# Comparing the VAE and the $\beta$-VAE under various latent bottleneck dimensionalities and data complexities

Chan Young Song[1]

Engineering Faculty
Department of Computer Science
MSc Machine Learning

Supervisors:
Prof. Bradley Love
Dr Brett Roads

Submission date: September 2018

**Abstract**

Unsupervised representational learning with the property of disentangled representation is a highly desirable property due to its offer of generality and its potential for better domain transfer. This is currently an active field of research and recently, the $\beta$-Variational Autoencoder($\beta$-VAE) by Higgins et al. (2017) has been shown to outperform the standard Variational Autoencoder(VAE)(Kingma and Welling, 2013) in learning disentangled representations on data sets such as the dSprites(Matthey et al., 2017). However, we are still in the early stages of understanding exactly under what context the $\beta$-VAE outperforms the VAE. Currently, there is no study on comparing the two models with respect to the incremental changes in data complexity, and this is what we wish to study in this project. Another important aspect we need to consider in order to run the experiments for the study above is the choice of the dimensionality of the latent space bottleneck, especially relative to the number of generative factors. This is an important question as it could be that the relative performances between the two models may be on par if the dimensionality of the latent space bottleneck is similar to the number of the generative factors, as the lack of extra free dimensions in the latent space can already provide enough pressure on the models to learn efficient and disentangled representations. The data sets we use are sets of synthetically generated images similar to dSprites but with varying complexity. We measure the performances using the disentanglement metric proposed by Kim and Mnih (2018). What we conclude from the results is that firstly, in contrast to our hypothesis on the dimensionality of the latent space, we have the difference in performance between the $\beta$-VAE and the VAE is greatest when the dimensionality is equal to the number of the generative factors and that the differences converge as the dimensionality increases. Secondly, as expected, the disentanglement performances between the two models diverge as the complexity increases but converge again after a certain point.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

The success of machine learning algorithms depends on the accurate statistical representation of data. From tasks such as handwriting recognition(LeCun et al., 1998) to beating Atari games(Mnih et al., 2013), a key aspect of these learning algorithms is to compress high dimensional data into a more meaningful lower dimensional representation. Numerous deep learning applications have been successful in this endeavour(He et al. (2016), Szegedy et al. (2015), Gregor et al. (2015), Van Den Oord et al. (2016), van den Oord et al. (2016), Mnih et al. (2015), Jaderberg et al. (2016), Silver et al. (2016)). However, as pointed out by Burgess et al. (2018), the representations learnt by these supervised and reinforcement learning algorithms are extraordinarily streamlined and–unlike human intelligence–does not generalise to transfer tasks(Lake et al., 2017). This can be seen as a form of overfitting to the given task.

To improve generalising the learnt representations, many approaches have re-framed the learning problem in terms of auxiliary tasks(Jaderberg et al., 2016) and data augmentation(Tobin et al., 2017), but a more promising approach arguably lies with the field of *disentangled representations*. A completely disentangled representation has a one-to-one correspondence between the individual latent dimensions and the generative factors of the input space. One of the first unsupervised methods which showed this property is the Variational Autoencoder(VAE) (Kingma and Welling, 2013). There have been other notable methods such as InfoGAN(Chen et al., 2016), but in this paper, we focus on the VAE based method $\beta$-Variational Autoencoder($\beta$-VAE) (Higgins et al., 2017). the $\beta$-VAE has been shown to outperform the original VAE model in terms of disentanglement on various data sets. Intriguingly, $\beta$-VAE only adds a single scalar hyperparamter on top of the original VAE. Surprisingly little work has been published that systematically investigates the conditions where the $\beta$-VAE outperforms the VAE. This work aims to better understand the disentanglement benefits of the $\beta$-VAE.

The first factor we are interested in is the dimensionality of the latent space relative to the size of the generative factors. This is an important study since in any $\beta$-VAE implementation requires a choice of the dimensionality of the latent space. Depending on how similar the latent space dimensionality is compared to the number of the generative factors, $\beta$-VAE's relative performance to the VAE may vary significantly. Therefore, the first experiment is comparing the

disentanglement performances between the two models as the dimensionality of the latent space become increasingly larger than the number of the generative factors.

The second factor we wish to consider is the data complexity and how it affects the relative performances between the two models. On extremely simple data sets with only one generative factors, both the VAE and the $\beta$-VAE should be able to disentangle very well if not perfectly. Now if $\beta$-VAE outperforms VAE on data set such as dSprites as shown by Higgins et al. (2017), this means that the two models' performance significantly diverge when the data becomes complex enough. Since there is no study yet on how the data complexity affects the models' performance in disentangling, this study will be a useful contribution to this field.

## 1.2 Outline of the paper

In Chapter 2, we do a brief literature survey of basic autoencoders, the VAE, and the $\beta$-VAE. In Chapter 3, we describe the aims of the experiments, the data sets we use, the methods of comparison, and the neural network architectures. In Chapter 4, we present the results from our experiments which are further discussed in Chapter 5. Finally, we summarise our work in Chapter 6 with some suggestions for future work. The codes used for the project are available https://github.com/chanyoungs/ucl_project_2018.

# Chapter 2

# Model reviews

## 2.1 Basic autoencoders

Within unsupervised learning, the study of autoencoders is a very popular field. An autoencoder is a neural network that takes an input vector from the input space $\boldsymbol{X}$ and embeds it into the latent space $\boldsymbol{Z}$ using the mapping defined by the *encoder*. The latent vector is then mapped to the output space $\boldsymbol{X'}$ with the same dimension as $\boldsymbol{X}$ by the *decoder*. An autoencoder is trained by minimising the *reconstruction loss* between the original input and the final output. Since the 1980s(Yann (1987), Bourlard and Kamp (1988), Hinton and Zemel (1994)), it has recently been promoted by Hinton and Salakhutdinov (2006).

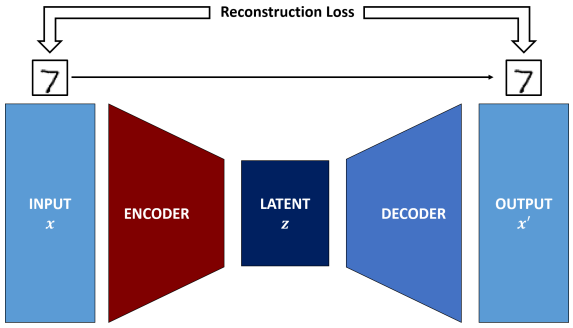



Figure 2.1: **Basic neural network structure of autoencoders**

There are many tasks autoencoders can achieve even in their basic form. The first is dimensionality reduction. If the dimension of the latent space is chosen to be less than that of the input space, then the autoencoder is forced to learn a compressed representation of the input space. In addition to the practical benefits, for example reducing the file size of images without losing too much quality, autoencoders perform a primitive form of feature learning.

With a slight modification, autoencoders can also be used for image denoising. Vincent et al. (2008) proposed a denoising autoencoder which was able to remove noise from an image. This was achieved by first purposefully adding random noise from a fixed distribution to the original image and feeding the noisy image into the autoencoder. Then the autoencoder was trained by

minimising the reconstruction loss between the final output image and the original (unaltered) image. As well as being able to remove noise from images, it was shown to be a more robust model and exhibited less overfitting than the original autoencoder.

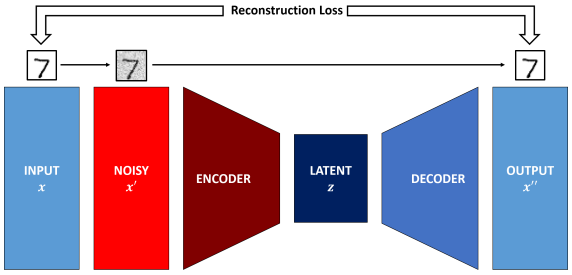



Figure 2.2: **Structure of a denoising autoencoder neural network**

Autoencoders have been extended in many ways, such as Stacked Denoising Autoencoder (Vincent et al., 2010), Contractive Autoencoder (Rifai et al., 2011), and k-Sparse Autoencoder (Makhzani and Frey, 2013). All of these approaches provided additional techniques for learning better performing models. However, basic autoencoders suffer from a fundamental limitation; they provided limited *interpretability* of the latent space. There are two levels of interpretability we will discuss in this project: continuity in the latent space, and disentangled representations. These concepts are discussed further in the following sections.

## 2.2 Variational Autoencoder(VAE)

### 2.2.1 Motivation

The VAE (Kingma and Welling, 2013) improves interpretability of the latent space by incorporating an additional loss term. As mentioned in the previous section, one limitation of the basic autoencoders is that their latent spaces have poor interpretability. For example, suppose we have a trained a basic autoencoder on a set of MNIST images. If we encode two different images of the digit '1' and get the corresponding latent vectors $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$, what would happen if we take the mean of the two latent vectors, call it $\boldsymbol{z_{avg}}$, and feed this vector into the decoder? In many of the basic autoencoders, it is likely to produce nonsensical output. This is because the autoencoder had not been encouraged to produce a continuous latent space in which the latent vectors in between the learnt latent vectors would interpolate in a meaningful way. Without this encouragement, the latent space is going to be sparse and composed of disjoint clusters that represent the training data. An important implication of this limitation is that it is very difficult to generate new images, let alone generating new images with specifiable features.

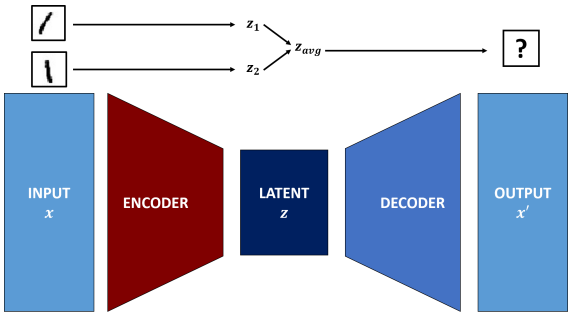



Figure 2.3: **Basic autoencoders have difficulty with interpolation**

On the other hand, VAEs produce a continuous latent space in which the interpolation between the latent vectors represent a much better interpolation between the original vectors. We will now discuss the theory behind the VAEs which makes this possible.

### 2.2.2 High level explanation of the VAE

VAE takes an input data point $\boldsymbol{x}$ and passes it through an encoder just like a basic autoencoder. However, instead of mapping the input data point as a latent data point, it maps it as a probability distribution $ENCODER(\boldsymbol{x}) \rightarrow q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})$, where $\boldsymbol{\theta_x}$ is the parameter vector defining the distribution $q$. The standard distribution for the VAE's latent space is the uncorrelated multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{\Sigma}$ is a diagonal covariance matrix. So $\boldsymbol{\theta_x} = \{\boldsymbol{\mu_x}, \boldsymbol{\Sigma_x}\}$ and since $\boldsymbol{\Sigma_x}$ is diagonal, the mean $\mu_{x_i}$ and variance $\sigma^2_{x_i}$ in each dimension are all the parameters required to describe the distribution. This is the key which makes the meaningful continuous latent space possible as the VAE's latent space is composed of continuously overlapping distributions. From this distribution $q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})$, a single latent vector is sampled at random $\boldsymbol{z'} \sim q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})$ which is then fed into the decoder to construct the output vector $DECODER(\boldsymbol{z'}) \rightarrow \boldsymbol{x'}$.

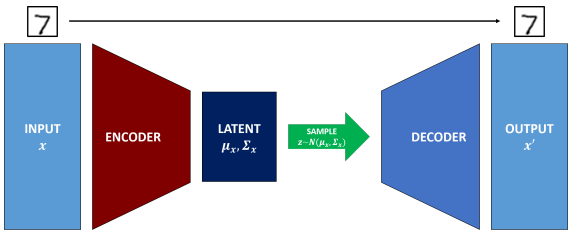



Figure 2.4: **VAE's architecture**

When training a VAE, the process is slightly more involved. Firstly, since the latent features are distributions as opposed to single points, we cannot calculate the reconstruction loss in the same way as with basic autoencoders. What should the model compare the original input to? One possible candidate is the output of the mean of the encoded distribution. However, this approach will not be able to take the variance of the distribution into account. Another candidate is the output of a randomly sampled point from the encoded distribution. Unfortunately, a single point is insufficient to convey the information of the whole distribution. What we need is to take a large

enough sample from the distribution and take the expectation of the reconstruction losses between the original input and the sampled outputs.

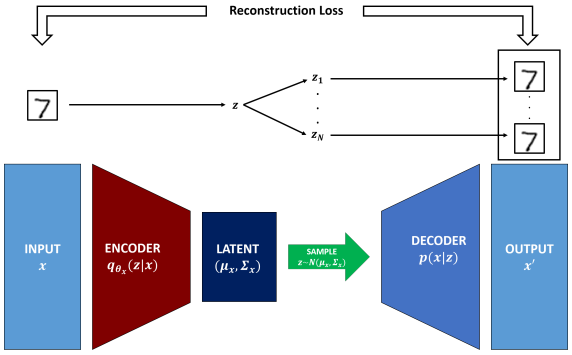



Figure 2.5: **VAE's reconstruction loss**

So far, what we have is a model which will learn to best represent the input space by modelling the latent space with Gaussian distributions. However, there is nothing stopping the model from using of all of the dimensions in the latent space and produce many disjoint Gaussians with very small variances. This issue is magnified if the dimension of the latent space is much larger than the number of features present in the input space. The consequence of this overfitting is that the model does not learn a continuous latent space. Instead, the model learns disjoint clusters which vary across the various dimensions. To combat this issue, the standard loss function is expanded to include a regulariser which penalises the model for using more dimensions that necessary as well as using Gaussians with small variance. The KullbackLeibler divergence (Kullback and Leibler, 1951)(KL divergence) is used to quantify the penalty. The exact mathematical definition and the derivation of the KL divergence's appearance in the loss function is discussed below, but for now, we will simply explain what effect the KL divergence has on the model. KL divergence is like a metric which measures the difference between two probability distributions. With this, we will measure the difference between $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_{in})$ and $p(\boldsymbol{z})$ where $p(\boldsymbol{z})$ is the standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Now if we add this KL divergence term to our loss, the model is pushed to keep the shape of $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}_{in})$ similar to a standard Gaussian, otherwise KL divergence will become large and incur a large loss. This has two main implications. First, VAE will try to keep as much latent dimensions to have standard Gaussian parameters which leads to using the least number of latent dimensions as possible. Secondly, VAE will be penalised when it tries to use small standard deviations to create small size clusters. This means that VAE is encouraged to cluster of a larger number of data points together, which results in a larger number of nearby data points being mapped to the same distribution. This will be explained in more detail in 2.3.2.

In summary, VAE is trained with the sum of the reconstruction loss, which is the expectation of the differences between the original input data and the samples of output data, together with the KL divergence between the latent uncorrelated Gaussian distribution and the standard Gaussian distribution:

$$\text{Loss}_{\text{VAE}} = \text{Reconstruction Loss} + \text{KL Divergence} \tag{2.1}$$

### 2.2.3 Probabilistic descriptions and mathematical derivations

We start with the observed space $\boldsymbol{X}$ composed of data points $\boldsymbol{x}$. We begin with an assumption that the each $\boldsymbol{x}$ are independently and identically distributed(iid) by some probability function of some corresponding latent variables $\boldsymbol{z}$ which are unobserved or hidden. Let $p(\boldsymbol{x}|\boldsymbol{z})$ be the probability of observing $\boldsymbol{x}$ given the latent $\boldsymbol{z}$, also known as the *likelihood* function. We also assume that each latent $\boldsymbol{z}$ are distributed with some function which we denote by $p(\boldsymbol{z})$, also known as the *prior*. The relationship can be illustrated with the following graphical model:

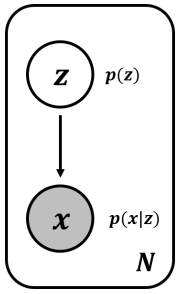



Figure 2.6: **The graphical model showing the relationship between $\boldsymbol{x}$ and $\boldsymbol{z}$**

Now given the prior and the likelihood, we can use the classical Bayes' theorem to express the *posterior*:

$$\begin{aligned} p(\boldsymbol{z}|\boldsymbol{x}) &= \frac{p(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{\int p(\boldsymbol{x},\boldsymbol{z})dz} \\ &= \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{\int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})dz} \end{aligned}$$

Unfortunately, the term which appears in the denominator of the posterior: $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})dz$, is intractable as the integral requires exponential computational time to calculate over all the possible values of the latents. Because of this problem of intractable posterior, VAE instead uses a family of tractable functions $q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})$ which best *approximates* the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$. Notice that the family of functions are parametrised by $\boldsymbol{\theta}_{\boldsymbol{x}}$ which in turn depends on the input $\boldsymbol{x}$. In the standard VAE, $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ are chosen to be Gaussians which means that $\boldsymbol{\theta}_{\boldsymbol{x}} = \{\boldsymbol{\mu}_{\boldsymbol{x}}, \boldsymbol{\Sigma}_{\boldsymbol{x}}\}$.

Now in order to make sure that $q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})$ closely approximates $p(\boldsymbol{z}|\boldsymbol{x})$, we can minimise:

$$KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right]$$

where the $KL\left[q(\boldsymbol{x})||p(\boldsymbol{x})\right]$ is the Kullback-Leibler(KL) divergence, measuring the difference be-

tween the two distributions, defined by:

$$KL\left[q(\boldsymbol{x})||p(\boldsymbol{x})\right] := \int_{\boldsymbol{x}\in\boldsymbol{X}} q(\boldsymbol{x}) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{x}$$

In other words, we wish to minimise:

$$KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right] = \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z}$$

The problem is not yet solved as the term still includes the intractable posterior $p(\boldsymbol{z}|\boldsymbol{x})$. To address this problem, we will minimise the KL divergence indirectly by using what's known as the *Evidence Lower BOund*($ELBO$):

$$ELBO(\boldsymbol{\theta}_{\boldsymbol{x}}) := \log p(\boldsymbol{x}) - KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right]$$

Keeping all other parameters fixed, we can see that maximising $ELBO$ with respect to $\boldsymbol{\theta}_{\boldsymbol{x}}$ minimises the KL divergence since the KL divergence is non-negative due to Jensen's inequality(Jensen, 1906). Doing some algebraic manipulation removes both intractable terms $p(\boldsymbol{x})$ and $p(\boldsymbol{z}|\boldsymbol{x})$ in the equation. We first begin by manipulating $\log p(\boldsymbol{x})$:

$$\begin{aligned}
\log p(\boldsymbol{x}) &= \log p(\boldsymbol{x}) \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z} && q \text{ is a probability density function} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{z} && \boldsymbol{z} \text{ does not appear in } \log p(\boldsymbol{x}) \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} && \text{Using } p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x},\boldsymbol{z})}{p(\boldsymbol{x})} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})} \frac{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} && \frac{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})} = 1 \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})} + q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} && \log \text{ property} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} + \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} && \text{Linearity of integrals} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} + KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right] && \text{Definition of } KL
\end{aligned}$$

Therefore, substituting the final form back into $ELBO$, we get:

$$
\begin{aligned}
ELBO(\boldsymbol{\theta_x}) &= \log p(\boldsymbol{x}) - KL\left[q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} + KL\left[q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right] - KL\left[q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log p(\boldsymbol{x}|\boldsymbol{z}) + q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log p(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z} + \int_{\boldsymbol{z}\in\boldsymbol{Z}} q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \mathbb{E}_{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + KL\left[q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]
\end{aligned}
$$

As claimed above, all terms in this expression are tractable and so we can use $ELBO$ as our loss function. We also notice that this is exactly what we had in 2.1:

$$
\boxed{\text{Loss}_{\text{VAE}} = \underbrace{\mathbb{E}_{q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{Reconstruction Loss}} + \underbrace{KL\left[q_{\boldsymbol{\theta_x}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]}_{\text{KL Divergence}}}
$$

### 2.2.4 The reparametrisation trick

There is one last difficulty that must be addressed in implementing the VAE as a neural network. In 2.4, one of the operation in the network is sampling from the latent distribution:

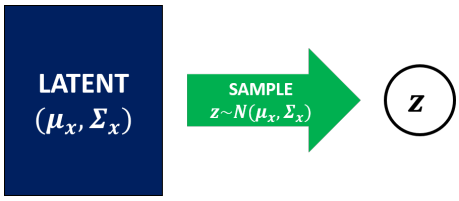



Figure 2.7: **We cannot back propagate through sampling**

Unfortunately, sampling is not an operation in which we can take the gradient for back propagation. To deal with this problem, Kingma and Welling (2013) takes advantage of the fact that the covariance $\boldsymbol{\Sigma_x}$ is a diagonal matrix and use a technique called *the reparametrisation trick* in which we arrive at the sampled latent $\boldsymbol{z}$ not directly from the latent Gaussian distribution, but indirectly by sampling a noise vector from a standard Gaussian distribution and then arriving at the latent $\boldsymbol{z}$ by scaling it by $\boldsymbol{\Sigma_x}$ and by translating it by $\boldsymbol{\mu_x}$. This has the same output as sampling directly from the latent distribution but now at least we can take the separate gradient through $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu_x}}$ and $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\Sigma_x}}$:

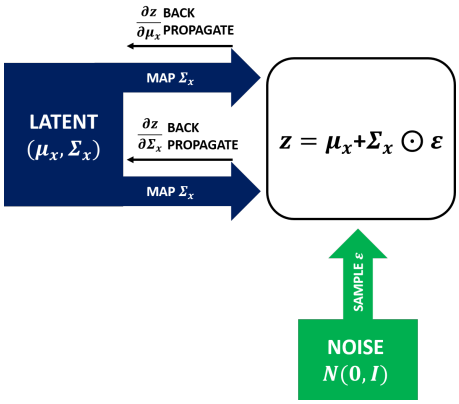



Figure 2.8: **The reparametrisation trick**

So the actual full architecture of VAE is the following.

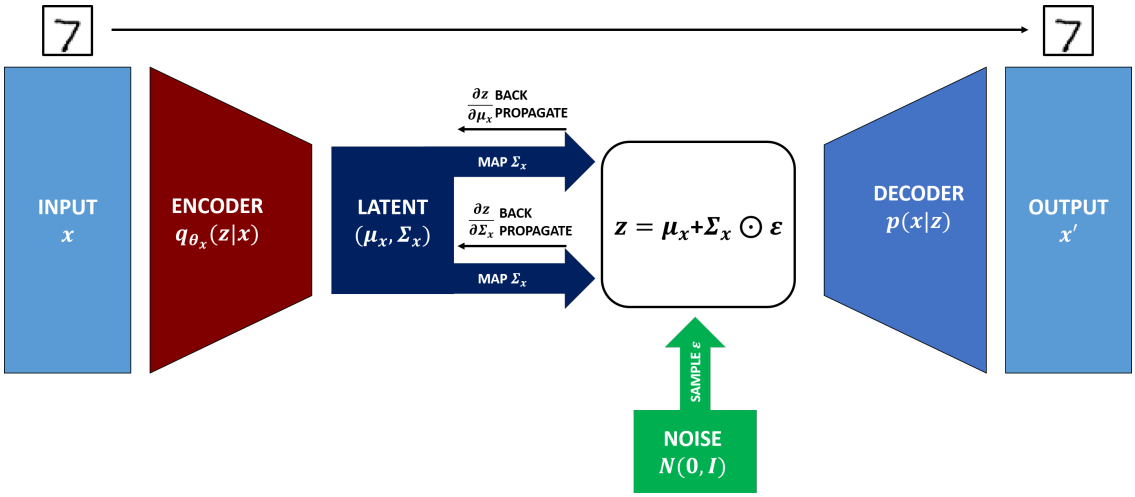
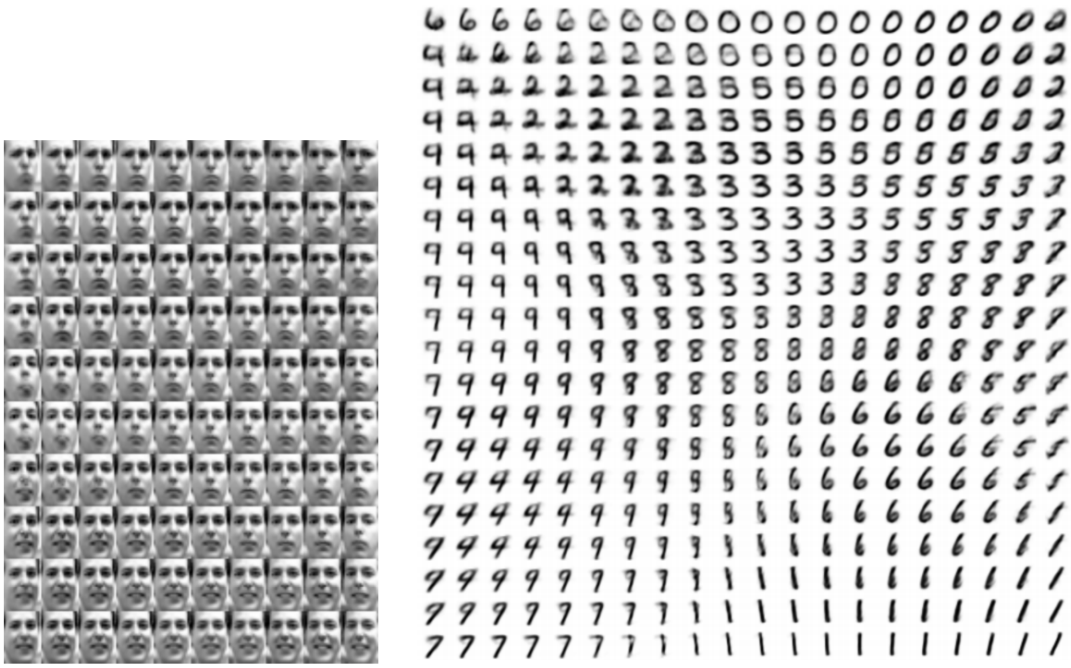



Figure 2.9: **The full VAE architecture**

### 2.2.5 Outputs

Kingma and Welling (2013) already showed that VAE produces latent spaces which satisfy our first condition of interpretability very well: it produces a continuous latent space in which locality in the input space is preserved in the latent space which allows for great interpolations between data points.



(a) Learned Frey Face manifold

(b) Learned MNIST manifold

Figure 2.10: **Figure by Kingma and Welling (2013) which shows the generated images from the latent values traversed uniformly across the 2D latent space.**

As we can see in the figure 2.10, the continously interpolated images make intuitive sense. If we consider the MNIST generated images, we even have interpolations between two different digits which seem almost plausible.

In fact, VAE also does well at producing latent spaces which satisfy the second condition of interpretability: disentangled representations. Bengio et al. (2013) defined the disentangled representation as that which has the property such that changes in a single latent dimension changes a single generative factor whilst leaving other generative factors mostly unchanged. Higgins et al. (2017) includes figures showing how much VAE's representations are disentangled compared to other models.

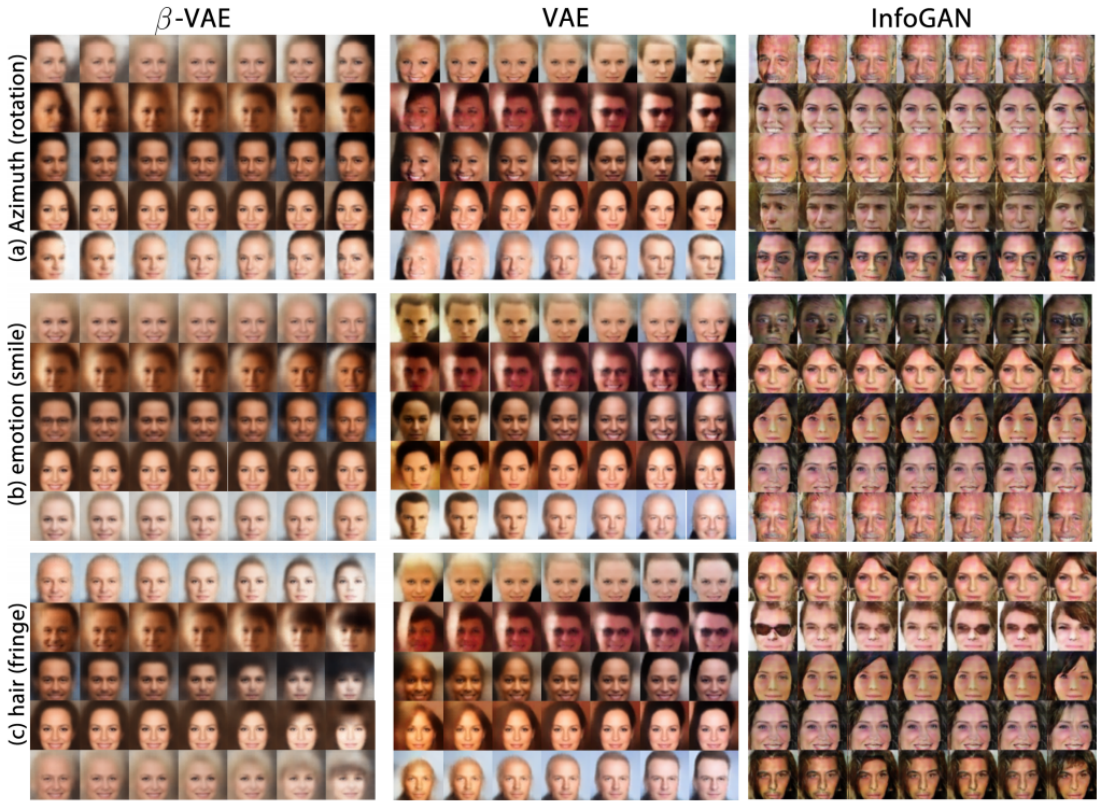



Figure 2.11: **Figures from Higgins et al. (2017). Qualitative comparison on disentanglement between $\beta$-VAE(Higgins et al., 2017), VAE(Kingma and Ba, 2014), and InfoGAN(Chen et al., 2016) on images of celebrity faces.**

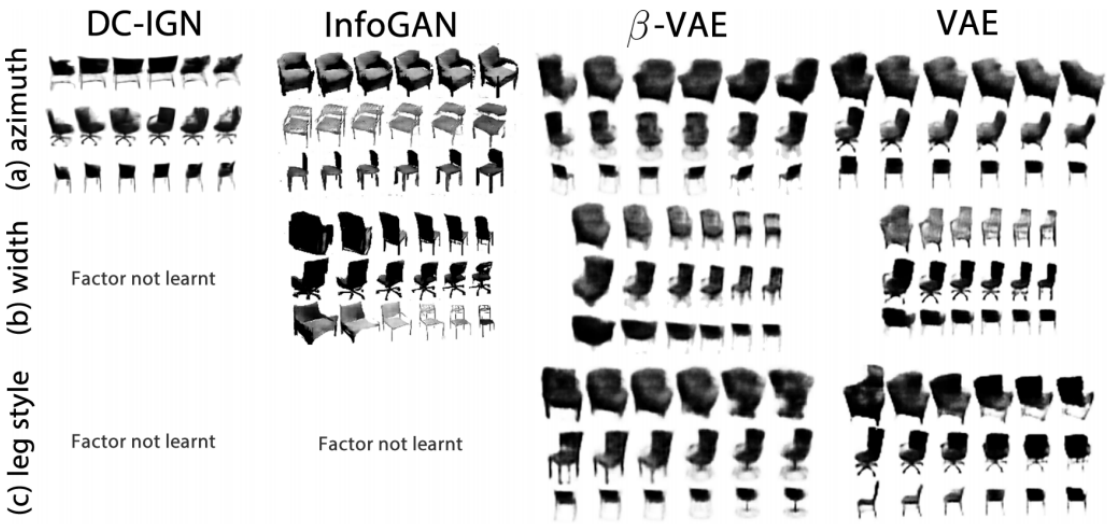



Figure 2.12: **Figures from Higgins et al. (2017). Qualitative comparison on disentanglement between DC-IGN(Kulkarni et al., 2015), InfoGAN(Chen et al., 2016), $\beta$-VAE(Higgins et al., 2017), and VAE(Kingma and Ba, 2014) on 3D images of chairs.**

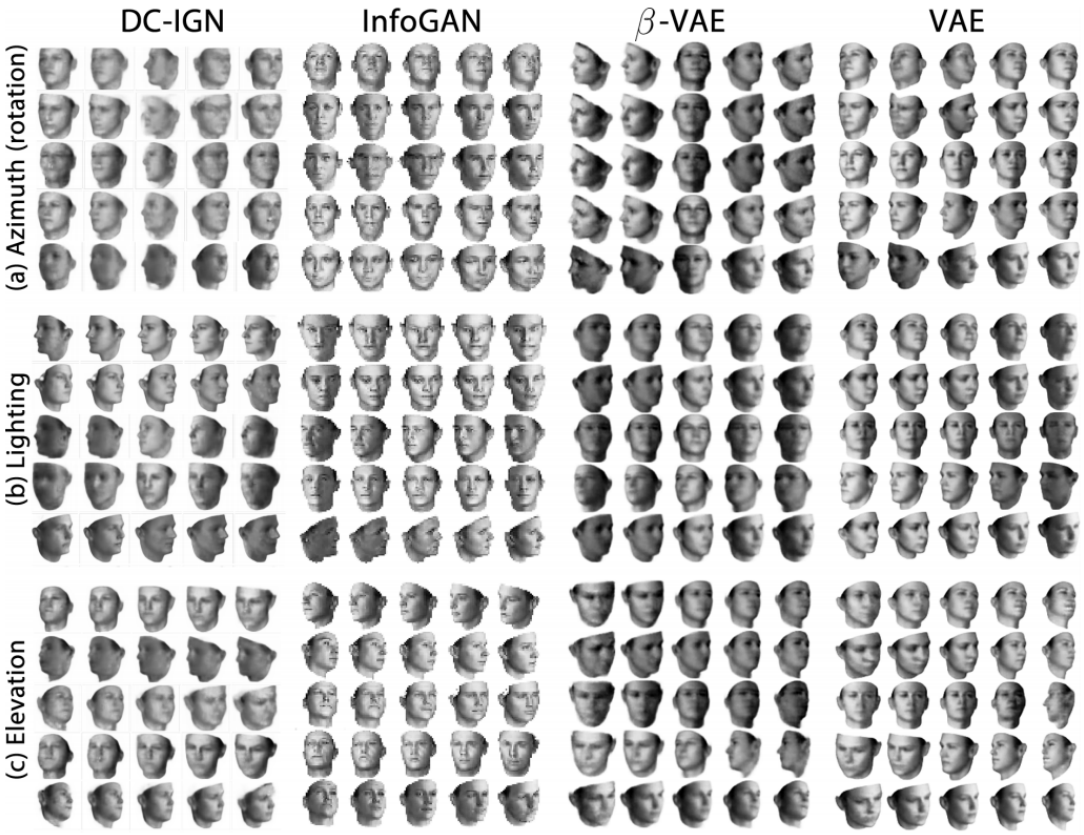



Figure 2.13: **Figures from Higgins et al. (2017). Qualitative comparison on disentanglement between DC-IGN(Kulkarni et al., 2015), InfoGAN(Chen et al., 2016), $\beta$-VAE(Higgins et al., 2017), and VAE(Kingma and Ba, 2014) on 3d images of faces.**

These figures show various input images acting as the seed and varying the value of a single latent dimension to show the change in a single generative factor. As we can see from the result images, VAE already does a good job at mapping some generative factors to single latent dimensions which is what is required for disentanglement. A more recent model called the $\beta$-VAE, expands on VAE and produces latent representations with better disentanglement.

## 2.3 Beta-Variational Autoencoder($\beta$-VAE)

### 2.3.1 Model description

The $\beta$-VAE (Higgins et al., 2017) introduces a small modification to the original VAE. The only difference is the introduction of a single scalar $\beta$ as a hyperparameter in the original loss function. Recall that the loss function used in the original VAE is:

$$\text{Loss}_{\text{VAE}} = \mathbb{E}_{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]$$

The loss function for the $\beta$-VAE is:

$$\text{Loss}_{\beta\text{-VAE}} = \mathbb{E}_{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + \beta KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]$$

By construction, $\beta = 1$ gives the standard VAE but Higgins et al. (2017) argues that the $\beta$-VAE, when $\beta > 1$ improves disentanglement.

### 2.3.2 Intuitive reasoning behind the $\beta$-VAE

Recall that we can interpret the original VAE's loss function as:

$$\text{Loss}_{\text{VAE}} = \text{Reconstruction Loss} + \text{KL Divergence}$$

Then the $\beta$-VAE's loss function can be interpreted as:

$$\text{Loss}_{\beta\text{-VAE}} = \text{Reconstruction Loss} + \beta\ \text{KL Divergence}$$

This makes the intent behind placing a scalar in front of the KL divergence clear enough. It is to be able to control the weight the KL divergence has on the total loss function. By increasing the value of $\beta$, the model will prioritise keeping all the latent distributions as close to the prior distribution (i.e., the standard Gaussian). By decreasing the value of $\beta$, the model will prioritise improving the reconstruction accuracy. To get the gist of the effect this priority has, it is enough to see one example from our results from the project below. The exact setup and the details will be explained in the following chapters, but for now, we wish to illustrate a basic comparison.

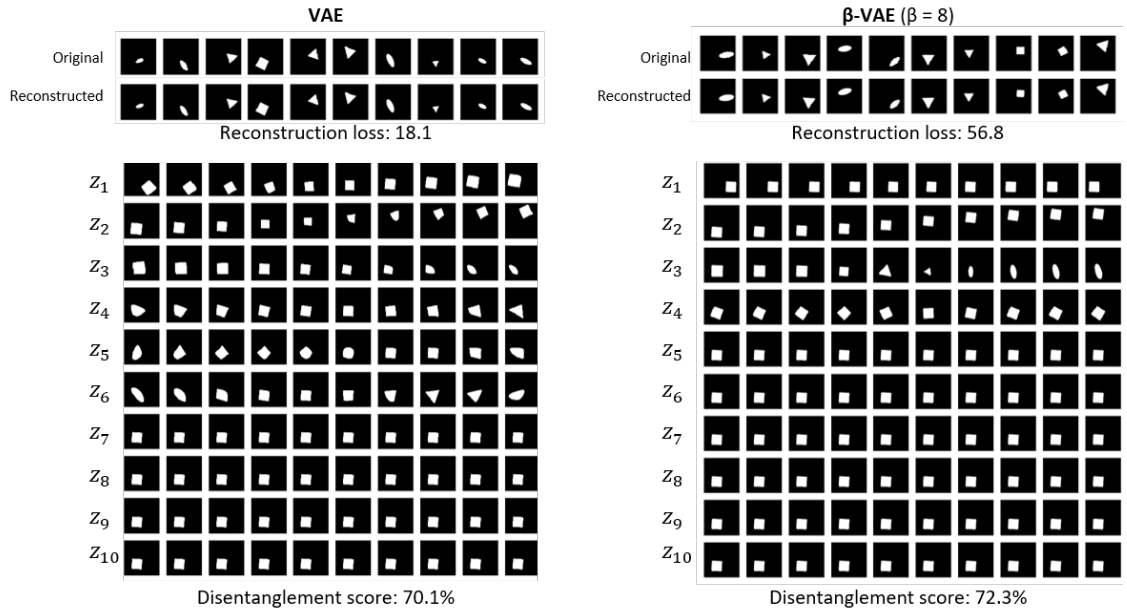



Figure 2.14: **Example of entangled representations vs disentangled representations.** Beginning with a seed of a single random input image which is encoded into the latent vectors, each row shows the generated images when we traverse through a single latent dimension from -3 to 3 standard deviations whilst keeping all others constant.

The above is an example of a clear illustration of the difference between a model which produced a relatively entangled representation of the input space and a model which produced a relatively disentangled representation of the input space.

**VAE**

| | x-pos. | y-pos. | Scale | Shape | Ori. |
|---|---|---|---|---|---|
| $z_1$ | ✓ | ✓ | ✓ | | ✓ |
| $z_2$ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $z_3$ | | | ✓ | ✓ | |
| $z_4$ | | | | ✓ | ✓ |
| $z_5$ | | | | ✓ | ✓ |
| $z_6$ | | | | ✓ | |
| $z_7$ | | | | | |
| $z_8$ | | | | | |
| $z_9$ | | | | | |
| $z_{10}$ | | | | | |

**$\beta$-VAE**

| | x-pos. | y-pos. | Scale | Shape | Ori. |
|---|---|---|---|---|---|
| $z_1$ | ✓ | | | | |
| $z_2$ | | ✓ | | | |
| $z_3$ | | | ✓ | ✓ | |
| $z_4$ | | | | | ✓ |
| $z_5$ | | | | | |
| $z_6$ | | | | | |
| $z_7$ | | | | | |
| $z_8$ | | | | | |
| $z_9$ | | | | | |
| $z_{10}$ | | | | | |

We can clearly see that the standard VAE is representing more than one feature per dimension and that features are represented across many dimensions. It is a many to many mapping which result in entangled representations. On the other hand, the $\beta$-VAE's($\beta = 8$) mapping is nearly one-to-one which results in more disentangled representations.

In some situations, the standard KL divergence may not provide a strong enough incentive to encode the latent space in the most information efficient way, instead focusing on reconstruction loss. This was exactly the case with the example above since the standard VAE produced better reconstructions than that of $\beta$-VAE. Indeed, most of the time, the value of the reconstruction loss is much bigger in magnitude than the value of the KL divergence. Therefore, it intuitively makes sense to adjust the weight that the KL divergence has on the total loss with the hope that a stronger weight on the KL divergence will produce a latent space which has better disentanglement.

However, there is yet to be a formal argument as to why this should improve disentanglement. Burgess et al. (2018)) offers an intuitive explanation based on the argument that the $\beta$-VAE loss function is closely related to the *information bottleneck principle* (Tishby et al. (2000), Achille and Soatto (2018), Alemi et al. (2016), Chechik et al. (2005)):

$$\max[I(Z;Y) - \beta I(X;Z)]$$

where $I(\cdot;\cdot)$ stands for mutual information and $\beta$ is a Lagrange multiplier. Optimising the above objective requires maximising the mutual information shared between the latent bottleneck $Z$ and the given task $Y$ whilst at the same time trying to remove all information $Y$ shared by $X$. Burgess et al. (2018) explains that we can see the posterior $q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})$ as the information bottleneck $Z$, and the reconstruction objective $\mathbb{E}_{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right]$ as the task $Y$ which encourages the latent distribution $q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})$ to *efficiently* represent the input $\boldsymbol{x}$ by jointly minimising the reconstruction loss and the $\beta$-weighted KL divergence. In the information theoretic perspective, the KL divergence can be seen as an upper bound on the amount of information that can be transmitted through the latent channels per data sample.

Burgess et al. (2018) continues to explain that the KL divergence encourages the embedding of the input space to have the property that points close in the input space are mapped to points close in the latent space. This is because to minimise the KL divergence between the posterior uncorrelated multivariate Gaussian and the prior standard multivariate Gaussian, the posterior Gaussian needs to do two things. First, the posterior mean values need to stay close to zero against the model's tendency to freely vary the mean values to best represent data. Secondly, the posterior standard deviations need to stay close to one against the model's tendency to have vary small standard deviations to partition the input space into as many small parts as possible which will decrease the reconstruction loss at the cost of losing the continuity in the latent space.

Now bearing in mind these two properties, consider the diagrams below. Each Gaussian distribution below represents the posterior distribution with respect to the input shown above as an example. We will illustrate the important role the KL divergence has and furthermore, the significance the *weight* of the KL divergence will have in the training.

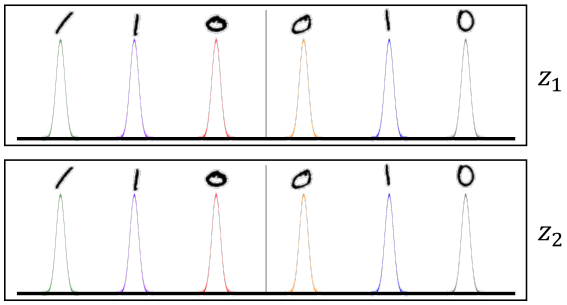



Figure 2.15: **Posterior distributions with no KL divergence.**

Figure 2.15 shows the kind of distributions we should expect to see if the final objective does not include the KL divergence. The reconstruction loss is maximised if the model "memorises" the input space by mapping distinct latent distribution for each input data point. As discussed in the previous section, this is undesirable outcome as the model will not produce an interpretable latent space.

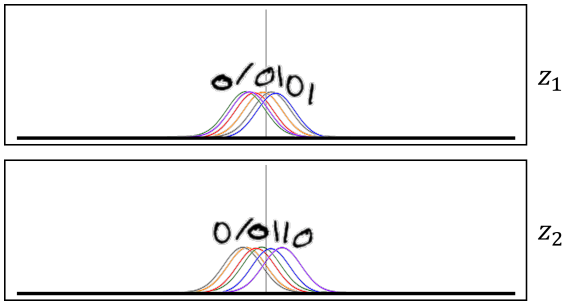



Figure 2.16: **Posterior distributions with too much weight on the KL divergence.**

Figure 2.16 shows the kind of distributions we should expect to see if the final objective includes too much weight on the KL divergence. Since the KL divergence encourages all the posterior distributions to have zero mean and 1 standard deviation, if we give too much weight

on the KL divergence, all the distributions will overlap as the standard Gaussian and will give a blurry output as an average between all the input points which is also a problem.

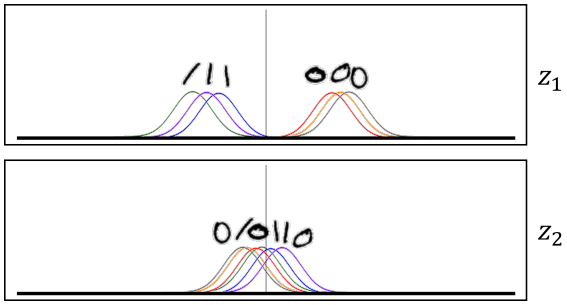



Figure 2.17: **Posterior distributions with a relatively large weight on the KL divergence.**

Figure 2.17 shows the kind of distributions we should expect to see if the final objective includes a relatively large weight on the KL divergence. The first latent dimension $z_1$ has learnt to discriminate between the digits. It was able to do this because the KL divergence encourages a large standard deviation and the mean values to be as close to 1 as possible. The model has learned to compromise between the reconstruction loss and the KL divergence by mapping similar input points near each other so that even if the model mistakenly samples a latent value from the "wrong" overlapping distribution, the output will not deviate much from the actual input. However, the large weight on the KL divergence means that the reconstruction gain from learning any other features through the second dimension was not worth the increase in the KL divergence by transforming the standard Gaussians. This model is much better than the previous two extremes but ideally, we would like to see it learn other minor but significant features such as slant.

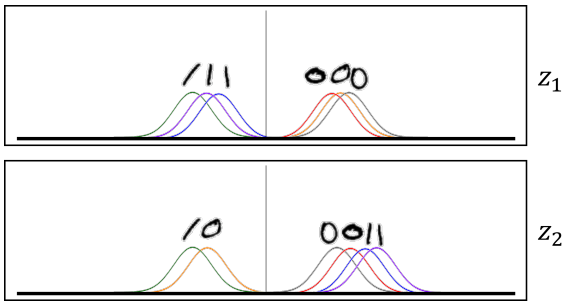



Figure 2.18: **Posterior distributions with the optimal weight on the KL divergence.**

Figure 2.18 shows the kind of distributions we should expect to see if the final objective includes an optimum weight on the KL divergence. The first latent dimension $z_1$ has learnt to discriminate between the digits as with the figure before. Now with a slightly smaller weight on the KL divergence, the second dimension had just the right amount of capacity to learn another feature(the slant in this case) at the cost of increasing the KL divergence but with the gain of less reconstruction loss. This is the ideal model which is the right balance between the reconstruction loss(how accurately we can retrieve the original input) and the KL divergence(interpretability).

In this perspective, it makes sense to put a hyperparameter $\beta$ to give a weight on the KL divergence for the total function. There is no reason to think that $\beta = 1$ as in the case of the standard VAE will always be the optimal balance between the reconstruction and the KL divergence.

# Chapter 3

# Methodology

## 3.1 Aims

Extending the experiments by Higgins et al. (2017), we aim to make further comparisons between the $\beta$-VAE and VAE by focusing on two variables. The first experiment tests how model performance depends on the size of the encoder bottleneck with the varying $\beta$ values. The second experiment tests how how model performance depends on the data complexity, by varying the number of generative factors, with the varying $\beta$ values.

The focus of the first experiment is on the relationship between the performance, latent bottleneck size and the $\beta$ values. This is an interesting question because it may be possible that when the dimension of the latent space(the bottleneck) is very close to the original number of generative factors, the role of KL divergence becomes less important. In another words, the role of the weight of the KL divergence may become more important as the latent space becomes significantly larger than that of the number of generative factors. If this hypothesis is the true, this has an important practical implication. In most real scenarios, we will not have access to the original number of generative factors of the data set. For example, how many generative factors are there for the ImageNet(Deng et al., 2009) dataset? Therefore, in most scenarios, we will have to make a safe overestimate of the number of learnable factors and set the number of latent dimensions to be generously large. Therefore, if the hypothesis is true, then in most real scenarios, the choice of $\beta$ will be important factor to consider. In order to test this, we will fix a data set with a known number of generative factors $n$. Then, starting with the bottleneck with size $n$, we will compare the two models' performances as we increase the size of the bottleneck.

The second experiment compares model performance when we vary the complexity of the input data by increasing the number of generative factors. Higgins et al. (2017) already showed some promising results that the $\beta$-VAE outperforms the standard VAE on certain data sets(celebA, 3D chairs, 3D faces, dSprites). However, it seems likely that the two models would perform similarly on extremely simple data sets, for example, if the data set only had a single position variable as the generative factor. The question then is at what level of complexity, and at what rate, do the two models' performances diverge? For this test, we will compare the model performances as we vary the complexity of the data set, by increasing the number of generative factors.

## 3.2 Data

The structure of the data set will be based on dSprites by Matthey et al. (2017) with small differences. Firstly, dSprites has an image for every single combination of each generative factors, each of which is from a fixed finite set:

- Colour: white
- Shape: square, ellipse, heart
- Scale: 6 values linearly spaced in [0.5, 1]
- Orientation: 40 values in $[0, 2\pi]$
- Position X: 32 values in [0, 1]
- Position Y: 32 values in [0, 1]

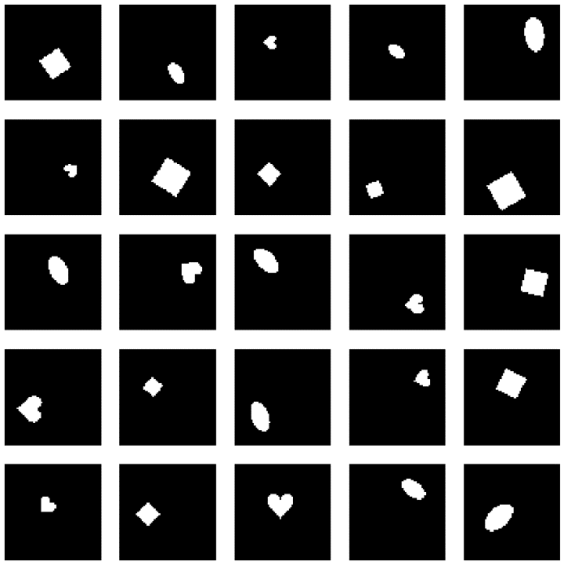


Figure 3.1: **dSprites images from Matthey et al. (2017)**

This means that there are $1 \times 3 \times 6 \times 40 \times 32 \times 32 = 737280 (\sim 1$ million) images. We plan to use data sets with more generative factors but then the file size becomes impractically large. So instead of having an image for each possible combination of all the factors, we decided to have a fixed number of image set size of one million and uniformly sample each factor for each image. As well as the practicality of the file size, another advantage of this approach is that the images could have the factor values from a continuous range instead of a small predefined finite set as with dSprites. Also, even though the data set will not have an image for every single combination of the factors, this should not be too much of a problem as the generative factors are all mostly uncorrelated factors and the sample size of a million should be large enough for the model to see enough examples for each partial combination of factors. These are the factors in order that will cumulatively be added on for the increasingly complex data sets:

1. Position $X \sim U[0, 1]$

2. Position $Y \sim U[0, 1]$

3. Scale $S \sim U[0, 1]$

4. Shape: square[cube], ellipse[ellipsoid], triangle[pyramid]

5. Rotation1: $Rotation_X \sim U[0, 2\pi]$

6. Colour: $HSV(h, s = 1, v = 1)$ where $h \sim U[0, 1]$

7. Rotation2: $Rotation_Y \sim U[0, 2\pi]$ (3D)

Note there is another minor difference of using a triangle instead of a heart as in dSprites. Also, it was a deliberate choice to choose a one dimensional colour scale by varying only the hue in HSV colour scheme:

Figure 3.2: **1 dimensional colour scale based on** $HSV(h, s = 1, v = 1)$ **where** $h \in [0, 1]$

The reason for this choice, instead of going for the standard colour schemes such as the 3 dimensional RGB system, is that there is more than one way to represent the multidimensional colour which may lead to a model learning a perfectly valid form of a disentangled representation of colour but may not correspond to the exact system used in generating the images. This could cause problem in comparing disentanglement between models. Similarly, only two dimensions of rotation are used instead of three dimensions for the 3D data sets since there is more than one way to produce the same orientation if we use three dimensions. The two dimension of rotations should be independent enough for good comparisons.

## 3.3 Comparison criteria

There are two criteria we can use to compare the models. The first is the qualitative inspection of the reconstructed images in the latent space through the method of latent traversals as we already saw in figure 2.14. The second is the quantitative disentanglement metric which aims to measure how well the model encodes the independent features using separate latent dimensions. The quantitative comparison is what we will rely on mainsy in this paper but the reconstructed images will be included in the appendix. We now discuss the trade-offs of two different disentanglement metrics proposed by Higgins et al. (2017) and Kim and Mnih (2018).

### 3.3.1 Disentanglement metric (Higgins et al., 2017)

The first proposal of a disentanglement metric was made by Higgins et al. (2017). In essence, a simple classifier is used to measure the one to one correspondence between the latent dimensions and the ground truth generative factors. Here are the steps from Higgins et al. (2017):

1. Randomly choose a generative factor $k \in [1, ..., K]$ where $K$ is the size of the latent space dimension.
2. For a batch of $l = 1, ..., L$ samples:
   (a) Sample two sets of latent vectors $\boldsymbol{v}_{1,l}$ and $\boldsymbol{v}_{2,l}$ making sure that the value of the latents in the $k^{th}$ coordinates are equal.
   (b) For each $n = 1, 2$, generate images $\boldsymbol{x}_{n,l}$ using the decoder of the model. Then infer $\boldsymbol{z}_{n,l} = \boldsymbol{\mu}_{\boldsymbol{x}_{n,l}}$ from the posterior distribution $N(\boldsymbol{\mu}_{\boldsymbol{x}_{n,l}}, \boldsymbol{\Sigma}_{\boldsymbol{x}_{n,l}})$ from the model's encoder.
   (c) Compute the absolute difference $\boldsymbol{z}^l_{diff} = |\boldsymbol{z}_{1,l} - \boldsymbol{z}_{2,l}|$
3. Use the average $\boldsymbol{z}_{avg} = \frac{1}{L}\sum_{l=1}^{L} \boldsymbol{z}^l_{diff}$ to predict $p(y|\boldsymbol{z}_{avg})$.

In the third step, Higgins et al. (2017) used a linear classifier with low VapnikChervonenkis dimension, i.e. low expressive power, to ensure that the classifier cannot do any complex non-linear disentangling itself. The diagram below illustrates these steps.

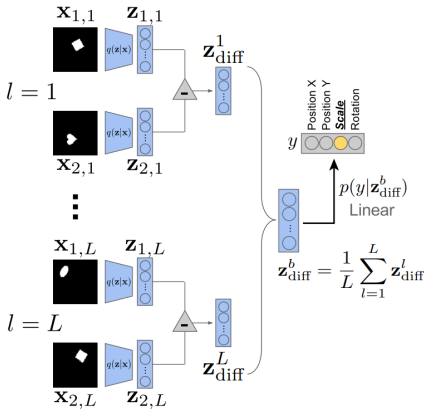



Figure 3.3: **Figure from Higgins et al. (2017). Schematic of the disentanglement metric by Higgins et al. (2017)**

The proposed metric serves its purpose because the model will need to produce a well disentangled latent space which makes a good one-to-one correspondence with the generative factors so that even a simple linear classifier can make high accuracy predictions as to which generative factor was held constant in the batch of samples. However, there are some limitations associated with this approach, as pointed out by Kim and Mnih (2018). Firstly, the metric could be sensitive to the choice of the linear classifier used, the hyperparameters, the weight initialisations, and the optimiser used for the classifier. Secondly, even if it's true that the linear classifier cannot make complex non-linear predictions, even a linear combination of the latent dimensions may in some cases be strong enough to do its own disentanglement which defeats the purpose of measuring disentanglement. Thirdly, this metric is computationally expensive since each new measurement of disentanglment requires training a new classifier. This is not ideal especially if one needs to calculate the metric thousands of times. Lastly, there are scenarios in which the metric gives a misleading score. Using the example by Kim and Mnih (2018), suppose we have a data set with 4 generative factors: x-position, y-position, scale and shape. Suppose the model has perfectly learned disentangled representations of the first three of the generative factors but has not learned shape. In this scenario, the linear classifier undesirably gives a disentangled metric score of 100% as the linear classifier can "correctly" predict that when the other latents' values corresponding to the first three generative factors are non-zero, it must be the last unlearnt factor which was kept constant.

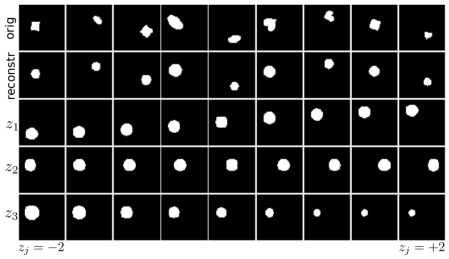



Figure 3.4: **Figure from Kim and Mnih (2018). A setting in which the metric by Higgins et al. (2017) incorrectly gives a 100% score without perfect disentanglement.**

### 3.3.2 Disentanglement metric (Kim and Mnih, 2018)

To address these limitations, Kim and Mnih (2018) propose a new metric. However, there is one key extra assumption that is required for this metric to work, which is that we need the full access to the ground truth image generator with which we can fully control of the generative features when generating the images. Of course, this is an unhelpful assumption for real life data sets but it can be a useful for experimenting with synthetic data sets as with this project.

1. Randomly choose a generative factor $k \in [1, ..., K]$ where $K$ is the size of the latent space dimension.
2. For a batch of $l = 1, ..., L$ samples:
   (a) Generate images $\boldsymbol{x}_l$ with the $k^{th}$ generative factor fixed.
   (b) Infer $\boldsymbol{z}_l = \boldsymbol{\mu}_{\boldsymbol{x}_l}$ from the posterior distribution $N(\boldsymbol{\mu}_{\boldsymbol{x}_l}, \boldsymbol{\Sigma}_{\boldsymbol{x}_l})$ from the model's encoder.
   (c) Normalise each dimension by its empirical standard deviation over the full data(or a large enough subset).
3. The index $d \in [1, ..., D]$ of the normalised latent dimensions with the lowest variance, together with $k$, provide one training $(d, k)$ = (input, output) example for the majority vote classifier. The accuracy of the classifier is the final disentanglement score.

The normalisation at step 2.c) is to ensure that the minimum index is invariant of the rescaling of of the representations in each dimension. The majority vote classifier $C$ used in the third step is defined as follows:

- Begin with a training set $(d_m, k_m)$ for $min[1, ..., M]$, where $M$ is the fixed size of the training set, $d = 1, ..., D$ where $D$ is the number of the latent dimensions, and $K$ is the number of the generative factors.
- Then for each $d \in [1, ..., D]$ and $k \in [1, ..., K]$, calculate $V_{d,k} = \sigma_{m=1}^{M} \mathbb{I}(d_m = d, k_m = k)$
- Finally, our classifier $C$ is defined by the equation: $C(d) = \text{argmin}_k V_{d,k}$.

Note that $D$ does not affect the metric because, for example, if the classifier chose at random, the accuracy would be $\frac{1}{K}$ independent of $D$.

Kim and Mnih (2018) explains that this approach addresses the problems faced by the metric by Higgins et al. (2017). Firstly, the majority vote classifier used for this metric is completely deterministic without the need for a choice of hyperparameters etc. Secondly, since the classifier is a simple majority vote, there is no room for any further disentanglement in calculating the score, even linear combinations between the latent dimensions. Thirdly, the classifier only needs to do simple counts to train, which significantly reduces the computation time. Finally, since the classifier needs to see the lowest variance in a latent dimension for a given factor to classify correctly, it avoids making misleading scores unlike the previous metric.

## 3.4 Model architectures

Below is the full model architectures used for this project. The implementation is based on Miyoshi (2017) and the actual codes used for the project are available https://github.com/chanyoungs/ucl_project_2018.

- Neural Network

**Encoder**

| **Input** $\boldsymbol{x}_{in}$: $64 \times 64 \times C$(Channels) | |
|---|---|
| $4 \times 4$ conv. 32 ReLU. stride 2 | |
| $4 \times 4$ conv. 32 ReLU. stride 2 | |
| $4 \times 4$ conv. 32 ReLU. stride 2 | |
| $4 \times 4$ conv. 32 ReLU. stride 2 | |
| FC. 256 ReLU | |
| FC. 256 ReLU | |
| $\boldsymbol{\mu}$: FC. $D$(Latent size) | $\boldsymbol{\Sigma}$: FC. $D$(Latent size) |

**Sampler**

| **Sample** $\boldsymbol{z}$ : $\boldsymbol{\mu} + \boldsymbol{\Sigma} \odot \boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim N(\boldsymbol{0}, \boldsymbol{I})$ |
|---|

**Decoder**

| **Input**: $D$ |
|---|
| FC. 256 ReLU |
| FC. 512 ReLU |
| $4 \times 4$ deconv. 32 ReLU. stride 2 |
| $4 \times 4$ deconv. 32 ReLU. stride 2 |
| $4 \times 4$ deconv. 32 ReLU. stride 2 |
| $4 \times 4$ deconv. 32 ReLU. stride 2 |
| **Output** $\boldsymbol{x}_{out}$: $64 \times 64 \times C$(Channels) |

- Number of input channels $C$: 1(BW) or 3(RGB)
- Dimensionality of the latent space $D$:

  - Experiment 1:
    For the first experiment as defined in section 3.1, we will vary $D$ on the fixed data set with the first 5 generative factors as defined in section 3.2. Since we have 5 generative factors, we will vary $D$ starting with 5 up to 20 in increments of 5.
  - Experiment 2:
    For the second experiment, we will keep $D = 20$ fixed which is safely larger than the maximum number of generative factors used in this project which is 7.

- Batch size: 64(fixed)
- Learning rate: 0.0005(fixed)
- Optimiser: Adam(fixed)
- Loss function: $\mathbb{E}_{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + \beta KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right]$(fixed)
- Hyperparameter $\beta$: 1, 2, 4, 8
- Max epochs:
  - 1 & 2 generative factors: 10 epochs
  - 3 generative factors: 20 epochs
  - 4 & 5 generative factors: 30 epochs
  - 6 & 7 generative factors: 60 epochs

# Chapter 4

# Results

## 4.1 Effect of the latent bottleneck dimension

### 4.1.1 Original hypotheses

There were two original hypotheses with regards to the first experiment. Firstly, we expected that there would be close to no effect the hyperparameter $\beta$ would make on improving disentanglement if the latent size, $D$ was equal to the number of generative factors, $K$. The reason for this was that since $D = K$, there is already the right amount of pressure for the model to efficiently learn to represent the generative factors into $K$ dimensions which should encourage the model to learn the disentangled factors. If this hypothesis is true, we should see the disentanglement scores across the models with varying $\beta$ values to be very similar when $D = K$.

Secondly, related to the first, we expected to see that the effect the hyperparameter $\beta$ would make on improving disentanglement would increase as $D$ becomes increasingly bigger than $K$. The reason for this was that as $D$ becomes larger than $K$, the model has more unnecessary extra degrees of freedom to represent the input space, and so we expected it to be necessary to put more pressure onto the model to represent the input space into less dimensions than was available. If this hypothesis is true, we should see the disentanglement scores across the model with varying $\beta$ values to vary more and more as $D$ becomes larger and larger relative to $K$.

### 4.1.2 Results

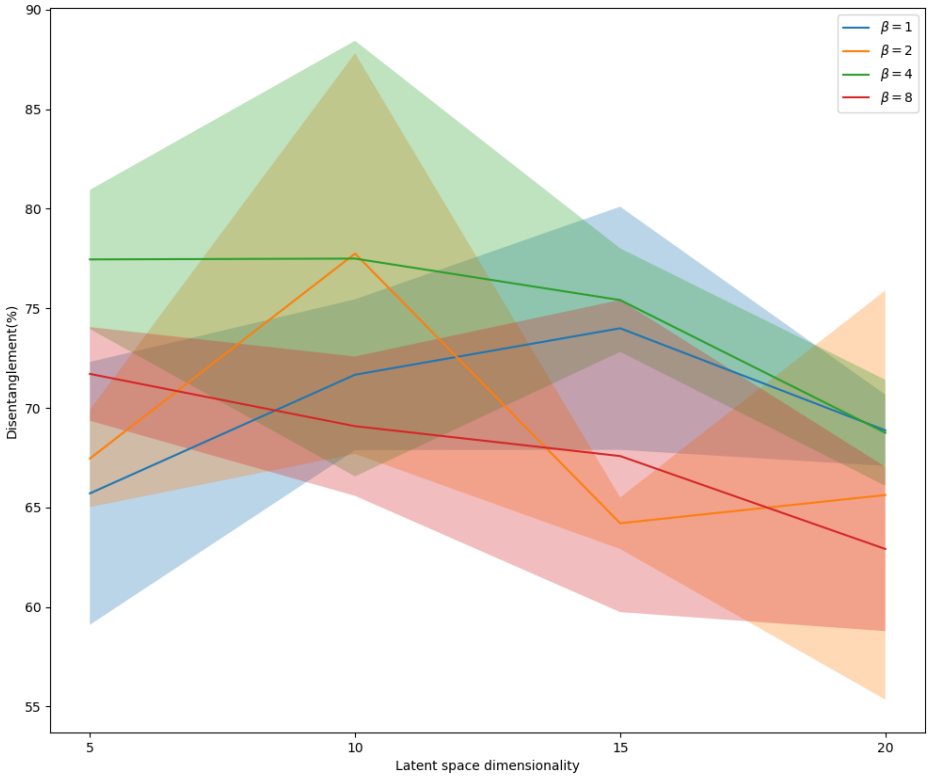



Figure 4.1: **Disentanglement scores on the final epoch over the varying latent sizes and $\beta$ values.** Coloured line graphs representing different $\beta$ values as shown in the legend. The x-values of the lines are the dimensionality of the latent space and the y-values of the lines represent the mean disentanglement scores across three runs. The shaded region around the solid lines represent the standard deviation of the disentanglement scores across the three runs.

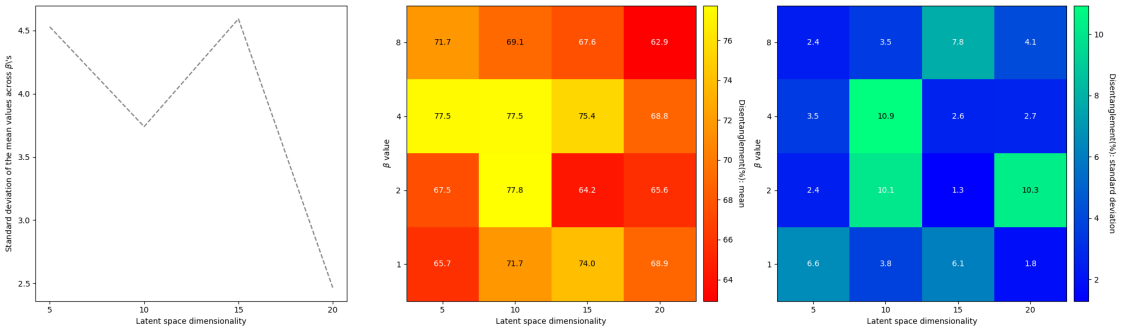



Figure 4.2: **Disentanglement scores on the final epoch over the varying latent sizes and $\beta$ values.** Left: line graph with x-values being the dimensionality of the latent space and y-values being the standard deviations of the mean disentanglement values across the models. Centre: heatmap of the mean disentanglement scores across the varying degrees of the latent space and $\beta$ values. Right: same as the bottom left figure except with the standard deviation.

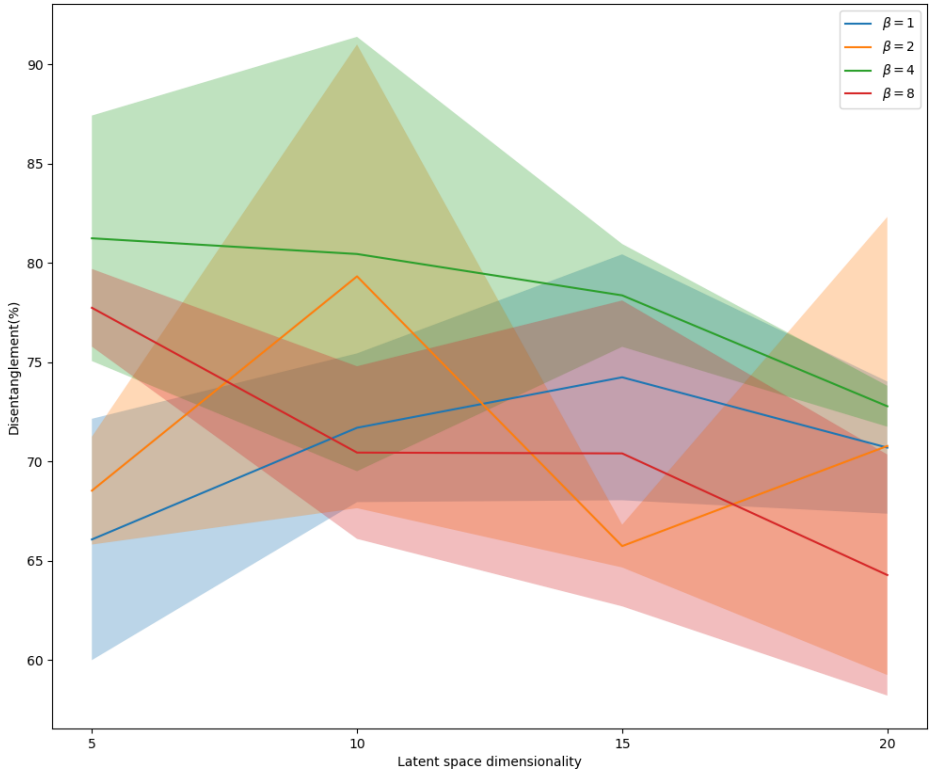



Figure 4.3: **Maximum disentanglement scores throughout the epochs over the varying latent sizes and $\beta$ values.** Same as the figure 4.1 above except we chose the maximum disentanglement score throughout the training.

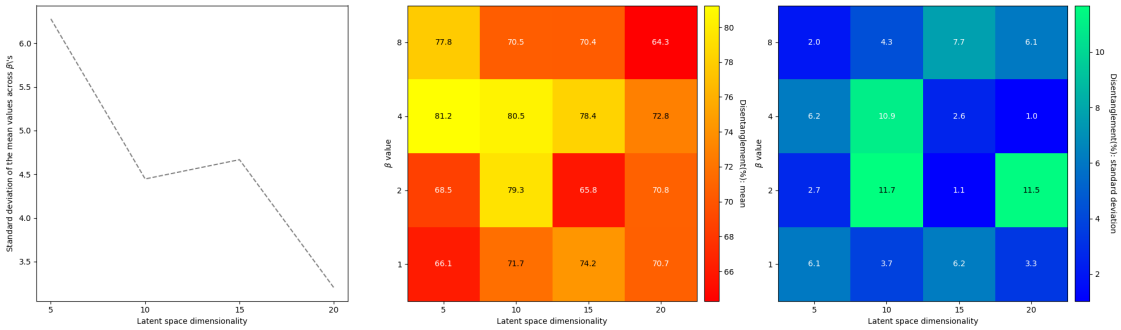



Figure 4.4: **Maximum disentanglement scores throughout the epochs over the varying latent sizes and $\beta$ values.** Same as the figure 4.2 above except we chose the maximum disentanglement score throughout the training.

### 4.1.3 Analysis

The results for the first experiment is very surprising and shows exactly the opposite trend to what we originally expected. If we see the first graphs in figures 4.2 and 4.4, we can see that in contrast to our first hypothesis, the disentanglement performances between the models vary the most when $D = K$. Similarly, unlike what we expected in our second hypothesis, the differences between the models *converged* as the latent space dimensionality increased relative to the number of the generative factors.

## 4.2 Effect of the data complexity

### 4.2.1 Original hypothesis

There were two original hypotheses with regards to the second experiment. Firstly, we expected that all models would have close to 100% disentanglement scores when $K = 1$ because the data set is simple enough for it to be encoded in a single dimension even without much pressure on efficiency. Secondly, related to the first, we expected to see that the effect the hyperparameter $\beta$ would make on improving disentanglement would increase as $K$ increases. As the results from Higgins et al. (2017) showed, the $\beta$-VAE outperformed VAE in disentangling on the dSprites data set which is equivalent to our data set with $K = 5$. This is why we expect the performance between the $\beta$-VAE and the VAE to increasingly diverge between $K = 1$ to $K = 5$. We also expect the divergence to continue for $K > 5$.

### 4.2.2 Results

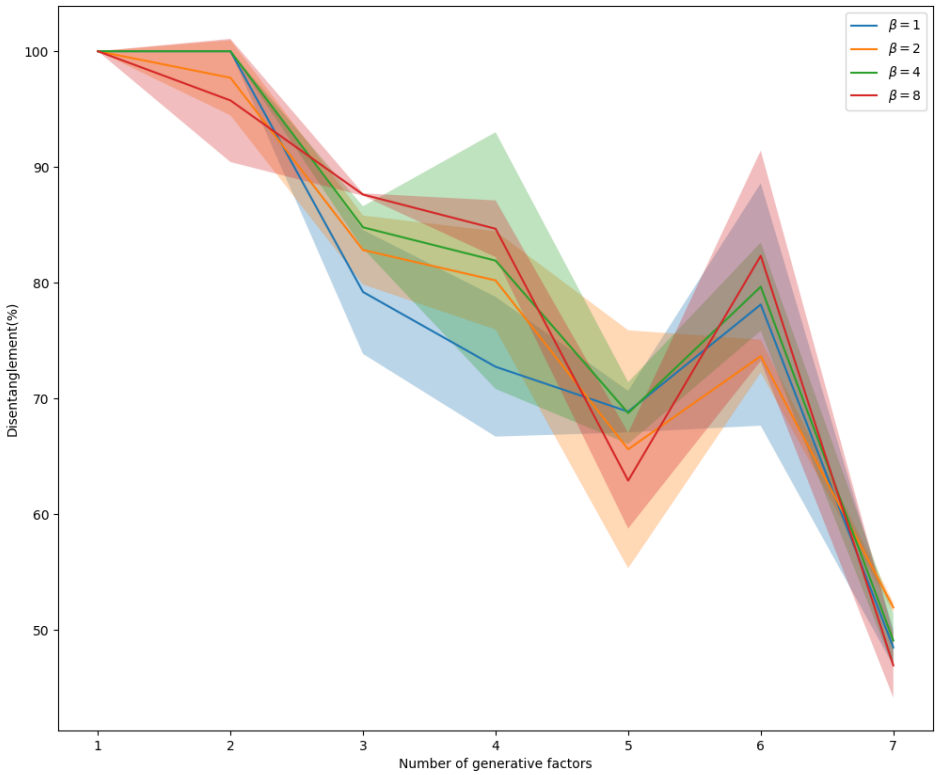



Figure 4.5: **Disentanglement scores on the final epoch over varying degrees of complexity and $\beta$ values.** Coloured line graphs representing different $\beta$ values as shown in the legend. The x-values of the lines are the number of generative factors and the y-values of the lines represent the mean disentanglement scores across three runs. The shaded region around the solid lines represent the standard deviation of the disentanglement scores across the three runs.

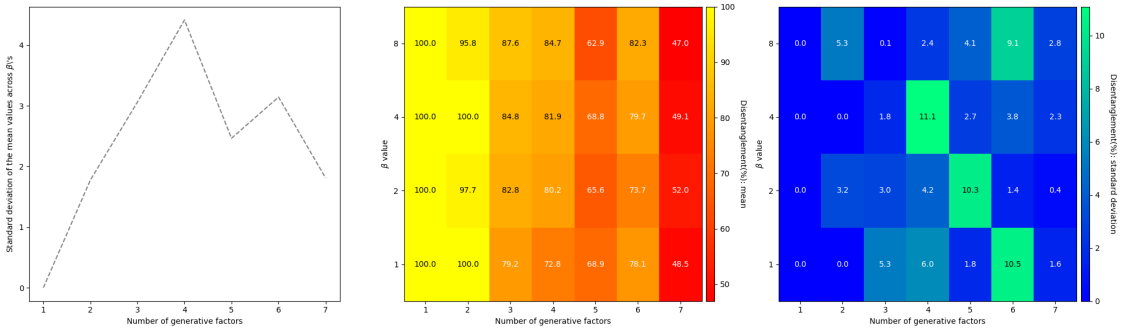



Figure 4.6: **Disentanglement scores on the final epoch over varying degrees of complexity and $\beta$ values.** Left: line graph with x-values being the number of generative factors and y-values being the standard deviations of the mean disentanglement values across the models. Centre: heatmap of the mean disentanglement scores across the varying numbers of generative factors and $\beta$ values. Right: same as the bottom left figure except with the standard deviation.

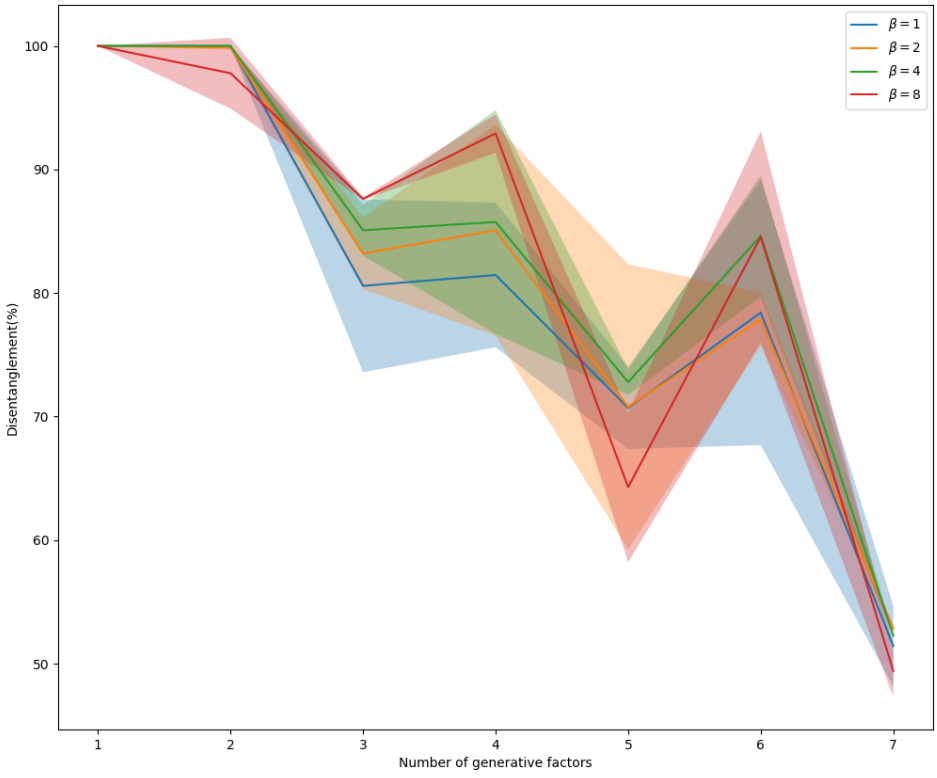



Figure 4.7: **Maximum disentanglement scores throughout the epochs over varying degrees of complexity and $\beta$ values.** Same as the figure 4.7 above

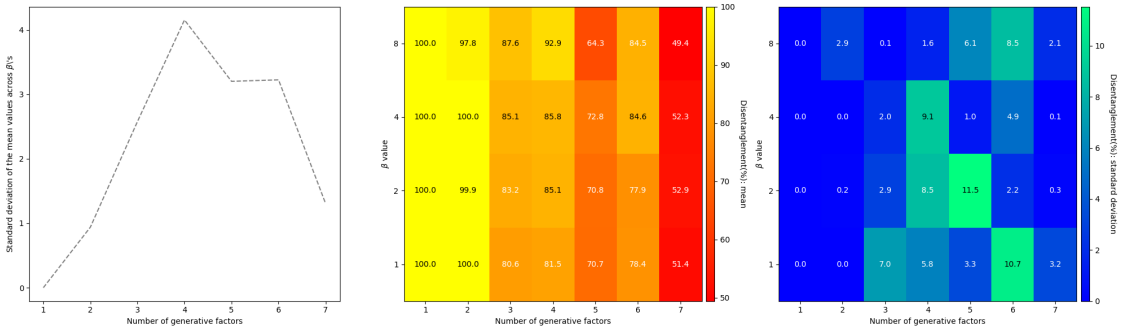



Figure 4.8: **Maximum disentanglement scores throughout the epochs over varying degrees of complexity and $\beta$ values.** Same as the figure 4.8 above

### 4.2.3 Analysis

If we see the first graphs in figures 4.6 and 4.8, the results support our first hypothesis as we can see that all models start with a 100% disentanglement score. The results also support our second hypothesis up to certain extent as we see that the disentanglement scores between the models vary more and more as the data complexity increases from one generative factor to four generative factors. However, after this point, we can see that the disentanglement scores between the models start to converge again.

# Chapter 5

# Discussions

## 5.1 Discussions on the results

### 5.1.1 Experiment 1

The results from the first experiment were very surprising because it seems to show exactly the opposite trend than expected. There are a few ways to interpret this result but in the hindsight, it looks like there may have been another important factor we should have taken into account. In the definition of the KL divergence, we take the sum over the dimensionality of the latent space. This means that the KL divergence already is a function of the dimensionality of the latent space. Higgins et al. (2017) proposes a way to normalise the $\beta$ constant:

$$\beta_{\text{norm}} = \frac{N\beta}{D}$$

where $N$ is the size of the input space and $D$ is the size of the latent space. Perhaps we would see the pattern that we expected had we varied the *normalised* $\beta$ values. We can test this in future research.

### 5.1.2 Experiment 2

It was good to see that the results clearly support our original hypotheses and if we look at the individual graphs in figures 4.5 and 4.7, we can clearly see the $\beta$-VAEs gradually outperforming the VAE model as the number of generative factors increase from one to four. It is however, interesting to see that unlike what we expected, the varying performances converge again as the data complexity passes a certain point. We can think of two interpretations for this phenomenon. First, it may be that this is the limitation of the $\beta$-VAE models. Perhaps the $\beta$-VAE models are not expressive enough to be able to continue to significantly outperform the VAE model in disentangling data with complexity beyond a certain point. Second, it may be that the range of the $\beta$ values used in this experiment do not cover large enough range to see further trends. It may be that with the number of generative factors beyond four, the optimal $\beta$ values need to be larger than eight, which is currently our maximum value, or that they need to be much smaller than one. These are what we could further look into in future research.

## 5.2 Ways to further improve

### 5.2.1 Model settings

Due to the constraint in time, we were unable to try more values of the latent space dimensions, number of complexities, and the $\beta$ values as discussed above. If we run the experiments with finer increments and/or bigger range of values, we expect to see more clear patterns emerging. Also, as is always the case, the results may improve by more carefully tuning the general model settings such as the batch size, learning rate, maximum epochs, and the optimiser which have been kept at constant at their commonly set values.

### 5.2.2 Disentanglement metric

There are some thoughts we wish to discuss on the disentanglement metric used in this project. By working with the metric proposed by Kim and Mnih (2018), we really found the computational simplicity really helpful, especially in saving time to run them. Also, the fact that the metric used a parameter-free deterministic classifier helped to have more consistent and robust standard of criteria for comparing models over various settings including the change of the size of the latent space.

However, we did make a simple observation which could potentially help to significantly improve the sensitivity of the metric. Recall, as explained in section 3.3.2, that the metric looks for the argmin across the normalised latent dimensions of the encoded values of images generated with a single generative factor fixed for calculating the disentanglement score. The limitation with this approach is that this results in the loss of information of *how small* the dimension with the argmin is. Consider two scenarios. The first scenario is when the model has ideally and perfectly made a disentangled representation of the input space. In this case, the normalised variance with the argmin will be zero. The second scenario is when the model learned a poor disentangled representation but has managed to disentangle enough for there to be a one-to-one correspondence between argmin dimensions and the generative factors. In this case, according to the metric, the model could potentially, though unlikely, achieve a high disentanglement score, because the metric does not care *how small* the minimum normalised variance is as long as it is *relatively* the smallest. Although in practice, getting a high score is still unlikely because if the model's latent encodings are strongly entangled, the argmin of the generated samples will highly be inconsistent across samples which leads to a lower disentanglement score. However, it seems like a waste of valuable information to not take into account how small the lowest variance was.

One way we could make use of the variances across the dimensions is by adding a weight to the votes which takes into account of the variances. For example, consider:

$$w = \frac{\left|\left(\sum_{d=1}^{D} \sigma_d\right) - \sigma_{min}\right|}{\sum_{d=1}^{D} \sigma_d}$$

With this weight $w$, the vote would weigh maximal if the relative variance at argmin is close to zero, and the weight of the vote would be minimal if the relative variance is close to the total variance. This can potentially provide more sensitive disentanglement metric which takes into account of *how small* the variance is at argmin instead of just taking argmin into account.

### 5.2.3 Loss function

Theoretically, as discussed in 2.3.2, it made much sense to let the loss function include a *weighted* KL-divergence term in order to control how efficiently we require the model to encode the images. However, the changes of the weight value $\beta$ did not always show recognisable differences in the experiments from this project. There were very specific settings in which the $\beta$ value did make a clear difference, but in many settings, there was not a big improvement as one hoped for.

More recent paper by Burgess et al. (2018) suggested some further changes to the loss function which could improve the performance. Let us recall what the $\beta$ value is encouraging the model to do. If $\beta = 0$, i.e. when there is no KL divergence, then the latent distributions will tend to be very sharp Gaussians, i.e. with very small variances, with the mean values heavily varying depending on the input values. Larger $\beta$ values enforce the Gaussians to have wider variances and the mean values to not vary too much by encouraging them to be close to zero. In the information bottleneck perspective(as discussed in section 2.3.2), increasing the $\beta$ decreases the information bottleneck by encouraging the KL divergence to be zero. Now it isn't that we actually want the distributions' KL divergence to be actually *zero*. Instead, by pushing the KL divergence towards zero by the appropriate amount, controlled by $\beta$, we want the KL divergence to arrive at an optimal value which is not too big, for it not to learn anything, nor too small, for it to learn in entangled manner. However, if what we what we are aiming for is encouraging the model to have some specific value of an optimum KL divergence, may be adjusting the pressure for the KL divergence to stay close to zero is not the most efficient way. This is exactly what the new loss function by Burgess et al. (2018) aims to address:

$$\text{Loss}_{\beta\text{-VAE2}} = \mathbb{E}_{q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + \gamma|KL\left[q_{\boldsymbol{\theta}_{\boldsymbol{x}}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right] - C| \tag{5.1}$$

Although the equation above may look similar to the original $\beta$-VAE loss function, there is a subtle difference in the objective it is trying to achieve. The new $\gamma$ term essentially plays the same role as the previous $\beta$ term: it acts as a weight to this newly transformed KL divergence. The new $C$ term is what controls the *capacity* of the information bottleneck. In the original $\beta$-VAE equation, the capacity term was zero and so the weight parameter was always directing the model towards a zero KL divergence. However, now in this new equation, we can specify what value of the KL divergence we want the model to aim for. Furthermore, Burgess et al. (2018) made the capacity term vary over time by linearly increasing it from a preset range. This was motivated by the observation that different features were learnt at different values of capacity as shown by the figure below:

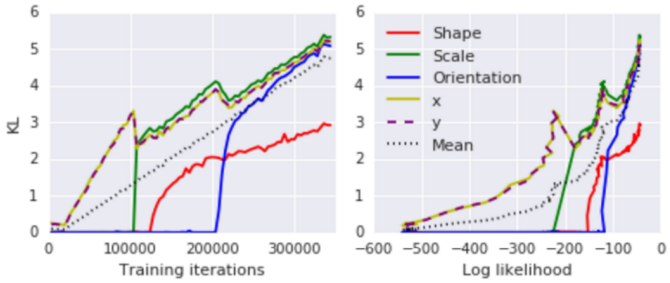



Figure 5.1: **Figure from Burgess et al. (2018): The effect the capacity has on the learning of various generative factors.** Left graph: the y-axis is the KL divergence(in nats) and the x-axis is the number of training iterations. The graphs show the KL divergence of the dimension which correspond to representing a specific generative feature. The once a dimension learns to represent a generative feature, the KL divergence increases as the distribution strays away from the standard Gaussian. The The grey dotted line is the mean value of the KL divergence which also represents the value of the increasing capacity $C$ over the iterations. At different values of capacity given, different generative features were learnt in the order depending on the KL divergence capacity required to learn. Right graph: same graph as the left graph except across the log reconstruction likelihood instead of the training iterations.

Making the loss a function of the reconstruction error, the weighted KL divergence, as well as the the target capacity and the training iteration seems like a step forward in improving the performance. As an extension for this project, we wish to experiment with this new loss function to test whether or not this makes a significant performance improvement over the original $\beta$-VAE.

One other factor which could be worth considering is to put constraints on the KL divergence across different dimensions *separately*. For example, as we can see from the figure 5.1 above, different features are learnt across different dimensions at different available capacities. Perhaps we could experiment what happens if we start with varied KL divergence capacity targets across the latent dimensions from the start. Another way we could use separate KL divergence weights across separate dimensions is to vary the weight according to how much KL divergence each dimension has at a given iteration. Increase in KL dimension in a given dimension is an indirect indication of it having learnt a generative feature. At this point, we probably want to increase the capacity across other unlearnt dimensions to allow it to learn features which require higher capacity. However, if we increase the capacity across the whole, the learnt dimension could use this capacity to unnecessarily learn more inefficient, entangled representations to increase the reconstruction accuracy. To prevent this, once a KL divergence in one dimension significantly increases and settles, we could set this KL divergence value to be the capacity target for this dimension only and give it an increasingly higher weight so that this dimension "stops learning" to give other dimensions the opportunity to learn other features by increasing the capacity across other dimensions.

# Chapter 6

# Conclusion

## 6.1 Summary of work

We have studied the theoretical details of the Variational Autoencoder(Kingma and Welling, 2013) to understand how its model design of encoding the input values on the continuous probability distributions with a KL divergence regulariser, encourages a continuous representation learning and also some disentangled representation learning to some extent. We also studied the motivations behind the new $\beta$-Variational Autoencoder(Higgins et al., 2017) to understand the motivations and the intuitive reasoning as to why the new design with a weighted KL divergence should improve in disentangled representation learning. With these insights, we formulated some hypotheses on the two models' relative disentangling performances and tested by varying the latent space dimensions and the data complexity by varying the number of generative factors. We made the quantitative comparisons based on the disentanglement metric by Kim and Mnih (2018).

The results from the first experiment are not exactly what we expected to see and we will need to study them in closer detail with different settings, for example, by trying with normalised $\beta$ values. The results from the second experiment did support our original hypotheses to a certain extent in that the degree in which the $\beta$-VAE outperforms the VAE increased as the data complexity increases upto a certain point. It also opened up another interesting point of study as we saw the differences in the performances converge again as the data complexity increased beyond a certain point.

The results from this project makes a useful contribution in this field of research. All of the papers making comparisons between models' disentanglement performances compare them on completely unrelated data sets such as dSprites, CelebA, 3dChairs, and 3dChairs. For future comparisons, it can be argued that comparing the performances on incrementally increasing data complexity is another important way to test the robustness of the models.

## 6.2 Extension for the future

Firstly, as discussed in section 5.2, there are plenty of immediate improvements we can make on the experiment settings alone. The key changes to try would be normalising the $\beta$ values across the latent dimensions especially for the first experiment, running experiments on more latent

dimension values and data complexities, and trying out various general model settings. Secondly, we discussed the possible limitations on the current metric that we used and suggested some ways to make the metric more sensitive to disentanglement by taking into account of the relative values of the variances across the normalised latent dimensions as well as the argmin. Next, we reviewed the new loss function by Burgess et al. (2018) and how it can improve the performance in disentangling by defining a target capacity. We also added a suggestion of making the loss function have separate weights across the latent dimensions depending on the KL divergence across each dimensions for further improvement. These new ideas for the loss function can be used for the experiments in the future.

Recently, many other VAE based models have been published since the $\beta$ including the Factor-VAE by Burgess et al. (2018), Disentangled Inferred Prior VAE(DIP-VAE) by Kumar et al. (2017), and Hierarchically Factorized VAE(HFVAE) by Esmaeili et al. (2018) to name a few. Another famous disentangling model which is not based on VAE is InfoGAN by Chen et al. (2016) which is based on a Generative Adversarial Network architecture with an added constraint on maximising mutual information between the encoded latent codes and the output of the generator. In the future studies, we wish to compare all of these models and see how they compare as we vary the latent sizes and the data complexity.

# Bibliography


Alessandro Achille and Stefano Soatto. Information dropout: Learning optimal representations through noisy computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

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

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in $\beta$-vae. *arXiv preprint arXiv:1804.03599*, 2018.

Gal Chechik, Amir Globerson, Naftali Tishby, and Yair Weiss. Information bottleneck for gaussian variables. *Journal of machine learning research*, 6(Jan):165–188, 2005.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. Imagenet: A large-scale hierarchical image database. In *In CVPR*, 2009.

Babak Esmaeili, Hao Wu, Sarthak Jain, Siddharth Narayanaswamy, Brooks Paige, and Jan-Willem Van de Meent. Hierarchical disentangled representations. *arXiv preprint arXiv:1804.02086*, 2018.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006. doi: 10.1126/science.1127647. URL http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google.

Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, pages 3–10, 1994.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Johan Ludwig William Valdemar Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30(1):175–193, 1906.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *arXiv preprint arXiv:1711.00848*, 2017.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Alireza Makhzani and Brendan Frey. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Kosuke Miyoshi. Replicating "understanding disentangling in -vae. https://github.com/miyosuda/disentangled_vae, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

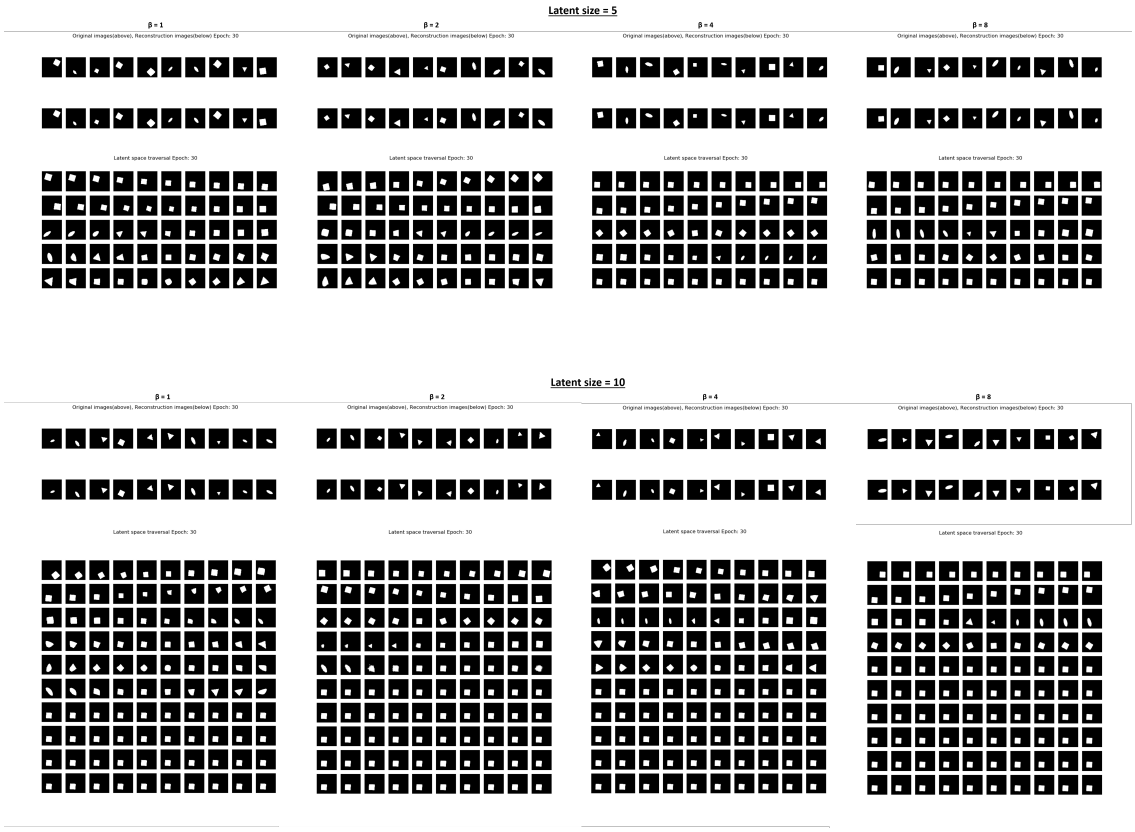
Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

L Yann. *Modeles connexionnistes de lapprentissage*. PhD thesis, PhD thesis, These de Doctorat, Universite Paris 6, 1987.

# Appendices

# Appendix A

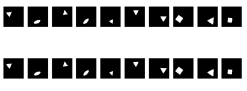
# Reconstructions of the latent traversals in Experiment 1



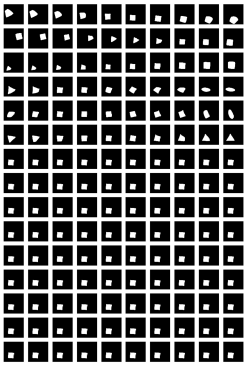
**Latent size = 15**

β = 1

Original images(above), Reconstruction images(below) Epoch: 30

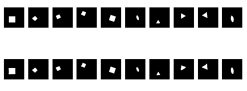


Latent space traversal Epoch: 30

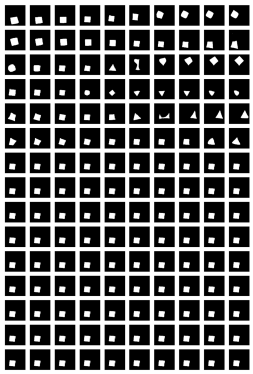


β = 2

Original images(above), Reconstruction images(below) Epoch: 30

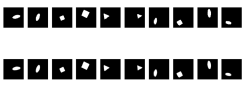


Latent space traversal Epoch: 30

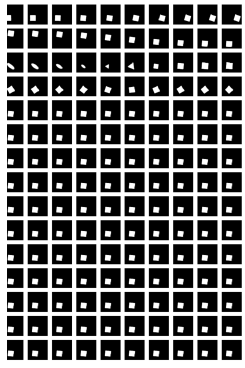


β = 4

Original images(above), Reconstruction images(below) Epoch: 30

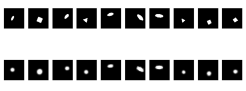


Latent space traversal Epoch: 30

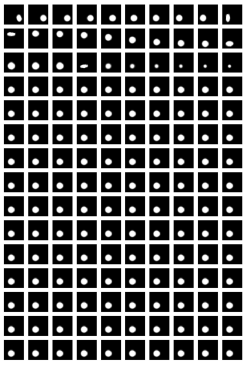


β = 8

Original images(above), Reconstruction images(below) Epoch: 30



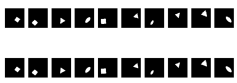
Latent space traversal Epoch: 30



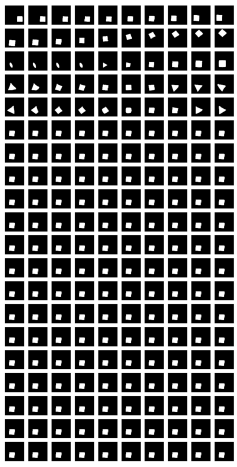
**Latent size = 20**

β = 1

Original images(above), Reconstruction images(below) Epoch: 30

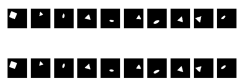


Latent space traversal Epoch: 30

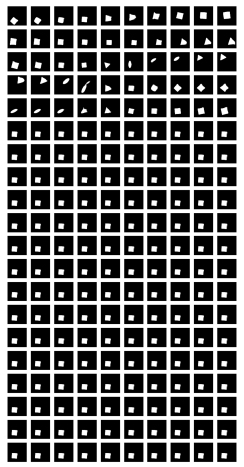


β = 2

Original images(above), Reconstruction images(below) Epoch: 30

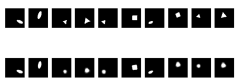


Latent space traversal Epoch: 30

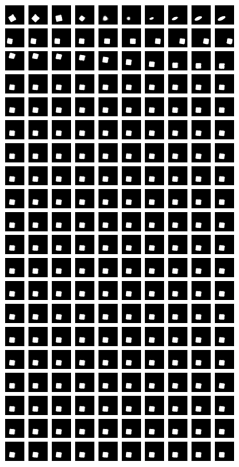


β = 4

Original images(above), Reconstruction images(below) Epoch: 30

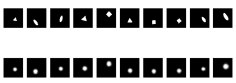


Latent space traversal Epoch: 30

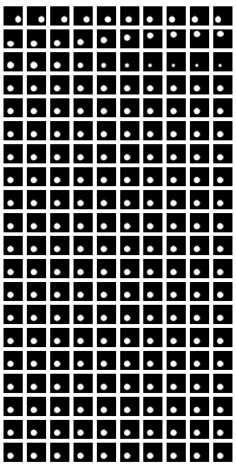


β = 8

Original images(above), Reconstruction images(below) Epoch: 30

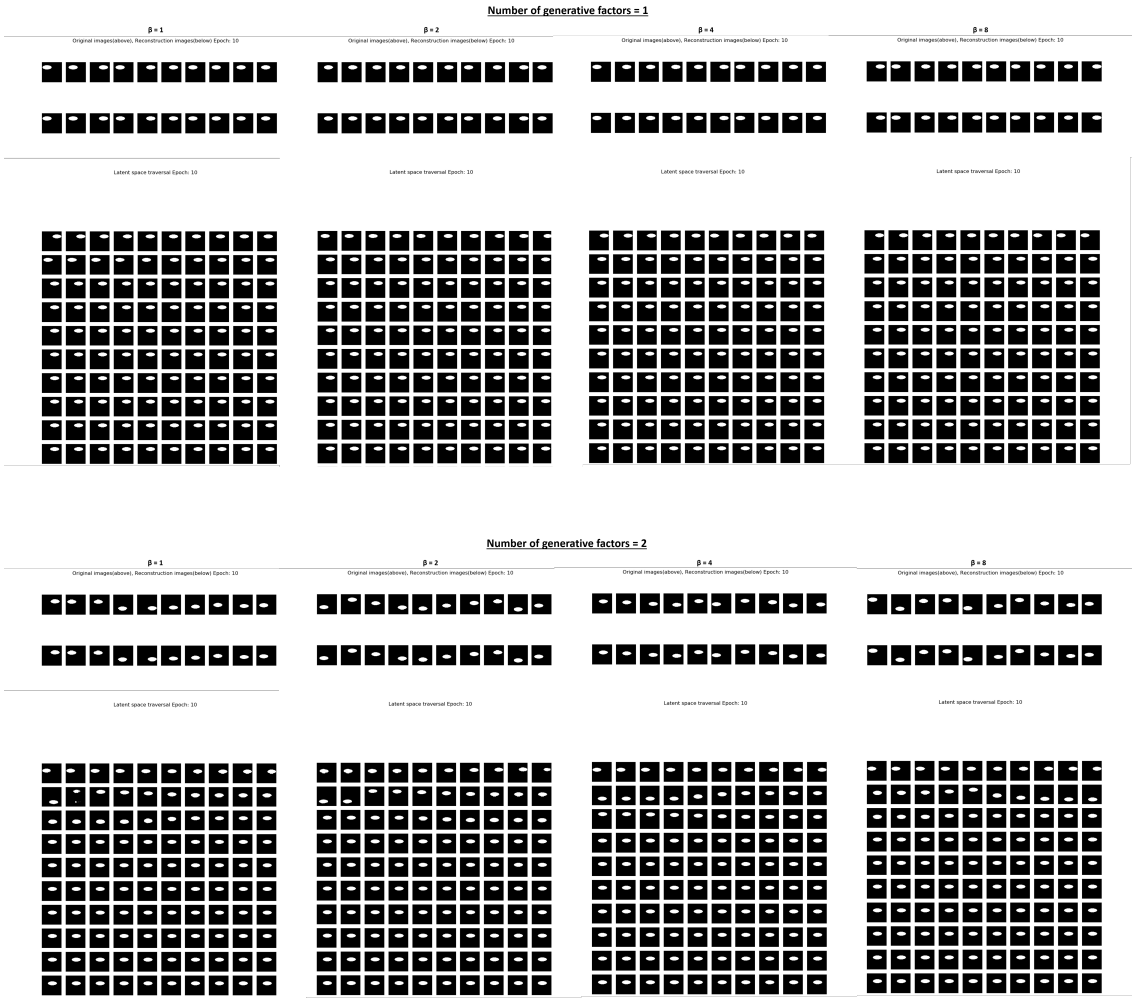


Latent space traversal Epoch: 30



# Appendix B

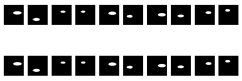
# Reconstructions of the latent traversals in Experiment 2



## Number of generative factors = 3

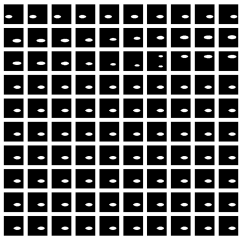
β = 1

Original images(above), Reconstruction images(below) Epoch: 20



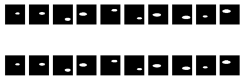


Latent space traversal Epoch: 20



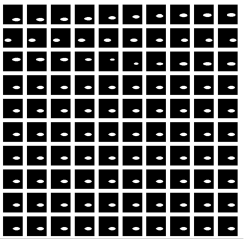
β = 2

Original images(above), Reconstruction images(below) Epoch: 20





Latent space traversal Epoch: 20

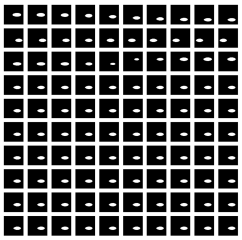


β = 4

Original images(above), Reconstruction images(below) Epoch: 20

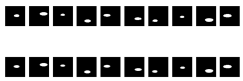


Latent space traversal Epoch: 20



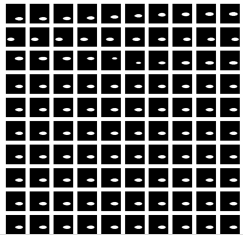
β = 8

Original images(above), Reconstruction images(below) Epoch: 20





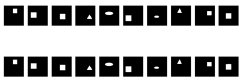
Latent space traversal Epoch: 20



## Number of generative factors = 4

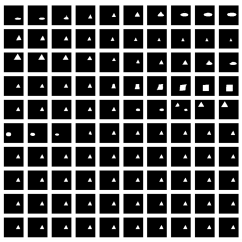
β = 1

Original images(above), Reconstruction images(below) Epoch: 30





Latent space traversal Epoch: 30

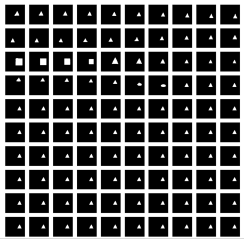


β = 2

Original images(above), Reconstruction images(below) Epoch: 30

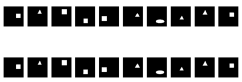


Latent space traversal Epoch: 30



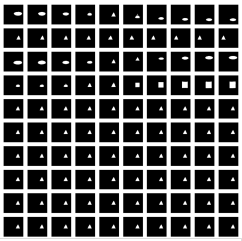
β = 4

Original images(above), Reconstruction images(below) Epoch: 30



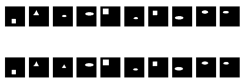


Latent space traversal Epoch: 30



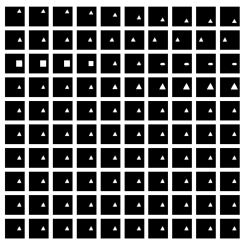
β = 8

Original images(above), Reconstruction images(below) Epoch: 30





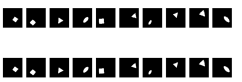
Latent space traversal Epoch: 30



## Number of generative factors = 5

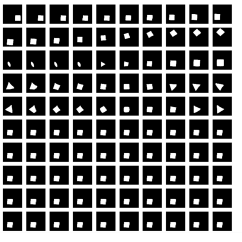
β = 1

Original images(above), Reconstruction images(below) Epoch: 30



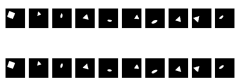


Latent space traversal Epoch: 30



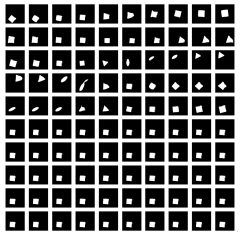
β = 2

Original images(above), Reconstruction images(below) Epoch: 30



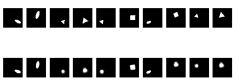


Latent space traversal Epoch: 30



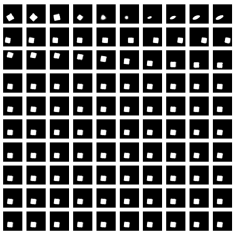
β = 4

Original images(above), Reconstruction images(below) Epoch: 30



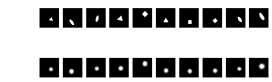


Latent space traversal Epoch: 30



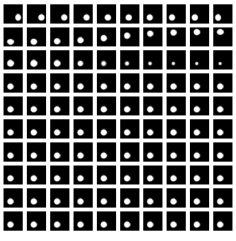
β = 8

Original images(above), Reconstruction images(below) Epoch: 30

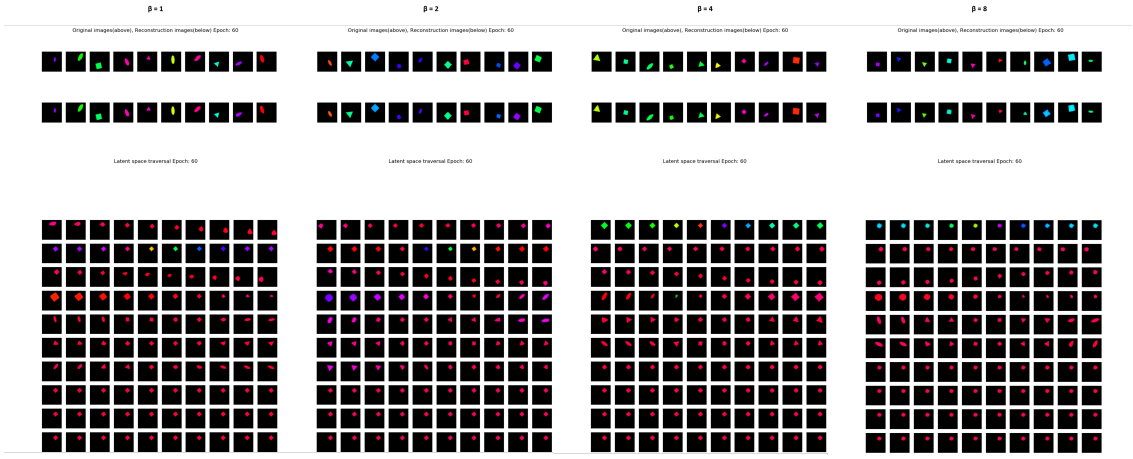




Latent space traversal Epoch: 30



**Number of generative factors = 6**




**Number of generative factors = 7**