Who Needs an Encoder?

Evaluating the Use of Random Latent Representations in Variational Autoencoders

Vrije Universiteit

Department of Computer Science
2nd quartile 2020-2021

Amsterdam, August 12, 2021

Final Version

Supervisor: Jakub Tomczak
2nd Supervisor: A.E. Eiben

Mick IJzer 2552611

Abstract

In recent years generative models have been the focus of many studies. As the name implies, generative models can generate novel data based on some initial data set. The most used and most well-known generative models are Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). While VAEs are often more stable to train, the images GANs produce tend to be of better quality. In VAEs the latent representation of a data point is learned by an encoder that is parameterized by a neural network. In this report, the learned latent representations are replaced by a randomly generated latent representations. This is achieved by freezing the weights of the encoder directly after initialization. Additionally, the effect of using random kitchen sinks instead of the regular encoder will be assessed as well. Results are obtained by training and evaluating 24 VAEs on the MNIST data set. The factors that are used are related to the encoder type (i.e. learnable, partially non-learnable, non-learnable, and random kitchen sinks), layer type (i.e. fully connected and convolutional), and prior (i.e. standard normal prior, Real-NVP flow-prior with 2 flows, and Real-NVP flow-prior with 6 flows). Results demonstrate that a decent quality of generated images can be produced when using randomly generated latent representations. However, the quality of generated images is not as high as that of the standard encoder, and the circumstances under which a VAE can use random latent representations are very specific. A fully convolutional VAE in combination with a Real-NVP flow-prior with 6 flows were needed to produce interpretable results. The main conclusions are that (1) it is possible to use randomly generated latent representations in VAEs, (2) the power of VAEs does not depend on the encoder, but rather the combination of the encoder, prior, and decoder, and (3) a VAE can be made more computationally efficient without losing generative performance by making the first layers of the encoder non-learnable. Implications, limitations, and future research directions are discussed.

Contents

Chapter 1

Introduction

1.1 Background

The foundations of modern deep learning all stem from the late 1950s when Rosenblatt (1958) invented the perceptron algorithm. The basic perceptron is based on the visual system of humans. It takes some input and performs several complex computations. Together with threshold and activation functions, the perceptron was well-substantiated by physiology and statistics. Unfortunately, it was only able to learn linearly separable patterns. Because of this, the initial interest in the perceptron dropped. However, after some time the multi-layer perceptron (MLP) became popular. An MLP, or feed-forward network, consists of, as the name implies, multiple layers of perceptrons (Murtagh, 1991). While MLPs contain the word perceptron, the perceptron is not the same as the one developed by Rosenblatt. Instead of the numerous computations, modern perceptrons use the dot-product of the input with a weight matrix. Between each layer, a non-linear activation function $h(\cdot)$, such as the sigmoid, tanh, or ReLU activation functions, is used. The non-linearity of the activation function allows the multilayer perceptron to be extremely flexible in approximating the target distribution. Besides, without the non-linear activation functions, the entire MLP could be rewritten as a single linear computation. The goal of an MLP, or any deep learning model, is to predict a target. This target can, for example, be a value (i.e. regression) or a class (i.e. classification). The computation of a single layer is schematically shown in Figure 1.1.1, note that an activation function still has to be applied to the output.

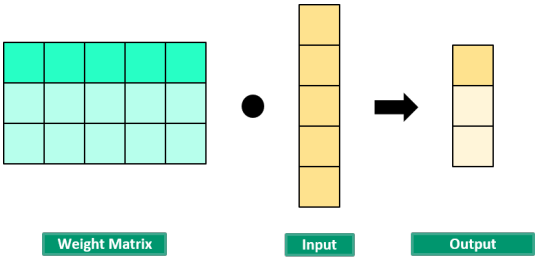


Figure 1.1.1: Computation of a Linear Layer.

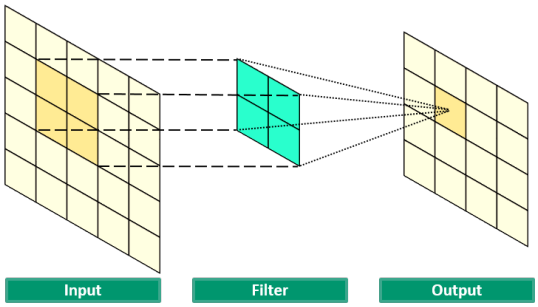


Figure 1.1.2: Computation of a Convolutional Layer.

MLPs are only able to process vectors as input. This means that they can handle for example images, however, these images have to be flattened into a single vector first. In that situation, the spatial structure, and thus part of the information, of the image is lost. Therefore, other types of deep learning models have been developed. One of them is the convolutional neural network. A convolutional neural network takes as input a matrix, such as an image, and uses a set of filters to extract relevant information (Fukushima, 1980; Cireşan et al., 2012). A filter is a small matrix of weights that is passed over the entire image. At each point, the dot-product with the section of the image is computed, which results in a processed matrix. This process is schematically shown in Figure 1.1.2. Usually, many distinct filters are used in a single convolutional layer. The filters in the first convolutional layers can learn to detect simple characteristics such as horizontal and vertical lines. The filters in later layers may detect more complex patterns or shapes. At the end of the convolutional neural network the information can not only be used for classification and regression, but also for object detection (Kumar and Srivastava, 2020), object localisation (Long et al., 2017), face verification (Bukovcikova et al., 2017), and face recognition (Wang and Li, 2018). The benefit of using a convolutional neural network is twofold; the spatial nature of an image (i.e. relative location of each pixel) is taken into account and these types of neural networks have a significantly reduced number of weights compared to fully connected neural networks.

The goal of a deep learning model is to approximate a target function (i.e. a mapping from input to target) (Flach, 2012; Hoogendoorn and Funk, 2018). For example, classifying an image to be either a cat or a dog, or predicting the travel duration of a ship given some information about its route. When the input is passed through the deep learning model it produces some output, which is the prediction of the model. Depending on the objective of the model, a specific loss function can be selected. For

instance, the most often used loss function is the cross-entropy loss for classification and the mean-squared error for regression. A deep learning model is trained to minimize the computed loss since a lower loss indicates that the prediction of the model is close to the target. In practice, many different loss functions can be chosen. It is even possible to develop a custom loss function for a specific objective. A custom loss function could for example be preferred when a false negative has significantly more severe consequences than a false positive. In that situation the loss associated with a false negative can be multiplied by a factor to increase its importance during training.

The first predictions of a deep learning model are probably not even close to the target (i.e. a high loss). Luckily, the weights of each layer in any deep learning model are trainable. This means that a weight can be updated such that the performance of the model improves. To find out in which way each weight should be updated to improve the performance, a batch of input vectors is passed through the model and the loss is computed. This computed loss is then backpropagated through the network and is used to update each weight proportionally. Intuitively backpropagation is the process of computing the strength of each weight's contribution to the final prediction. Fortunately, a mathematical optimization algorithm exists that can do that; namely gradient descent (Bishop, 2006; Witten et al., 2017). Gradient descent is the most used learning method for training any deep learning model. It takes the derivative of the loss with respect to the weights. The result indicates the strength of the contribution as well as the optimal direction in which the weight should be updated. With this knowledge, it is possible to update the weights proportionally and in the right direction. Additionally, there is one more important factor that influences the size of the updates; the learning rate. A high learning rate corresponds to larger steps and thus faster learning. However, a high learning rate is also more volatile than a low learning rate. This means that the updates might be too big and 'overshoot' the optimal value of a weight.

1.2 Generative Modeling

Two relatively basic deep learning models (i.e. fully connected and convolutional neural networks) have been described in the previous section. The Variational AutoEncoder (VAE) is also a deep learning model. However, VAEs are categorized as a latent-variable deep generative model. As the name implies, generative models can generate new data and a deep generative model makes use of neural networks (Ruthotto and Haber, 2021). These type of models have been applied to topics such as molecular design (Wei

et al., 2020; Shin et al., 2021), language generation (Pang et al., 2021), and even video generation (Mazaheri and Shah, 2021). A generative model is a powerful way of learning any kind of data distribution using unsupervised learning. Deep generative models can be divided into three categories; autoregressive models, flow-based models, and latent-variable models. Autoregressive models are models that predict the next token in a sequence given the preceding tokens (Bond-Taylor et al., 2021). These tokens can for example be the characters in a word, or even words themselves. The nice thing about autoregressive modeling is that no target has to be provided, since the target is just the input shifted by one time step. The downside of autoregressive models is that it generates data sequentially, which means that the sampling is relatively slow. Flow-based models utilize normalizing flows. A normalizing flow consists of a sequence of invertible mappings that transform an initial distribution (e.g. input data) into a valid distribution (e.g. standard normal distribution) (Rezende and Mohamed, 2015). Sampling can then be done from the valid distribution after which the generation is simply applying the inverse of each mapping in reversed order.

VAEs fall into the last category, latent-variable models. Latent-variable models are the most-used and efficient generative models. The two most popular models are VAEs (Kingma and Welling, 2014) and Generative Adversarial Networks (GANs) (Goodfellow et al., 2020). While both GANs and VAEs are latent-variable models differ in the sense that GANs are an implicit latent-variable model and VAEs are a prescribed latent-variable model. The distinction between an implicit model and a prescribed model is that for a prescribed model a prior distribution is used. This means that the latent variables are 'forced' to follow a prescribed distribution. In contrast, the latent variables of GANs are randomly drawn from a chosen distribution. The result of this difference is that VAEs are often more stable in training, while the generated images of GANs tend to be less blurry. Another benefit of using VAEs compared to GANs is that information compression is possible for VAEs. This means that data can be both compressed and reconstructed using a VAE, which could for example save storage space.

A VAE consists of three main components; the encoder, the decoder, and the prior. Some input vector, for example an image, is given to the encoder, which in turn provides a low-dimensional latent representation. The latent representation should be likely to be sampled from the prior, an a-priori selected distribution. Furthermore, the latent representation should contain meaningful information about the input, since the decoder uses the latent representation to attempt to reconstruct the original image. It

is important that the distribution of the latent representations is similar to the prior distribution because it makes it possible to sample new latent representations and decode them to generate new data. The description of the architecture in this paragraph is a simplified description. A more in-depth analysis of the architecture of VAEs is given in Section 2.1.

1.3 Research Question

While the purpose of the prior and decoder is singular, the encoder has two purposes; (1) extracting meaningful information from the input, and (2) representing this information as a latent distribution from which sampling is possible. With respect to the second purpose, the use of a learnable prior (see Section 2.2) relieves some pressure on the encoder to produce outputs according to an isotropic distribution. The consequence of this is that the encoder has more freedom to extract meaningful information from the input. Research has been done to completely remove the second purpose of the encoder by simply using an autoencoder combined with ex-post density estimation of the resulting latent representation (Ghosh et al., 2019b). Ex-post density estimation refers to fitting a density estimation on the latent variables after the Autoencoder is trained. In other words, the model itself can solely focus on reconstructing the input image without considering the latent distribution. From this ex-post density estimation samples can be drawn from which new data can be generated. Ghosh and colleagues found that by removing the constraint on the latent distribution new data can be generated of equal quality to state-of-the-art VAE architectures.

Since Ghosh et al. (2019b) showed that the encoder of a VAE does not necessarily have to provide a latent distribution for a given input, it can be concluded that the most important purpose of the encoder is to extract a meaningful representation from the input. Based on this conclusion it becomes interesting to examine how important an encoder is at all. In other words, the question arises whether extracting meaningful information from the input is an irreplaceable part of a VAE. However, it is impossible to reconstruct an image without a latent representation. So some form of dimension reduction is still needed. Therefore, instead of removing the encoder, the encoder will be made non-learnable. This means that the neural network of the encoder is initialized but its weights are not updated at all during training. The consequence of this change is that the latent representation consists of randomly generated features that may or may not contain any meaningful information about the input. Based on these random features

the decoder will still attempt to reconstruct the original input. The purpose of this report is to investigate the impact on the performance of VAEs when gradually replacing the regular encoder with randomly generated features. The main goal is to obtain a VAE that uses random latent representations and is still able to both reconstruct the input data and generate new data. Additionally, it is important to investigate under which circumstances or conditions a VAE can reach a satisfying level of performance when dealing with random latent representations. For instance, an encoder that cannot be trained is incapable in providing its output according to a standard normal distribution. Thereby, a VAE with such an encoder is probably unable to generate new data. In contrast, if such a VAE is combined with a learnable prior the quality of generated images might improve significantly.

Chapter 2

Related Work

2.1 The Variational Autoencoder

In this section, a detailed explanation of the architecture and workings of VAEs is given. The architecture of a VAE consists of several important parts and it would be difficult to introduce all of them at the same time. Therefore it makes sense to start with the AutoEncoder (AE) (Kramer, 1991), which is shown in figure 2.1.1. An AE consists of two parts; the encoder and the decoder. Both the encoder and the decoder tend to be neural networks and are parameterized by ϕ and θ respectively. These neural networks can consist of linear (see Figure 1.1.1) or convolutional layers (see Figure 1.1.2) or a combination of the two. An input vector is given to the encoder, which returns a low-dimensional representation of the input $q_\phi(z|x)$. The decoder then uses this low-dimensional representation to attempt to reconstruct the original input $p_\theta(x|z)$. The low-dimensional representation is also known as the latent representation and exists within the latent space. The loss of an AE can be as simple as the difference between the original input and the reconstructed input. AEs can be used for compression of the input. However, AEs are not generative models, since there is no way to sample from the latent space. Nevertheless, the architecture of an AE is the basis from which the VAE can be built.

The next step towards the VAE architecture is to use distributions in two ways as shown in Figure 2.1.2 (Kingma and Welling, 2014). Firstly, the encoder does not directly produce the latent representation as output. Instead, it computes a vector of means and a vector that represents a measure of variation (e.g. standard deviation or

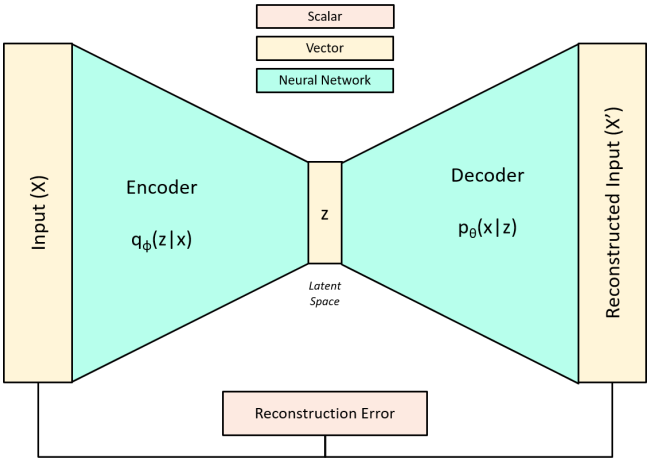


Figure 2.1.1: Architecture of an AutoEncoder.

variance). The combination of these two vectors describes a distribution around the latent representation. The latent representation of the input vector can then be drawn from this distribution. The other distribution that is used is the prior distribution, which is the reason why VAEs are considered a prescribed generative model. The prior, a chosen distribution for the latent space, is compared to the sampled latent representation. This is done by using a regularization term. In the most basic instance of a VAE, the prior is a standard normal distribution and the regularization term is the Kullback-Leibler Divergence. The Kullback-Leibler Divergence is a metric that measures the distance between two distributions. In other words, the distance between the latent distribution and the distribution of the prescribed prior should be as small as possible. Therefore the encoder is 'forced' to provide values for the latent distribution that align with the distribution of the prescribed prior. If the regularization term is properly minimized it is possible to draw samples from the prescribed distribution and pass them directly to the decoder to generate new data. It must be noted that priors are not necessarily static distributions, it is also possible to have trainable parameterized priors that are optimized to fit the latent distribution.

There is one issue with the architecture in Figure 2.1.2. The sampling procedure from the latent distribution is non-differentiable. This means that the loss cannot be propagated from the decoder to the encoder. To be able to differentiate the complete process the reparameterization trick is used (Kingma and Welling, 2014). The reparameterization trick is given by $z_x = \epsilon\sigma_x + \mu_x$, where z is the vector containing the latent variables, $\epsilon \sim N(0, 1)$ is the sample noise, and σ_x and μ_x are the output of the encoder. The

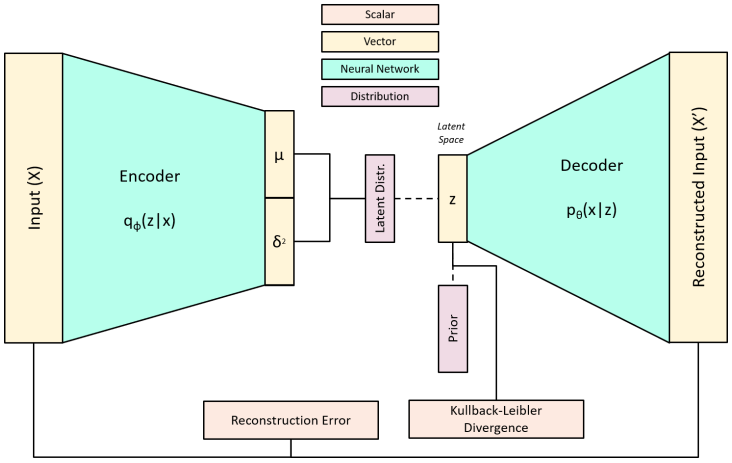


Figure 2.1.2: Architecture of a VAE.

reparameterization trick has practically the same output as directly sampling from μ and σ. However, by sampling $\epsilon \sim N(0, 1)$ the VAE becomes fully differentiable, which means that the computed loss can be backpropagated from the decoder to the encoder. The final VAE with reparameterization trick is shown in Figure 2.1.3. The objective of VAEs, and thus loss-function, differs from that of MLPs or convolutional neural networks. Considering the goal of reconstructing the original input and being able to generate new data, the loss function consists of two intuitive parts; the reconstruction error and the regularization term. Together they form the Evidence Lower Bound (ELBO) given by Equation 2.1.1, where $p(z)$ is the prior distribution. The full derivation of the ELBO for the standard normal prior can be found in Appendix A.1. Note that the objective of a VAE is to maximize the ELBO.

$$ELBO = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Error}} - \underbrace{KL[q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]}_{\text{Regularization Term}} \tag{2.1.1}$$

2.2 Advances in Variational Autoencoders

Three primary directions are researched concerning VAEs; (1) applying them to real-world problems, (2) improving the quality of the generated images, and (3) improving disentanglement (Wei et al., 2020). Some examples of real-world application of VAEs are, as mentioned earlier, molecular design (Wei et al., 2020; Shin et al., 2021), and language generation (Eisenstein, 2018). While these applications both use the generative

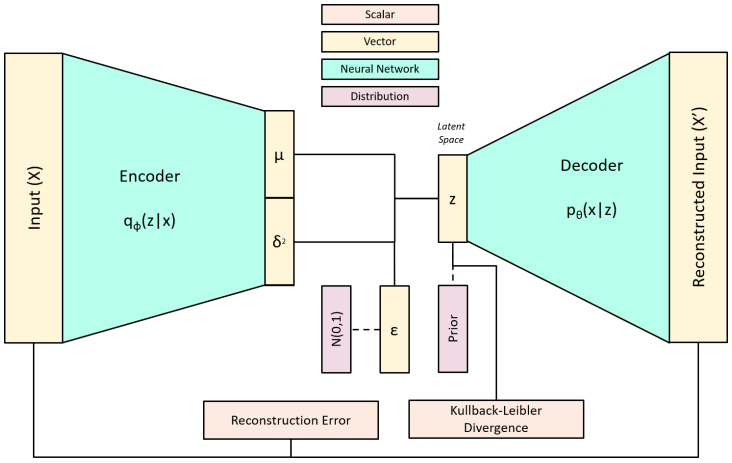


Figure 2.1.3: Architecture of a VAE with reparameterization.

aspect of VAEs, VAEs can be used for other applications as well. For example, the Variational Fair Autoencoder can learn representations that are explicitly invariant to some known characteristic of a data set while retaining as much information as possible (Louizos et al., 2016). In other words, a certain aspect of the data, such as the gender or race of a person, can be excluded from the latent representation so that certain biases are not learned. The low-dimensional representation of a data point can also be used for online malware detection (Tumpach and Holena, 2020) and to replace databases with storage-intensive and error-prone 3d-models by their learned low-dimensional latent representation (Saito et al., 2018).

Concerning the second research direction, improving the quality of generated images, many different VAE variants have been proposed. Researchers have attempted to modify the architecture of VAEs by, for example, combining the VAE architecture with the GAN architecture (e.g. VAE-GAN (Larsen et al., 2015) and f-VAEGAN-D2 (Xian et al., 2019)). Instead of modifying the architecture, it is also possible to modify the loss function to balance the reconstruction error and regularization term (e.g. INFO-VAE (Zhao et al., 2017) and β-VAE (Higgins et al., 2016)). A third way to improve the quality of generated images is to modify the way the VAE is regularized by choosing a different non-isotropic trainable prior (e.g. VaDE (Jiang et al., 2016) and GMVAE (Reynolds, 2015)). So far, only the standard normal prior has been discussed, while the possibility of a trainable prior has only been mentioned. A trainable prior is beneficial because it can bridge the gap between the latent representation (i.e. output of the encoder) and a valid distribution (e.g. the standard normal distribution).

For this report only the latter, thus choosing a different prior, is of interest. The use of a standard normal prior forces the encoder to provide latent representations that could be sampled from the standard normal distribution. However, the optimal latent representation for reconstructing the input might not exactly follow a standard normal distribution. This means that samples could be drawn from the standard normal distribution that are not similar to the representation of any of the previously seen data points. These 'gaps' in the latent space can heavily impact the quality of generated images. Furthermore, the constraint of a unimodal distribution does not allow for more complex representations of the input, which can in turn reduce the quality of the reconstructions. An intuitive substitute for the standard normal prior is a prior that does not use one Gaussian distribution, but a mixture of multiple Gaussians. Dilokthanakul et al. (2016) developed the Gaussian Mixture VAE (GMVAE) that does exactly that. They assumed that the observed data was generated from a mixture of Gaussians. For each Gaussian in the mixture of Gaussians they used three learnable parameters; the mean, the variance, and a parameter that finetunes the mixing of Gaussians. The learned clusters in the latent space correspond to meaningful characteristics of the original data. The class of a data point (e.g. dog/cat) can then be easily inferred by finding the cluster from which the latent distribution was generated. Similarly, when sampling from a certain cluster, the generated images share high-level characteristics, such as being the same animal.

Another more flexible prior that has been studied uses normalizing flows, which is the same idea as the flow-based generative models that were mentioned in the previous chapter (Rezende and Mohamed, 2015). However, instead of using normalizing flows to directly generate data, normalizing flows are used as an invertible mapping from the latent representations to a valid distribution. Many different types of normalizing flows have been proposed, such as Sylvester Normalizing Flows (Van Den Berg et al., 2018), Residual Flows (Behrmann et al., 2019; Chen et al., 2019), Householder Flows (Tomczak and Welling, 2016), and Invertible DenseNets (Perugachi-Diaz et al., 2020). However, the most promising normalizing flow is the Real-Valued non-Volume Preserving Flows (RealNVP) (Dinh et al., 2017).

Real-Valued non-Volume Preserving Flows consist of two types of layers; coupling and permutation layers. The coupling layers are the main component of RealNVP and are responsible for the transformation of the input. The first step towards using a coupling layer is to split the input into two same-size vectors $x = [x_a, x_b]$. The computation of the coupling layer is then given by equations 2.2.1 and 2.2.2, where $s(\cdot)$ and $t(\cdot)$ are

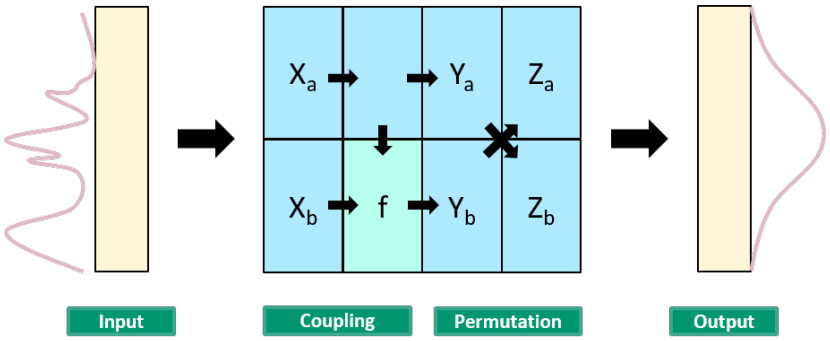


Figure 2.2.1: Computation of a Single Flow of RealNVP.

arbitrary neural networks called scaling and transition networks respectively.

$$y_a = x_a \tag{2.2.1}$$

$$y_b = exp(s(x_a)) \odot x_b + t(x_a) \tag{2.2.2}$$

These transformations are invertible by design, see equations 2.2.3 and 2.2.4. However, the coupling layer only transforms half of the input vector. To transform the complete input permutation layers are used. Permutation layers are as simple as switching the order of y_a and y_b. Multiple flows are then needed to map the entire input vector into a valid distribution. The process of the RealNVP is schematically shown in Figure 2.2.1, where f is equal to equation 2.2.2. Finally, the sampling process of a RealNVP consists of sampling from the valid output distribution (e.g. standard normal distribution) and applying the reverse permutation layer and inverse equations 2.2.3 and 2.2.4 to the sampled vector. The result of these computations is then a latent representation that could have been sampled from the same distribution as the original data.

$$x_a = y_a \tag{2.2.3}$$

$$x_b = (y_b - t(y_a)) \odot exp(-s(y_a)) \tag{2.2.4}$$

One of the research directions that were identified by Wei and colleagues (Wei et al., 2020) has not yet been discussed; improving the disentanglement for VAEs. The majority of available data is unlabeled and unorganized. Learning useful representations using unsupervised learning methods can be the solution to properly label and organize data. Unfortunately, learning a useful representation is not easy and is one of the challenges of artificial intelligence. One way to obtain meaningful representations is to use feature engineering or feature extraction. However, feature engineering is a manual time-consuming task that heavily relies on the professional knowledge of the person that does the feature engineering (Flach, 2012). Therefore, much research has been done to explore automated feature engineering (Hoogendoorn and Funk, 2018; Eisenstein, 2018). Automated feature extraction does not necessarily have to be the result of artificial intelligence, for example, the resulting components of a principal component analysis are also a low-dimensional representation that has been computed in an unsupervised manner (Whitley et al., 2012). An important thing to note is that disentangled representations are preferred. A disentangled representation has variables that are narrowly defined and are encoded as separate dimensions. This means each variable in the representation is related to a separate and unique feature of the original data point and is independent of other variables within the representation. With respect to VAEs learning a useful and relevant (latent) representation of a data point is implicit to the architecture of a VAE. A latent representation of a properly trained VAE that is able to accurately reconstruct the input data contains the important information of the original data. However, the latent features are not guaranteed to be disentangled. So researchers have attempted to improve disentanglement by for example adversarial training schemes (Carbajal et al., 2021).

2.3 Random Features

As mentioned in the previous section there are several ways to extract information from data in unsupervised ways. However, some methods, such as principal component analysis, are computationally expensive (Witten et al., 2017). Principal component analysis is expensive because the time it takes to find the eigenvectors and covariance matrix is cubic in the number of dimensions. This observation in combination with the large number of attributes that are present in most data sets makes it infeasible to use these computationally expensive dimensionality reduction methods. In the book written by Witten, the use of random projections of the data to a low-dimensional space is briefly mentioned. Random projections preserve distance relationships between data

points, which makes them an efficient way to extract a low-dimensional representation when using certain methods, such as nearest neighbours or decision trees. It makes sense that the performance of a model that was trained on random projections is worse than a model that was trained using the components extracted by principal component analysis. However, the difference between the performance of these two models decreases as the number of random projections increase.

Rahimi and Recht (2007) have researched a way to map input data to a randomized low-dimensional feature space. After this mapping fast linear methods can be applied. They describe two sets of random features; random Fourier bases and random map partitions over the input space. Both of these random feature sets are non-learnable and deterministic. Rahimi and Recht empirically showed that these random feature sets can be used to achieve competitive results with state-of-the-art kernel machines. They used the feature sets as input for linear learning algorithms and compared the performance on the accuracy, training time, and evaluation time. After the paper was published their random features were dubbed 'Random Kitchen Sinks'. Rahimi and Recht further researched these random kitchen sinks in a follow-up study (Rahimi and Recht, 2009). In this study, they found that randomly generating the non-linearities (i.e. features generated by random kitchen sinks) produces classifiers that are equally as accurate as an Adaboost classifier. While the number of randomly generated features is many times larger, it is possible to reach equal performance in significantly less time. Therefore, they concluded that random kitchen sinks can be seen as a powerful way to generate useful random features.

Random Kitchen Sinks can be implemented in just a few lines of code since it is the dot product of the input with a randomly initialized weight matrix. When using packages such as SKLearn, TensorFlow, and PyTorch, it can be simulated by a non-learnable linear layer. However, Morrow et al. (2017) showed that Random Kitchen Sinks can also be computed in a convolutional way. The convolutional random kitchen sinks were evaluated on predicting transcription factor binding sites from DNA sequences. By using convolutional kitchen sinks in combination with a single trainable linear layer Marrow was able to outperform state-of-the-art deep learning methods. In addition, the computation of convolutional kitchen sinks and not needing to update their weights resulted in faster computation times compared to the state-of-the-art deep learning models.

In summary, random features have been used in models such as decision trees, nearest neighbours, feed-forward neural networks, and convolutional neural networks. However, within the context of generative modeling random features have not been researched yet. Based on the successes that have been booked by using random features, it is probably possible to use them instead of a learnable encoder without significantly reducing the performance.

Chapter 3

Method

3.1 Encoders

The main goal of this report is to develop a VAE that can reconstruct its inputs and generate new data by using random latent representations. The random latent representations will be obtained by making the encoder non-learnable. In other words, the weights in the encoder are randomly initialized and not updated at all during training. In total four different types of encoders are evaluated; learnable, partially non-learnable, non-learnable, and random kitchen sinks. A learnable encoder coincides with how deep learning models are normally trained. The weights of the partially non-learnable encoder are initialized and only the weights in the final layer will be updated. The weights in the first layers are not updated at all. The output of these two encoders is still a vector of means and a vector that contains a measure of variation since the encoder should be able to adapt and learn to output a distribution.

The weights of the non-learnable and random kitchen sinks encoders are not updated at all. This has one crucial implication; these encoders cannot learn to output valid means and measures of variance. This is somewhat comparable to the constant-variance encoders described by Ghosh et al. (2019b). The encoders of constant-variance VAEs only output a mean value, while the variance is a fixed scalar. These type of models have been adopted for variational image compression (Ballé et al., 2017) and adversarial robustness (Ghosh et al., 2019a). Based on these studies it was decided to discard the measure of variance as output and choose a fixed scalar for the variance. The variance is a hyperparameter and should be chosen with care since a too small variance reduces

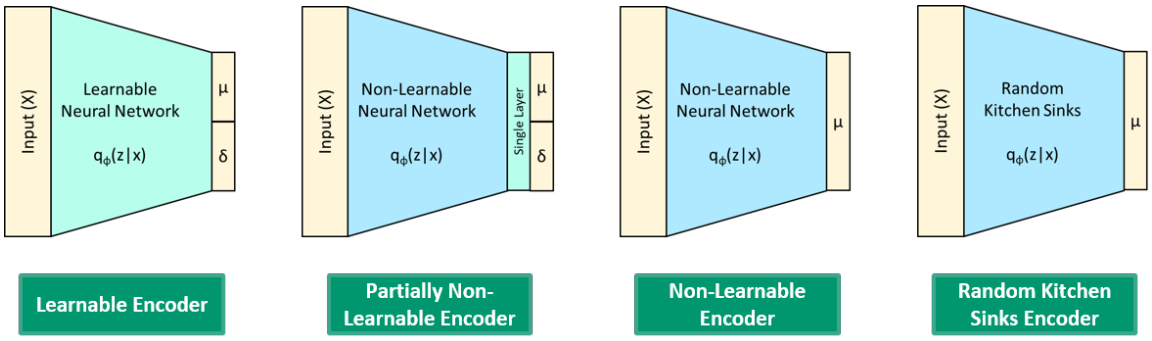


Figure 3.1.1: Schematic overview of the four different encoders.

generative capabilities. However, a variance that is too large makes it hard for the decoder to distinguish between latent representations, thus impairing its reconstructive ability. The four different encoders that were described in this section are schematically shown in Figure 3.1.1.

3.2 Layer Types and Priors

Several aspects of the model architecture might influence the performance of the VAEs. More specifically, as mentioned, if the encoder outputs a randomly generated deterministic latent representation it loses the ability to extract meaningful information and the ability to provide a latent distribution. The randomly generated representations do contain information about the input, however, the question is whether the information is meaningful and informative enough for the decoder to reconstruct the original image. So, to enable some form of meaningful information the use of a fully convolutional VAE (Fan et al., 2020) could be beneficial compared to a VAE with only fully connected layers. A fully convolutional VAE, as the name implies, only uses convolutional layers, which means that some of the spatial information of the input might be retained in the latent representation even if the latent representation consists of random features. An additional benefit of using a fully convolutional VAE is that the number of weights is not heavily influenced by the size of the input or the size of the latent representations. In other words, the scalability of a fully convolutional VAE is superior to that of a VAE with fully connected layers.

In Section 2.1 the basic architecture of VAEs was explained. Within the basic architecture, the prior that is normally chosen is the standard normal prior. The standard normal

prior is used to force the encoder to provide latent representations that align with the standard normal distribution. However, the encoder is only able to learn this if it has trainable weights in its final layer. This means that the non-learnable and random kitchen sinks encoders are unable to learn proper latent representations that could be sampled from the standard normal distribution. Therefore, the use of a learnable prior, that can approximate the latent distribution, could make it possible to generate new samples. The prior that is chosen is the Real-Valued Non-Volume Preserving flow-prior (see Section 2.2) (Dinh et al., 2017). However, approximating randomly generated latent representations could prove difficult. Thus, the flexibility of the trainable prior might play an important role in the performance of the model as well. The flexibility and power of a Real-NVP flow-prior are determined by both the number of flows and the size of the neural networks in each flow. Therefore, two separate flow-priors will be evaluated; one with two flows and one with six flows. The scaling and transition networks will be kept equal for both flow-priors.

3.3 Evaluation Metrics

The performance of the VAEs will be evaluated by several evaluation metrics. The first few metrics are those of the objective functions (see Section 4.3). In other words, the total loss (i.e. negative ELBO) and its components; the regularization term and reconstruction error, will be evaluated. These objective measures are commonly used to gain insights into the quantitative performance of the models (Tomczak and Welling, 2016; Jiang et al., 2019; Joo et al., 2020; Guo et al., 2020; Van Den Berg et al., 2018; Liu et al., 2020). Additionally, the Fréchet Inception Distance (FID-score) will be computed (Heusel et al., 2017). The FID-score reflects the similarity between two data sets of images. More specifically, it compares the distribution of generated data to the distribution of original data that was used to train the VAE. A FID-score of 0 means that the distribution of the generated data is equal to that of the original data. On the other hand a high FID-score corresponds to a large difference between the two distributions. To be able to reliably compute the FID-score it is recommended to use at least 10.000 samples from both data sets. Both the regularization term and the FID-score are related to the quality of generated images. However, the FID-score directly represents the quality of generated images, whereas the regularization term represents the ability of the prior to approximate the latent distribution. This is why the FID-score is considered to be more informative and why it is generally used in the field of generative modeling (Asperti and Trentin, 2020; Ghosh et al., 2019b).

The next two measures are the computation time per epoch and the number of trainable weights. Note that the computation time is not an absolute measure, but a relative one, because it depends on the quality of the code as well as the hardware that was used. However, since it is a relative measure it is possible to compare the different models to each other (Wei et al., 2020; Jiang et al., 2019). Concerning the number of trainable weights; the models with non-learnable and random kitchen sinks encoders will have less trainable weights compared to the models with partially non-learnable and learnable encoders. Furthermore, using a Real-NVP flow-prior also adds a significant number of weights for each flow used. Still, it might be interesting to evaluate whether the increase/decrease in trainable weights is proportional to the differences in performance of the models.

Lastly, the quality of reconstructed and generated images and the interpolation between generated images will be evaluated manually. The manual evaluation is a regular practice in the field of generative modeling (Ghosh et al., 2019b; Joo et al., 2020; Liu et al., 2020; Dilokthanakul et al., 2016; Wei et al., 2020; Jiang et al., 2019). The quality of reconstructed images is of course correlated to the reconstruction error, and the quality of generated images to the regularization term and FID-score. However, visually inspecting the quality of the reconstructed and generated images can result in useful insights. Visually evaluating the quality of generated images is a subjective task. Fortunately, the quality of generated images is relative, so it is still possible to conclude that the quality of generated images is better or worse for one model compared to another model. Additionally, the premise of a VAE is that the latent space is smooth. Therefore the interpolation between generated images will be checked on smoothness and inconsistencies.

Chapter 4

Experimental Setup

4.1 Design

A three-factor experimental design will be used to answer the research question. The three factors are (1) encoder type, (2) layer type, and (3) prior. The reasoning behind these factors has been explained in Chapter 3. For the encoder type four different encoders are used; learnable, partially non-learnable, non-learnable, and random kitchen sinks. Two layer types are evaluated, namely fully connected layers and convolutional layers. Lastly, the standard normal prior, a Real-NVP flow-prior with 2 flows, and a Real-NVP flow-prior with 6 flows are evaluated. Altogether this results in a 4x2x3 factorial design and a total of 24 models that will be trained and evaluated. These 24 models will be trained and evaluated on a relatively simple data set (MNIST). Afterward, one layer type and one prior will be chosen to be further evaluated on a more complex data set (SVHN) to investigate whether the results generalize. This means that four more models (i.e. the four different encoder types) will be trained and evaluated on the SVHN data set.

4.2 Model Architecture

The model architectures are kept as consistent as possible. This means that the difference between a model with fully connected layers and a normal prior and a model with fully connected layers and a Real-NVP flow-prior with 6 flows is only the used prior. Thus, the number of layers and the number of weights in each layer for both the encoder and decoder are equal for these two models. All of the encoders and decoders (excluding the

random kitchen sinks encoder) have four and five layers respectively. The number of weights in each layer depends on the layer type. The exact layers, activation functions, and the input/output sizes of each layer can be found in Appendix B. The random kitchen sinks encoder consists of a single linear layer or a single convolutional layer depending on the layer type. The output shape (i.e. latent representation) of the random kitchen sinks encoders is equal to that of the other encoders.

To be able to properly compare different models it was decided to use a latent representation with 16 features for the MNIST data set. However, the shape of the latent representation differs for the models that use convolutional layers compared to those that use fully connected layers. More specifically the latent representation has shape 4x2x2 and 16 when using convolutional layers and fully connected layers respectively. Intuitively this means that the convolutional models have four features for four distinct locations of the image, while the fully connected models have 16 features for the entire image. The models that will be further evaluated on the more complex SVHN data set will have a doubled size for their latent representations (i.e. 8x2x2 and 32 for the convolutional and fully connected layer types respectively).

The last part of the model architecture is the prior. As mentioned two Real-NVP flow-priors with a different number of flows (i.e. 2 flows and 6 flows) will be evaluated. Besides the number of flows, there are no differences in the neural networks that are used in a flow. Each flow consists of two neural networks (i.e. transition and scaling) that both have three linear layers with 64 nodes in the hidden layer. A leaky ReLU activation function is used after the first two layers and a Tanh activation function for the last layer. Note that the size of the input and output is equal to the size of the latent representation. However, the shape of the latent representation for the models with convolutional layers is three-dimensional. Therefore, the latent representation for these models is flattened before giving them as input to the prior.

4.3 Objective Function

The objective function that will be minimized during training is the negative ELBO. The negative ELBO is minimized, because the true objective is to optimize the ELBO as given by Equation 4.3.1. Its derivation is given in Appendix A.1. The ELBO consists of the reconstruction error and the regularization term and the negative ELBO is the go-to objective loss function for VAEs since it captures both the reconstructive purposes and generative purposes of VAEs. The reconstruction error will be computed by using

the mean-squared error between the original input and the reconstructed input. For the models with the standard normal prior, the regularization term is equal to the Kullbeck-Leibler divergence.

$$ELBO = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Error}} - \underbrace{KL[q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]}_{\text{Regularization Term}} \tag{4.3.1}$$

However, the ELBO of the models with a flow-based prior is slightly different. Compared to the standard ELBO an additional term is added for these models; the log determinant of the jacobian matrix. Therefore the loss function is extended to Equation 4.3.2. Note that this makes it impossible to directly compare the loss values between the different priors. The derivation of the ELBO for the RealNVP flow-prior is given in Appendix A.2. The goal of training a VAE is to maximize the ELBO. However, since deep learning models are used to minimize a loss, the objective function becomes the negative ELBO. The two negative ELBOs will be minimized by stochastic gradient descent.

$$ELBO = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Error}} \underbrace{- KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\lambda(\mathbf{z})] - \sum_{i=1}^{K}(\sum_{j=1}^{D-d} s_k(x_a^k)_j)}_{\text{Regularization Term}} \tag{4.3.2}$$

4.4 Data

The data sets that will be used to answer the research question are the MNIST and SVHN data sets. Some samples from both data sets are shown in Figure 4.4.1 and Figure 4.4.2 respectively. The MNIST data set consists of 60.000 28x28 pixels greyscale images of hand-written digits (LeCun and Cortes, 2010). The Street View House Numbers (SVHN) data set is a real-world data set with images of house numbers that were extracted from Google Street View images (Netzer and Wang, 2011). It contains over 600.000 three-colour channel images. However, the images of the cropped SVHN data set will be used. The cropped SVHN data set consists of 73.257 images of 32x32 pixels. To be able to use the same model architecture the images of the MNIST data set will be padded by two pixels so that the shape will become 32x32 pixels.

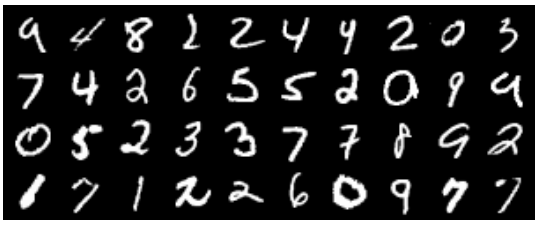


Figure 4.4.1: Samples from MNIST.

Figure 4.4.2: Samples from SVHN.

4.5 Approach

There are several important aspects to be able to properly evaluate the performance of a model and compare the performance between models. First of all, the data has to be split into a train set, a validation set, and a test set. The size of each data set is provided in Table 4.1. The validation set is used to choose the global (hyper)parameters, such as the batch size, the learning rate, and the initialization technique. Next, all models are trained using the train set. During training, the negative ELBO, its components; the reconstruction error and the regularization term, and the computation time per epoch will be saved. After training each model, the test set will be used to compute the negative ELBO and its components again, and to extract the reconstructed images. Next, new images will be sampled from the prior and interpolations between generated images will be computed. Lastly, the FID-score is computed based on 10.000 of the newly generated samples and a subset of 10.000 images of the original data set.

Since non-learnable models are evaluated, the weight initialization of the non-learnable encoders may influence the performance of the models. To combat these random effects the full experiment, excluding (hyper)parameter selection, is carried out five times. The evaluation metrics per model are then averaged over the five experiments. Of course, averaging the number of trainable weights in each model doesn't make sense since it is a static number. At the start of each experiment, the data is randomly split into a train set, a validation set, and a test set. All VAEs that are part of an experiment are trained on the same training data and evaluated on the same test data. During the experiments, the validation set is not used.

Since the experiment is carried out five times it is possible to use statistical analyses to compare models. These statistical analyses are carried out using SPSS (IBMCorp., 2017). All of the analyses are done with the encoder type and the layer type as independent variables. For evaluation metrics where the scale depends on the prior (i.e. the negative ELBO and regularization term), the prior is considered a covariate. For the other

dependent variables, the prior is added as an independent variable. The first analyses that will be done are several ANOVAs with as dependent variables the negative ELBO, reconstruction error, and regularization term on the test set. For these analyses, main effects and interaction effects are considered. The next analyses are to compare the differences between the train set and test set for these metrics to check for possible overfitting. These analyses are carried out using an ANOVA with the difference between the train set and test set performance as dependent variables. Only main effects will be tested. Then the FID-scores and computation times will be analysed with two more ANOVAs. Each ANOVA will both contain main effects and interaction effects. Several one-way ANOVAs are used to detect significant differences for the models that were trained on the SVHN data set. The only independent variable for these ANOVAs is the encoder type. For all statistical analyses, a significance bound of $\alpha = 0.01$ will be used.

4.6 Hyperparameters and Design Decisions

Most characteristics, such as the number of layers of the encoder and decoder, latent representation size, layer types, priors, and encoder types, of the 24 models have been described in the previous sections. However, there are still (hyper)parameters and design decisions that are left. To prevent unfair comparisons all the (hyper)parameters and design decisions are equal for all VAEs that will be trained. These are shown in Table 4.1. The size of the train set and validation set were chosen to maximize computational efficiency. The result is that the test set consists of all the data that is not part of the train and validation set. The batch size, number of epochs, learning rate, initialization, and constant variance were chosen based on manual non-empirical hyperparameter tuning on the validation set. These values were optimized for the performance of the non-learnable and random kitchen sinks encoders.

The chosen ADAM optimization algorithm is an extension to the widely used stochastic gradient descent, where the per-parameter learning rate is adaptive (Kingma and Ba, 2015). The per-parameter learning rates are computed based on estimates of the first and second moments of the gradients. Lastly, the weights of the encoder and decoder for all models were initialized using the Xavier Uniform distribution (Glorot and Bengio, 2010). The bounds of Xavier Uniform initialization are given by $\left[-\sqrt{\frac{6}{fan\_in+fan\_out}}, \sqrt{\frac{6}{fan\_in+fan\_out}} \right]$, where fan_in and fan_out are equal to the number of inputs and outputs of a layer respectively. Other initialization techniques

that were non-empirically evaluated were the standard initialization of PyTorch, Xavier Normal initialization, He Uniform initialization, and He Normal initialization (He et al., 2015).

Parameter	Value
Train size	25.000
Validation size	10.000
Test size	25.000 [a]
Batch size	256
Epochs	250 [b]
Constant Variance σ^2	$1e^{-6}$ [c]
Optimizer	ADAM
Learning Rate	0.001
Initialization	Xavier Uniform

Note. [a] The test size for the SVHN data set is 38.257.
[c] The number of Epochs for the SVHN data set is 500.
[c] The constant variance σ^2 is only used for the non-learnable and random kitchen sinks encoders.

Table 4.1: General Parameter Settings.

Chapter 5

Results

The experiments were carried out according to the methodology and experimental setup as described in Chapter 3 and Chapter 4. In this section, the results of the 24 VAEs that were trained on the MNIST data set and the results of the 4 VAEs that were trained on the SVHN data set will be discussed. Note that for all tables in this chapter abbreviations are used to indicate the type of the encoder, the type of layer, and the prior. The abbreviations are *L*, *PNL*, *NL*, and *RKS* for the regular learnable encoder, the partially non-learnable encoder, the non-learnable encoder, and the random kitchen sinks encoder respectively. Furthermore, the layer types are indicated by *FC* and *Conv* for the fully connected layers and convolutional layers. Lastly, the different priors are *Normal*, *Flow (2)* and *Flow (6)*, for the standard normal prior, the Real-NVP flow-prior with 2 flows, and the Real-NVP flow-prior with 6 flows respectively. For all statistical analyses, visual inspection has been performed regarding the assumptions of the used analysis (e.g. normality, linearity, and homoscedasticity).

5.1 MNIST

First, the negative ELBO is analysed. Table 5.1 schematically shows the average and standard deviation of the achieved negative ELBO on the test set for all 24 models. The average negative ELBO of for example the fully connected layer type, with a normal prior and a learnable encoder is 9.84 (std = 0.05). As mentioned in Chapter 5 the negative ELBO can only be compared given a prior since the regularization term of the negative ELBO depends on which prior is used. Note that the negative ELBO is minimized, so a lower value indicates a better model. An ANOVA was carried out to

Layer	Prior	Encoder Type							
		L		PNL		NL		RKS	
		M	Std	M	Std	M	Std	M	Std
FC	Normal	9.84	0.05	21.77	0.39	36.67	0.47	33.52	0.91
	Flow (2)	-6.11	0.05	5.78	0.32	20.78	0.87	16.48	0.23
	Flow (6)	-36.90	0.17	-26.06	0.47	-7.70	1.22	-2.53	0.63
Conv	Normal	13.08	0.29	20.04	2.35	35.37	2.04	31.52	1.81
	Flow (2)	-2.16	0.53	2.65	0.88	21.01	4.77	14.26	0.61
	Flow (6)	-31.91	0.32	-25.43	1.83	-10.74	4.92	-13.31	1.24

Table 5.1: Overview of the negative ELBO (MNIST).

determine significant effects. The prior, as a covariate, had a significant effect on the negative ELBO, $F(2, 110) = 3459.90$, $p < .000$. Additionally, the interaction effect between encoder type and layer type was significant, $F(3, 110) = 17.79$, $p < .000$. This interaction effect can be interpreted as; the learnable encoder has a better performance when using fully connected layers and the random kitchen sinks encoder has a better performance when using convolutional layers. For the partially non-learnable encoder and the non-learnable encoder, there is no difference in performance between fully connected layers and convolutional layers. Finally, a main effect of encoder type was found, $F(3, 110) = 700.46$, $p < .000$, which showed that all encoder types significantly differ from one another (i.e. L < PNL < RKS < NL).

The negative ELBO consists of the regularization term and reconstruction error. The averages and standard deviations for these two evaluation metrics are shown in Table 5.2 and Table 5.3 respectively. Note that subtracting the regularization term from the reconstruction error, as described in Equation 4.3.1 and Equation 4.3.2, yields the negative ELBO in Table 5.1. Also note that while a lower reconstruction error indicates better reconstructive performance, a higher regularization term is preferred for better generative performance. For both the regularization term and reconstruction error an ANOVA was carried out. However, the used prior is a covariate in the ANOVA with

Layer	Prior	Encoder Type							
		L		PNL		NL		RKS	
		M	Std	M	Std	M	Std	M	Std
FC	Normal	-1.83	0.02	-1.98	0.01	-6.46	0.01	-6.48	0.01
	Flow (2)	13.76	0.01	13.64	0.01	9.53	0.01	9.32	0.02
	Flow (6)	44.82	0.02	44.66	0.01	38.07	0.38	28.80	0.17
Conv	Normal	-2.66	0.04	-2.88	0.06	-6.43	0.02	-6.42	0.00
	Flow (2)	12.89	0.03	12.86	0.05	9.58	0.01	9.54	0.01
	Flow (6)	42.97	0.01	42.90	0.03	41.08	0.34	38.74	0.21

Note. The regularization term is equal to the Kullbeck-Leibler Divergence for the models with a standard normal prior. For the models with a flow prior the log determinant of the jacobian matrix is part of the regularization term.

Table 5.2: Overview of the regularization term (MNIST).

the regularization term as the dependent variable, while it is an independent variable in the other ANOVA. The first ANOVA that will be discussed is that of the regularization term. The same significant effects were found for the prior as covariate, $F(2, 110) = 6300.92$, $p < .000$, the interaction between encoder type and layer type, $F(3, 110) = 10.95$, $p < .000$, and the main effect of encoder type, $F(3, 110) = 83.64$, $p < .000$, as in the ANOVA with the negative ELBO as dependent variable. The interaction effect showed that the non-learnable and random kitchen sinks encoders perform better with convolutional layers, while the learnable and partially non-learnable encoders perform better with fully connected layers. The main effect of the encoder type showed that the learnable and partially non-learnable encoders perform better than the non-learnable and random kitchen sinks encoders. Additionally, the regularization term of the random kitchen sinks encoder is significantly lower (i.e. worse) than that of the non-learnable encoder (L & PNL > NL > RKS).

Next, the results of the ANOVA with the reconstruction error as dependent variables will be discussed. An interaction effect between the encoder type and layer type was

Layer	Prior	Encoder Type							
		L		PNL		NL		RKS	
		M	Std	M	Std	M	Std	M	Std
FC	Normal	8.01	0.05	19.79	0.39	30.22	0.48	27.04	0.91
	Flow (2)	7.66	0.05	19.43	0.31	30.31	0.87	25.80	0.23
	Flow (6)	7.92	0.18	18.60	0.46	30.38	1.21	26.27	0.70
Conv	Normal	10.42	0.26	17.15	2.33	28.94	2.05	25.09	1.81
	Flow (2)	10.74	0.55	15.52	0.88	30.60	4.78	23.80	0.61
	Flow (6)	11.06	0.32	17.48	1.84	30.34	5.20	25.43	1.40

Note. The reconstruction error is computed by the mean squared error between the original input and its reconstruction.

Table 5.3: Overview of the reconstruction error (MSE) (MNIST).

found, $F(3, 96) = 13.61$, $p < .000$. The reconstruction error was significantly higher (i.e. worse) for the learnable encoder when using convolutional layers compared to fully connected layers. For all other encoder types, the reverse was true. Furthermore, the main effect of the encoder type was significant $F(3, 96) = 809.61$, $p < .000$. All differences in reconstruction error between every encoder type were significant (L < PNL < RKS < NL). An important observation is that there is no significant main effect or interaction effect of the prior when it comes to the reconstruction error. This means that the reconstructive performance does not depend on the prior that was used.

The results that are shown so far are all based on the performance of the models on the test set. As mentioned in Section 4.5, the negative ELBO and its components were saved during training. The difference between the performance measures of the last epoch on the train set and the performance measures on the test set reflects how a VAE overfits the training data. Most deep learning models obtain a worse performance on the test set than on the training set, so this is expected for the evaluated models in this report as well. However, it is interesting to investigate whether the magnitude in which the training performance differs from the test performance depends on the used prior,

layer types, and encoder types. For all three metrics, an ANOVA was carried out with the prior, the layer type, and the encoder type as independent variables. The means and standard deviations that are mentioned refer to the estimated marginal means. Note that only main effects are considered.

Two main effects were found with respect to the negative ELBO. The average difference between training performance and test performance was signficantly higher for the fully connected layer type (M = 12.03, std = 0.38) compared to that of the convolutional layer type (M = 1.26, std = 0.38), $F(1, 113) = 395.31$, $p < .000$. This means that the models with fully connected layers were overfitting more than those with convolutional layers. Additionally, the non-learnable (M = 10.04, std = 0.54) and random kitchen sinks (M = 9.64, std = 0.54) encoder were significantly more overfitting than the learnable (M = 1.66, std = 0.54) and partially non-learnable encoders (M = 5.23, std = 0.54), $F(113) = 53.73$, $p < .000$. The exact same patterns were found with respect to the reconstruction error (all p < .000). However, this was not the case for the regularization term. The difference in regularization term on the test set compared to the train set was significantly worse for the random kitchen sinks encoder (M = .64, std = 0.08) compared to all other encoders (M < 0.12, std < 0.08), $F(3, 113) = 13.97$, $p < .000$. Lastly, a main effect of the used prior was found, $F(2, 113) = 26.14$, $p < .000$, where the flow (6) prior (M = 0.61, std = 0.067) overfitted more than both the flow (2) prior (M = 0.02, std = 0.067) and standard normal prior (M = 0.00, std = 0.067).

After training each model the FID-score was computed based on 10.000 generated samples. The 10.000 generated samples were compared to 10.000 randomly selected images from the original data set. Note that a lower FID-score means that the generated samples resemble the original samples. The mean and standard deviation of the FID-scores of each model are shown in Table 5.4. Three observations become apparent when inspecting Table 5.4; (1) for all encoder types the best performance is seen when using convolutional layers and a Real-NVP flow-prior with 6 flows, (2) the standard deviation seems to be significantly larger for the models that use convolutional layers compared to fully connected layers, (3) and the standard deviations for the non-learnable and random kitchen sinks encoders seem to be a lot bigger than that of the learnable and partially non-learnable encoders. An ANOVA with as dependent variable the FID-scores and as independent variables the used prior, layer type, and encoder type was carried out. All main effects and interaction effects were considered. All effects were found to be significant $p < .000$. However, only the three-way interaction between encoder type, prior, and layer type, $F(6, 96) = 17.37$, $p < .000$, and the three main effects, $F(1,$

Layer	Prior	Encoder Type							
		L		PNL		NL		RKS	
		M	Std	M	Std	M	Std	M	Std
FC	Normal	92.39	2.87	153.11	1.78	212.92	4.45	175.85	2.94
	Flow (2)	70.39	1.56	138.15	2.31	165.32	5.79	142.73	1.77
	Flow (6)	54.68	0.64	112.23	1.67	107.11	8.98	120.72	10.82
Conv	Normal	90.05	6.25	128.32	7.52	242.73	24.38	209.22	6.84
	Flow (2)	45.71	9.45	42.67	4.74	220.86	30.27	138.68	15.39
	Flow (6)	40.96	5.20	39.29	7.29	78.17	26.78	57.29	11.54

Table 5.4: Overview of the FID-scores (MNIST).

96) $>$ *69.60,* $p <$ *.000*, are interpreted since the interpretation of two-way interaction effects becomes difficult when there is a significant three-way interaction effect. The models with fully connected layers benefit less from a more complex prior than the models with convolutional layers. In other words, the FID-scores for models with fully connected layers do become lower when using a more complex prior, but the reduction is less significant than that of the models with convolutional layers. Additionally, the non-learnable and random kitchen sinks encoder in combination with the convolutional layer type perform show a significantly larger increase in performance when using a flow (6) prior compared to the learnable and partially non-learnable encoders. Furthermore, the flow (2) prior improves the performance compared to the standard normal prior for all models, except for the model with convolutional layers and a non-learnable encoder. Altogether this means that the performance of a VAE with random latent representations (i.e. non-learnable encoder and random kitchen sinks encoder) is best when using convolutional layers and a Real-NVP flow-prior with 6 flows. The main effects showed that the FID-scores were significantly lower for the models with convolutional layers than for the models with fully connected layers, that all three priors significantly differed from one another (Flow (6) $<$ Flow (2) $<$ Normal), and that all encoders had a significantly different performance (L $<$ PNL $<$ RKS $<$ NL).

Layer	Prior	Encoder Type							
		L		PNL		NL		RKS	
		M	Std	M	Std	M	Std	M	Std
FC	Normal	0.71	0.01	0.63	0.00	0.59	0.00	0.56	0.00
	Flow (2)	1.08	0.02	0.99	0.01	0.95	0.01	0.92	0.01
	Flow (6)	2.02	0.01	1.96	0.01	1.87	0.02	1.86	0.01
Conv	Normal	1.87	0.01	1.45	0.00	1.40	0.00	1.20	0.01
	Flow (2)	2.04	0.00	1.62	0.01	1.56	0.00	1.39	0.01
	Flow (6)	2.64	0.00	2.20	0.00	2.11	0.01	2.06	0.01

Table 5.5: Overview of the computation time per epoch (MNIST).

The next two measures are the computational time per epoch (in seconds) and the number of trainable weights. The results of these measures are shown in Table 5.5 and Table 5.6. Statistical analyses can only be carried out on the computational time. Thus, an ANOVA was carried out with the average computational time per epoch as the dependent variable. Only the main effects of the used prior, encoder type, and layer type are considered. All three factors showed a significant main effect, which can be explained by the low variability in computation times (i.e. low standard deviations). The computational time for the flow (6) prior is significantly higher than that of the flow (2) and standard normal prior, $F(2, 113) = 538.17$, $p < .000$. Additionally, the flow (2) prior is also significantly slower than the standard normal prior. The main effect of the encoder type showed that the computation time per epoch significantly differed between all encoders (RKS < NL < PNL < L), $F(3, 113) = 39.80$, $p < .000$. The last main effect is that of the layer type, which showed that the computation time of the models with convolutional layers is significantly higher than that of the models with fully connected layers, $F(1, 113) = 528.99$, $p < .000$. All of these findings correspond with the number of computations that are required for a single batch to pass through the model.

		Encoder Type			
Layer	Prior	L	PNL	NL	RKS
FC	Normal	10.256	7.952	7.824	7.824
	Flow (2)	11.888	9.584	9.456	9.456
	Flow (6)	15.152	12.848	12.720	12.720
Conv	Normal	521	434	396	396
	Flow (2)	3.689	3.602	3.564	3.564
	Flow (6)	10.025	9.938	9.900	9.900

Table 5.6: Overview of the number of trainable weights (MNIST).

At the end of training 8 previously unseen images from the test set were given to the VAE and the reconstructions were saved to be manually evaluated. These reconstructions can be seen in Table 5.7, where the top row of each entry contains the original images and the bottom row the reconstructions. The quality of the reconstructions is directly linked to the reconstruction errors in Table 5.3. Per the results of the ANOVA with as dependent variable the reconstruction error, it can be seen that the reconstructions of the learnable encoder are the best. Additionally some reconstructions, especially from the convolutional models with standard prior and from the fully connected model with non-learnable and random kitchen sinks encoders, are more blurry than the rest. However, in general, all reconstructions seem to capture the main characteristics of the original images.

The next measure that will be evaluated is the quality of generated images. Table 5.8 shows 64 generated images for each model. The quality of the generated images is directly linked to the regularization terms in Table 5.2 and the FID-scores in Table 5.4 and is limited by the reconstructive performance in Table 5.3. The generated images from the models with convolutional layers and a standard normal prior stand out. Some shapes that resemble digits can be seen, but in general, most of the images are not interpretable. The same goes for the models with fully connected layers, a standard normal prior, and non-learnable and random kitchen sinks encoders, and the model with convolutional layers, a flow (2) prior and non-learnable encoder. Additionally, the

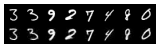
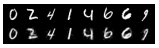
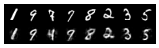
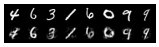
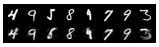
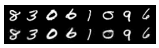
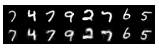
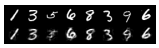
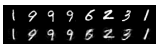
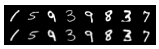
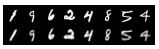
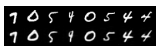
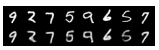
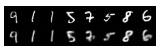
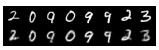
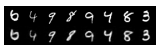
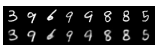
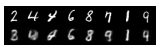
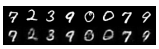
Layer	Prior	Encoder Type			
		L	PNL	NL	RKS
FC	Normal				
	Flow (2)				
	Flow (6)				
Conv	Normal				
	Flow (2)				
	Flow (6)				

Table 5.7: Reconstructed Images (MNIST).

generated images from the models with fully connected layers, a flow (6) prior and a non-learnable and random kitchen sinks encoder show almost no variation in generated images. Moreover, the performance of the partially non-learnable encoder is similar to that of the learnable encoder for all layer types and priors. For the research goal, the most interesting VAE is the one with convolutional layers, a flow (6) prior, and a non-learnable encoder. The generated images for this VAE may be blurry, but they are interpretable. Likewise, the best performing model with a random kitchen sinks encoder also uses convolutional layers and a flow (6) prior.

Finally, the interpolation between generated images is evaluated, see Table 5.9. The first and last columns of each entry show a randomly generated image. This means that the quality of the images directly relates to the quality of generated images from Table 5.8. Next, the distance between the latent representations of the two outermost images is split into six. For each of the points in between an image is generated by the decoder. If the quality of the images is not taken into account, it becomes clear that for each layer type, used prior, and encoder type the latent space is smooth. The transitions from the first to the last column are natural, and no obvious discrepancies are found.

Layer	Prior	Encoder Type: L	PNL	NL	RKS
FC	Normal				
	Flow (2)				
	Flow (6)				
Conv	Normal				
	Flow (2)				
	Flow (6)				

Table 5.8: Generated Images (MNIST).

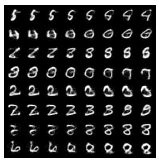
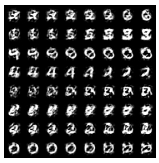
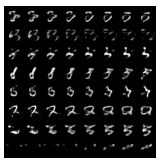
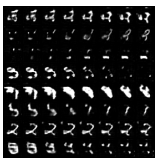
Layer	Prior	Encoder Type: L	PNL	NL	RKS
FC	Normal				
	Flow (2)				
	Flow (6)				
Conv	Normal				
	Flow (2)				
	Flow (6)				

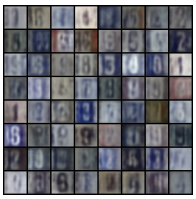
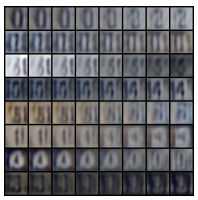
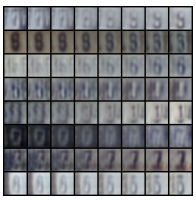
Table 5.9: Interpolation between Generated Images (MNIST).

5.2 SVHN

Based on the results obtained by training the 24 models a selection of layer type and prior was made to be evaluated on the SVHN data set. Since the goal is to obtain presentable results using random latent representations it makes sense to choose the combination of prior and layer type that was able to generate the best images on the MNIST data set. Hence, the Real-NVP flow-prior with 6 flows and the convolutional layer type were selected, which means that four more models with convolutional layers and a flow-prior with 6 flows were trained and evaluated on the SVHN data set. Two things are important to note concerning this part of the experiment; (1) the size of the latent representation was doubled compared to the models trained on the MNIST data set, and (2) the model architecture is the same as that from the models trained on the MNIST data set and is thus not optimized for the SVHN data set. The same objective function and evaluation metrics were used. This part of the experiment was also carried out five times to combat random effects.

All evaluation metrics for the four models are shown in Table 5.10. For each of the negative ELBO, regularization term, reconstruction error, FID-score, and computation time a one-way ANOVA was carried out with the encoder type as the only independent variable. All ANOVA models were found to be significant, $F(3, 19) > 21.76$, $p < .000$. Post-hoc testing showed that there were significant differences between the negative ELBO scores of all models (L < PNL < RKS < NL). For the regularization term, the learnable, partially non-learnable, and non-learnable encoders achieved a significantly better score than random kitchen sinks encoders (NL & PNL & L > RKS). For the reconstruction error, only the partially non-learnable and random kitchen sinks encoders did not significantly differ (L < PNL & RKS < NL). The FID-score of the non-learnable encoder was significantly worse than all other encoder types (L & PNL & RKS < NL). Finally, the computation time significantly differed for all encoder types (RKS < NL < PNL < L). These effects correspond to the effects found for the models trained on the MNIST data set. Some important observations that are apparent when comparing the SVHN models to the MNIST models are that (1) the reconstruction error and computation times are nearly equal for a given encoder, (2) the FID-scores are worse for the learnable, partially non-learnable, and random kitchen sinks encoders for the SVHN data set, while the FID-scores for the non-learnable encoders are very similar, and (3) the standard deviations of the FID-scores are significantly lower for the SVHN data set. Especially the third observation is relevant since it shows that the consistency of the generative performance between experiments is larger for the SVHN data set.

The findings concerning the reconstruction error align with the quality of reconstructed images. More specifically, the reconstructions of the learnable encoder are the most accurate, while the reconstructions of the non-learnable encoder are the most blurry. Furthermore, the generated images of the learnable and partially non-learnable encoders are of better quality than those of the non-learnable and random kitchen sinks encoders. Nonetheless, both VAEs with random latent representations were able to generate interpretable new samples. In other words, most of the generated images may not look like the original images, but the digits in the images are still readable. However, the variance of the regularization term for the non-learnable encoder is many times higher than that of the other encoder types. As a result the quality of generated images varied the most between the experiments for the non-learnble encoder. Additionally, the brightness and contrast of the images generated by the non-learnable encoder are significantly higher, which can be the explanation for the higher FID-score. Whereas the generated images for the non-learnable encoder show a lot of contrast, the images generated by the random kitchen sinks encoder are more monotonous. Lastly, as was the case for the MNIST models, the interpolations between generated images show a smooth latent space with no discrepancies.

Metric	Encoder Type							
	L		PNL		NL		RKS	
	M	Std	M	Std	M	Std	M	Std
Negative ELBO	-79.23	0.11	-74.09	2.66	-62.60	3.36	-67.90	1.19
RT	90.81	0.01	90.89	0.04	90.30	1.28	88.00	0.23
RE	11.58	0.11	16.80	2.63	27.69	2.36	20.10	1.37
FID-score	59.15	1.77	57.68	1.52	78.08	5.46	65.40	4.00
Time	2.65	0.01	2.21	0.01	2.13	0.01	2.08	0.00
Weights	10.339		10.250		10.196		10.196	
Reconstruction								
Generated								
Interpolation								

Note. The network architecture is convolutional and the prior is a Real-NVP flow-prior with 6 flows for all four models. RT = Regularization Term, RE = Reconstruction Error.

Table 5.10: Overview of Performance Metrics on the SVHN data set.

Chapter 6

Discussion

The purpose of the current research is to investigate the impact on the performance of VAEs when gradually replacing the regular learnable encoder with randomly generated features. Two research goals have been defined; (1) to train a VAE using random latent representations that can reconstruct inputs and generate new data, and (2) to identify the conditions under which a VAE can be trained using random latent representations. To achieve these two goals several VAEs with different architectures were trained. More specifically, four encoder types (i.e. learnable, partially non-learnable, non-learnable, and random kitchen sinks encoders) in combination with two layer types (i.e. fully connected layers and convolutional layers) and three different priors (i.e. standard normal, flow (2), and flow (6)) were trained and evaluated.

Results showed that it is possible to achieve good performance when training a VAE using random latent representations. However, of the six architectures (i.e. 3 priors times 2 layer types) that were evaluated only one was able to generate interpretable images. This one architecture consisted of a fully convolutional VAE (Fan et al., 2020) and a Real-NVP flow-prior with six flows (Dinh et al., 2017). Using only convolutional layers allows the random latent representation to contain some spatial information that is otherwise not present when using fully connected layers. As expected the flow-prior had difficulties in approximating the randomly generated latent representations when it consisted of only two flows, but with six flows it was able to approximate the latent distribution in a sufficient manner. Moreover, training additional models on a more complex data set showed that the reconstructive and generative capabilities of the VAE trained using random latent representations generalizes. This was the case for both the non-learnable encoder as well as the random kitchen sinks encoder. Though the variation

in the generated samples was significantly larger for the non-learnable encoder compared to the random kitchen sinks encoder. This variation was also present when rerunning the experiment multiple times. Whereas all other encoder types displayed a consistent performance over the experiments, the performance of the non-learnable encoder varied significantly. The use of multiple non-learnable layers likely reduces the amount of information that can be conveyed to the decoder. At the same time, these non-learnable layers may reduce the variation in the latent representations between samples. Combined these two hypotheses could explain why the regularization term for the non-learnable encoder is better than that of the random kitchen sinks encoder while the reconstruction error for the non-learnable encoder is worse than that of the random kitchen sinks encoder. A surprising result was that the generative performance (i.e. FID-score) of the random kitchen sinks encoder did not significantly differ from the partially non-learnable and learnable encoders. However, visual inspection of the generated samples did show that the generated images of the random kitchen sinks encoder show less contrast (i.e. less well-defined) than those of the other two encoders.

Although it was not part of the defined goals, the results of the partially non-learnable encoder require attention as well. The partially non-learnable encoder, where only the weights of the final layer of the encoder are updated, displayed a performance that was nearly equal to that of the normal learnable encoder. More specifically, there were no differences in FID-scores for these two encoders when using convolutional layers and a Real-NVP flow-prior with six flows. It must be noted that there were differences in the reconstructive performance. However, whereas the FID-scores were equal, the computation time per epoch was around 16.5% faster. This means that the partially non-learnable encoder achieves the same generative performance while the total computational resources needed are significantly smaller. These findings coincide with the findings regarding random features (Rahimi and Recht, 2007, 2009; Morrow et al., 2017), where equal performances and significant decreases in computational time were found compared to the 'standard' deep learning models. In the current research and the aforementioned studies, the randomly generated features were given to a single trainable layer. Though the random features that were used in the mentioned studies were generated by random kitchen sinks while the random features of the partially non-learnable encoder were the result of multiple non-trainable layers. It could be interesting to study the performance of the partially non-learnable encoder when the random features are generated by random kitchen sinks. The computational speed will likely increase while the generative performance remains the same.

Combined, the results of the partially non-learnable, non-learnable, and random kitchen sinks encoders, show that the decoder and prior can take over the tasks of the encoder if the encoder is heavily impaired in terms of trainable weights. Ghosh et al. (2019b) already demonstrated that the variational part of a variational autoencoder can be 'replaced' by ex-post density estimation to approximate the latent distribution. They dubbed their model Regularized Autoencoder. In this research, it is shown that learned meaningful latent representations can be 'replaced' by randomly generated latent representations. Taking these two pieces of information together results in the conclusion that the power of a VAE is not conditional on the encoder, but the combination of the encoder, the prior, and the decoder. Modifying the architecture of the Regularized Autoencoder by replacing the encoder by either randomly generated latent representations or a partially non-learnable encoder could be of interest. The latent distribution can then be approximated by ex-post density estimation (e.g. a mixture of Gaussians). By using a non-learnable or random kitchen sinks encoder in combination with the Regularized Autoencoder the output of the encoder will become deterministic non-variational and randomly generated. In other words, the encoder would be stripped of all of its power. On the other hand, by using a partially non-learnable encoder the generative performance could potentially become equal to the 'normal' Regularized Autoencoder (i.e. state-of-the-art).

However, the use of randomly generated latent representations, in their current form, might not be feasible for VAEs. The power of random features is that they are computationally more efficient than learned features, especially when using a large number of features. Additionally, while the information that is contained in a randomly generated feature is significantly less than the information of a learned feature, the use of a large number of random features compensates for this concern. This is visible when comparing the generative performance on the MNIST and SVHN data sets. Whereas, the non-learnable encoders performed significantly worse on the MNIST data set, the random kitchen sinks encoder performed similar to the learnable encoder on the SVHN data set. This is probably due to the larger size of the latent representation. Another significant benefit of using a larger latent representation is that the variation between experiments was a lot lower. In other words, the use of random latent representations becomes more consistent and reliable if there are more latent variables in the latent representation. Unfortunately, increasing the number of random features in the latent representation heavily impacts the ability of a prior to approximate the latent distribution. Two possible solutions to mitigate this problem are to use a different prior (i.e. other than the RealVNP flow-prior) or to use a single trainable layer that can compress the information

of a large number of random features into a low-dimensional latent representations (i.e. similar to the partially non-learnable encoder). While the RealNVP flow-based prior can very accurately approximate an unknown distribution, it is at the cost of a significant amount of extra trainable parameters. Thus increasing the number of random features means that the number of extra trainable parameters of the flow-prior should increase as well. Therefore, the use of different priors, such as a mixture of Gaussians (Dilokthanakul et al., 2016) or VampPrior (Tomczak and Welling, 2018), could potentially be more suitable for approximating latent representations that contain more latent variables. The other solution resembles the partially non-learnable encoder, however, in that situation the number of inputs to the final trainable layer would be significantly higher. For example, instead of having as an input a matrix of shape 4x4x16 the matrix would have shape 4x4x1028. The main task of the final trainable layer would then be to reduce the the number of random features to a number that is easier to approximate by a given prior.

The outcomes of the experiments that were conducted can be seen as a proof of concept that random features and random latent representations can be used in the context of VAEs. It is not definite proof since the architecture of the VAEs that was used was relatively simple. However, the training and evaluation of each VAE was carried out multiple times which shows that random effects in initializing the non-learnable layers do not have a major effect on the eventual performance of the models. It must be noted that there was more variation in the performance of the VAEs with a non-learnable encoder, but the differences were not substantial. Moreover, by training and evaluating the VAEs on a more complex data set it is shown that the findings generalize. It is still relevant to evaluate the performance on data sets with larger images as this could potentially influence the ability of the decoder to reconstruct the inputs from randomly generated latent representations. In addition, with respect to the convolutional layers that were used, scalability would not be a problem. The lack of linear layers makes it so that increasing the size of the latent representation or the size of the input does not have a noteworthy impact on the number of weights. However, as mentioned, increasing the size of the latent representation also means that the power of the prior has to increase to still be able to approximate the latent distribution. In addition, the performance of all of the trained models is not state-of-the-art. However, achieving state-of-the-art performance was not one of the goals of this research. When comparing the used architectures with those that were used to achieve state-of-the-art performance there is a noteworthy difference in for instance the number of trainable weights. More

specifically, the last layer of the encoder and first layer of the decoder of the VAEs trained in Ghosh et al. (2019b) were linear. Together these two layers contain almost five times the number of trainable weights (1.114.112) than the total number of trainable weights of **all** of the models that were evaluated in this report combined (224.590). The computational time per epoch was not mentioned in the study by Ghosh and colleagues, but one can assume that it is drastically higher for their models. The combination of convolutional and fully connected layers, as is commonly used in VAE research, is not likely to improve the performance of the non-learnable encoders. However, it might be beneficial when using a partially non-learnable encoder.

Besides the research directions that have been mentioned already in this chapter, other directions might be of interest. For instance, the initialization that was used in this report was chosen based on a non-empirical evaluation of initialization techniques. Several initializations were briefly used to train a non-learnable encoder and the one with the most promising results was chosen. Empirically evaluating the effect initialization techniques, such as He initialization (He et al., 2015), have on the random feature generation could result in new insights. Additionally, a very basic VAE architecture was used. Thus, it might be interesting to combine a non-learnable or partially non-learnable encoder with more complex VAE architectures such as VAE-GAN (Larsen et al., 2015), f-VAEGAN-D2 (Xian et al., 2019), or selfVAE (Gatopoulos and Tomczak, 2020). Finally, deep learning models often benefit from using dropout (Hinton et al., 2012) and batch normalization (Ioffe and Szegedy, 2015), neither of which were used in the models that were trained and evaluated. The idea behind dropout is that by randomly deleting units and their connections during training the deep learning model is less likely to overfit. The results have shown that certain VAE architectures are more prone to overfitting than others, so it might be worthwhile to investigate the effects of dropout. However, for regular VAE architectures dropout does not seem to improve the quality of reconstructions and generated images (Yeung et al., 2017). Yet, in the context of a non-learnable encoder, this conclusion might not hold. Investigating the effect of using dropout in non-learnable layers is of interest since it could introduce some stochasticity in an otherwise deterministic encoder. Batch normalization ensures that each unit in a layer is normalized to have a mean of zero and unit variance within a batch. Other research has shown that batch normalization and warm-up always increase the generative performance in VAEs (Raiko et al., 2016). Whether this conclusion holds for randomly generated latent representations is not clear.

Chapter 7

Conclusion

Variational Autoencoders are a latent-variable generative model. In practice, the latent representation of a data point is learned by an encoder parameterized by a neural network. In this research, the learned latent representation was replaced by a randomly generated latent representation. This was achieved by freezing the weights of the encoder directly after initialization. It was shown that, in combination with a powerful prior, the use of random latent representation did not visibly impair the quality of reconstructions and generated images. This conclusion was initially found when training VAEs on a relatively simple data set. When using a more complex data set similar results were found, implying that the use of random latent representations is not limited to simple tasks. Additionally, if the encoder consists of non-learnable layers in combination with a final learnable layer the generative performance is equal to that of an encoder with only trainable layers, while the computational efficiency increases by more than 15%.

These conclusions give rise to novel research questions regarding the use of non-learnable and partially non-learnable encoders in VAEs. For example, some new research questions could be whether it is possible to achieve state-of-the-art performance when using a non-learnable encoder, whether using a less computationally intensive prior, such as a mixture of Gaussians, could produce similar results, and whether design decisions that were not explicitly studied in this research, such as batch normalization and dropout, have a significant impact on the performance of a VAE with a non-learnable encoder. Of course, evaluating the partially non-learnable encoder and non-learnable encoder on an even more complex data set such as CelebA or CIFAR10 would be of relevance as well. A more provocative research objective is to combine Regularized AutoEncoders with a non-learnable encoder. This combination removes **all** of the power of the encoder since the

output of the encoder will be a randomly generated non-variational latent representation. However, the most important research direction for non-learnable encoders is to improve the reliability of the generative performance. Compared to the other encoders that were evaluated the non-learnable encoders showed a significantly larger variation in the quality of generated images between experiments. Therefore, investigating ways to increase the reliability of, or decrease the variation in, the generative performance is the key topic that has to be researched. For the partially non-learnable encoder, it is interesting to investigate its benefits in VAE variants with a different architecture, such as VAE-GAN, f-VAE-GAN-D2, or selfVAE. Based on the conclusions that were drawn from this research it is likely that the performance of these models will be the same when using a partially non-learnable encoder. However, with a partially non-learnable encoder, the computational time per epoch will probably significantly decrease. Altogether, researching the use of non-learnable encoders and partially non-learnable encoders is of importance to understand where the power of a VAE comes from, and to improve the computational efficiency of training VAEs respectively.

Bibliography

Asperti, A. and Trentin, M. (2020). Balancing reconstruction error and kullback-leibler divergence in variational autoencoders. *IEEE Access*, 8:199440–199448.

Ballé, J., Laparra, V., and Simoncelli, E. P. (2017). End-to-end optimized image compression. In *5th International Conference on Learning Representations*. International Conference on Learning Representations, ICLR.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J. H. (2019). Invertible residual networks. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, pages 894–910. International Machine Learning Society (IMLS).

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag, New York, first edition.

Bond-Taylor, S., Leach, A., Long, Y., and Willcocks, C. G. (2021). Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models.

Bukovcikova, Z., Sopiak, D., Oravec, M., and Pavlovicova, J. (2017). Face verification using convolutional neural networks with Siamese architecture. In *Proceedings Elmar - International Symposium Electronics in Marine*, volume 2017-Septe, pages 205–208. Croatian Society Electronics in Marine - ELMAR.

Carbajal, G., Richter, J., and Gerkmann, T. (2021). Disentanglement Learning for Variational Autoencoders Applied to Audio-Visual Speech Enhancement. *2021 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*.

Chen, R. T., Behrmann, J., Duvenaud, D., and Jacobsen, J. H. (2019). Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, volume 32. Neural information processing systems foundation.

Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column Deep Neural Networks for Image Classification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.

Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2016). Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders. *ICLR 2017*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Eisenstein, J. (2018). *Natural Language Processing*. MIT Press, Cambridge.

Fan, Y., Wen, G., Li, D., Qiu, S., Levine, M. D., and Xiao, F. (2020). Video anomaly detection and localization via Gaussian Mixture Fully Convolutional Variational Autoencoder. *Computer Vision and Image Understanding*, 195.

Flach, P. (2012). *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, Cambridge, first edition.

Fukushima, K. (1980). Biological Cybernetics Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biol. Cybernetics*, 36:202.

Gatopoulos, I. and Tomczak, J. M. (2020). Self-Supervised Variational Auto-Encoders. *Entropy*, 23(6):747.

Ghosh, P., Losalka, A., and Black, M. J. (2019a). Resisting adversarial attacks using Gaussian mixture variational autoencoders. In *33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 541–548. AAAI Press.

Ghosh, P., Sajjadi, M. S. M., Vergari, A., Black, M., and Schölkopf, B. (2019b). From Variational to Deterministic Autoencoders. *ICLR 2020*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Journal of Machine Learning Research*, volume 9, pages 249–256.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Guo, C., Zhou, J., Chen, H., Ying, N., Zhang, J., and Zhou, D. (2020). Variational Autoencoder with Optimizing Gaussian Mixture Model Priors. *IEEE Access*, 8:43992–44005.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 Inter, pages 1026–1034.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *Advances in Neural Information Processing Systems*, 2017-December:6627–6638.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Hoogendoorn, M. and Funk, B. (2018). *Machine Learning for the Quantified Self: On the Art of Learning from Sensory Data*, volume 35. Springer, Cham, first edition.

IBMCorp. (2017). *IBM SPSS Statistics for Windows*. IBM Crop, Armonk, NY.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd International Conference on Machine Learning, ICML 2015*, volume 1, pages 448–456. International Machine Learning Society (IMLS).

Jiang, S., Chen, Y., Yang, J., Zhang, C., and Zhao, T. (2019). Mixture variational autoencoders. *Pattern Recognition Letters*, 128:263–269.

Jiang, Z., Zheng, Y., Tan, H., Tang, B., and Zhou, H. (2016). Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. *IJCAI International Joint Conference on Artificial Intelligence*, 0:1965–1972.

Joo, W., Lee, W., Park, S., and Moon, I. C. (2020). Dirichlet Variational Autoencoder. *Pattern Recognition*, 107.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243.

Kumar, A. and Srivastava, S. (2020). Object Detection System Based on Convolution Neural Networks Using Single Shot Multi-Box Detector. In *Procedia Computer Science*, volume 171, pages 2610–2617. Elsevier B.V.

Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. (2015). Autoencoding beyond pixels using a learned similarity metric. *33rd International Conference on Machine Learning, ICML 2016*, 4:2341–2349.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*.

Liu, S., Liu, J., Zhao, Q., Cao, X., Li, H., Meng, D., Meng, H., and Liu, S. (2020). Discovering influential factors in variational autoencoders. *Pattern Recognition*, 100.

Long, Y., Gong, Y., Xiao, Z., and Liu, Q. (2017). Accurate object localization in remote sensing images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 55(5):2486–2498.

Louizos, C., Swersky, K., Li, Y., Welling, M., and Zemel, R. (2016). The variational fair autoencoder. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Mazaheri, A. and Shah, M. (2021). Video Generation from Text Employing Latent Path Construction for Temporal Modeling.

Morrow, A., Shankar, V., Petersohn, D., Joseph, A., Recht, B., and Yosef, N. (2017). Convolutional Kitchen Sinks for Transcription Factor Binding Site Prediction.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197.

Netzer, Y. and Wang, T. (2011). Reading digits in natural images with unsupervised feature learning. *Nips*, pages 1–9.

Pang, B., Nijkamp, E., Han, T., and Wu, Y. N. (2021). Generative Text Modeling through Short Run Inference. *EACL 2021 - 16th Conference of the European Chapter*

of the Association for Computational Linguistics, Proceedings of the Conference, pages 1156–1165.

Perugachi-Diaz, Y., Tomczak, J. M., and Bhulai, S. (2020). i-DenseNets. *3rd Symposium on Advances in Approximate Bayesian Inference*, pages 2020–2021.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines — Proceedings of the 20th International Conference on Neural Information Processing Systems.

Rahimi, A. and Recht, B. (2009). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, pages 1313–1320.

Raiko, C. K., Maaløe, T. ., Sønderby, L., and Winther, S. K. (2016). How to Train Deep Variational Autoencoders and Probabilistic Ladder Networks. Technical report.

Reynolds, D. (2015). Gaussian Mixture Models. *Encyclopedia of Biometrics*, pages 827–832.

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *32nd International Conference on Machine Learning, ICML 2015*, volume 2, pages 1530–1538. International Machine Learning Society (IMLS).

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ruthotto, L. and Haber, E. (2021). An introduction to deep generative modeling. *GAMM Mitteilungen*, 44(2).

Saito, S., Hu, L., Ma, C., Ibayashi, H., Luo, L., and Li, H. (2018). 3D hair synthesis using volumetric variational autoencoders. In *SIGGRAPH Asia 2018 Technical Papers, SIGGRAPH Asia 2018*. Association for Computing Machinery, Inc.

Shin, J. E., Riesselman, A. J., Kollasch, A. W., McMahon, C., Simon, E., Sander, C., Manglik, A., Kruse, A. C., and Marks, D. S. (2021). Protein design and variant prediction using autoregressive generative models. *Nature Communications*, 12(1):1–11.

Tomczak, J. M. and Welling, M. (2016). Improving Variational Auto-Encoders using Householder Flow.

Tomczak, J. M. and Welling, M. (2018). VAE with a vampprior. In *International*

Conference on Artificial Intelligence and Statistics, AISTATS 2018, pages 1214–1223. PMLR.

Tumpach, J. and Holena, M. (2020). Online Malware Detection with Variational Autoencoders. *ceur-ws.org*.

Van Den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. (2018). Sylvester normalizing flows for variational inference. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, volume 1, pages 393–402. Association For Uncertainty in Artificial Intelligence (AUAI).

Wang, J. and Li, Z. (2018). Research on Face Recognition Based on CNN. In *IOP Conference Series: Earth and Environmental Science*, volume 170, page 32110. Institute of Physics Publishing.

Wei, R., Garcia, C., El-Sayed, A., Peterson, V., and Mahmood, A. (2020). Variations in Variational Autoencoders - A Comparative Evaluation. *IEEE Access*, 8:153651–153670.

Whitley, B. E., Kite, M. E., and Adams, H. L. (2012). *Principles of research in behavioral science*. Taylor and Francis, New York, third edition.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2017). *Data Mining Practical Machine Learning Tools and Techniques*. Elsevier, Cambridge, fourth edition.

Xian, Y., Sharma, S., Schiele, B., and Akata, Z. (2019). f-VAEGAN-D2: A Feature Generating Framework for Any-Shot Learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:10267–10276.

Yeung, S., Kannan, A., Dauphin, Y., and Fei-Fei, L. (2017). Tackling Over-pruning in Variational Autoencoders.

Zhao, S., Song, J., and Ermon, S. (2017). InfoVAE: Information Maximizing Variational Autoencoders.

Appendix A

Derivation of ELBO

A.1 ELBO for Standard Normal Prior

The source of data is some distribution. When modeling a stochastic process one can choose to model it with, for instance, a normal distribution, exponential distribution, or Poisson distribution. However, data that originates from the real world tends to follow unknown distributions. For instance, the images (read pixels in the images) that are part of the MNIST data set have a certain distribution that is nearly impossible to be modeled by a random variable. The goal of a generative model is to approximate this specific distribution $p_\vartheta(x)$ in an unsupervised manner. This approximation takes place via a latent representation z. The marginal probability distribution can then be formalized.

$$p(x, z) = p(x|z)p(z) \tag{A.1.1}$$

The marginal likelihood function with respect to x can then be obtained by integrating over z.

$$p_\vartheta(x) = \int p(x|z)p(z)dz \tag{A.1.2}$$

While this integral is tractable, it is not easy to compute. Therefore a type of approximate inference is generally used; namely variational inference. In the context of VAEs the decoder (i.e. $p(x|z)$) and the prior (i.e. $p(z)$) tend to be parameterized by θ and λ respectively. In addition, the logarithm is taken of this equation.

$$ln\, p_\vartheta(x) = ln \int p_\theta(x|z) p_\lambda(z) dz \tag{A.1.3}$$

The next step is to simply multiply the inner part of the integral by $\frac{q_\phi(z|x)}{q_\phi(z|x)} = 1$. The term $q_\phi(z|x)$ can be seen as the encoder that computes z, parameterized by ϕ.

$$ln\, p_\vartheta(x) = ln \int \frac{q_\phi(z|x)}{q_\phi(z|x)} p_\theta(x|z) p_\lambda(z) dz \tag{A.1.4}$$

Now the expectation with respect to z is taken and Jensen's inequality is applied. If X is a random variable and φ is a convex function, then Jensen's inequality states $\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(x)]$.

$$ln\, p_\vartheta(x) = ln\, \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\frac{p_\theta(x|z) p_\lambda(z)}{q_\phi(z|x)}] \tag{A.1.5}$$

$$ln\, p_\vartheta(x) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, \frac{p_\theta(x|z) p_\lambda(z)}{q_\phi(z|x)}] \tag{A.1.6}$$

The next two steps are simply rewriting the terms of the equation.

$$ln\, p_\vartheta(x) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\theta(x|z) + ln\, p_\lambda(z) - ln\, q_\phi(z|x)] \tag{A.1.7}$$

$$ln\, p_\vartheta(x) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\theta(x|z)] - \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\lambda(z) - ln\, q_\phi(z|x)] \tag{A.1.8}$$

Finally the ELBO is obtained, consisting of the reconstruction error and the regularization term.

$$ELBO = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[ln\, p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Error}} - \underbrace{KL[q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]}_{\text{Regularization Term}} \tag{A.1.9}$$

A.2 ELBO for RealNVP flow-prior

The basic idea of deriving the ELBO for RealNVP flow-prior is the same as the derivation for the standard ELBO in A.1. However, an additional term is added; the logarithm of the Jacobian-determinant. Intuitively the Jacobian-determinant reflects

the change in volume. A Jacobian-determinant of 1 implies no change of volume, while a Jacobian-determinant larger than 1 means that the volume has increased. Lastly a Jacobian-determinant between 0 and 1 shows that the volume has decreased. For a flow in a RealNVP flow-prior the permutation layer has a Jacobian-determinant of 1, because it does not affect the volume. In contrast volume changes may appear in the coupling layers. The Jacobian is relatively easy to compute:

$$\mathbf{J} = \begin{bmatrix} \mathbf{I}_{d\ x\ d} & \mathbf{0}_{d\ x\ (D-d)} \\ \frac{\delta y_b}{\delta x_a} & diag(exp(s(x_a))) \end{bmatrix} \tag{A.2.1}$$

where, $s(x_a)$ refers to the scaling network of the coupling layer. Note that $D-d$ reflects the cutoff point of $x = [x_a, x_b] = [x_{1:d}, x_{d:D}]$. The determinant is then given by:

$$det(\mathbf{J}) = \prod_{j=1}^{D-d} exp\left(s(x_a)\right)_j = exp\left(\sum_{j=1}^{D-d} s(x_a)_j\right) \tag{A.2.2}$$

The only two steps that are left are to take the logarithm and to show what happens if there are multiple flows. By taking the logarithm the exponent cancels out and with K flows the logarithm of the Jacobian-determinant is simply summed for each flow. Thus in the equation below $s_i(x_j^i)$ refers to the scaling network of the ith flow.

$$ln\ det(\mathbf{J}) = \sum_{i=0}^{K} \sum_{j=1}^{D-d} s_i(x_a^i)_j \tag{A.2.3}$$

Appendix B

Detailed Model Descriptions

	Layer (Activation)	Input Shape	Output Shape
Encoder	Linear (Leaky ReLU)	32*32*C	256
	Linear (Leaky ReLU)	256	128
	Linear (Leaky ReLU)	128	64
	Linear	64	M*N
Decoder	Linear (Leaky ReLU)	M	128
	Linear (Leaky ReLU)	128	256
	Linear (Leaky ReLU)	256	512
	Linear (Leaky ReLU)	512	1024
	Linear	1024	32*32*C

Note. C = Number of colour channels. M = Number of latent features. N = 2 if the encoder provides a mean and measure of variance.

Table B.1: Architecure of the VAEs with fully connected layers.

	Layer (Activation)	Input Shape	Output Shape
Encoder	Conv2d (Leaky ReLU)	32x32xC	32x32x4
	Maxpool	32x32x4	16x16x4
	Conv2d (Leaky ReLU)	16x16x4	16x16x8
	Maxpool	16x16x8	8x8x8
	Conv2d (Leaky ReLU)	8x8x8	8x8x16
	Maxpool	8x8x16	4x4x16
	Conv2d	4x4x16	2x2x(M*N)
Decoder	Conv2d (Leaky ReLU)	2x2xM	2x2x64
	Upsample	2x2x64	4x4x64
	Conv2d (Leaky ReLU)	4x4x64	4x4x32
	Upsample	4x4x32	8x8x32
	Conv2d (Leaky ReLU)	8x8x32	8x8x16
	Upsample	8x8x16	16x16x16
	Conv2d (Leaky ReLU)	16x16x16	16x16x8
	Upsample	16x16x8	32x32x8
	Conv2d	32x32x8	32x32xC

Note. All Conv2d layers have kernel size 3, stride 1, and padding 1, except for the last Conv2d layer of the encoder, which has padding 0. C = Number of colour channels. M = Number of latent features. N = 2 if the encoder provides a mean and measure of variance.

Table B.2: Architecure of the VAEs with convolutional layers.

Appendix C

List of Figures

Appendix D

List of Tables