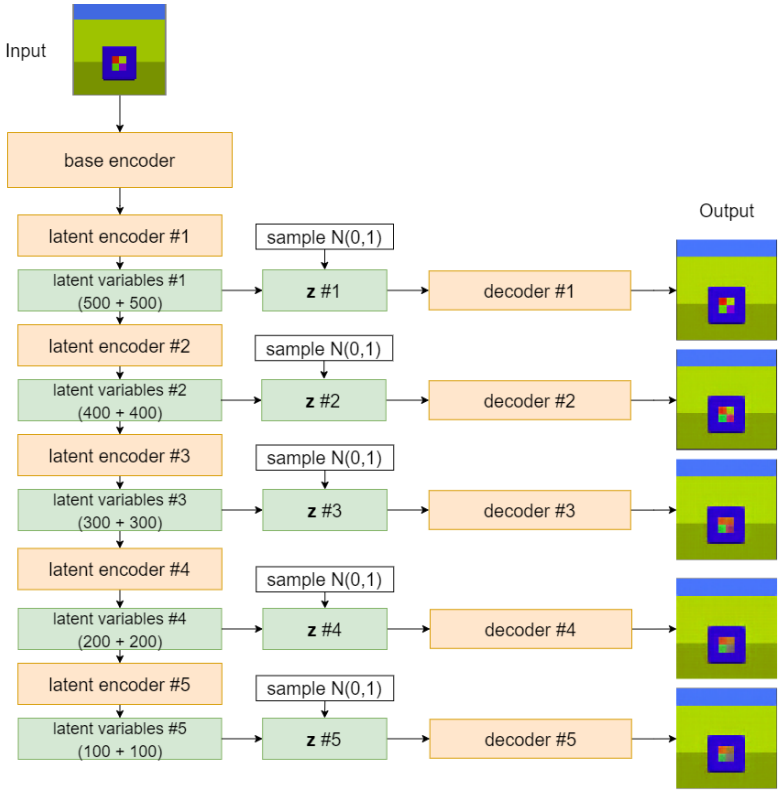

Figure 19. 5-layer variational autoencoder. Every latent parameters layer is stacked on top of the previous layer, connected with an additional encoder. Every latent layer has its own decoder.

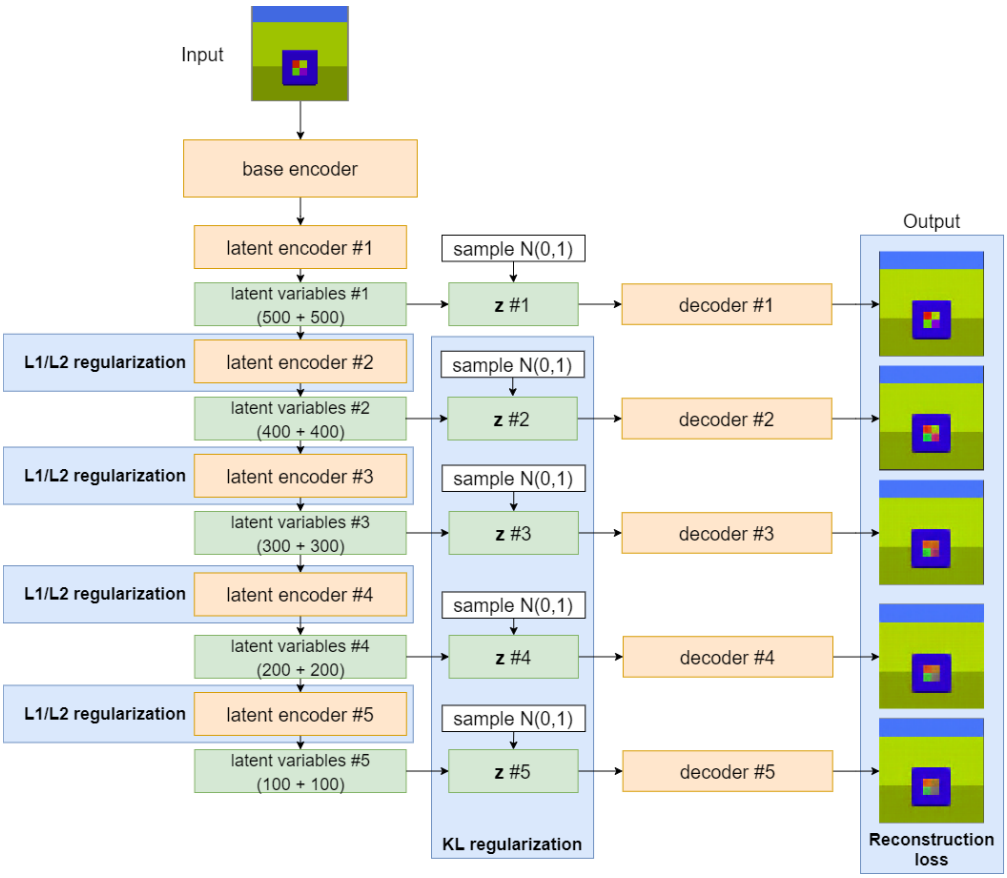

Figure 20. Loss terms on proposed architecture. L1 and L2 regularisation on the encoder weights between latent parameters. KL regularisation and reconstruction loss as in β-VAE.

ReLU) relationships. This setup should enable the architecture to learn the dependencies it thinks are optimal, and constraints from the architecture would be minimal. Base encoder, latent encoder #1 and decoder architectures are available in Appendix I.

Linear	[2 x number of latent variables in the previous layer]
Leaky ReLu	[0.3]
Linear	[2 x number of latent variables in current layer]

Table 1. Additional encoder between latent variable layers. It contains two linear layers and a non-linear activation function Leaky ReLu between them (with negative values slope 0.3). The first linear layer fully connects all previous layer neurons to the same amount of neurons. It will allow the architecture to combine the signal from all latent variables. The architecture also does not limit the model to only combining mean or variance values from the previous layer (it is allowed to mix the signal). In the second linear layer, all nodes are connected to all current latent layer mean and variance nodes.

5.2 Loss function

Our training loss function was similar to β-VAE. We did not explicitly force the model to disentangle on the first latent variables layer, as we want this layer to capture as much as possible from the input and not lose details. On the follow-up layers, we add regularisation to enforce more disentanglement. The first latent layer objective function is defined as:

$$L_{\mathrm{LEVEL1}}(\theta) = -\mathbb{E}_{\mathbf{z_1}\sim q_\theta^{(1)}(\mathbf{z_1}|\mathbf{x})} \log p_\theta^{(1)}(\mathbf{x} \mid \mathbf{z_1}). \tag{10}$$

We included the Kullback-Leibler divergence regularisation term on the second to fifth layer (see also Figure 20):

$$L_{\mathrm{LEVEL2-5}}(\theta, \beta) = -\sum_{i=2}^{5} \left(\mathbb{E}_{\mathbf{z_i}\sim q_\theta^{(i)}(\mathbf{z_i}|\mathbf{x})} \log p_\theta^{(i)}(\mathbf{x} \mid \mathbf{z_i}) + \beta \cdot D_{\mathrm{KL}} \left(q_\theta^{(i)}(\mathbf{z_i} \mid \mathbf{x}) \| p_\theta^{(i)}(\mathbf{z_i}) \right) \right). \tag{11}$$

In our architecture, $q_\theta^{(i)}(\mathbf{z_i} \mid \mathbf{x})$ defines our probabilistic encoder of each layer, which produces multivariate Gaussian distribution with a diagonal covariance structure. Each layer has its own probabilistic decoder $p_\theta^{(i)}(\mathbf{x} \mid \mathbf{z_i})$, and its prior $p_\theta(\mathbf{z_i})$ is defined to be from normal Gaussian distribution $\mathcal{N}(0, 1)$.

The fully connected layer weights between latent parameters were regularised. We applied L1 and L2 regularisation to reduce over-fitting and control that we would not combine weak and noisy signals to our latent variables. L1-regularisation is the sum of all weights' absolute values applied on specific weights' layers. It forces the model

during training unused weights closer to zero. L2-regularisation is the sum of square weights applied on specific weights' layers. It helps avoid over-fitting and keeps the weights in lower values and similar ranges.

$$L_{\text{TOTAL}} = L_{\text{LEVEL1}} + L_{\text{LEVEL2}-5} + \gamma_1 \cdot L1_{loss} + \gamma_2 \cdot L2_{loss} \tag{12}$$

Total loss is the sum of level 1, other layers losses and L1, L2 regularisation terms (γ_1 and γ_2 are hyperparameters for L1 and L2 loss).

5.3 Evaluation Results

We evaluated our architecture with four different β values (0.5, 1.5, 2.5, 5.0) on all three $\mathrm{Boxhead}$ datasets. We also run β-VAE with the same encoder-decoder and latent variables size 300. For our proposed architecture, we used fixed coefficients for L1 and L2 regularisation (0.0001 and 0.05). They have not been selected by using hyperparameter tuning. Instead, they were selected based on weight values monitored during exploratory experiments training. Experiment setup was also discussed in Chapter 3.1.

5.3.1 Reconstruction

The reconstruction loss on the first layer of the proposed architecture is the lowest. For example, if we look at Figure 21 $\beta = 1.5$ subplot (second from left), then scores on layer L1 are lower than on other layers. A similar trend can be seen in all other subplots of Figure 21. We did not force any disentanglement on that layer's loss function; therefore, the result is expected. We see a slight increase in the reconstruction loss in layers 2-5. In the case of $\beta = 5$, we see a pretty constant reconstruction error from the second to fifth layer. It indicates that we lose already on layer 2 some details, probably micro-level generative factors related ones. It is confirmed by visual inspection of the reconstruction images. In Figure 22 (last column), we can see that the $\mathrm{Boxhead}$ micro-level factors are not reconstructed starting from the second layer. In all other β values, we can see some differences between layers, and the reconstruction loss increases layer by layer. β-VAE architecture reconstruction loss (see Figure 16 reconstruction images) also follows a similar pattern, except on $\beta = 5$ on a single $\mathrm{Boxheadsimple2}$ dataset, where the construction loss value is higher than for other datasets.

5.3.2 Disentanglement

In micro-level feature capturing, our proposed architecture has a better joint model DCI disentanglement score for lower β values compared to β-VAE experiments. In Figure 23, subplots where β values are 0.5 and 1.5, the score of the first layers are higher than the β-VAE model scores at any β experiments. From Figure 23, we can see that our proposed architecture also enforces disentanglement through the architecture. Based

Figure 21. Reconstruction loss of our proposed architecture and original β-VAE for different β values and datasets. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. The loss on the y-axis is a sum of reconstruction losses of every training set sample (evaluated after training).

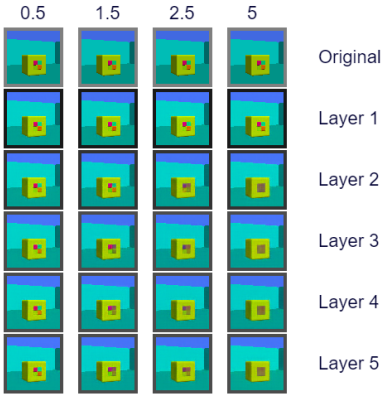

Figure 22. Original reconstruction images on different hyperparameter β values and on different layers of the proposed architecture.

Figure 23. Boxhead micro-level factors disentanglement scores - joint model. Each subplot groups together one specific experiment. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. On the y-axis, the metric score is shown for each experiment dataset.

Figure 24. Boxhead micro-level factors disentanglement scores - per latent variable model. Each subplot groups together one specific β experiment. On the x-axis L1 - L5 indicate the layers 1 to 5 of our proposed architecture. On the y-axis metric score is shown for each experiment dataset (scale 0-1). In $\beta = 5$ some of the boxheadsimple2 metric scores are 0 as no latent variable captured the micro-level generative factors (micro-level factor DCI metric is defined in Chapter 3.2.1).

on the metric, it also disentangles level 1 layer latent variables, although in the loss function, we do not explicitly enforce it. For 5-layer architecture, the highest score for boxheadsimple2 dataset is for $\beta = 0.5$, which is better than any compared β-VAE experiments (compared against all β-VAE experiments using the same dataset). When comparing the results against regular β-VAE architecture, we must take into account that the same β in different architectures applies different pressure on the model. For other datasets, the best score is not from the first layer. The best disentanglement for the boxheadsimple dataset is on the $\beta = 0.5$ third layer. Boxhead_07 dataset has the best disentanglement score on $\beta = 1.5$ third layer.

We do not see a similar trend for per latent variable model disentanglement DCI scores. The original β-VAE performs as good as or better on each dataset than our proposed architecture. See Figure 24, where for each dataset, β-VAE architecture scores are as good as or better than any of our proposed architecture results. Note also that the scores values are very low.

We do not have a consensus between our metrics; we checked the findings using latent traversals. Latent traversal images show latent variables where the KL-divergence value is over 0.1 nats (lower values do not seem to have much impact on the output that could be visually inspected). In Figure 25, we have latent traversal images for 5-layer architecture second layer (L2) trained on Boxheadsimple2 dataset, where $\beta = 1.5$. This layer's joint model DCI disentanglement score was second-best based on Figure 23. The best one was on $\beta = 0.5$ first layer, but this is not regularised with KL. Therefore it has hundreds of latent variables holding information, and the content is visually hard to grasp). For comparison, β-VAE traversal in Figure 26 uses the same input image and the same $\beta = 1.5$ value as our proposed architecture example. We chose this β-VAE model for comparison as the metric result is slightly better than $\beta = 0.5$. Also, we have the details of the micro-level factors visible (as β = 2.5 already loses many details

from micro-level generative factors (visible in Figure 16)). While reviewing the latent traversals, we do not notice much difference, except that our architecture has learned the features cleaner: β-VAEuses 15 latent variables to describe most of the info, and our architecture uses 12. We think that our proposed architecture had a better DCI joint model disentanglement score because many latent variables hold a small proportion of information. An example is seen in Figure 25, last row, column GB. Similar to this for many latent variables, the KL-divergence is close to 0 (but not 0), which boosts the metric a bit as it captures only a single generative factor. It happens probably due to back-propagating loss from lower layers which also adjust weights in related layers.

The best disentanglement metric scores (which show the score for the single best latent variable) for $\mathrm{Boxheadsimple2}$ dataset are shown in Figure 27. There are very few cases where any model has captured one of the eye generative factors. The hidden generative factor results are similar between the 5-layer architecture and the regular β-VAE model. Best disentanglement score heatmaps of other experiment's datasets have been added to Appendix III.

The best disentanglement scores of the $\mathrm{Boxheadsimple2}$ dataset for all generative factors (Figure 28) align well with the latent traversal visualisation. Our architecture has identified some high-level features (box size) in a cleaner manner.

When evaluating disentanglement with respect to all generative factors, DCI joint model and DCI per latent variable model metrics do not show the same trends. Joint models disentanglement metric (Figure 29) indicates that learned features in 5-layer have slightly better disentanglement compared to β-VAE. However, compared to the per latent variable model (Figure 30) metric shows the opposite. The trends of the score metrics do not align.

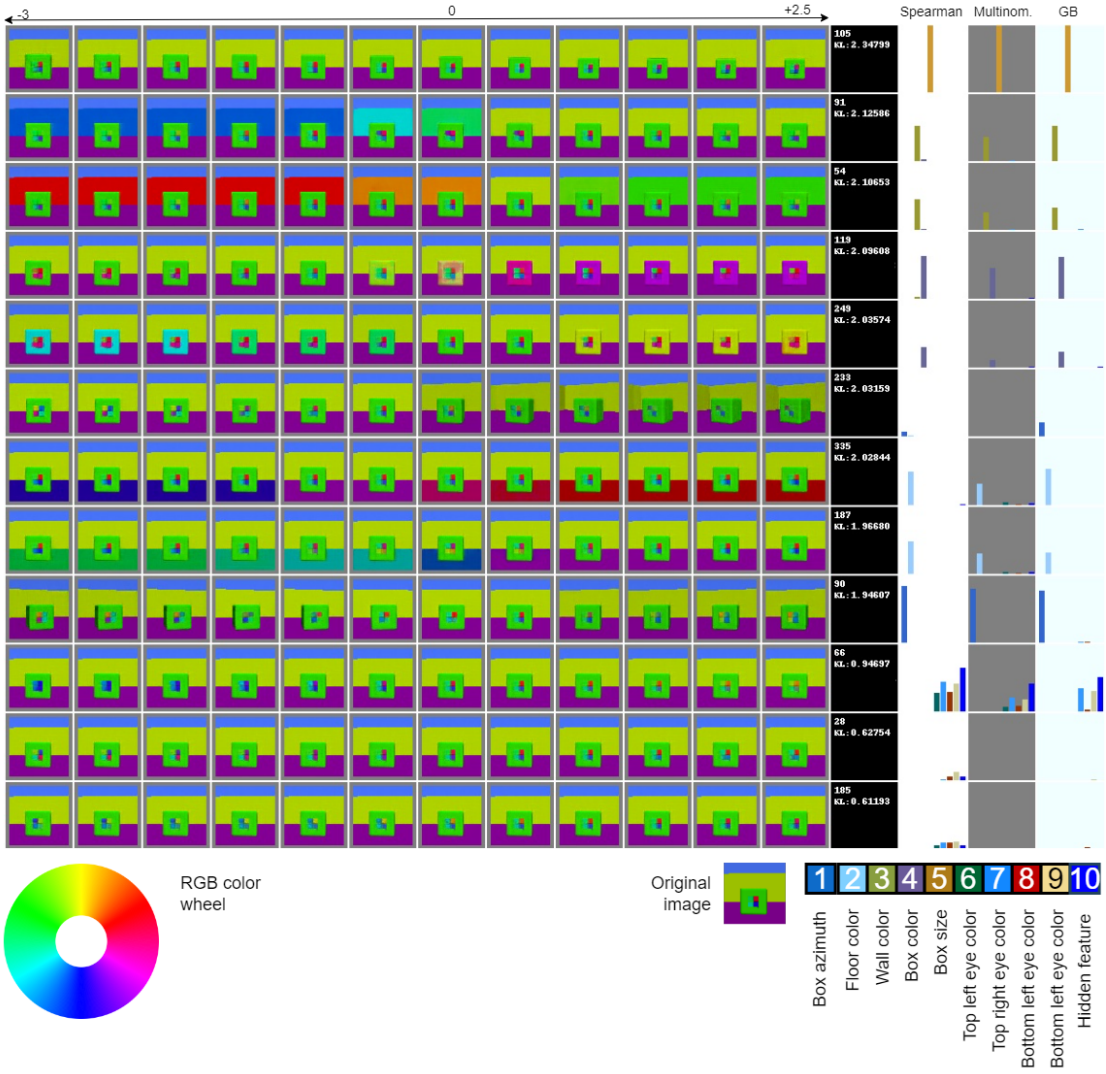

Figure 25. β = 1.5 proposed 5-layer architecture second latent layer latent variables single sample traversal images. It is limited to showing only latent variables where the KL loss value is over 0.1 nats. Additionally RGB color wheel is shown. For example, second and third row latent variables have colors encoded in the same order as in the RGB color wheel.

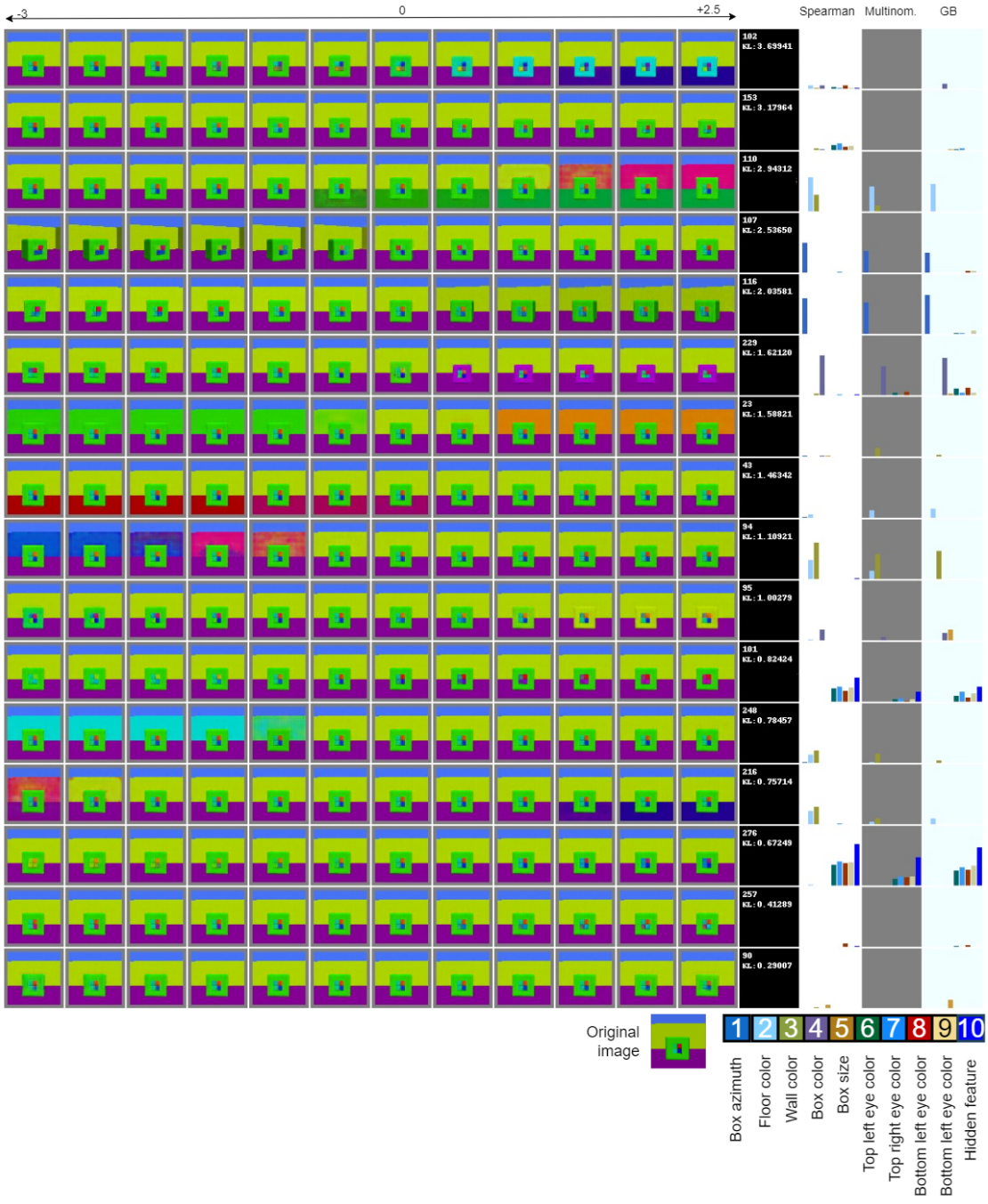

Figure 26. β = 1.5 β-VAE architecture latent variables single sample traversal images. It only shows latent variables where the KL loss value is over 0.1 nats.

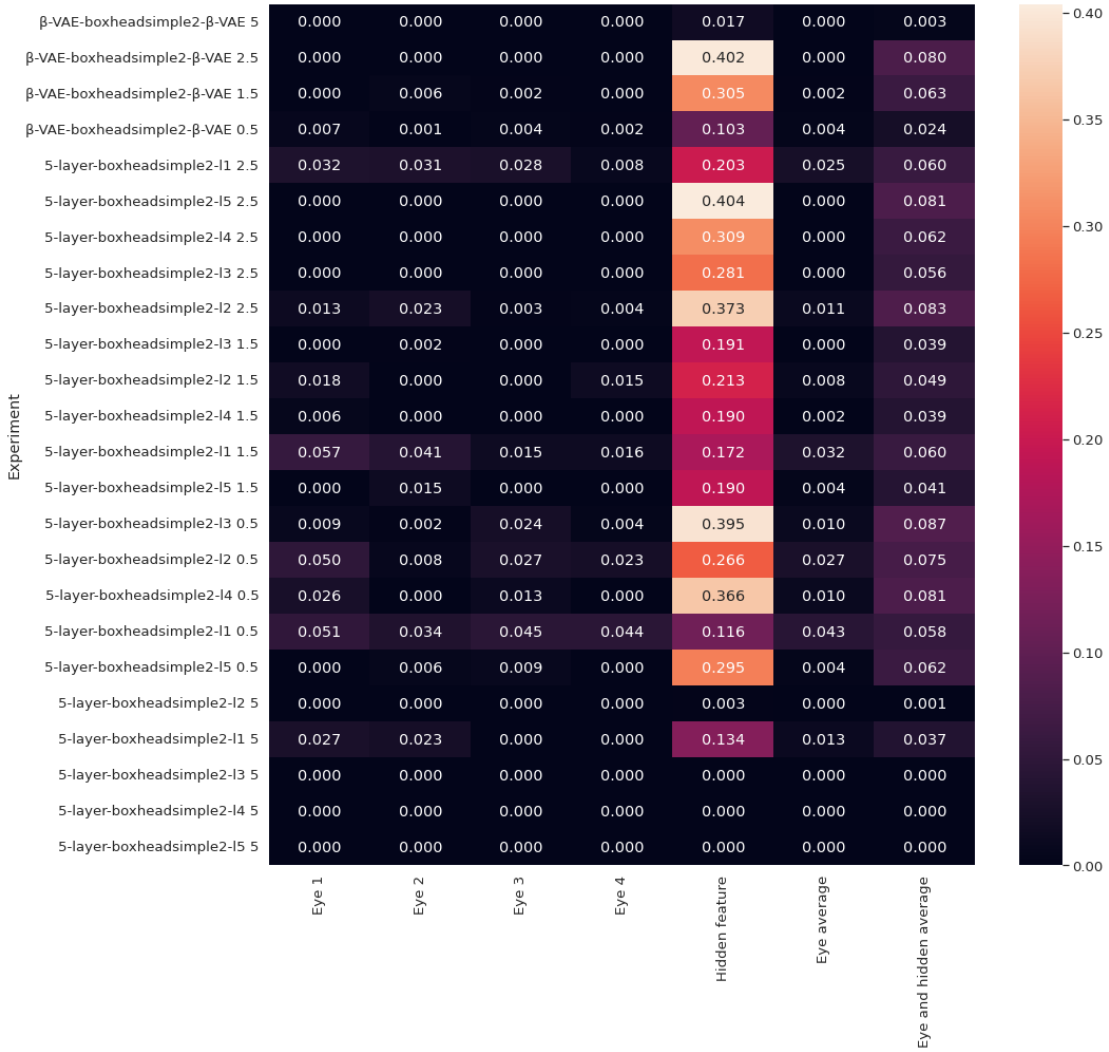

Figure 27. Boxheadsimple2 dataset micro-level generative factors heatmap of the best disentanglement score. On each row experiments name (type - dataset - latent layer - beta). There is no significant difference in the generative factors captured between our proposed architecture and β-VAE.

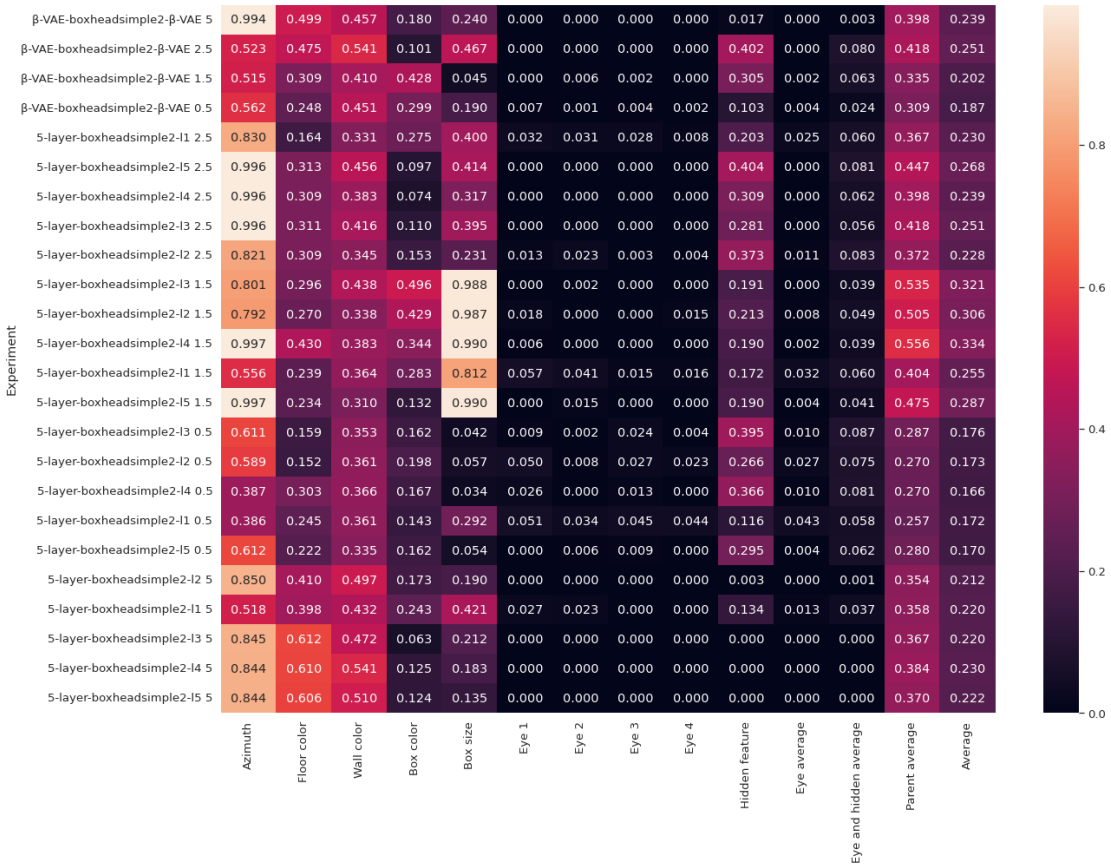

Figure 28. Boxheadsimple2 dataset all generative factors heatmap of best disentanglement score. On each row experiments name (type - dataset - latent layer - beta). Box size generative factor scores better on our proposed architecture.

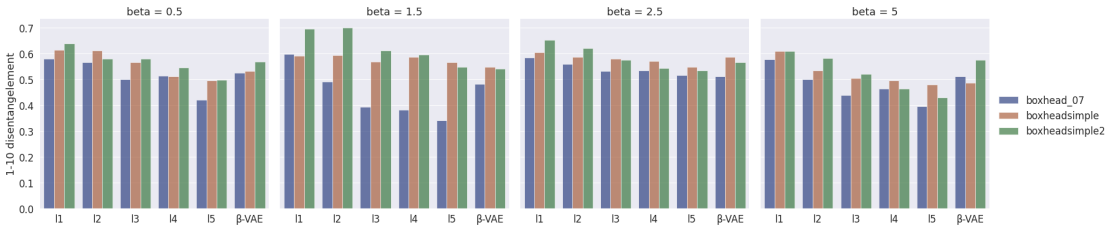

Figure 29. Boxhead all generative factors disentanglement scores - joint model. Each subplot groups together experiments at one specific β value. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. On the y-axis metric score is shown for each experiment dataset.

Figure 30. Boxhead all generative factors disentanglement scores - per latent variable model. Each subplot groups together experiments at one specific value. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. On the y-axis metric score is shown for each experiment dataset.

Figure 31. Boxhead micro-level generative factors completeness scores - joint model. Each subplot groups together experiments at one specific β value. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. On the y-axis metric score is shown for each experiment dataset.

5.3.3 Completeness

For completeness on micro-level features, β-VAE captures the generative factors as good as (or better) than our architecture based on the results of DCI by joint model DCI metric. See Figure 31, where for every dataset exists a β-VAE experiment that is better than any of our proposed architecture experiments. When comparing all generative factors metrics, DCI completeness joint model score is slightly better for β-VAE architecture experiments. (See Figure 32 β-VAE models give as good as or better results than layered architecture).

For per latent variable model DCI in Figure 33, we see a clear trend of improving completeness score over latent layers 2-5 where we apply KL-regularisation. For all datasets, a layer from our proposed architecture outperforms the β-VAE experiment result (see Figure 33 fifth layer (L5) results and compare to best β-VAE results).

Figure 32. Boxhead all generative factors completeness scores - joint model. Each subplot groups together experiments at one specific β value. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. On the y-axis metric score is shown for each experiment dataset.

Figure 33. Boxhead all generative factors completeness scores - per latent variable model. Each subplot groups together experiments at one specific β value. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture. On the y-axis metric score is shown for each experiment dataset.

Figure 34. Boxhead micro-level generative factors disentanglement scores (joint model DCI). Boxheadsimple2 dataset with only images without azimuth changes. On the x-axis, L1 - L5 indicate the layers 1 to 5 of our proposed architecture ($\beta = 1.5$). Baseline β-VAE label given together with hyperparameter β value.

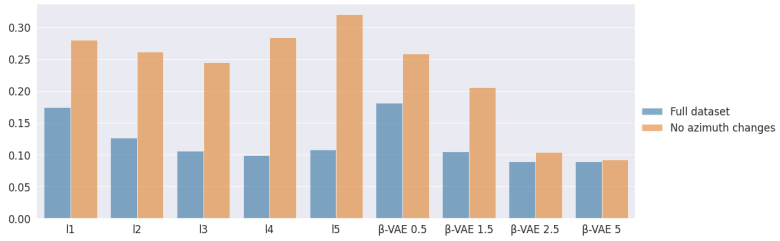

Figure 35. Boxhead micro-level generative factors disentanglement scores (per latent variable model DCI). Boxheadsimple2 dataset with only images without azimuth changes. On the x-axis, L1 - L5 indicate the layers 1 to 5 of our proposed architecture ($\beta = 1.5$). Baseline β-VAE label given together with hyperparameter β value.

5.4 Additional experiment - no azimuth feature

Based on the experiments, we think it might be hard for β-VAE based architectures to capture generative factors where pixel positions vary, and the feature amount of pixels is small (all micro-level generative factors). We did additional experiments on the boxheadsimple2 dataset, where we included only images with no box rotation. Therefore, the small micro-level generative factors would always be in a similar position in the image. The results show that both architectures improve capturing micro-level generative factors based on DCI disentanglement metrics per joint model (in Figures 34) and per generative factor model (Figure 35). The 5-layer model latent traversal images (see Figure 36) show that the last seven latent variables have captured different parts of the micro-level generative factors. In the original β-VAE model traversal images (provided in Appendix IV. Figure 56), we can also identify the disentanglement of micro-level generative factors. However, they have been captured in a bigger number of latent variables. Each latent variable seems to capture a smaller proportion of the generative factor. Both images identify the micro-level factors more successfully compared to experiments with rotation samples. The heatmap of the best disentanglement score in Figure 37 also confirms the results: there are latent variables that significantly capture the eyes' generative factors.

The DCI completeness scores of micro-level generative factors (see Figures 38) had slightly worse results compared to the experiments using the full dataset. It can be due to using a significantly smaller number of samples in no-azimuth experiments (51328 vs 5200) or indicating that the experiment result depends on the initial state of the network weights.

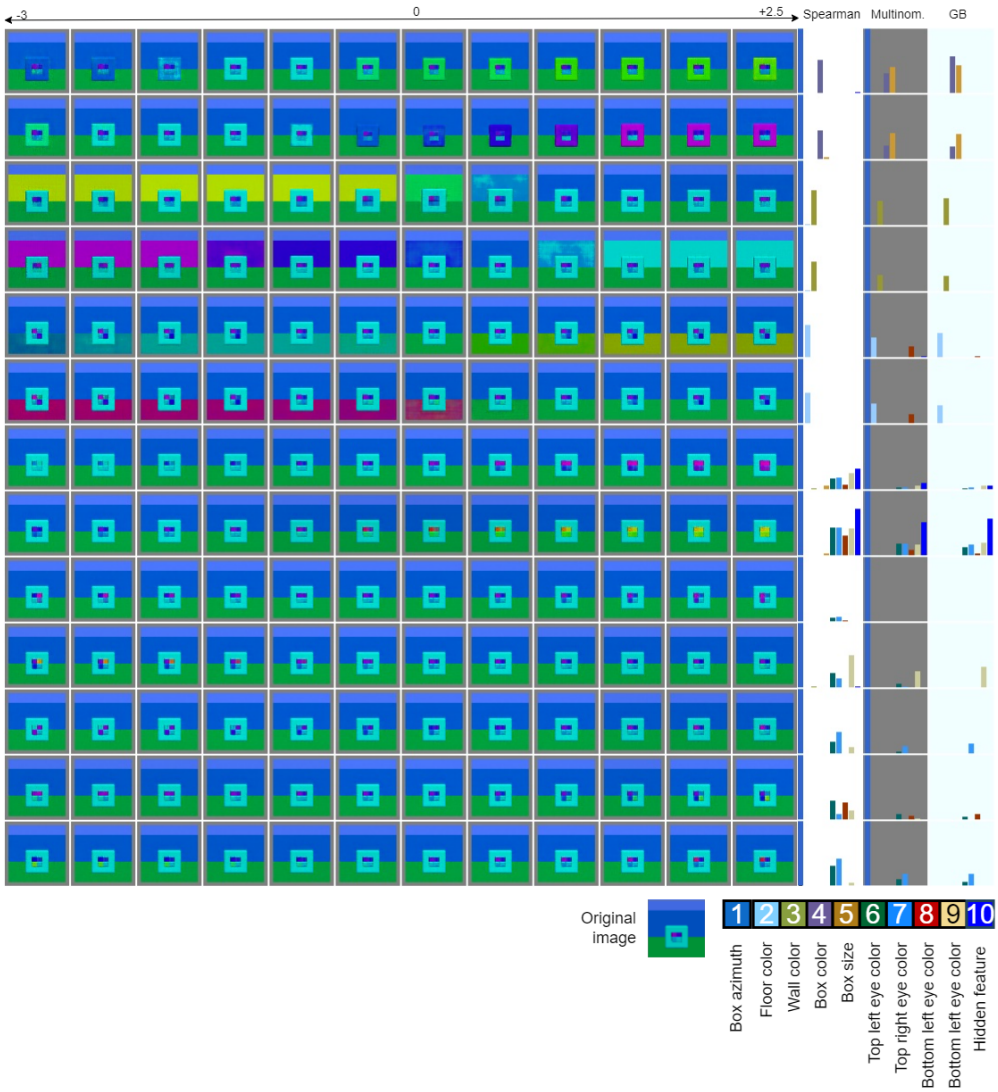

Figure 36. β = 1.5 proposed 5-layer architecture second latent layer latent variables single sample traversal images. Limited to show only latent variables where KL loss value is over 0.1 nats. The model was trained on Boxheadsimple2 dataset, where samples had no azimuth changes.

6 Discussion

We compared our proposed 5-layer architecture to a similar encoder/decoder β-VAE architecture. One of our goals was to improve disentanglement. In Figure 39, on layers 2-4, our average best disentanglement score improves. On layer one, we also have

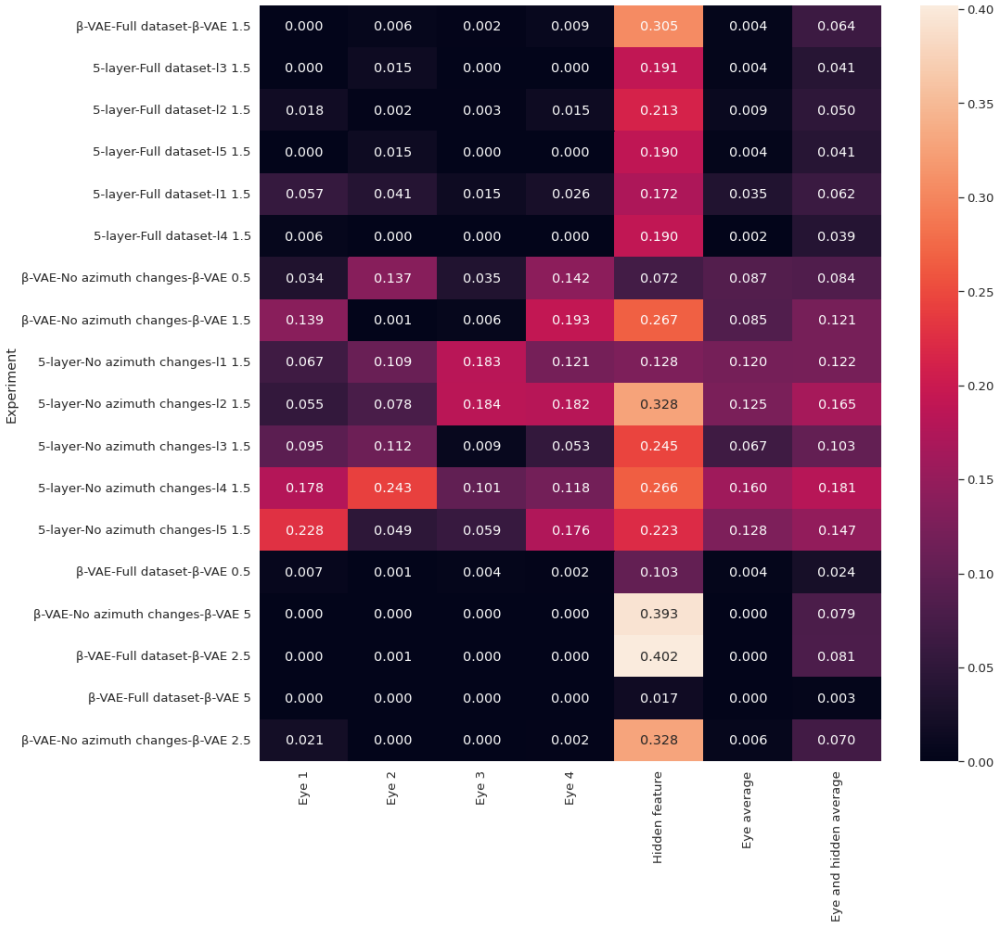

Figure 37. Boxheadsimple2 dataset micro-level generative factors best disentanglement scores heatmap. On each row experiments name (type - dataset - latent layer - beta). No-azimuth dataset with good reconstruction ($\beta < 2.5$) also captures micro-level generative factors.

good best disentanglement scores. However, as no KL-divergence is directly applied as regularisation (it was still partially regularised due to being related to deeper layers), it has many overlapping latent variables carrying the same or similar info (latent traversal example given in Appendix IV, Figure 47).

Although, our proposed architecture applies every following latent variable layer with a stronger regularisation. We did not identify any improvement in disentanglement based on latent traversal images evaluation nor based on DCI disentanglement per latent variable model scores. DCI disentanglement score joint model improved, but it might be a side-effect of using a layered architecture. We had a disagreement between commonly

Figure 38. Boxhead micro-level generative factors completeness scores (joint model). Boxheadsimple2 dataset with only images without azimuth changes. On the x-axis, L1 - L5 indicates layers 1 to 5 of our proposed architecture ($\beta = 1.5$). Baseline β-VAE label given with hyperparameter β value.

.

used joint model DCI scores and our proposed per latent variable model based DCI metrics. The first one uses all latent variables to predict a single generative factor value - assigns high scores to only the best latent variables. In per latent variable model DCI, we evaluate each latent variable goodness independently from others. Therefore, the latent variables do not compete to get a better score. Notably, our architecture has many latent variables which carry almost no information. However, it can be that due to back-propagation still going through these latent variables, some information is still there, although not used for reconstruction. Small proportions of info in many latent variables could explain why disentanglement scores for the joint model were higher for multi-layered architecture. The use of DCI metrics for many latent variables should be reviewed for future experiments, and possibly other metrics like Mutual Information Gap (MIG) [Zaidi et al., 2020] should be considered.

Interestingly, for best disentanglement scores (which describes completeness), our proposed architecture has captured better some of the independent factors (Boxheadsimple2 dataset feature azimuth (see Figure 28)). This finding is not confirmed by the joint model DCI completeness score. Latent traversal images and DCI completeness per latent variable model metric support this finding.

Our results show that the architecture added additional pressure on KL-regularisation over different layers, but this did not improve the separation of generative factors based on currently used metrics. The objective function for our proposed architecture was very similar to the β-VAE one. The ability not to disentangle Boxhead datasets features could be a limitation of the loss function. There is competition between reconstruction and disentanglement/compression in the loss function. Therefore, due to the small number of pixels in the Boxhead micro-level generative factors, we are unable to disentangle them.

Also, It might be that our encoder, which has convolutions, cannot capture the signal from the small micro-level generative factors due to the pixel positions changing due to box rotation. The hypothesis is partially confirmed by our additional experiments

conducted on a small dataset, where we included only images with no box rotation (see Figures 34, 35, 36, 56). Both architectures capture now, to some extent, the small micro-level generative factors, but our 5-layer architecture has captured the features into fewer latent variables - confirmed by latent traversal images. As the encoder contains a set of convolutional layers, the ability of the encoder to capture small features should be evaluated [Geirhos et al., 2019].

We should also evaluate if the micro-level generative factors we are trying to disentangle are separable into independent latent variables theoretically or if there is a need for some weak supervision in the process [Träuble et al., 2021]. It might be that some generative factors cannot be identified based on data (example given in Introduction).

Current experiments were performed once. Repetitive experiments should be conducted with different initial weights to understand the impact of a random starting point for our training process and have stronger statistical support to make conclusions.

Boxhead dataset has very specific properties. The micro-level generative factors cover a very small part of the image, but β-VAE based objective function currently focuses first on features covering larger areas. Additional experiments should be made with bigger dependent features to understand the objective function capabilities in disentangling them and if needed reviewing the pixel-wise reconstruction loss in the objective function.

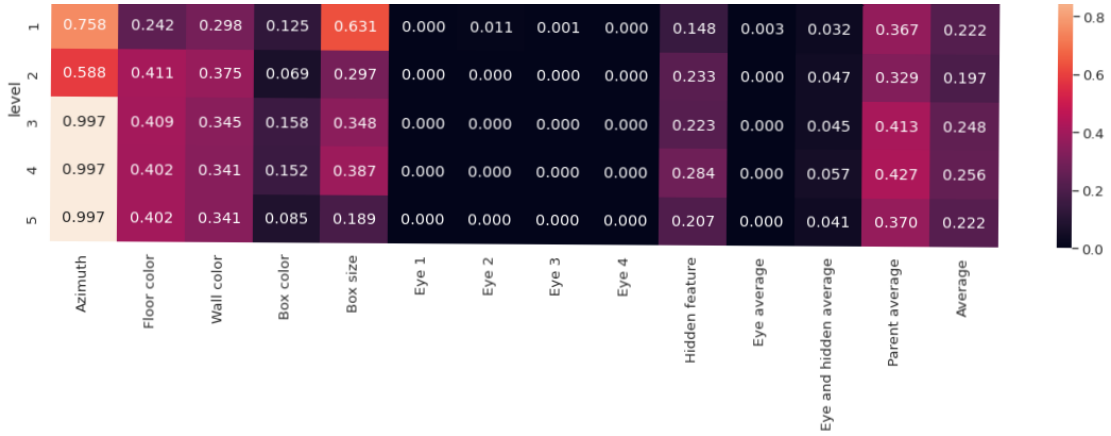

Figure 39. Boxheadsimple dataset best disentanglement score over layers by a generative factor

.

7 Conclusion and future work

We conducted a set of experiments on β-VAE to understand its behaviour on $\mathrm{Boxhead}$ dataset. We confirmed that it was not able to capture well small dependent features in images. Based on exploratory experiments, we proposed an architecture similar to β-VAE that is not restricted to a small number of generative features and would learn multiple layers of latent variables at once and, in deeper layers, push the latent variables towards better disentanglement. The approach did not improve the dependent generative factors capturing compared to the standard β-VAE approach on $\mathrm{Boxhead}$ datasets. Based on latent traversal images, it visually seems that our proposed architecture often uses fewer latent variables to encode macro-level generative factors information compared to regular β-VAE architecture.

During our research process, we introduced two new metrics to evaluate disentanglement in our setup due to possible side-effects of layered architecture on the DCI score. First, we introduced metrics similar to DCI, but the importance matrix of a latent variable was calculated independently from other latent variables. Secondly, a metric to get the generative factor best disentanglement score. Both metrics results are well aligned with the results from latent traversal images.

Based on our results, we hypothesised that our current architecture might have difficulties capturing micro-level features when their positions change on different images. We conducted an additional exploratory experiment with $\mathrm{Boxhead}$ dataset without rotation of the box object. Initial results show that this significantly improved capturing micro-level features on our proposed and original β-VAE architecture based models.

Our proposed architecture should be evaluated on a dataset with pixel-wise larger dependent variables. Current $\mathrm{Boxhead}$ dataset dependent variables cover a relatively small area compared to other features. Motivated by our additional experiment of a dataset without box rotations, an additional evaluation should be done on the encoder and decoder to check if they can capture the details about features if their position changes significantly on the image. In order to improve capturing details of the image, also objective function should be evaluated. Currently, pixel-level comparison of input and reconstruction guides the model to learn pixel-wise larger features first.

References

[Altosaar, Jaan , 2016] Altosaar, Jaan (2016). Tutorial - what is a variational autoencoder? https://jaan.io/what-is-variational-autoencoder-vae-tutorial. Last checked on May 14, 2022.

[Bengio et al., 2012] Bengio, Y., Courville, A., and Vincent, P. (2012). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[Burgess et al., 2019] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., Lerchner, A., and London, D. (2019). Understanding disentangling in β-VAE.

[Burgess, Chris and Kim, Hyunjik, 2018] Burgess, Chris and Kim, Hyunjik (2018). 3d shapes dataset. https://github.com/deepmind/3d-shapes. Last checked on January 15, 2022.

[Chen et al., 2021] Chen, Y., Träuble, F., Dittadi, A., Bauer, S., and Schölkopf, B. (2021). Boxhead: A Dataset for Learning Hierarchical Representations.

[Corti et al., 2019] Corti, L. M., Denmark, C., Fraccaro, M., Liévin, V., and Winther, O. (2019). BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling.

[Deng, 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Eastwood and Williams, 2018] Eastwood, C. and Williams, C. K. I. (2018). A framework for the quantitative evaluation of disentangled representations. *ICLR*.

[Geirhos et al., 2019] Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. (2019). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.

[Grid.ai, Inc, 2022] Grid.ai, Inc (2022). Pytorch lightning. https://www.pytorchlightning.ai. Last checked on January 17, 2022.

[Higgins et al., 2018a] Higgins, I., Amos, D., Pfau, D., Racaniere, S., Matthey, L., Rezende, D., and Deepmind, A. L. (2018a). Towards a Definition of Disentangled Representations.

[Higgins et al., 2021] Higgins, I., Chang, L., Langston, V., Hassabis, D., Summerfield, C., Tsao, D., and Botvinick, M. (2021). Unsupervised deep learning identifies semantic disentanglement in single inferotemporal face patch neurons. *Nature Communications 2021 12:1*, 12(1):1–14.

[Higgins et al., 2017] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A., and Deepmind, G. (2017). β-VAE: Learning basic visual concepts with a constrained variational framework.

[Higgins et al., 2018b] Higgins, I., Sonnerat, N., Matthey, L., Pal, A., Burgess, C. P., Bošnjak, M., Shanahan, M., Botvinick, M., Hassabis, D., and Lerchner, A. (2018b). SCAN: Learning hierarchical compositional visual concepts.

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441.

[Hyvärinen and Oja, 2000] Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

[Lee, WonKwang and Metger, Tony, 2022] Lee, WonKwang and Metger, Tony (2022). 1konny/beta-vae. `https://github.com/1Konny/Beta-VAE`. Last checked on January 15, 2022.

[Locatello et al., 2019] Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2019). Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations. In *International Conference on Machine Learning*, pages 4114–4124.

[Matthey et al., 2017] Matthey, L., Higgins, I., Hassabis, D., and Lerchner, A. (2017). dSprites: Disentanglement testing Sprites dataset. https://github.com/deepmind/dsprites-dataset/.

[Mika et al., 1999] Mika, S., Ratsch, G., Weston, J., Scholkopf, B., and Muller, K. R. (1999). Fisher discriminant analysis with kernels. *Neural Networks for Signal Processing - Proceedings of the IEEE Workshop*, pages 41–48.

[Rocca, 2019] Rocca, J. (2019). Understanding Variational Autoencoders (VAEs) | by Joseph Rocca | Towards Data Science.

[Rolínek et al., 2018] Rolínek, M., Zietlow, D., and Martius, G. (2018). Variational Autoencoders Pursue PCA Directions (by Accident).

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning Internal Representations by Error Propagation.

[Schönfeld et al., 2018] Schönfeld, E., Ebrahimi, S., Sinha, S., Darrell, T., and Akata, Z. (2018). Generalized Zero-and Few-Shot Learning via Aligned Variational Autoencoders.

[Sepliarskaia et al., 2021] Sepliarskaia, A., Kiseleva, J., and De Rijke, M. (2021). How to Not Measure Disentanglement.

[Slavin Ross and Doshi-Velez, 2021] Slavin Ross, A. and Doshi-Velez, F. (2021). Benchmarks, Algorithms, and Metrics for Hierarchical Disentanglement.

[Sønderby et al., 2016] Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., Winther, O., and Dk, O. (2016). Ladder Variational Autoencoders. Technical report.

[Träuble et al., 2021] Träuble, F., Creager, E., Kilbertus, N., Locatello, F., Dittadi, A., Goyal, A., Schölkopf, B., and Bauer, S. (2021). On Disentangled Representations Learned from Correlated Data.

[Vahdat and Kautz, 2020] Vahdat, A. and Kautz, J. (2020). NVAE: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, volume 2020-December. Neural information processing systems foundation.

[Weights and Biases, Inc, 2022] Weights and Biases, Inc (2022). Weights and biases. `https://docs.wandb.ai`. Last checked on May 01, 2022.

[Weng, Lilian, 2018] Weng, Lilian (2018). From autoencoder to beta-vae | lil'log. `https://lilianweng.github.io/posts/2018-08-12-vae/`. Last checked on May 14, 2022.

[Zaidi et al., 2020] Zaidi, J., Boilard, J., Gagnon, G., and Carbonneau, M.-A. (2020). Measuring Disentanglement: A Review of Metrics.

[Zhao et al., 2017] Zhao, S., Song, J., and Ermon, S. (2017). Learning Hierarchical Features from Generative Models.

I. Encoder/Decoder architecture

conv2d	[64, 4, 2]
Leaky ReLu	[0.3]
Batch normalisation	
conv2d	[128, 4, 2]
Leaky ReLu	[0.3]
Batch normalisation	
conv2d	[256, 2, 2]
Leaky ReLu	[0.3]
Batch normalisation	
conv2d	[512, 2, 2]
Leaky ReLu	[0.3]
Batch normalisation	
Flatten	
Linear	[1024]

Table 2. Encoder base architecture: convd2d layer given as [number of filters, kernel size, stride]. Architecture adapted from [Chen et al., 2021] (Large Architecture).

Batch normalization	
Linear	[2 * number of latent variables in layer 1]

Table 3. Additional encoder to first level latent variables.

Linear	[1024]
Batch normalization	
Linear	[8192]
Batch normalization	
Reshape	[512, 4, 4]
Conv trans 2d	[256, 4, 2]
Leaky ReLu	[0.3]
Batch normalization	
Conv trans 2d	[256, 4, 1]
Leaky ReLu	[0.3]
Batch normalization	
Conv trans 2d	[128, 4, 2]
Leaky ReLu	[0.3]
Batch normalization	
Conv trans 2d	[64, 4, 1]
Leaky ReLu	[0.3]
Batch normalization	
Conv trans 2d	[64, 3, 4]
Leaky ReLu	[0.1]

Table 4. Decoder architecture. Architecture adapted from [Chen et al., 2021] (Large Architecture).

II. No azimuth - additional charts

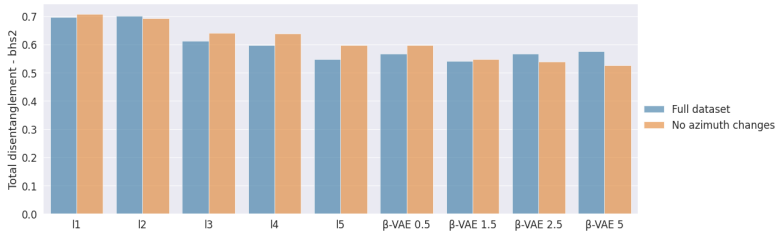

Figure 40. Boxhead all features disentanglement scores (joint model). Boxheadsimple2 dataset with only images without azimuth changes.

Figure 41. Boxhead all features disentanglement scores (per latent variable model). Boxheadsimple2 dataset with only images without azimuth changes.

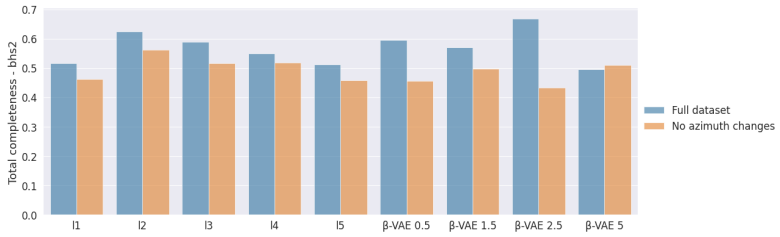

Figure 42. Boxhead all features completeness scores (joint model). Boxheadsimple2 dataset with only images without azimuth changes.

.

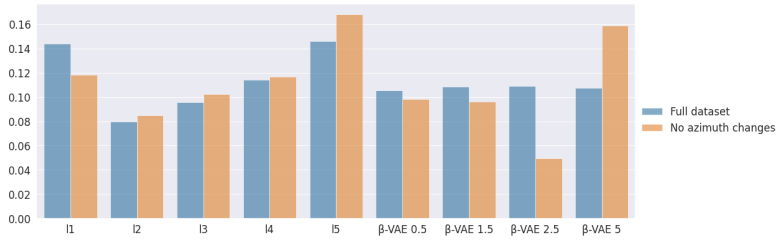

Figure 43. Boxhead all features completeness scores (per latent variable model). Boxheadsimple2 dataset with only images without azimuth changes.

.

III. Best disentanglement - heatmaps

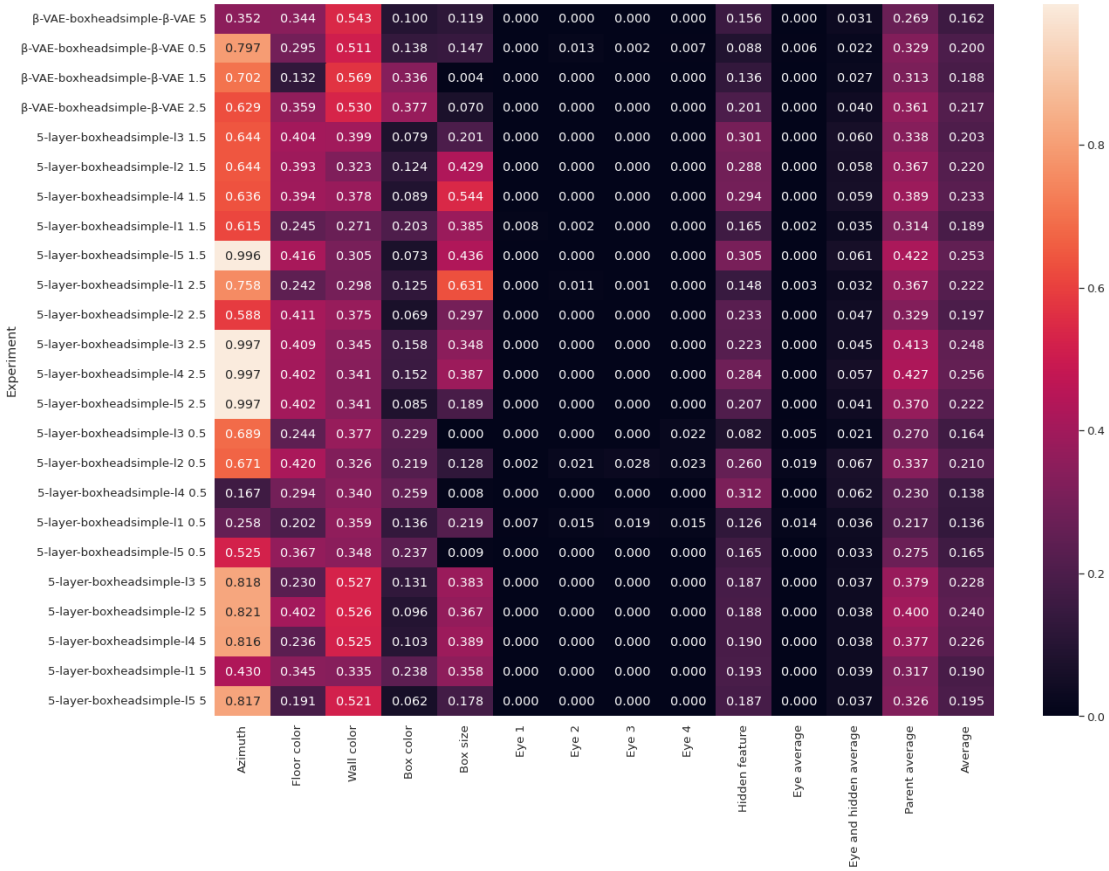

Figure 44. Heatmap of Boxheadsimple dataset best disentanglement scores.

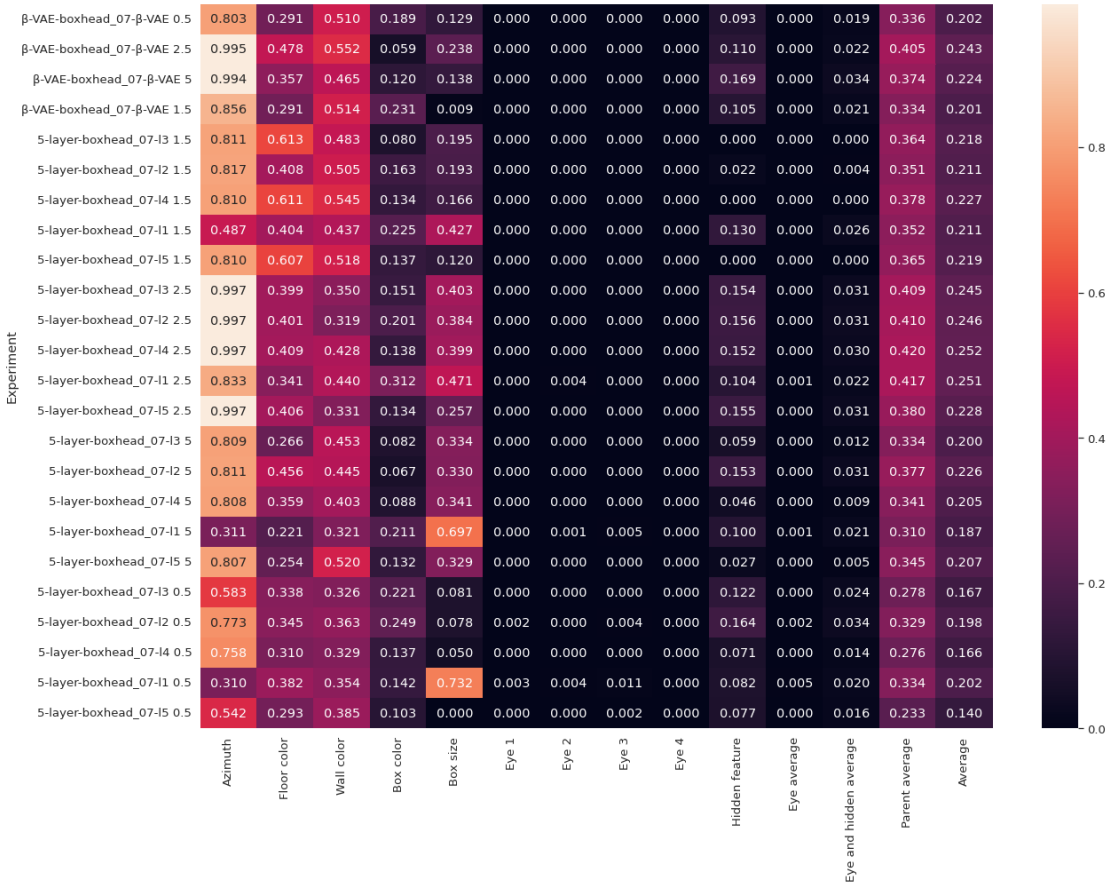

Figure 45. Heatmap of Boxhead_07 dataset best disentanglement scores.

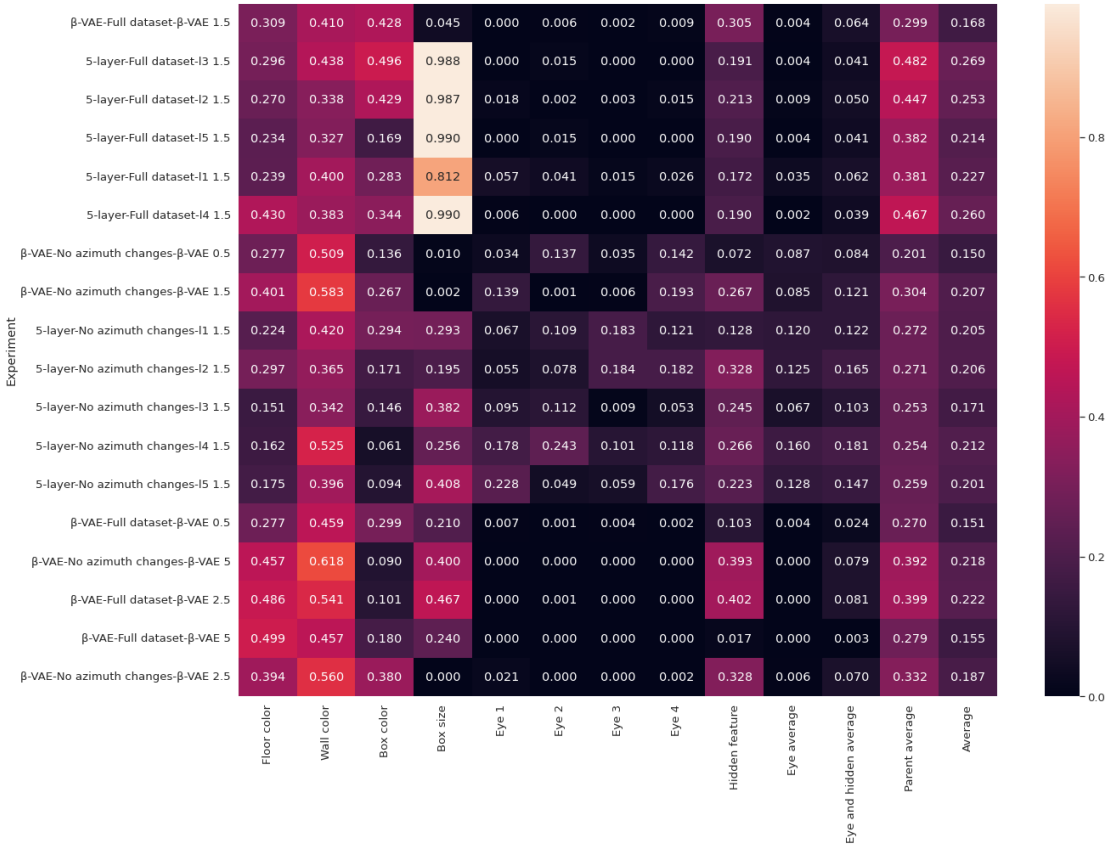

Figure 46. Heatmap of Boxheadsimple2 (without azimuth changes) dataset best disentanglement scores.

IV. Latent traversals

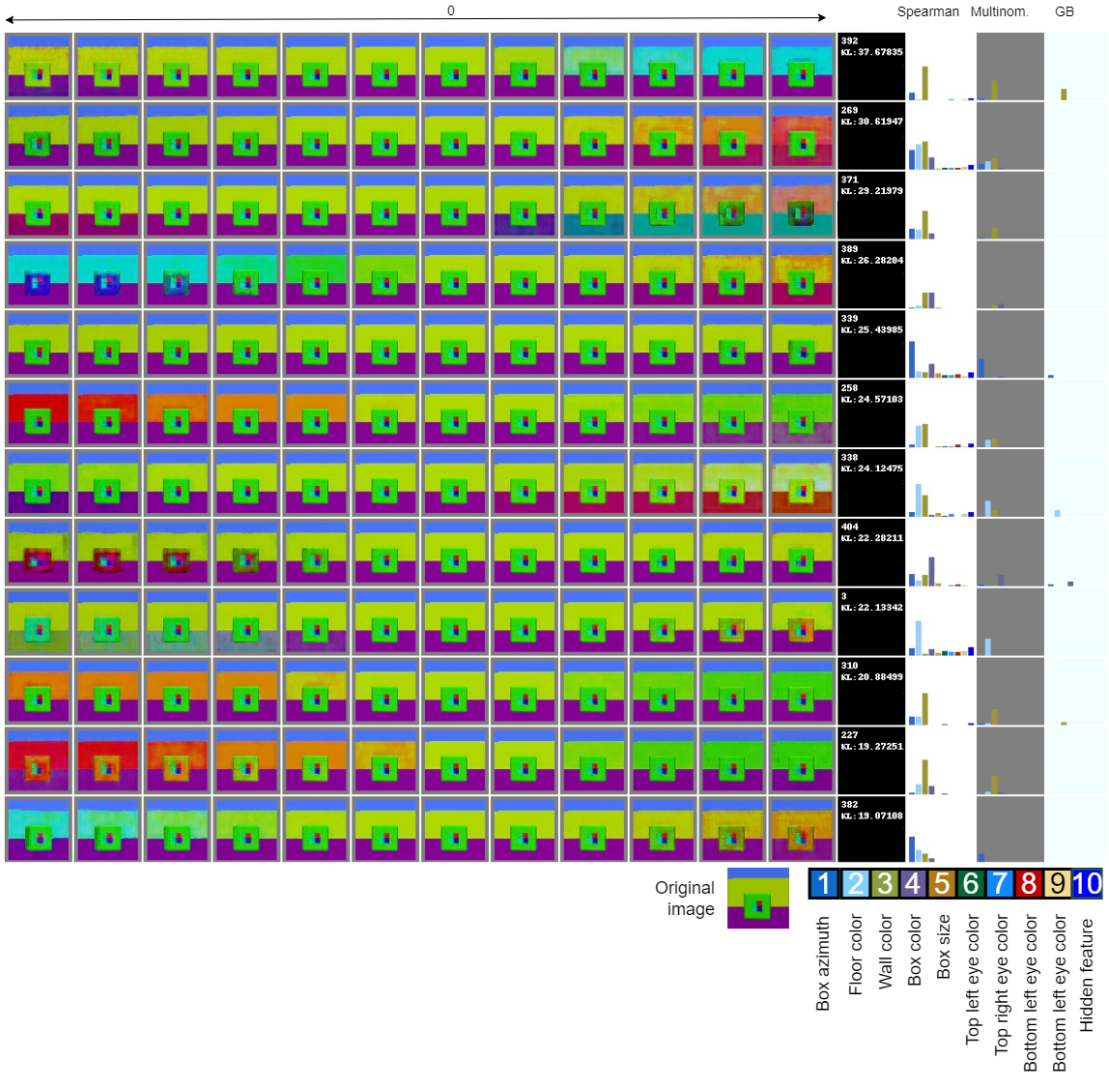

Figure 47. $\beta = 0.5$ proposed 5-layer architecture first 12 latent variables of layer L1 (based on KL of latent variable).

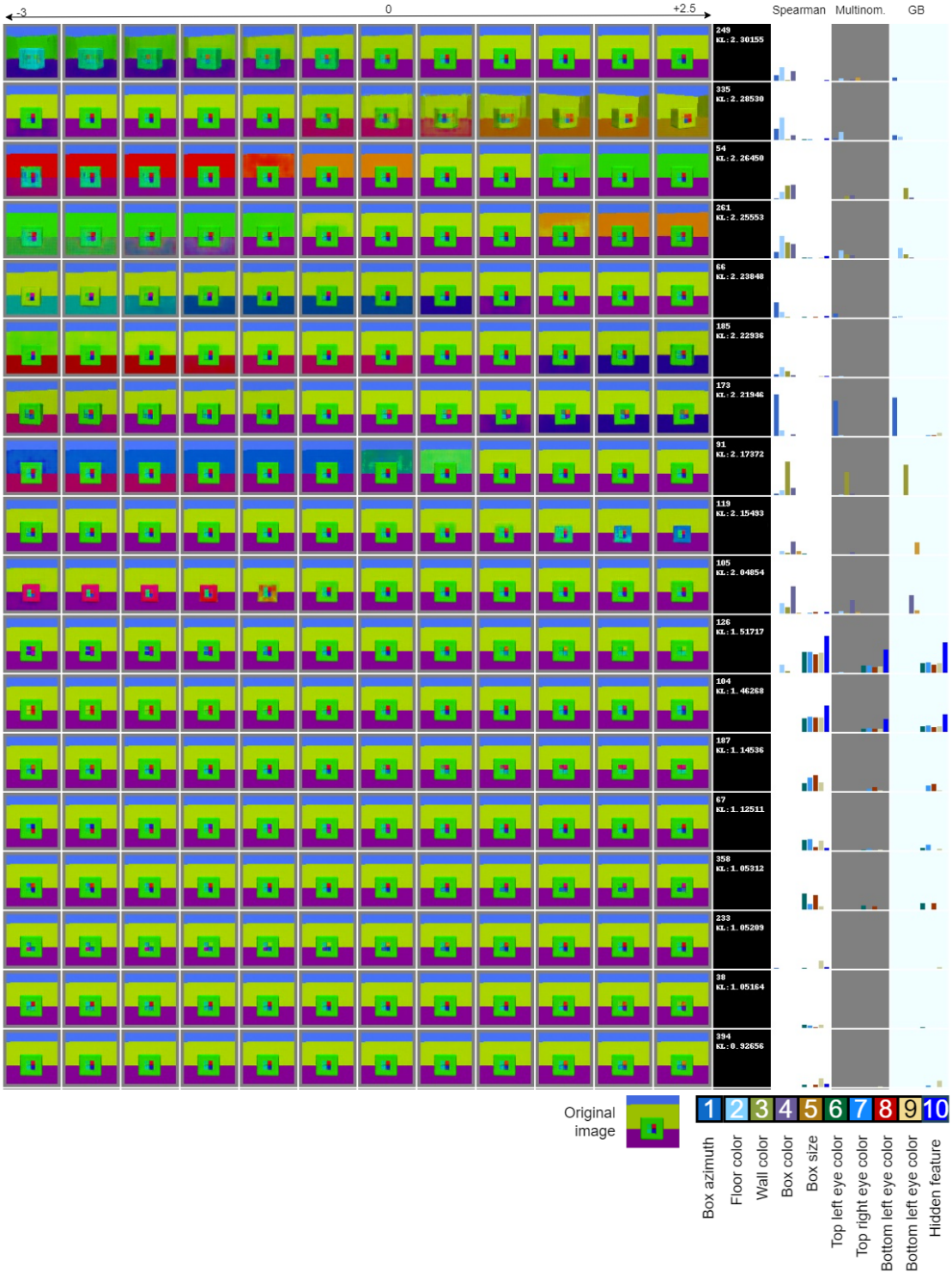

Figure 48. Latent variables layer L2, 5-layer architecture β=0.5.

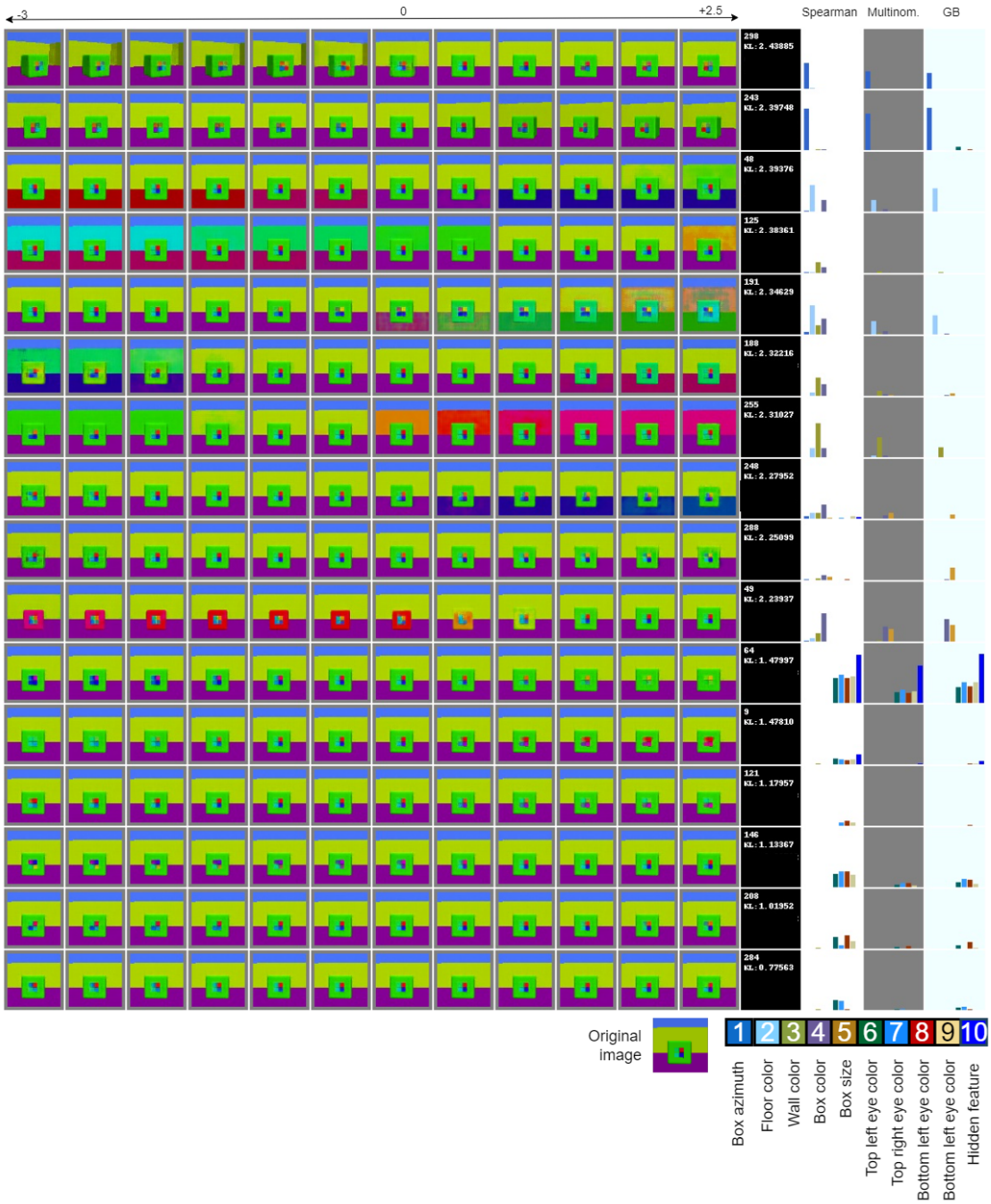

Figure 49. Latent variables layer L3, 5-layer architecture β=0.5.

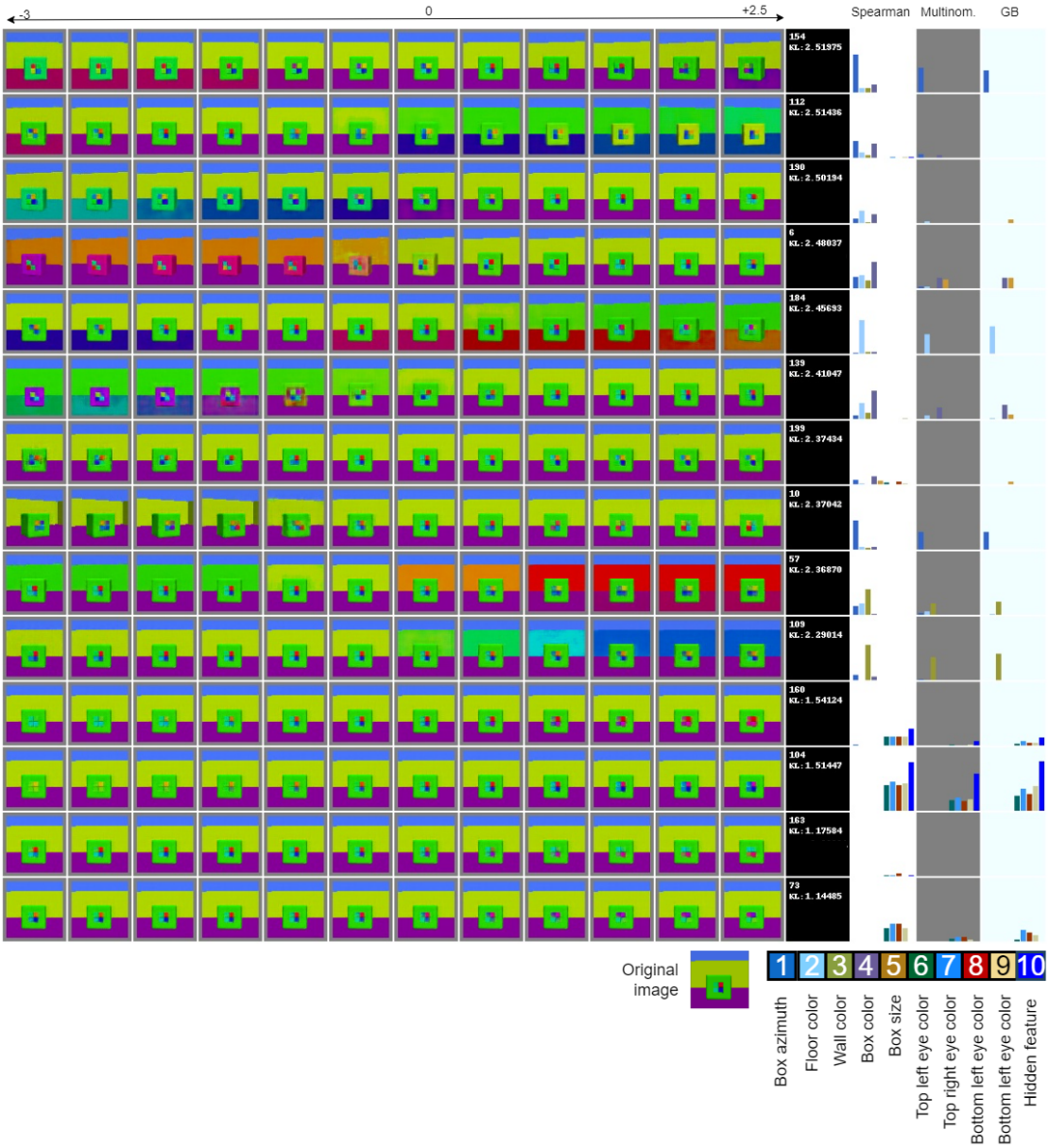

Figure 50. Latent variables layer L4, 5-layer architecture β=0.5.

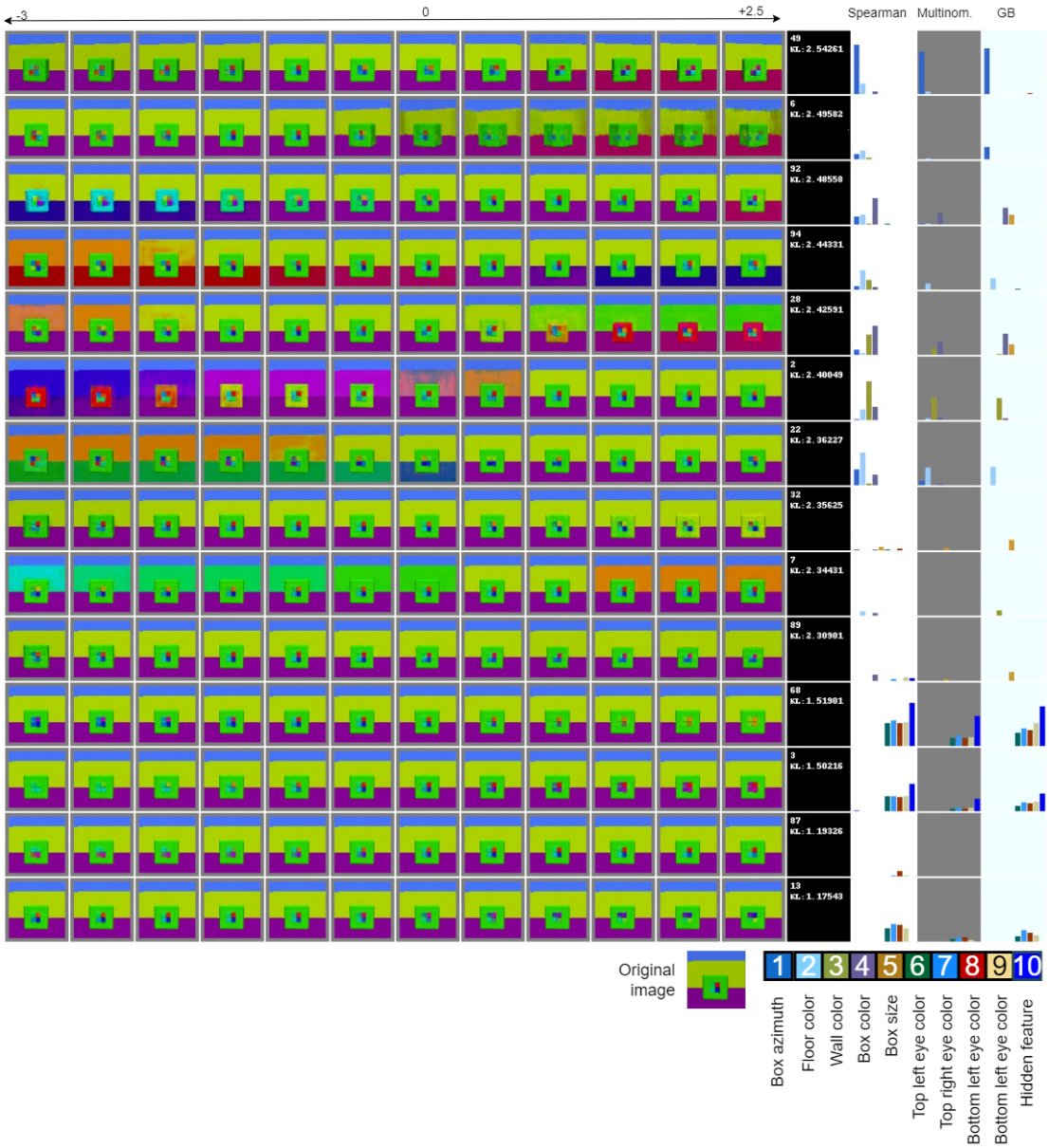

Figure 51. Latent variables layer L5, 5-layer architecture β=0.5.

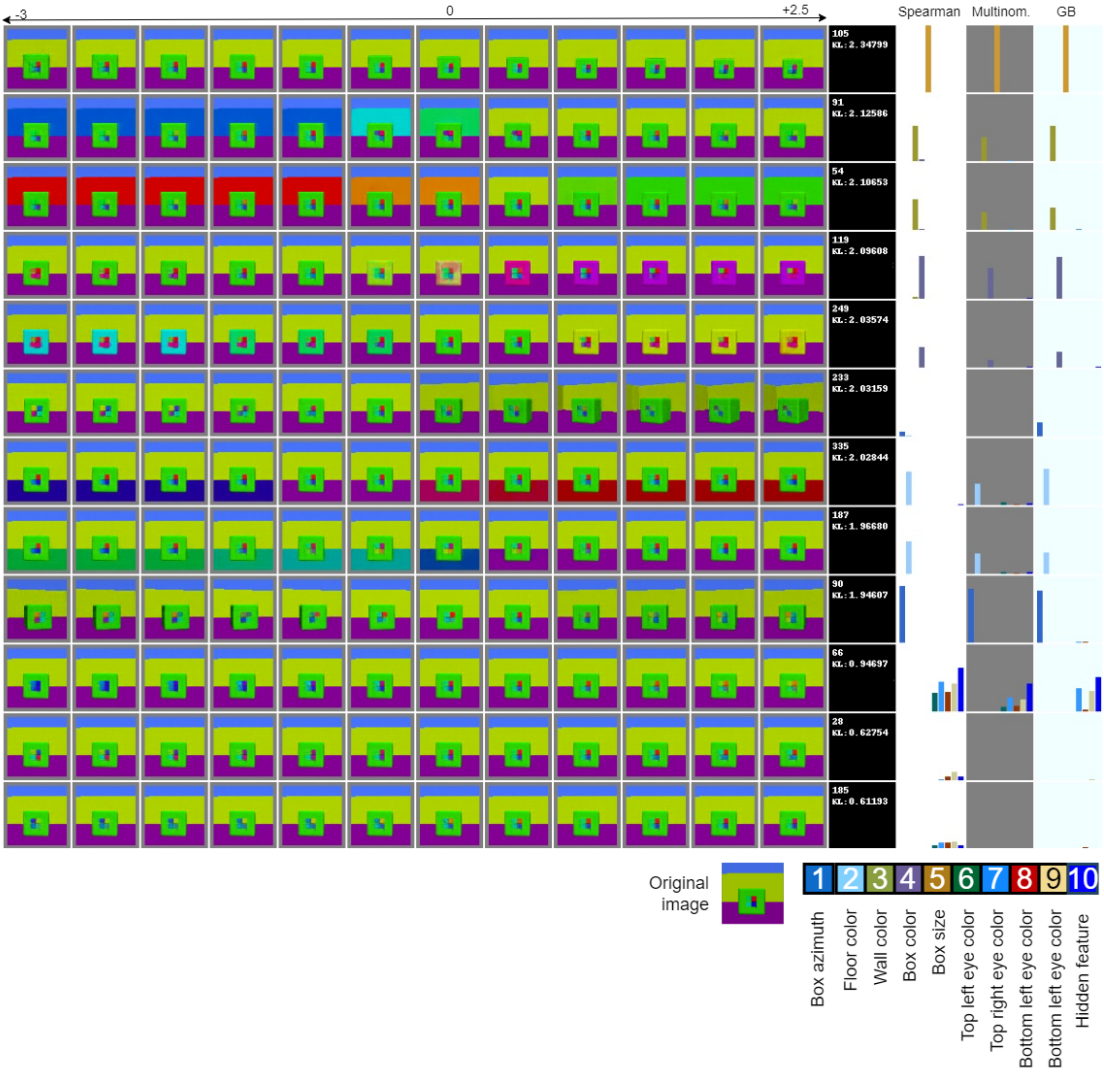

Figure 52. Latent variables layer L2, 5-layer architecture β=1.5.

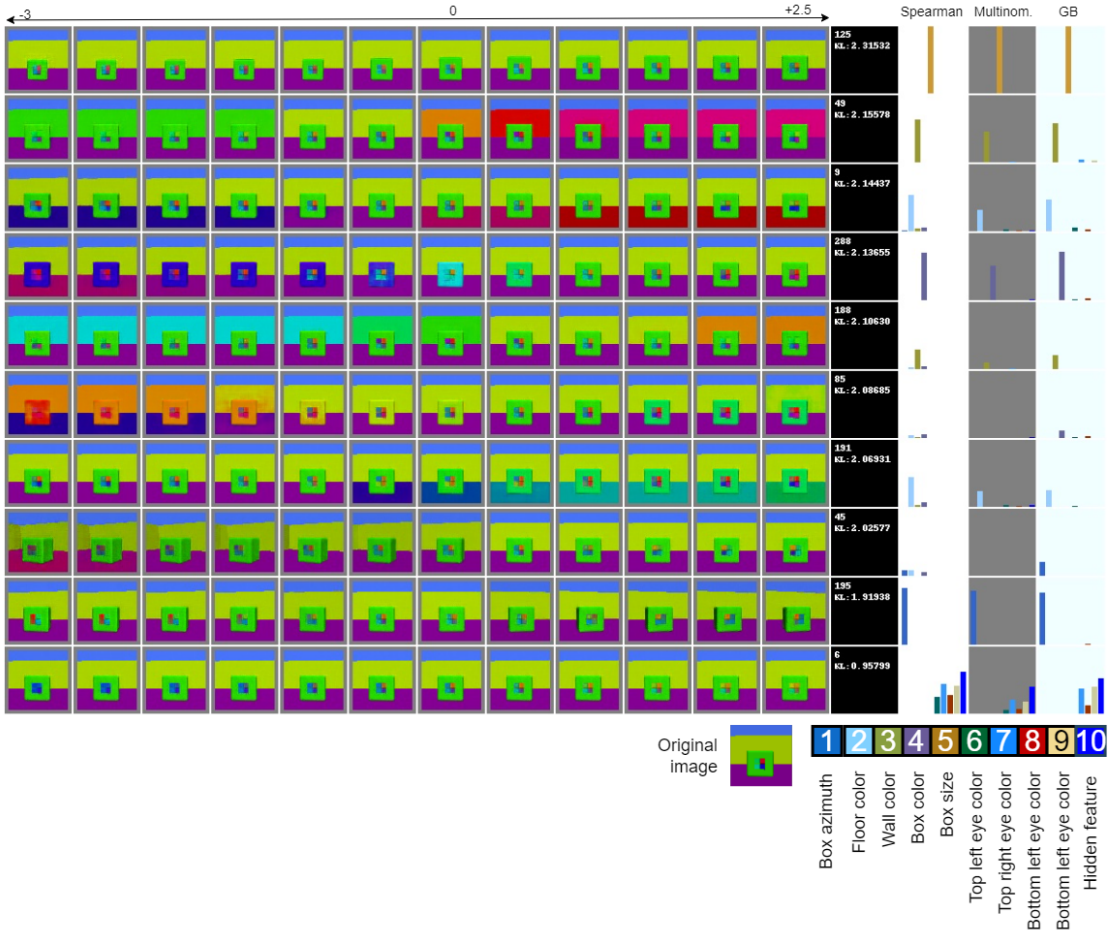

Figure 53. Latent variables layer L3, 5-layer architecture β=1.5.

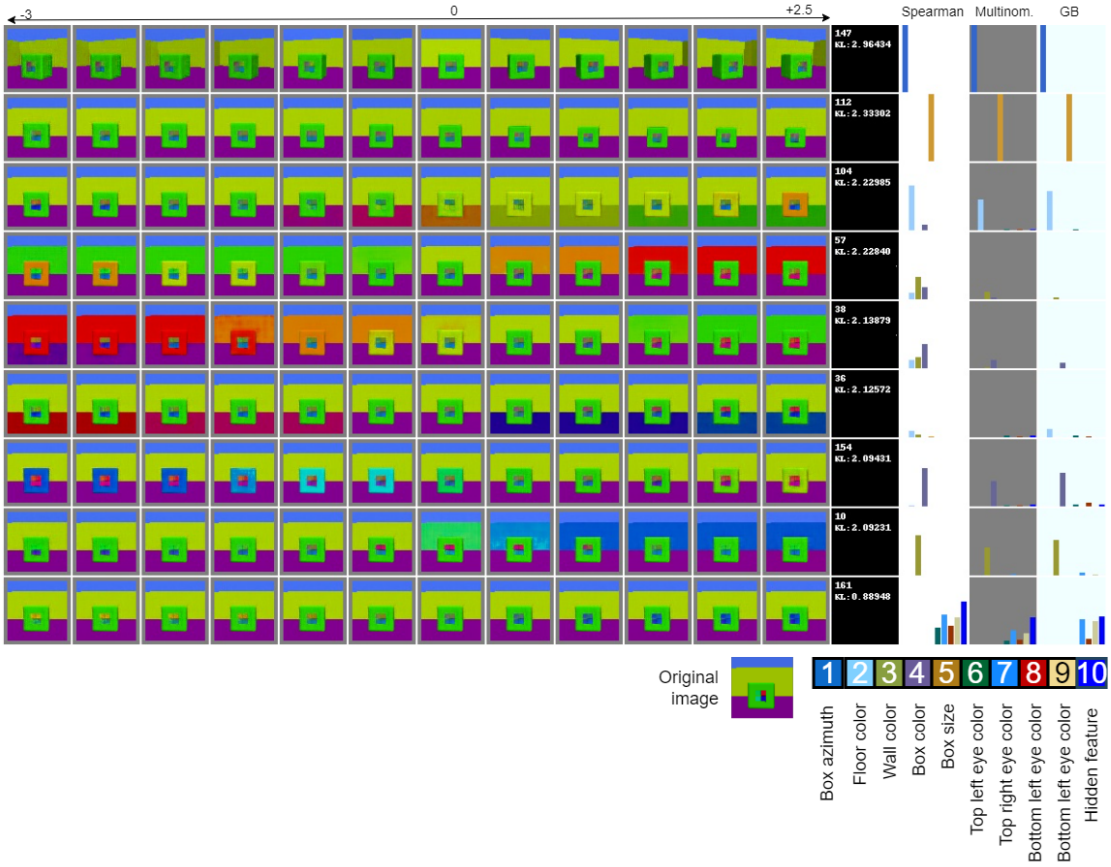

Figure 54. Latent variables layer L4, 5-layer architecture β=1.5.

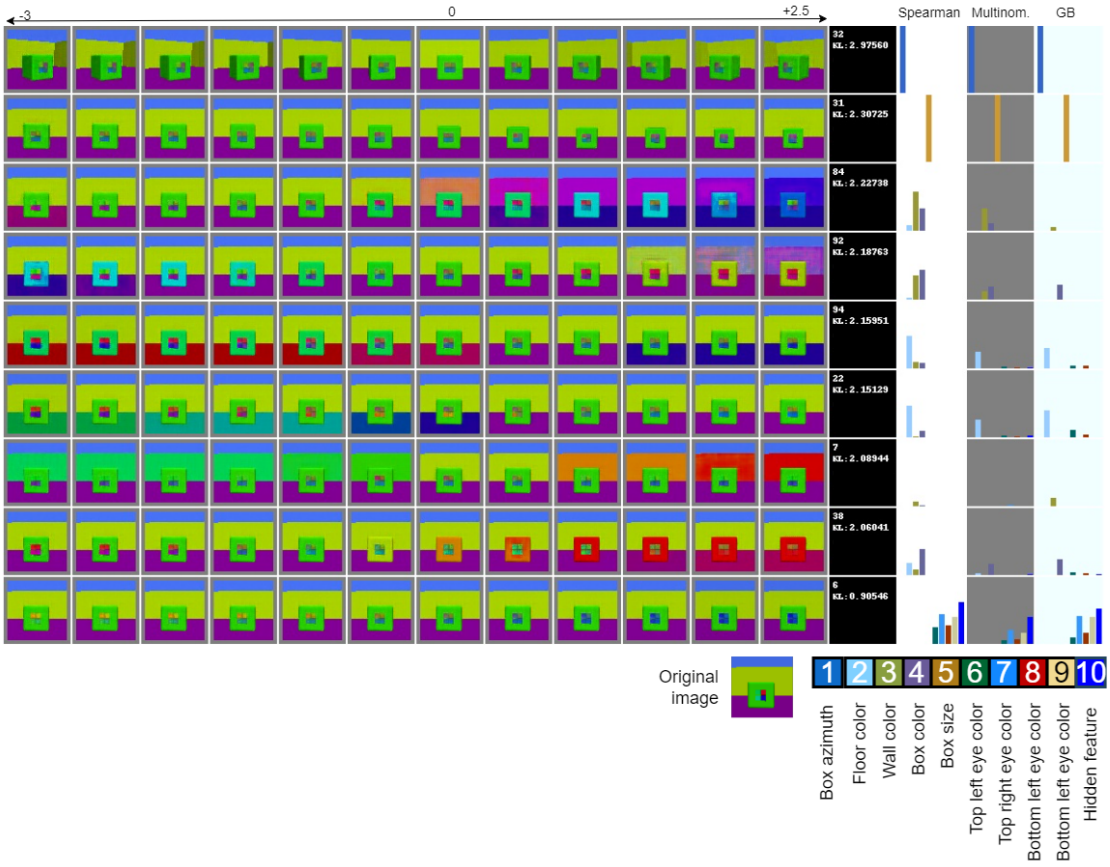

Figure 55. Latent variables layer L5, 5-layer architecture β=1.5.

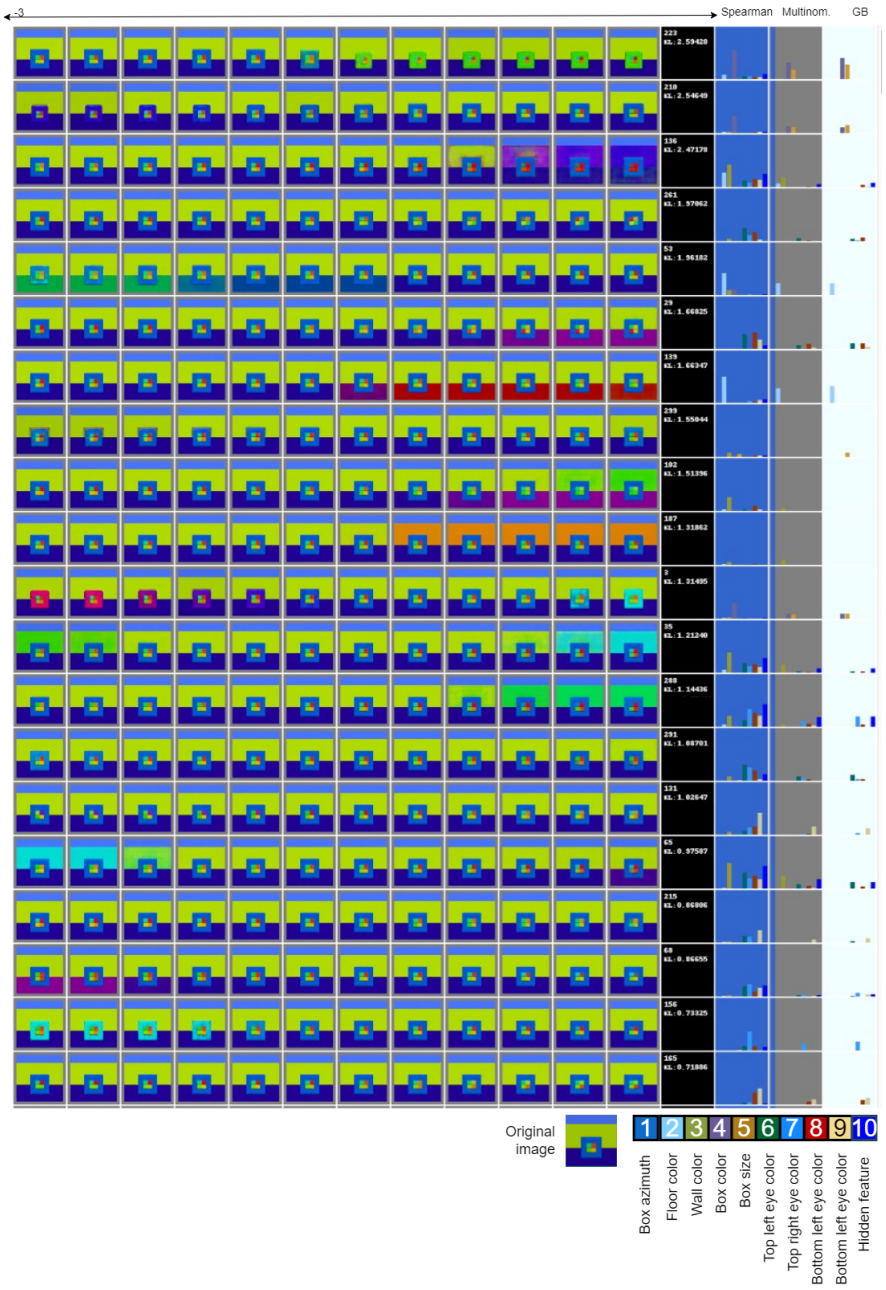

Figure 56. $\beta = 0.5$ β-VAE architecture latent variables single sample traversal images. The figure only shows latent variables where the KL loss value is over 0.1 nats. The model was trained on Boxheadsimple2 dataset, where samples had no azimuth changes.

V. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Kaarel Tark**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Disentanglement of features in variational autoencoders,
   (title of thesis)

   supervised by Meelis Kull.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kaarel Tark
17/05/2022