Unsupervised Feature Extraction of Clothing Using Deep Convolutional Variational Autoencoders

FREDRIK BLOM
Unsupervised Feature Extraction of Clothing Using Deep Convolutional Variational Autoencoders

FREDRIK BLOM
Abstract

As online retail continues to grow, large amounts of valuable data, such as transaction and search history, and, specifically for fashion retail, similarly structured images of clothing, is generated. By using unsupervised learning, it is possible to tap into this almost unlimited supply of data. This thesis set out to determine to what extent generative models – in particular, deep convolutional variational autoencoders – can be used to automatically extract representative features from images of clothing in a completely unsupervised manner. In reviewing variations of the autoencoder, both in terms of reconstruction quality and the ability to generate new realistic samples, results suggest that there exists an optimal size of the latent vector in relation to the image data complexity. Furthermore, by weighting the latent loss and generation loss in the loss function, it was possible to disentangle the learned features such that each feature captured a unique defining characteristic of clothing items (here t-shirts and tops).
Sammanfattning

I takt med att E-handeln fortsätter att växa och kunderna i ökad utsträckning rör sig online, genereras stora mängder värdefull data, exempelvis transaktions- och sökhistorik, och specifikt för klädeshandeln, välstrukturerade bilder av kläder. Genom att använda oövervakad maskininlärning (unsupervised machine learning) är det möjligt att utnyttja denna, nästan oberänsade mängd data. Detta arbete syftar till att utreda i vilken utsträckning generativa modeller, särskilt djupa självkodande neurala faltningsnätverk (deep convolutional variational autoencoders), kan användas för att automatiskt extrahera definerande drag från bilder av kläder. Genom att granska olika varianter av självkodaren framträder en optimal relation mellan storleken på den latenta vektorn och komplexiteten på den bilddata som nätverket tränades på. Vidare noterades att dragen kan fördelas unikt på variablerna, i detta fall t-shirts och toppar, genom att vikta den latenta förlustfunktionen.
Acknowledgements

Firstly, I would like to thank Mårten Björkman for being my supervisor. I am grateful for his input and guidance, but also for introducing me to image processing and computer vision through his courses at KTH. Without them, I wouldn’t have chosen this particular field. Furthermore, I also want to thank Danica Kragic Jensfelt for being my examiner.

Secondly, I want to thank Yann Hendel for our biweekly meetings. His council and support throughout the semester has helped me to continuously push the thesis forward.

Thirdly, I want to thank Eva Henje Blom, Jonathan Weisblatt and Johanna Elming for proof-reading and support. I would also want to thank Philip Hammarskiöld and Rolf Blom for various types of counsel.

Finally, I want to take this opportunity to thank my friends at KTH: Harald Agering, Marcus Ahlgren, Axel Bååthe, Patric Bonnier, Jacob Jonsson, Adam Ridemar, Adam Von Peltzer and Oscar Xing Luo. All of whom have contributed to make my time at KTH a joyful and rewarding experience.
## Contents

1 Introduction ............................................. 1  
1.1 Objective ............................................. 2  
1.2 Research Question ...................................... 2  
1.3 Scope .................................................. 3  
1.4 Ethical and Societal Aspects, and Sustainability .......... 3  

2 Background .............................................. 5  
2.1 Fashion Retail ........................................... 5  
2.1.1 Industry Insights ...................................... 5  
2.1.2 Clothing Features ...................................... 6  
2.2 Artificial Neural Networks (ANN) ......................... 6  
2.2.1 Convolutional Neural Networks (CNN) ................ 8  
2.2.2 Autoencoders .......................................... 9  
2.2.3 Techniques of Neural Networks ....................... 11  
2.3 Generative Models ....................................... 12  
2.3.1 Maximum Likelihood Estimation (MLE) ............... 12  
2.3.2 Explicit Density Estimation Models .................. 13  
2.3.3 Implicit Density Estimation Models .................. 15  
2.3.4 Variational Autoencoders (VAE) ..................... 15  

3 Related Work ............................................. 20  
3.1 Deep Convolutional Neural Network Architecture ........ 20  
3.2 Generative Model Architecture .......................... 21  
3.3 Generative Model Methods .............................. 22  
3.3.1 Constrained Variational Autoencoders (β-VAE) ....... 22  
3.4 Performance and Evaluation ............................. 23  

4 Method .................................................... 24  
4.1 Hardware and Software .................................. 24  
4.1.1 Software .............................................. 24
Chapter 1

Introduction

Online retail is growing rapidly and an increasing portion of total revenue is generated from online sales. More global and accessible stores create a competitive landscape where it is crucial to understand the latest trends and what you and your competitors are currently selling. During 2017, retailers in the US closed a record number of stores to focus on their most attractive locations [1]. As retailers transition online, the collection and use of data become ever more important to analytics. The recent developments in Convolutional Neural Networks (CNNs) and the large amount of similarly structured image data generated from online stores (as seen in Figure 1.1) create a unique opportunity for computer vision and machine learning. Many of the state-of-the-art methods currently used in machine learning are so-called super-

\[ \text{Figure 1.1: Two examples of black t-shirts with print and short sleeves. Is it possible to extract the features that make them similar/unique and even compare/cluster them? The images were downloaded from (a)[2] and (b)[3].} \]
vised and require a large amount of labeled data. Acquiring such data can be expensive and time-inefficient, so in this regard, unsupervised methods are preferable. In addition, it is possible that the traditional labels used when describing generic features of clothing items such as color, sleeve length or print, might not contain enough semantic information to capture more latent features involved in buying-decisions and brand association such as texture, feel and appeal.

1.1 Objective

The aim of the thesis is to develop a method capable of automated feature extraction from clothing photos that can be utilized to compare clothing images and draw conclusions about similarities and differences. Unsupervised deep learning is an active area of research with a variety of sub-fields. This thesis will study one such sub-field, namely generative models and in particular different variations of the autoencoder[4][5][6]. It is essential to understand how to optimize feature extraction, specifically when working with similarly structured images of clothing, and how these extracted features can be used to compare and potentially map items.

The thesis will review relevant state-of-the-art methods in machine learning and generative models. A successful comparison technique of unlabeled image data could help create an effective and unique overview of the current market trends through mapping and help a company to understand its own inventory, its competitors and unexploited areas of clothing categories within different product types.

1.2 Research Question

To what extent can autoencoders be used for feature extraction in the context of clothing images? What is the best approach when working with unlabeled similarly structured images of clothes? In addition, what visualization techniques can be utilized to make the extracted features more comprehensible? The latter is important for image-based comparison and mapping of clothing items.
1.3 Scope

The thesis is strictly limited to off-model product photos of t-shirts and tops (only clothes on a monotone generic bright background). Any evaluation of the method will be done with this in mind. The focus of the thesis is well-structured images rather than images in a "non-studio" setting or images in-the-wild. The primary focus is feature extraction and hence not optimizing with respect to image quality in any generated images. The purpose of using different variations of the autoencoder is to investigate the effect on the extracted features and what they represent, as well as to study whether it is possible to disentangle the features. The dataset is composed of images from online retailers, with the exception of possible additions of similar non-licensed images from other sources, such as ImageNet, to facilitate the publishing of photos. All of the included images are chosen in order to eliminate the potential effects from lighting, background colour, angle etc. Furthermore, the thesis will focus on answering the following questions:

- What is the most suitable evaluation techniques for the different versions of the autoencoder?
- What is the effect of different dimensions of the latent vector, both in term of possible effects on reconstruction quality but also what the features represent?
- How can different visualization techniques be used to gain a deepened understanding of how the autoencoders operate and what type of features are captured?

1.4 Ethical and Societal Aspects, and Sustainability

Working with generative models raises some questions regarding licensing. Is a reconstructed image still under copyright or can it be deemed as new art-work? If so, small, incremental adjustments, over time, can be made to eventually cause the image to become unrecognizable from the first. It seems as if there is no clear and easy answer
to this question. Another potential problem is the integrity of individuals in the images. Even if this is not a problem for this thesis, similar methods can be used on different datasets that include people.

As online retail is getting bigger, so are the retail giants such as Amazon. They will have access to extreme amounts of data such as transaction and search history, product specifications and image data. These data points can be used to create sophisticated suggestions, as well as marketing and ordering algorithms that might prove difficult to compete with. As fast-fashion becomes even faster, combined with the ability to more quickly identify trends, it could fuel unnecessary and impulsive consumption, which in turn could negatively impact the environment.
Chapter 2

Background

The following chapter provides some relevant information to facilitate the comprehension of the thesis as a whole. The first section offers some potentially necessary information regarding the fashion retail industry in order to understand the academic and commercial contribution, and the connection between them. The second section is more technical and presents general information regarding machine learning and neural networks, as well as autoencoders and its implementation. The third and final section builds on the previous two and explains in more depth generative models which can prove helpful in understanding the contributions discussed in the Related Work (see Section 3), as well as the specific methods used in this thesis.

2.1 Fashion Retail

The fashion retail industry has a central role in this thesis. The method was chosen because of its suitability when working with fashion clothing, and not the other way around. The following section begins with high-level industry insights, followed by online-specific information and finally connects to the importance of clothing features.

2.1.1 Industry Insights

On opposite sides of the fashion industry are luxury and fast-fashion. Starting with luxury fashion, there are two fashion companies in the top five largest luxury goods companies by global sales, namely Moët Hennessy-Louis Vuitton SE (LVMH) and Kering SA. LVMH owns lux-
ury brands, Louis Vuitton, Fendi, Bvlgari and Marc Jacobs; and Kering owns Gucci, Bottega Veneta and Saint Laurent, to name a few examples [7]. These brands have a few shows each year where they present new collections which dictate much of the global fashion trends. Following these trends are the fast-fashion retailers, the two largest being Zara and H&M. These fast-fashion brands have come under new competition from even faster competitors. For example, ASOS (a major online retailer) is capable of producing products with just 2–4 weeks lead-time. In relation to Zara’s 5-week or H&M’s 6-9-month cycles this creates new, heretofore unseen challenges for the more traditional retailers [8][9]. These short cycles create a new "retailer dilemma of product shortages versus excessive inventory, and ensuing markdowns and lower margins" [8]. It is therefore crucial to closely monitor what is being sold and to optimize inventory.

2.1.2 Clothing Features

When browsing a fashion e-commerce website, there is often a variety of filtering options which might include: men or women, tops or bottoms, t-shirts, shirts, tank-tops or sweaters. In addition to these high-level features there might be more detailed options such as long/short sleeve and print/clean. Apart from helping customers to navigate the website, it also plays an essential role in solving the retailer dilemma stated above. The features can be used to keep track of products which are being sold and assess whether it is likely that there will be any cannibalization in sales from new items on current ones. Two products can have similar descriptions but be completely different and hence cause either poor or time-inefficient allocation decisions. It is also crucial in understanding customers, such as predicting likely future purchases given previous ones.

2.2 Artificial Neural Networks (ANN)

In order to keep this section short, it will only briefly introduce the theory behind Artificial Neural Networks (ANN) . For a more in-depth review, there is a plenitude of introductory material (for example [10]).

Successfully predicting the output of unseen data relies on the method’s ability to learn and represent the functional mapping between input
and output. Increased complexity of this underlying correlation puts tougher demands on more complex function approximations. In some sense this is the power of ANN, its ability to learn arbitrarily complex functional mappings [11]. The most basic ANN generally consists of an input layer, one or several hidden layers, and an output layer. If the network consists of two or more hidden layers it is often referred to as being deep. Furthermore, if all neurons between two adjacent layers are connected it is called fully connected (a deep neural network with fully connected layers is shown in Figure 2.1).

**Figure 2.1:** An Artificial Neural Network graph consisting of a fully connected input layer with eight neurons, three fully connected hidden layers with nine neurons each and a fully connected output layer with four neurons. The image was downloaded from [12].

The outputs from the previous layer are weighted and summed up in each of the following neurons (in a fully connected layer the output from each neuron is added to each of the following neurons, represented above with lines). The sum is then passed through activation functions (also called non-linearity functions) and continues as input to the following layer; this process is often referred to as forward propagation. These activation functions have two important properties. The first is non-linearity; without it, the network would simply become a linear regression model and hence lose the ability to learn and represent more sophisticated functional mappings. The second important property is differentiability: in training, the weights of the network are updated in order to reduce the discrepancy between out-
put layer approximation and the ground truth. This process of tracing back through the network is called back-propagation. The activation function being differentiable allows for effective derivation and backward traversal to identify how to update the weights. There are plenty of activation functions to choose from, but the most widely used is called Rectified-Linear-Unit or simply ReLU and is defined as follows [11][10]:

\[
\text{ReLU}\left( \sum_i w_i x_i + b \right) = \max\left( 0, \sum_i w_i x_i + b \right) \tag{2.1}
\]

where \( w_i \) are the weights, \( x_i \) are the outputs from the previous layers and \( b \) is a bias that is often used.

### 2.2.1 Convolutional Neural Networks (CNN)

The CNN architecture was first successfully implemented in the 90’s in a network called LeNet [13] but CNNs did not gain traction until a network called AlexNet [14] won ImageNet Large Scale Visual Recognition Competition (ILSVRC) challenge [15] in 2012. CNNs are currently widely used when working with visual images due to two properties: (1) the number of trainable weights scale well with larger image sizes; and (2) the ability to handle spatial information. The following section will briefly describe the basic operations involved in CNNs when handling 3D visual images. The first two dimensions, height and width, are somewhat obvious and are usually given in number of pixels. The third dimension, depth, is given in the number of color channels. A grey-scale image only has one while images in color often have three channels: Red, Green and Blue (or simply RGB).

In the center of CNN is the convolution operation. It includes a kernel (also called filter) of variable size that is convolved (or slided) across the image applying dot products to the pixel values. Generally, a number of different kernels with trainable weights are applied and the dot products are collected in so-called activation maps. These activation maps are themselves convolved into new activation maps throughout the network layers. Figure 2.2 illustrates how an input image of size 32x32x3 is convolved with six 5x5x3-kernels to form the first activation map. Each element in the activation maps is transformed with an activation function as previously described.
These convolution layers are often alternated with sub-sampling operations to reduce the dimensionality of the data. This is usually done through max or average pooling. For example, in max pooling only the largest value in a given domain, say 2x2x1, is saved and thereby effectively reducing the image dimension by two.

\[ \text{The result of taking a dot product between the filter and the small 5x5x3 chunk of the image, i.e. 5x5x3 = 75 dimensional dot product + bias.} \]

Figure 2.2: An illustration of a 32x32x3 input image (32 pixels wide, 32 pixel high and three color channels deep) being convolved with six 5x5x3 kernels. The image and accompanying text was downloaded from [16].

Incomplete convolutions near the edges are controlled by using different variants of padding. Either the incomplete convolutions are omitted or a so-called zero-padding is added to the edges to control the output size. At the end of a succession of convolution and pooling layers, the activation map is often larger in depth than in width and height. If necessary, these activation maps are often flattened and formed into a fully connected layer. The final output vector can now be used much like the output vector of an ANN such as for classification [10].

2.2.2 Autoencoders

Before introducing autoencoders, it is important to make a clear distinction between supervised and unsupervised learning. Supervised
learning often handles labeled data in order to make predictions on similarly structured unseen data. For example, an output vector of a classifier would compare the computed output vector with a labeled one-hot vector in order to calculate a loss that will mandate how the network weights are updated. If the input data is defined as \( x \) and the output data as \( y \), the goal is to estimate \( p(y|x) \) (what is \( y \) given \( x \)).

One major disadvantage is the requirement of marked or labeled data. Since neural networks often require huge datasets, finding or creating such data can be both time-inefficient and costly (not seldom one needs to manually label thousands or tens of thousands of images). This is arguably the biggest advantage of unsupervised learning: it tries to learn some underlying, and possibly hidden, distribution of the data \( p(x) \). It does not require labels but can instead learn from the data itself (effectively having unlimited access to training data)[4]. One such powerful unsupervised method is the autoencoder (see Figure 2.3 for visualization).

![Figure 2.3: A visualization of an autoencoder that uses colored images as input and output. The image was downloaded from [17].](image)

The idea behind autoencoders is relatively straightforward and works almost like a copying machine. It consists of three distinct parts: an encoder (also called convolution network), a latent vector (also called bottle-neck or code) and a decoder (also called deconvolution network). The encoder network reduces the dimensionality of the input image; for example, a CNN would apply a series of convolution and pooling operations. The output from the encoder is, if necessary, flattened and usually connected to the latent vector through a fully connected layer. The dimensionality of the latent vector is variable but should generally
be of lower dimension than the input image in order to enforce encoding (or compression). If each of the input pixels were random in the sense that they were drawn from an i.i.d. Gaussian, encoding the data with fewer variables than pixels would be difficult. However, as often is the case with images, there are some correlated features that can be found. The latent vector is then fed into the decoder network which in some sense is an inverse of the encoder network and again increases the dimensionality. In a CNN, this would include applying a series of deconvolutions and unpooling layers. The output, or reconstruction, should be of identical dimension as the input image. A loss function is then calculated as the difference between the input image and the reconstructed output image; for example, the pixel-wise L2-norm. The network weights are subsequently updated in order to minimize the loss, effectively improving the reconstruction quality. As discussed, the latent vector should arguably encode some latent features of the image needed in order to successfully reconstruct the image. This latent vector can be used in a variety of ways such as clustering or to train a classifier [18]. An additional way to view the autoencoder is in terms of distributions. The encoder is trying to learn $p(z|x)$ where $z$ is the latent vector, and the decoder in turn tries to learn $p(x|z)$ [4].

2.2.3 Techniques of Neural Networks

The previous two sections briefly introduced concepts surrounding ANNs and CNNs. This section offers complementing information regarding training and implementation of networks.

Hyperparameters

The settings or design variables that determine the workings of a network are often referred to as hyperparameters. They includes anything from how much the weights should be adjusted to kernel sizes or number of layers.

Learning rate is the magnitude with which you update the weights in the network. There is often an ideal learning rate that can be unique for different architectures. A too-large learning rate can cause the network to diverge, and a too-small can hinder the network from reaching its optimum, and training generally takes longer. Learning rate is often somewhere between $10^{-2} - 10^{-5}$. 
Kernel size is the size of the filter that is being convolved. It is something that needs to be empirically tested. For example, it depends on stride length and the application of the network. More information can be found in Related Work.

2.3 Generative Models

The following section provides some high-level information regarding generative models and some motivation for their use. The section is, to a large extent, inspired by Ian Goodfellow’s tutorial on Generative Adversarial Networks [19] and the Deep Learning Book [4]. A generative model is here defined: a model is considered generative if it is capable of learning an estimated probability distribution model $p_{\text{model}}$ (or density estimation) of training data that is sampled from some true but hidden distribution $p_{\text{data}}$. Depending on what the intended use is, there are different generative models with both implicitly and explicitly stated density estimations.

Generative models have several interesting features which make them an interesting field within machine learning. The estimated model can be used to gain insights into both representation and manipulation of high-dimensional density estimation. Furthermore, it can be used in reinforcement learning, for example, when working with time-series, it can be used to estimate future states. Another interesting area of research is semi-supervised learning, where a portion of the labels is missing. The model can be trained on the unlabeled data and integrated to labeled datasets to draw desired conclusions.

2.3.1 Maximum Likelihood Estimation (MLE)

To facilitate the comparison of models in this work, all generative models will use Maximum Likelihood as estimation method (MLE). This is not necessarily the case, however most models can be modified to do so [19].

Consider a dataset $X$ consisting of data points independently sampled from $p_{\text{data}}(x)$ (as defined above). Now set $p_{\text{model}}(x; \theta)$ to the parametric probability distribution estimation of $p_{\text{data}}(x)$ over $\theta$. The MLE for $\theta$ is
equivalent to:

\[ \theta_{ML} = \arg\max_{\theta} \mathbb{E}_{\hat{p}_{data}} \left[ \log p_{model}(x^{(i)}; \theta) \right] \]  

(2.2)

where \( \hat{p}_{data} \) is the empirical distribution defined by \( X \). The optimization problem above is essentially a minimization problem of the discrepancy between \( \hat{p}_{data} \) and \( p_{model} \). Hence, another way of looking at the MLE is through KL divergence, defined as follows:

\[ D_{KL}(\hat{p}_{data}||p_{model}) = \mathbb{E}_{x \sim \hat{p}_{data}} \left[ \log \hat{p}_{data}(x) - \log p_{model}(x) \right]. \]  

(2.3)

The right side only depends on the model, and hence training is equivalent to minimizing:

\[-\mathbb{E}_{x \sim \hat{p}_{data}} \left[ \log p_{model}(x) \right] \]  

(2.4)

or interchangeably minimizing the cross-entropy between the distributions [4].

The following two sections will briefly traverse down the two sides of the taxonomy tree shown in Figure 2.4, explicitly and implicitly defined density models, and, in doing so, cover their advantages and disadvantages, as well as method of likelihood maximization.

### 2.3.2 Explicit Density Estimation Models

The main advantage with explicitly defined density models is the simplicity of likelihood maximization (computed by directly evaluating Equation 2.3). However, this imposes a trade-off between tractability (the ability to compute the gradient) and complexity (the ability to model more complex functional mappings). As seen in Figure 2.4, there are two popular approaches that include using either tractable or approximate models.

#### Tractable Models

Tractable models are defined to ensure tractability and therefore require certain constraints on model structure. One of the most widely used tractable methods is Fully Visible Belief Nets (FVBN) [20]. It uses the chain rule to represent the probability distribution:

\[ p_{model}(x) = \prod_{i=1}^{n} p_{model}(x_i|x_1, \ldots, x_{i-1}) \]  

(2.5)
where each iteration depends on the previous. Therefore, it cannot be computed in parallel, which limits computational speed. State-of-the-art implementations use deep ANNs to compute each of the probability distributions. Another popular method is Nonlinear Independent Components Analysis (or nonlinear ICA) \[21\]. It uses a combination of precisely crafted transformations with simpler distributions over some latent variable to create complex distributions over \(x\). The major disadvantage of this method is the strict constraints on the transformations such as the requirement of the latent variable being of equal dimension as \(x\) \[4\].

**Approximate Models**

To bypass some of the drawbacks associated with the tractable models, one can instead use approximations of intractable density models. The two most commonly used approximation methods are variational methods and Markov Chain (stochastic). Variational approximations define a so-called lower bound as:

\[
\mathcal{L}(x; \theta) \leq \log p_{model}(x; \theta)
\]  

\[ (2.6) \]
and by maximizing $L$ it forced the smallest possible value (or simply lower bound) of the log-likelihood to increase (compare Equation 2.3). In cases where the log-likelihood is intractable, this can be utilized by defining a tractable $L$. The most widely used method is the Variational Autoencoder introduced by Kingma [5] and is further discussed in Section 2.3.4. One disadvantage worth mentioning of these types of methods is that the difference between $L$ and the true log-likelihood can cause the $p_{model}$ to learn something different than $p_{data}$, even under ideal conditions. The last explicitly defined density method to mention is Markov chain approximation. By sampling from $x' \sim q(x'|x)$ and adjust $q$ accordingly, convergence can be guaranteed (at least to some sample from $p_{model}$). However, it may be slow and computationally expensive [19].

2.3.3 Implicit Density Estimation Models

The right side of the taxonomy tree consists of models with implicit density functions. This does not allow for direct interaction with the model, and training is instead often done by sampling from $p_{model}$. This sampling can be done through Markov Chains (as discussed above) or by using a Generative Adversarial Network (GAN). The GAN framework was first developed by Ian Goodfellow [22].

2.3.4 Variational Autoencoders (VAE)

The Autoencoder was first introduced in Section 2.2.2 and the context, rather than explanation, of the VAE was later presented in Section 2.3.2. It is therefore appropriate here to offer a more in-depth review of the VAE. The VAE was first introduced in the paper Auto-Encoding Variational Bayes [5] by Diederik P. Kingma and Max Welling in 2014 and is perhaps the single most important publication for this thesis. Its contributions are reviewed with the help of [23][4].

Consider a generative model with the objective to generate data samples similar to training data. In the case of hand-written digits $^1$ the model can benefit from first deciding what digit to generate before it starts to adjust pixel values. The process might proceed as follows: (1)

$^1$Hand-written digits are commonly seen in literature related to generative models due to a popular dataset called MNIST containing ten thousands labeled samples
sample the code $z$ (or latent variable) from some uniform discrete prior distribution $p(z) = \{0, 1, \ldots, 9\}$; (2) generate the digit and ensure that it resembles the features seen in the training data [23].

More formally, consider an i.i.d. dataset $X = \{x^{(1)}, \ldots, x^{(n)}\}$ in which each datapoint $x^{(i)}$ has been generated from $z$ that is drawn from the prior $p(z; \theta_t) = p_{\theta_t}(z)$. This value is then used in some likelihood $p_{\theta_t}(x|z)$. The true parameter $\theta_t$ is hidden along with the latent variables $z^{(i)}$. The goal is to find an estimate $\theta$ of these parameters by maximizing the probability $p_{\theta}(x)$, which is given by law of total probability (marginalization):

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz. \quad (2.7)$$

In other words, increase the likelihood of the model producing data-points in the training set; and, in doing so, it will be more likely to generate other similar samples. However, the likelihood is intractable along with the posterior, the latter of which is given by Bayes’s theorem:

$$p_{\theta}(x|z) = \frac{p_{\theta}(z|x)p_{\theta}(x)}{p_{\theta}(z)} \quad (2.8)$$

The true intractable posterior is instead approximated with a recognition model $q_{\phi}(z|x)$ and the parameter $\phi$ is jointly learned with $\theta$. To connect back to Section 2.2.2, $q_{\phi}(z|x)$ is often referred to as an encoder and $p_{\theta}(x|z)$ as a decoder. The relationship between the encoder and the true posterior is important and can be investigated through the KL-divergence (the parameters and indices are dropped momentarily for convenience):

$$D_{KL}(q(z|x)||p(z|x)) = \mathbb{E}_{q(z|x)}\left[ \log q(z|x) - \log p(z|x) \right]. \quad (2.9)$$

An expectation over $p(z|x)$ of Bayes’s theorem yields:

$$\mathbb{E}_{q(z|x)}\left[ \log q(z|x) - \log p(x|z) - \log p(z) \right] + \log p(x) \quad (2.10)$$

Some additional manipulation gives the following equation:

$$\log p(x) = D_{KL}(q(z|x)||p(z|x)) + \mathbb{E}_{q(z|x)}\left[ \log p(x|z) \right] - D_{KL}(q(z|x)||p(z)) \quad (2.11)$$

or simply:

$$\log p(x) = D_{KL}(q(z|x)||p(z|x)) + \mathcal{L}(x) \quad (2.12)$$
where:

\[
L(x) = \mathbb{E}_{q(z|x)} \left[ \log p(x|z) \right] - D_{KL}(q(z|x) || p(z)).
\] (2.13)

Here, \( L \) is referred to as the variation lower bound. The objective is to maximize \( L(\phi; \theta; x^{(i)}) \) over parameters \( \phi \) and \( \theta \). However, gradient estimators over \( \phi \) in non-trivial and popular techniques i.e. (Naïve) Monte Carlo exhibit too high variance (see paper for details). The authors instead introduced the so-called reparameterization trick which offers an alternative way to sample from \( q_\phi(z|x) \) by using a differentiable transformation \( G_\phi(\epsilon, x) \) of the auxiliary noise variable \( \epsilon \). In order to keep this section as short as possible, the remaining part will focus on implementations using ANNs. A graph illustrating the implementation is found in Figure 2.5

Let \( p(z) = \mathcal{N}(z; 0, I) \) and set \( p_\theta(x|z) \) to be a multivariate Gaussian

\[
p_\theta(x|z) \text{ is intractable but is assumed to be approximately Gaussian and having an approximately diagonal co-variance. Let the variational approximate poste-}
\]

**Figure 2.5:** A graph illustrating a variational autoencoder with (right) and without (left) the reparameterization trick. The red coloured boxes are non-differentiable (sampling) while the blue are differentiable (losses). In this case the likelihood is Gaussian. The image was downloaded from [23].

or Bernoulli (depending on continuous or binary data) where the parameters are given from a fully connected layer. \( p_\theta(z|x) \) is intractable but is assumed to be approximately Gaussian and having an approximately diagonal co-variance. Let the variational approximate poste-
prior be defined as:

$$\log q_\phi(z^{(i)}|x) = \log \mathcal{N}(z; \mu^{(i)}, \Sigma^{(i)} I) \quad (2.14)$$

where:

$$\mu = W_ch + b_c$$
$$\log \Sigma^2 = W_bh + b_b \quad (2.15)$$
$$h = \text{activation}(W_ax + b_a).$$

The $z$ are sampled from $q_\phi(z|x^{(i)})$ by using the reparametrization trick:

$$z^{(i,l)} = G_\phi(x^{(i)}, \epsilon^{(l)}) = \mu^{(i)} + \Sigma^{(i)} \cdot \epsilon^{(l)} \quad (2.16)$$

where $\epsilon^{(l)} \sim \mathcal{N}(0, I)$. Since both the prior and the approximate posterior are Gaussian the KL divergence can be evaluated without approximation:

$$\int q_\theta(z) \log p(z) dz = \int \mathcal{N}(z; \mu, \Sigma^2) \log \mathcal{N}(z; 0, I) dz$$
$$= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_j (\mu_j^2 + \Sigma_j^2) \quad (2.17)$$

where $J$ is the dimension of the latent variable $z$. Furthermore:

$$\int q_\theta(z) \log q_\theta(z) dz = \int \mathcal{N}(z; \mu, \Sigma^2) \log \mathcal{N}(z; \mu, \Sigma^2) dz$$
$$= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_j (1 + \log(\Sigma_j^2)). \quad (2.18)$$

One can now form part of the lower bound:

$$-D_{KL}(q_\theta(z|x)||p_\theta(z)) = \int q_\theta(z) \left[ \log p_\theta(z) - \log q_\theta(z) \right] dz$$
$$= \frac{1}{2} \sum_j \left( 1 + \log(\Sigma_j^2) - \mu_j^2 - \Sigma_j^2 \right). \quad (2.19)$$

Finally the lower bound estimator is given by:

$$\mathcal{L}(\phi; \theta; x^{(i)}) \simeq \frac{1}{2} \sum_j \left( 1 + \log(\Sigma_j^2) - \mu_j^2 - \Sigma_j^2 \right) + \frac{1}{L} \sum_l \log p_\theta(x|z^{(l)}) \quad (2.20)$$
where $L$ is the number of samples per datapoint. Note that the second term is an approximation of the expectation in found in Equation 2.14 through sampling. As a closing remark, it is important to identify the property of the two terms in Equation 2.20. The first term functions as a regularizer and is often called latent loss. The second term is the expected reconstruction loss (or interchangeably called generation loss).
Chapter 3

Related Work

The following chapter builds on the more fundamental information previously presented. The purpose is to provide concrete findings gathered from relevant publications and literature that have been influential in developing the method. The chapter consists of two sections about network architecture and generative models.

3.1 Deep Convolutional Neural Network Architecture

In the paper Striving for simplicity: The All Convolutional Net [24] the authors found that often-used max-pooling layers could be replaced with a convolutional layer of stride length larger than one without affecting the accuracy in a number of image recognition benchmarks.

Very Deep Convolutional Networks for Large-Scale Image Recognition [25] investigates how the depth of CNNs affects accuracy when used in image recognition networks. It found that by using small convolutional filters (3x3) it allowed for a deeper network (16-19 layers) that achieved better results than previous state-of-the-art architectures. The results underline the benefits associated with increased depth when working with visual representations.
3.2 Generative Model Architecture

*Learning to Generate Chairs with Convolutional Neural Network* [26] found that their network could learn important features associated with reconstruction images of chairs rather than "learn all images by heart". The authors could generate new "unseen" viewpoints of chairs and even generate completely new chairs. They used 5x5 convolutional filters with unpooling for upsampling and ReLU activation functions. Furthermore, they implemented two models with two different image input sizes, 128x128 and 64x64. The larger image used 5 convolutional/unconvolutional layers while the smaller used only 4. The network was trained in a supervised fashion since it required labeled data.

*Variational Autoencoders for Deep Learning of Images, Labels and Captions* [27] implemented a deep convolutional variational autoencoder and utilized latent vectors for additional tasks, apart from reconstructing the image. It was used in a Bayesian Support Vector Machine (SVM) for labeling and a Recurrent Neural Network (RNN) for captioning.

*Deep Feature Consistent Variational Autoencoder* [28] used an alternative loss to the variational autoencoder. In terms of architecture, it uses a 4-layer-deep encoder network, followed by a latent vector of size 100. The encoder network uses an input image of size 64x64x3 with 4x4 filters and stride=2 (no dedicated pooling layer) with Leaky-ReLU as activation function. The final activation map is flattened and connected to a fully-connected layer of size 100. The design is not symmetric in the sense that the decoder network somewhat differs. It uses upsampling operations and 3x3 filters and stride=1. In addition, they use batch-normalization at each convolutional layer. As the height and width are reduced by a factor 2, number of filters is increased by a factor 2. See Figure 3.1 for graphs displaying the network architecture.
3.3 Generative Model Methods

3.3.1 Constrained Variational Autoencoders (β-VAE)

β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework [6] introduces a new unsupervised approach to automatically extract factorized latent representation. It introduces an additional hyperparameter $\beta$ to the variational autoencoder to balance the reconstruction and latent loss:

$$
L(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - \beta D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).
$$

The addition is straightforward but nonetheless impactful. With $\beta = 1$ there is no difference from the normal VAE, and setting $\beta = 0$ is similar to a normal autoencoder (essentially excluding to minimization of the latent loss in training). A fine-tuned $\beta$ allows for unsupervised disentangled feature learning and in contrast to other methods, the constrained VAE is stable in training and makes few assumptions. $\beta$ is tuned either with the use of labeled data or simply by visual inspection. Figure 3.2 shows the difference in learned features from a vanilla and constrained VAE. The left plot (A) shows how a constrained VAE is successful in disentangling the features (each latent variable representing a unique and distinguishable feature). The variable $z_2$ captures vertical position, $z_6$ horizontal position, $z_1$ scale, and $z_5$ and $z_7$ rotation. In contrast to the constrained VAE, the vanilla VAE is unsuccessful in disentangling the features. In the right plot (B), it is not possible to distinguish what each of the variables represent.
Figure 3.2: The image illustrates the difference in learned features between a traditional (right) and constrained (left) VAE. The image was downloaded from [6].

3.4 Performance and Evaluation

A Note on the Evaluation of Generative Models [29] offers insights into some of the difficulties involved with evaluating generative models. It highlights the importance of “understanding the trade-offs between different measures” and that different applications require different metrics. It makes the case that evaluations based on "visual fidelity" tends to be biased towards over-fitting models. On the other hand, high likelihood does not guarantee good-looking samples. There is no universal assessment of generative model performance.
Chapter 4

Method

The following chapter consists of three parts: hardware and software, feature extraction, and datasets. The first section offers some insights into the tools used in implementing the autoencoders. The second section provides some motivation to particular design choices and presents the network architecture, and the final part introduces the three datasets used.

4.1 Hardware and Software

4.1.1 Software

The programming language Python was used and implemented using Jupyter Notebooks executed locally and on a virtual machine. The machine learning package TensorFlow was used. TensorFlow is developed by Google and is available both in high-level and low-level API with a variety of programming of languages [30]. The structure from the Github repository [31] was used to initially set up the code structure of the network.

4.1.2 Hardware

Training deep CNNs requires a large amount of computations and can therefore be a time-consuming task when performed locally on personal computers. For this reason, after setting up the basic architecture on a smaller training set, a virtual machine was used to reduce the training time. Below are the specifications of the local computer
and virtual machine used:

**Computer**

MacBook Pro (13-inch, 2017) 3.5 GHz Intel Core i7, 8 GB 2133 MHz LPDDR3, Intel Iris Plus Graphics 650 1536 MB.

**Virtual Machine**

Amazon Web Services, AMI ID Deep Learning AMI (Ubuntu) Version 7.0. Instance Type: c4.4xlarge.

### 4.2 Feature Extraction

The generative model of choice is the variational autoencoder. However, three different versions of it will be used to investigate how to best optimize its use. The features are extracted as the variables of the latent vector between 5 convolutional and 5 deconvolutional layers.

#### 4.2.1 Motivation

**Generative Model**

As mentioned in Related Work, the VAE is a relatively straightforward method that offers a means to study its behavior on simpler datasets before finally experimenting on the dataset comprised of t-shirts and tops.

**Image Dimension**

Determining the appropriate resolution of the input images is non-trivial and was done empirically. Larger images allow the network to capture more details but also require more weights. As previously seen, larger images either require additional layers or simply a larger fully connected layer just before the latent vector. The fully connected layer is used to reduce the flattened version on the last activation map into the latent vector. This means that for each added dimension of that flattened vector, the additional number of required weights scale linearly with the dimension of the latent vector.
Number of Layers

Each convolutional layer in the encoder network is met with a corresponding deconvolution layer in the decoder network, and hence the increasing depth adds at least two new layers. A variety of image size and depth combinations have been evaluated before deciding on the current architecture. The autoencoder was more successful in reconstructing the images in terms of color and shapes, while the last fully connected layer was kept relatively small.

Code Dimension

There is no set code dimension; it will, instead, be varied across different datasets. This will be discussed at length in Experiments and Results (see Section 5). There is an important note to make regarding the impact of image input size and latent variable size when evaluating the performance of autoencoders. Since both the latent loss and reconstruction loss are defined as the sum over the elements (pixels or latent dimensions), an increase will effectively increase the loss, which aggravates comparisons.

4.2.2 Architecture

Table 4.1 displays the architecture of the encoder network. It consists of 5 layers with 5x5 filters and stride=2 using ReLU as activation function. The output size from each layer should be read as Height x Width x Depth. Table 4.2 displays how the last activation map from the encoder network is transformed into the latent vector $z$. The latent dimension is in this example set to 10 (for clarification revisit Figure 2.5 and Equation 2.14). Table 4.3 displays how the output vector of size 1x768 is transformed into a tensor of shape 3x2x128. Note that the network is symmetrical with equal number of layers, filters size, stride length and activation as the encoder network.
CHAPTER 4. METHOD

Encoder Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Activation</th>
<th>Channels</th>
<th>Stride</th>
<th>Filter</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input image</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>96x64x3</td>
</tr>
<tr>
<td>Conv1 - ReLU</td>
<td></td>
<td>8</td>
<td>2x2</td>
<td>5x5x3</td>
<td>48x32x8</td>
</tr>
<tr>
<td>Conv2 - ReLU</td>
<td></td>
<td>16</td>
<td>2x2</td>
<td>5x5x8</td>
<td>24x16x16</td>
</tr>
<tr>
<td>Conv3 - ReLU</td>
<td></td>
<td>32</td>
<td>2x2</td>
<td>5x5x16</td>
<td>12x8x32</td>
</tr>
<tr>
<td>Conv4 - ReLU</td>
<td></td>
<td>64</td>
<td>2x2</td>
<td>5x5x32</td>
<td>6x4x64</td>
</tr>
<tr>
<td>Conv5 - ReLU</td>
<td></td>
<td>128</td>
<td>2x2</td>
<td>5x5x64</td>
<td>3x2x128</td>
</tr>
<tr>
<td>Reshape - Flatten</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1x768</td>
</tr>
</tbody>
</table>

Table 4.1: Encoder Network architecture displaying activation function, number of channels, stride and filter for each layer in the encoder network.

Example of Latent space: 10

<table>
<thead>
<tr>
<th>Layer</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flattened vector</td>
<td>None</td>
</tr>
<tr>
<td>Fully connected z-mean &amp; z-std</td>
<td>None</td>
</tr>
<tr>
<td>Generated latent vector</td>
<td>None</td>
</tr>
<tr>
<td>Fully connected - Activation: ReLU</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 4.2: An example of how the latent space is structured with a dimension equal to 10. Note that z-mean and z-std are separate fully connected layers. They are placed on the same row since they are computed in parallel (with respect to depth).

Decoder Deconvolution Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Activation</th>
<th>Channels</th>
<th>Stride</th>
<th>Filter</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReLU, Fully-connected</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1x768</td>
</tr>
<tr>
<td>Reshape - Tensor form</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3x2x128</td>
</tr>
<tr>
<td>Deconv1 - ReLU</td>
<td></td>
<td>64</td>
<td>2x2</td>
<td>5x5x128</td>
<td>6x4x64</td>
</tr>
<tr>
<td>Deconv2 - ReLU</td>
<td></td>
<td>32</td>
<td>2x2</td>
<td>5x5x64</td>
<td>12x8x32</td>
</tr>
<tr>
<td>Deconv3 - ReLU</td>
<td></td>
<td>16</td>
<td>2x2</td>
<td>5x5x32</td>
<td>24x16x16</td>
</tr>
<tr>
<td>Deconv4 - ReLU</td>
<td></td>
<td>8</td>
<td>2x2</td>
<td>5x5x16</td>
<td>48x32x8</td>
</tr>
<tr>
<td>Deconv5 - ReLU</td>
<td></td>
<td>3</td>
<td>2x2</td>
<td>5x5x8</td>
<td>96x64x3</td>
</tr>
</tbody>
</table>

Table 4.3: Decoder Network architecture displaying activation function, number of channels, stride and filter for each layer in the encoder network.

4.3 Datasets

The same network architecture presented in the previous section is used to investigate autoencoders using three datasets with different
properties. Each dataset consists of 10,000 or more samples with dimensions 96x64x3. The complexity of the data increases with each dataset. The first two datasets were artificially generated, while the third and perhaps most interesting one is comprised of real images of t-shirts and tops. The purpose of the different datasets is to investigate how the dimensionality of the latent space affects generation and latent loss of the autoencoders, while increasing the complexity of the underlying data and how different weighing or constraining of the latent loss might impact how features are distributed on the latent variables. All datasets are split into training and validation sets (roughly 10-to-1).

4.3.1 Dataset 1

The dataset is comprised of 10,000 images of circles with varying radii and centers (one circle per image). The dataset is generated by first drawing the radius, \( r \), from a uniform distribution between 1 and 20, followed by pixel-coordinates, \( x \) and \( y \), drawn from two other uniform distributions between 20 and 44. All pixel-values where initialized equal to zero and all pixel-values inside the circle is set to 1 in all three channels. An example of 4 images is seen in Figure 4.1(a). While each image contains over 18-thousand values (96x64x3), it can easily be represented using three values \((x, y, r)\) (also referred to as explaining variables).

4.3.2 Dataset 2

The dataset bears obvious similarities with dataset 1, both in terms of size (10,000 samples of 96x64x3) and how it was generated. The main difference is that the sampling process is repeated for each of the three color channels (RGB). An example of 4 images is seen in Figure 4.1(b). This adjustment adds color and significantly increases the complexity of the data. The number of explaining variables goes up from three to nine \((x_R, x_G, x_B, y_R, y_G, y_B, r_R, r_G, r_B)\).

4.3.3 Dataset 3

The dataset contains roughly 13,200 similarly structured images of t-shirts and tops collected for this thesis. It was processed in two ways: (1) re-scaling the size with anti-aliasing, and (2) normalizing the values
CHAPTER 4. METHOD

(a) Dataset 1

(b) Dataset 2

Figure 4.1: Examples of images from Dataset 1 and 2. Each plot consists of 4 (2x2) samples.

from 0-255 (RGB values) into 0-1 by dividing all pixel-values with 255. Deciding what size to use is, as previously mentioned, a trade-off between detail and number of weights. A number of larger and smaller images has been experimented with. One important characteristic of the particular dimension is how well it divides by two. Since each convolutional layer uses stride 2 to down-sample, the size is effectively reduced by 2: 96x64 → 48x32 → 24x16 → 12x8 → 6x4 → 3x2

(a) Dataset 3

Figure 4.2: Examples of images from Dataset 3. The plot consists of 4 (2x2) samples. The images were downloaded from [32].
Chapter 5

Experiments and Results

The results from three different experiments are presented below, followed by a more general discussion connecting them.

5.1 Experiments

The following section contains experiments in which different variations of the autoencoder was trained on three different datasets. The first two datasets are artificially created in order to study the correlation between data complexity and design features for the network. The third dataset contains real images; in this case it is much more difficult to make quantifiable claims about complexity. The data found in the tables below is the respective average loss over the last five epochs. The loss over each epoch is itself an average over the last batch. Each experiment starts with a variational autoencoder where the latent loss has been weighted to zero ($\beta = 0$). This is not identical to a vanilla autoencoder, since it still samples the latent variable from the mean and standard deviation vector. However, since there are not any constraints, it allows the mean to take on any value, while reducing the standard deviation. In this sense it still acts as a counter-weight to putting tougher constraints on the latent loss and will therefore be referred to as a vanilla autoencoder. The second part of each experiment includes putting $\beta=1$ and studying the impact on the generation (or reconstruction) and latent loss. In the third and final part, different $\beta$ were experimented with in an attempt to disentangle the latent representations. The autoencoder is abbreviated AE, the variational autoencoder VAE, and the constrained variational autoencoder $\beta$-VAE. When
a specific value of $\beta$ is used, the value is stated after the $\beta$ ($\beta4$-VAE means $\beta=4$). Following the autoencoder type is the latent dimension; for example, VAE9 means the latent dimension is equal to nine. Note that $\beta1$-VAE is identical to VAE.

### 5.1.1 Experiment 1

The main objective of the first experiment was to study whether the models could learn to represent data generated from simpler distributions. It was found that using unnecessarily large latent dimensions did not significantly improve the reconstruction quality. Furthermore, the samples generated from the VAE (in the second part) became somewhat odd-looking when the latent vector was too large (in relation to data complexity). Constraining the VAE resulted in superfluous latent variables being left unused, which in turn improved the generation of more realistic samples. It was, however, unsuccessful in fully disentangling the latent representation. The following settings were used in training:

- Learning rate: 0.001
- Number of epochs: 50
- Batch size: 50

#### The Vanilla Autoencoder

Table 5.1 contains the losses for a vanilla autoencoder trained on dataset 1. As mentioned above, the underlying distribution of the data "should" require three variables to reconstruct the data. It was found that the autoencoder with two latent variables (AE2) achieves relatively poor reconstruction quality compared to an autoencoder using three latent variables (AE3). Furthermore, increasing the latent dimension ($d_z$) beyond three variables does not significantly improve the results, as seen with AE10, which uses ten dimensions. It seems as if the ability to reconstruct the data is correlated to the relation between complexity (explaining variables) and latent dimension. Notice that the latent loss is displayed even if it was not an objective to minimize it during training. It can offer some insight into how the network is learning. Rather than learning some underlying representation, AE2, seems to learn the samples, rather than distribution, and should, therefore, generalize poorly.
Table 5.1: The loss for autoencoders trained on dataset 1. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

<table>
<thead>
<tr>
<th>AE</th>
<th>(d_z=2)</th>
<th>(d_z=3)</th>
<th>(d_z=10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>514</td>
<td>61.6</td>
<td>56.3</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>568</td>
<td>69.6</td>
<td>68.3</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>5060</td>
<td>139</td>
<td>4550</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>2260</td>
<td>148</td>
<td>4860</td>
</tr>
</tbody>
</table>

Figures 5.1 - 5.3 show reconstructed images from AE2, AE3 and AE10, respectively. If the sub-images are named (1) to (5) from left to right. (1) is an input image and (2) is its reconstruction. (5) is another input image and (4) is its reconstruction. The middle image (3) is the "average" of the (2) and (4) and is computed by feeding the average latent vector from (1) and (5) to the second part of the network (the decoder). In some sense, it illustrates how well the images generalize. Figure 5.1 displays similar results as seen in Table 5.1. The reconstruction is somewhat blurry and the average latent vector does a poor job of creating what can be deemed an average of the two images. Comparing Figure 5.2 and 5.3, both have good reconstruction. However, AE3 achieves better generalization than AE10 (the radius being closer to an average radius).
The Variational Autoencoder

Table 5.2 contains the losses for a variational autoencoder trained on dataset 1. Again, the variational autoencoder with latent dimension equal to two (VAE2) achieves relatively poor reconstruction compared to VAE3 and VAE10. For all three VAE, the latent loss has been forced down, as expected, since it is now part of the training objective. The reconstruction qualities of VAE3 and VAE10 are very similar. However, the latent loss is somewhat higher for VAE10. In order to further study the difference between VAE3 and VAE10 we can generate samples by sampling the latent vector from $\mathcal{N}(0, I)$ and afterwards feed it to the decoder network. The resulting output images are shown in Figures 5.4 - 5.6. The first image on the left is a zero vector of equal dimensions as the latent space and acts as a reference (it is also somewhat of an average sample since the latent vector should be roughly zero-mean). The generated samples from VAE2 are blurry when compared to the samples of VAE3 and VAE10. However, it seems as if VAE10 generates some images that are not circles but rather ellipses.

<table>
<thead>
<tr>
<th>VAE</th>
<th>$d_z=2$</th>
<th>$d_z=3$</th>
<th>$d_z=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>516</td>
<td>66.0</td>
<td>64.5</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>506</td>
<td>66.6</td>
<td>72.0</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>24.3</td>
<td>29.7</td>
<td>52.3</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>25.2</td>
<td>29.6</td>
<td>52.7</td>
</tr>
</tbody>
</table>

Table 5.2: The loss for variational autoencoders trained on dataset 1. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.
The Constrained Variational Autoencoder

Table 5.3 contains the losses for a constrained variational autoencoder trained on dataset 1. Comparing Table 5.2 with Table 5.3, the reconstruction is similar, while the latent loss of the constrained variational autoencoder with 10 latent variables using $\beta = 4$ ($\beta$4-VAE10) is almost cut in half. As seen in Figure 5.7 this yielded better-looking samples.

<table>
<thead>
<tr>
<th>$\beta$4-VAE</th>
<th>$d_z=2$</th>
<th>$d_z=3$</th>
<th>$d_z=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>563</td>
<td>79.0</td>
<td>89.2</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>617</td>
<td>80.8</td>
<td>94.1</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>16.6</td>
<td>24.0</td>
<td>28.0</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>16.2</td>
<td>24.2</td>
<td>27.2</td>
</tr>
</tbody>
</table>

Table 5.3: The loss for constrained variational autoencoders trained on dataset 1. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

However, using $\beta = 4$ did not succeed in fully disentangling the latent variables for VAE3. Additional designs were therefore tested, as displayed in Table 5.4. Increasing $\beta$ continues to drive down latent
loss at the expense of poorer reconstruction. VAE2 is excluded due to its poor performance in relation to that of VAE3.

![Random samples from \(\beta_{4}\text-VAE10\) trained on Dataset 1.](image)

**Figure 5.7:** Random samples from \(\beta_{4}\text-VAE10\) trained on Dataset 1.

<table>
<thead>
<tr>
<th>(\beta)-VAE3</th>
<th>(\beta = 1)</th>
<th>(\beta = 2)</th>
<th>(\beta = 4)</th>
<th>(\beta = 8)</th>
<th>(\beta = 16)</th>
<th>(\beta = 32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>66.0</td>
<td>65.4</td>
<td>79.0</td>
<td>102</td>
<td>157</td>
<td>245</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>66.6</td>
<td>72.9</td>
<td>80.8</td>
<td>99.6</td>
<td>157</td>
<td>256</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>29.7</td>
<td>26.2</td>
<td>24.0</td>
<td>20.9</td>
<td>17.6</td>
<td>13.6</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>29.6</td>
<td>26.9</td>
<td>24.2</td>
<td>20.9</td>
<td>17.5</td>
<td>13.2</td>
</tr>
</tbody>
</table>

**Table 5.4:** The loss for constrained variational autoencoders trained on dataset 1. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

In order to study the disentanglement, it is necessary to plot output images. Figures 5.8 - 5.10 show three grid-plots of output images from a constrained VAE with \(\beta = 1, 4\) and 32. The latent variable \(z_1\) on the top row is incrementally increased from \(-3\) to \(3\) with step-size equal to 1, while the other variables are kept at zero. The second row similarly iterates \(z_2\) and so on. By plotting these output images, it illustrates what features each of the latent variables have learned. Using S to abbreviate Scale, H for Horizontal movement, V for Vertical movement, and N for Non-existing features (i.e. rotation or stretch) and parentheses () for secondary or minor feature, the latent representations are summarized in Table 5.5.
Table 5.5: A summary of the latent representations for VAE3, β4-VAE3 and β32-VAE3 trained on Dataset 1

<table>
<thead>
<tr>
<th>β-VAE3</th>
<th>β = 1</th>
<th>β = 4</th>
<th>β = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>z₁</td>
<td>S + (V) + (H)</td>
<td>S + V + (H)</td>
<td>V</td>
</tr>
<tr>
<td>z₂</td>
<td>V + S + (H)</td>
<td>S + (H)</td>
<td>S + H</td>
</tr>
<tr>
<td>z₃</td>
<td>V</td>
<td>S + H</td>
<td>S + H</td>
</tr>
</tbody>
</table>

Figure 5.8: Grid-plot of VAE3 trained on Dataset 1. Each row represents a latent variables and each column a step-change in that variable.

Figure 5.9: Grid-plot of β4-VAE3 trained on Dataset 1. Each row represents a latent variables and each column a step-change in that variable.

As seen in Figures 5.6 and 5.7, using $\beta > 1$ created better-looking random samples for β4-VAE10. A similar analysis of VAE with regards to disentanglement and feature representation is shown in Table 5.6 for $\beta = 1, 4$ and 16 with the corresponding output images being presented in Figure 5.11.
Figure 5.10: Grid-plot of $\beta$32-VAE3 trained on Dataset 1. Each row represents a latent variable and each column a step-change in that variable.

<table>
<thead>
<tr>
<th>$\beta$-VAE10</th>
<th>$\beta = 1$</th>
<th>$\beta = 4$</th>
<th>$\beta = 16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_1$</td>
<td>S + V</td>
<td>H</td>
<td>-</td>
</tr>
<tr>
<td>$z_2$</td>
<td>S</td>
<td>-</td>
<td>S + V + (H)</td>
</tr>
<tr>
<td>$z_3$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$z_4$</td>
<td>H + (N)</td>
<td>-</td>
<td>S</td>
</tr>
<tr>
<td>$z_5$</td>
<td>N</td>
<td>-</td>
<td>V + H + (S)</td>
</tr>
<tr>
<td>$z_6$</td>
<td>N</td>
<td>S + (H)</td>
<td>-</td>
</tr>
<tr>
<td>$z_7$</td>
<td>N + (H)</td>
<td>V + (S)</td>
<td>N</td>
</tr>
<tr>
<td>$z_8$</td>
<td>N</td>
<td>S</td>
<td>-</td>
</tr>
<tr>
<td>$z_9$</td>
<td>S + V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$z_{10}$</td>
<td>S + H</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.6: A summary of the latent representations for VAE10, $\beta$4-VAE10 and $\beta$16-VAE10 trained on Dataset 1

The $\beta$ seems to force some superfluous latent dimensions to not capture any features. However, it did not properly succeed in disentangling the latent variables.
Figure 5.11: Grid-plots of VAE10, β4-VAE10 and β16-VAE trained on Dataset 1. Each row represents a latent variables and each column a step-change in that variable.

5.1.2 Experiment 2

The main objective of the second experiment was to study whether similar results could be obtained when increasing the latent dimension and the complexity of the data. It was found that a similar drop in generation loss appeared for latent space equal or larger than 9 and that the larger latent dimensions generated odd-looking samples. Similarly, these could be improved by constraining the latent loss, which resulted in a number of unused variables. Once again, no fully disentangled latent representation could be obtained. The following settings were used in training:

- Learning rate: 0.001
- Number of epochs: 50
- Batch size: 50

The Vanilla Autoencoder

Table 5.7 contains the losses for a vanilla autoencoder trained on Dataset 2. The underlying distribution of the data "should" now instead require 9 variables, three for each color channel. It is found that AE3
achieves poor reconstruction quality, compared to AE9 and AE20. But AE9 and AE20 are relatively similar. Comparing the reconstruction losses between the different AE3 for Datasets 1 and 2, reconstruction loss increases with increased complexity of the data.

<table>
<thead>
<tr>
<th>AE</th>
<th>$d_z=3$</th>
<th>$d_z=9$</th>
<th>$d_z=20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>1800</td>
<td>161</td>
<td>139</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>2020</td>
<td>246</td>
<td>192</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>176</td>
<td>653</td>
<td>8030</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>148</td>
<td>635</td>
<td>7980</td>
</tr>
</tbody>
</table>

Table 5.7: The loss for autoencoders trained on Dataset 2. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

Figures 5.12 - 5.14 show reconstructed images from AE3, AE9 and AE20 respectively. Figure 5.12 agrees with the values found in Table 5.7. The reconstruction is very blurry and does a poor job of capturing essential features of the images. Comparing Figure 5.13 and Figure 5.14, both have good reconstruction. However, AE9 successfully captures the green circle in image (4). AE20 achieves a sharper reconstruction but also warps the circles in image (3).

![Figure 5.12: Reconstructions from AE3 trained on Dataset 2.](image1)

![Figure 5.13: Reconstructions from AE9 trained on Dataset 2.](image2)
Figure 5.14: Reconstructions from AE20 trained on Dataset 2.

The Variational Autoencoder

Table 5.8 contains the losses for different VAE trained on Dataset 2 with same latent space as the different AE above. Similar results are again found between VAE3 and VAE9. However, the reconstruction loss of VAE20 is larger than that of VAE9. In addition, the validation loss is very much larger than the training loss, which can imply overfitting.

<table>
<thead>
<tr>
<th>VAE</th>
<th>(d_z=3)</th>
<th>(d_z=9)</th>
<th>(d_z=20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>1690</td>
<td>200</td>
<td>306</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>1810</td>
<td>268</td>
<td>529</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>22.4</td>
<td>66.9</td>
<td>123</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>22.7</td>
<td>70.0</td>
<td>118</td>
</tr>
</tbody>
</table>

Table 5.8: The loss for variational autoencoders trained on Dataset 2. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

The generated samples from VAE3 agree with those from AE3. The samples are very blurry and include some non-existing color. It does not look like something that could originate from Dataset 2. VAE9 does a much better job of generating realistic samples, and Figure 5.16 shows that the randomly generated samples have a strong resemblance with samples from the dataset. Note that image (1) again is the zero vector. VAE20, much like VAE9 for Dataset 1, generates samples that are odd-looking. It includes blurry objects that are not necessarily round circles.
Comparing Table 5.9 with 5.8, the reconstruction quality somewhat suffers for $\beta$-VAE3 and $\beta$-VAE9. Comparing VAE9 with $\beta$-VAE9, the reduction of the latent loss is relatively limited. The latent loss for VAE20 is again almost cut in half, which should yield better-looking samples. However, it is still not generating samples that are flawless (see Figure 5.18(b)). Note that the zero vector sample is much better in being an average sample image compared to the one generated from VAE20. Furthermore, the discrepancy between training and validation loss is also reduced.

Studying the disentanglement becomes trickier as the complexity of the data and the latent space dimension increase. In order to more carefully study this, each color channel is displayed separately for different values of $\beta$ for VAE9 in Figures A.1-A.4 (found in Appendix A). Similar information as the one presented in Table 5.6 that depicts the captured features is simply not possible here due to the large number of variables. More generally, it seems as if VAE9 and $\beta$-VAE9
Table 5.9: The loss for constrained variational autoencoders trained on Dataset 2. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

<table>
<thead>
<tr>
<th>β-VAE (β=4)</th>
<th>$d_z=3$</th>
<th>$d_z=9$</th>
<th>$d_z=20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>1920</td>
<td>285</td>
<td>398</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>2030</td>
<td>375</td>
<td>429</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>16.6</td>
<td>50.8</td>
<td>54.5</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>15.9</td>
<td>51.1</td>
<td>54.1</td>
</tr>
</tbody>
</table>

Figure 5.18: Random samples from β4-VAE3, β4-VAE9 and β4-VAE20 trained on Dataset 2.

are equally unsuccessful in disentangling the features, since changes can be seen across all three color channels simultaneously when $z$ is changed. Furthermore, it seems as if β16-VAE9 is marginally more disentangled such that $z_2$ is mostly red-vertical movement, $z_3$ is mostly blue-scale and $z_5$ is mostly blue-vertical movement. Table 5.10 includes the losses for the different designs.
Table 5.10: The loss for constrained variational autoencoders trained on Dataset 2. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

<table>
<thead>
<tr>
<th>β-VAE9</th>
<th>β = 1</th>
<th>β = 4</th>
<th>β = 16</th>
<th>β = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>200</td>
<td>285</td>
<td>560</td>
<td>933</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>268</td>
<td>375</td>
<td>639</td>
<td>1040</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>66.9</td>
<td>50.8</td>
<td>29.9</td>
<td>18.8</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>70.0</td>
<td>51.1</td>
<td>30.8</td>
<td>19.6</td>
</tr>
</tbody>
</table>

Table 5.11 includes the losses from β-VAE20. β16-VAE20 further forces down the latent loss and now seems capable of generating more realistic samples (see Figure 5.19).

In Experiment 1, it was found that $β > 1$ seemed to force superfluous latent dimensions not to capture any features. The same effect occurs for β16-VAE20 and β32-VAE20 (with output plots displayed in Figure A.8 and A.9 respectively). While VAE20 and β4-VAE20 captures weird-looking shapes it is interesting to note that β16-VAE20 only utilizes 9 dimensions. It seems as if $z_3, z_4, z_6, z_7, z_{12}, z_{13}, z_{14}, z_{15}, z_{17}, z_{18}$ and $z_{19}$ are unused. In addition, β32-VAE20 too only utilizes 9 dimensions with $z_2, z_5, z_6, z_{10}, z_{11}, z_{13}, z_{15}, z_{17}, z_{18}$, $z_{19}$, and $z_{20}$ unused. β32-VAE20 seems somewhat successful in disentangling a subsection of the variables ($z_1, z_3, z_8, z_{14}$ and $z_{16}$).

<table>
<thead>
<tr>
<th>β-VAE20</th>
<th>β = 1</th>
<th>β = 4</th>
<th>β = 16</th>
<th>β = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>306</td>
<td>398</td>
<td>589</td>
<td>777</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>529</td>
<td>429</td>
<td>647</td>
<td>783</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>123</td>
<td>54.5</td>
<td>30.6</td>
<td>21.2</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>118</td>
<td>54.1</td>
<td>29.6</td>
<td>21.6</td>
</tr>
</tbody>
</table>

Table 5.11: The loss for constrained variational autoencoders trained on Dataset 2. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.
5.1.3 Experiment 3

The main objective of the third experiment was to study whether the results from Experiments 1 and 2 held for data with arbitrary complexity. Previous data had a specific number of variables that was theoretically sufficient to "explain" the data. Making the same claim for photographs or images is more difficult since there is not necessarily such a number. The different autoencoders studied in previous experiments will here be trained on Dataset 3 that includes images of t-shirts and tops. It was found that a similar drop in generation loss could not be obtained for Dataset 3. The constrained VAE had similar effects as previously seen on generating samples. In addition, with large enough $\beta$ the latent representation was successful in disentangling the features. The following settings were used in training the networks:

- Learning rate: 0.001
- Number of epochs: 50
- Batch size: 50

The Vanilla Autoencoder

Table 5.12 includes the losses from a autoencoder with a large variety of different latent dimensions trained on Dataset 3. Note that there is no significant drop in generation loss as seen for Datasets 1 and 2. It seems as if the generation loss somewhat plateaus around AE9. Since these samples are more complex, it requires visual inspection to spot the difference in reconstruction quality, examples of which are seen in Figure 5.20 - 5.26. The difference in reconstruction quality between AE2 and AE3 is minor and they both perform poorly. It seems as if AE2 reconstructs "a" t-shirt rather than "the" t-shirt. AE2 does a poor job of capturing sleeve length and color. However, keep in mind that the complete image is reconstructed using only two variables. Similar
AE & $d_z=2$ & $d_z=3$ & $d_z=9$ & $d_z=20$ & $d_z=40$ & $d_z=100$ & $d_z=200$ \\ 
Train Gen. & 5590 & 5170 & 4930 & 4890 & 5060 & 4880 & 5010 \\ 
Valid Gen. & 5540 & 5480 & 5040 & 5010 & 5030 & 4880 & 4960 \\ 
Train Lat. & 147.3 & 100 & 600 & 935 & 1550 & 2730 & 4850 \\ 
Valid Lat. & 135.9 & 97.4 & 660 & 921 & 1610 & 2690 & 4810 \\

Table 5.12: The loss for autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

Remarks can be made about AE3. The first substantial leap in performance comes with AE9. It not only does a better job with colors but manages both to identify the print and shapes properly. It is interesting to note that with AE9 and beyond, the average image starts to blend different features from the two t-shirts. Including the print from image (1), changing color to dark blue, slimming down arm width and widening the neck. AE40 is somewhat off in color and AE100 adds some non-existing color to the print.

Figure 5.20: Reconstructions from AE2 trained on Dataset 3.

Figure 5.21: Reconstructions from AE3 trained on Dataset 3.
The Variational Autoencoder

Table 5.13 includes the losses of VAE with similar latent dimension
as shown above in Table 5.12. Again, there is no substantial drop in generation loss as seen in the previous experiments. The mid-sized latent spaces VAE20 and VAE40 have somewhat similar generation and latent loss as that seen in Table 5.12. VAE100 and VAE200 experiences an increase in generation loss, while dramatically reducing the latent loss.

<table>
<thead>
<tr>
<th>VAE</th>
<th>$d_z=2$</th>
<th>$d_z=3$</th>
<th>$d_z=9$</th>
<th>$d_z=20$</th>
<th>$d_z=40$</th>
<th>$d_z=100$</th>
<th>$d_z=200$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>5500</td>
<td>5230</td>
<td>5040</td>
<td>5110</td>
<td>5020</td>
<td>4990</td>
<td>5220</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>5700</td>
<td>5460</td>
<td>5400</td>
<td>5000</td>
<td>5020</td>
<td>5190</td>
<td>5200</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>18.1</td>
<td>22.9</td>
<td>46.3</td>
<td>64.1</td>
<td>66.7</td>
<td>78.7</td>
<td>99.6</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>16.5</td>
<td>22.8</td>
<td>46.6</td>
<td>63.8</td>
<td>67.0</td>
<td>79.1</td>
<td>102</td>
</tr>
</tbody>
</table>

Table 5.13: The loss for variational autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

In order to further study how the generation and latent loss is impacted, random samples are generated in a similar manner as before and examples of which are found in Figure 5.27 - 5.33. VAE2 and VAE3 does a decent job of generating samples. It is a clear improvement with VAE9, VAE20, and VAE40. The larger latent dimensions seem to have some problems. VAE100 seems to be able to capture more detail than previous designs, but at the expense of some odd-looking shapes. VAE200 is blurry, but in comparison with VAE2 and VAE9, it is less uniform and instead seems to struggle with putting the features together. The image to the very left, previously referred to as image (1), is again the generated output from when the decoder network is fed a zero-vector. Therefore, the output can, in some sense, be regarded as the average t-shirt/top.

Figure 5.27: Random samples from VAE2 trained on Dataset 3.
Figure 5.28: Random samples from VAE3 trained on Dataset 3.

Figure 5.29: Random samples from VAE9 trained on Dataset 3.

Figure 5.30: Random samples from VAE20 trained on Dataset 3.

Figure 5.31: Random samples from VAE40 trained on Dataset 3.

Figure 5.32: Random samples from VAE100 trained on Dataset 3.

Figure 5.33: Random samples from VAE200 trained on Dataset 3.
The Constrained Variational Autoencoder

Table 5.14 includes the losses of a constrained variational autoencoder with $\beta = 4$. The latent loss for $\beta$-VAE40, $\beta$-VAE100 and $\beta$-VAE200 are similar, with $\beta$-VAE200 having the lowest latent loss of the three.

<table>
<thead>
<tr>
<th>$\beta$-VAE ($\beta = 4$)</th>
<th>$d_z=2$</th>
<th>$d_z=3$</th>
<th>$d_z=9$</th>
<th>$d_z=20$</th>
<th>$d_z=40$</th>
<th>$d_z=100$</th>
<th>$d_z=200$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>5570</td>
<td>5280</td>
<td>5080</td>
<td>5180</td>
<td>5040</td>
<td>5300</td>
<td>5050</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>5620</td>
<td>5570</td>
<td>5080</td>
<td>5250</td>
<td>5100</td>
<td>5280</td>
<td>5410</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>13.1</td>
<td>16.6</td>
<td>29.0</td>
<td>36.2</td>
<td>31.5</td>
<td>33.2</td>
<td>31.2</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>12.8</td>
<td>17.0</td>
<td>29.8</td>
<td>36.2</td>
<td>31.5</td>
<td>33.5</td>
<td>31.4</td>
</tr>
</tbody>
</table>

Table 5.14: The loss for constrained variational autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

Inspecting Figures 5.34 - 5.40 reveals some interesting results. It seems as if the random samples from the constrained variational autoencoders improved across all VAE. $\beta$-VAE200 and $\beta$-VAE100 become less blurry and more precise in shape, but on the other hand they are almost grey-scale.

Figure 5.34: Random samples from $\beta$-VAE2 trained on Dataset 3.

Figure 5.35: Random samples from $\beta$-VAE3 trained on Dataset 3.

In an effort to reduce the amount of $\beta$ and latent dimension combinations, the smallest latent dimension that will be studied is 9.
Table 5.15 contains the losses of several \( \beta \)-VAE9 with varying weight of the latent loss with \( \beta \) equal to 1, 4, 8, 16 and 32. The \( \beta \)-VAE9 achieves a low latent loss at the expense of reconstruction quality, which also can be seen in the random samples generated in Figure 5.41.
Table 5.15: The loss for constrained variational autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

<table>
<thead>
<tr>
<th>β-VAE9</th>
<th>β = 1</th>
<th>β = 4</th>
<th>β = 8</th>
<th>β = 16</th>
<th>β = 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>5040</td>
<td>5080</td>
<td>5160</td>
<td>5270</td>
<td>5260</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>5400</td>
<td>5080</td>
<td>5150</td>
<td>5220</td>
<td>5352</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>46.3</td>
<td>29.0</td>
<td>22.3</td>
<td>15.5</td>
<td>9.40</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>46.6</td>
<td>29.8</td>
<td>21.0</td>
<td>14.6</td>
<td>9.88</td>
</tr>
</tbody>
</table>

In order to study the feature captured by the VAE, similar grid-plots as the ones presented in Experiments 1 and 2 are used. Here, each row starts with $z_i$ equal to -4 in the left-most image and for each step to the right $z_i$ incrementally increased by 1 (to finally in the right-most image equal to 4). Line 1 only adjusts $z_1$ and leaves all other latent variables equal to 0, line 2 adjusting only $z_2$, and so on. The different features captured in VAE9 are shown in Figure 5.42. They are not disentangled, since several of what one might deem as unique features in a clothing item change simultaneously. For example, $z_1$ seems to adjust both sleeve length, height and color, $z_2$ sleeve length and collar/neck and $z_3$ sleeve length, height and color.

The grid-plots for β4-VAE9 are shown in Figure 5.43. It, too, is unsuccessful in disentangling all the features. $z_1$ changes both sleeve length and collar; and, $z_2$ seems to capture print and color; one might deem $z_3$ as a somewhat disentangled feature in capturing neck/collar; $z_5$ captures sleeve length in addition to minor color-changes; $z_6$ darkness/lightness; $z_7$ mainly height but also color and print; and, finally $z_8$ bottom/top color.

The results from β8-VAE9 are shown in Figure 5.44. Again, the features are not completely disentangled, but it is successful in learning some
Figure 5.42: Grid-plot of VAE9 trained on Dataset 3. Each row represents a latent variables and each column a step-change in that variable.

previously unidentified features such as $z_3$ capturing print. Detailed sleeve length in $z_6$ and $z_5$ seems to capture dark-sleeve/tank-top.
Figure 5.43: Grid-plot of $\beta 4$-VAE$^9$ trained on Dataset 3. Each row represents a latent variables and each column a step-change in that variable.

The results from $\beta$16-VAE$^9$ are shown in Figure 5.45. Here the reconstruction quality begins to degenerate. However, the features are somewhat successfully disentangled. $z_1$ captures sleeve length (and to some extent height), $z_2$ darkness/lightness, $z_3$ sleeve color, $z_4$ height (going over to dress in the left), $z_5$ color, $z_6$ print, $z_7$ neck/collar, $z_8$ vague bottom/top color and $z_9$ size in frame. $\beta$32-VAE$^9$ was also tested but did not yield any interesting results, with poor reconstruction quality and many indistinguishable and entangled features.
The next step involves increasing the latent space from 9 to 20 and repeating the same tests with different $\beta$. Table 5.16 includes the losses for $\beta$-VAE20 with the same $\beta$ as above (1, 4, 8, 16 and 32). Again, $\beta$32-VAE20 gives a large reconstruction loss while having a low latent loss. $\beta$8-VAE achieves a relatively low validation generation loss. Since these plots are all 20 lines long and cover almost the entire page, the plots will be included in Appendix B. They also span between [-5,5], are 11 images wide and are split into two images.

The grid-plot for VAE20 is shown in Figure B.1 - B.2. It shows similar results as seen above with VAE9. It seems as if each latent dimension captures at least one feature but in the majority of the cases more. Do note that in the images to the very left and right, the latent value is ±5 which is far from 0 in a Gaussian distribution with standard de-
Figure 5.45: Grid-plot of $\beta_{16}$-VAE9 trained on Dataset 3. Each row represents a latent variables and each column a step-change in that variable.

Table 5.16: The loss for constrained variational autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

<table>
<thead>
<tr>
<th>$\beta$-VAE20</th>
<th>$\beta = 1$</th>
<th>$\beta = 4$</th>
<th>$\beta = 8$</th>
<th>$\beta = 16$</th>
<th>$\beta = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>5110</td>
<td>5180</td>
<td>5220</td>
<td>5310</td>
<td>5550</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>5000</td>
<td>5250</td>
<td>5050</td>
<td>5440</td>
<td>5270</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>64.1</td>
<td>36.2</td>
<td>20.9</td>
<td>14.5</td>
<td>9.82</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>63.8</td>
<td>36.2</td>
<td>21.9</td>
<td>14.9</td>
<td>9.89</td>
</tr>
</tbody>
</table>

The grid-plot for $\beta_{4}$-VAE20 is shown in Figure B.3 - B.4. It seems as if
some latent dimensions start fading or taking on more subtle features such as $z_4$ and $z_{15}$. With $\beta$-VAE20 and beyond, a large portion of the features starts to fade, and the same is true for VAE40. The grid-plots for $\beta$-VAE20 are found in Figure B.5 - B.6. In Experiments 1 and 2 this happened when the dimension was unnecessarily large. It seems as if the capacity for this particular network is somewhere between 9-20 dimensions. For this reason, further plots and results regarding disentanglement for larger dimension 40, 100 and 200 are left out. The losses for $\beta$-VAE40 and $\beta$-VAE100 can be found in Table 5.17 and 5.18.

<table>
<thead>
<tr>
<th>$\beta$-VAE40</th>
<th>$\beta = 1$</th>
<th>$\beta = 4$</th>
<th>$\beta = 8$</th>
<th>$\beta = 16$</th>
<th>$\beta = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>5020</td>
<td>5040</td>
<td>5030</td>
<td>5340</td>
<td>5390</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>5020</td>
<td>5100</td>
<td>5360</td>
<td>5260</td>
<td>5380</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>66.7</td>
<td>31.5</td>
<td>21.7</td>
<td>14.4</td>
<td>9.49</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>67.0</td>
<td>31.5</td>
<td>22.5</td>
<td>14.3</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Table 5.17: The loss for constrained variational autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.

<table>
<thead>
<tr>
<th>$\beta$-VAE100</th>
<th>$\beta = 1$</th>
<th>$\beta = 4$</th>
<th>$\beta = 8$</th>
<th>$\beta = 16$</th>
<th>$\beta = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Gen.</td>
<td>4990</td>
<td>5300</td>
<td>5320</td>
<td>5400</td>
<td>5570</td>
</tr>
<tr>
<td>Valid Gen.</td>
<td>5190</td>
<td>5280</td>
<td>5320</td>
<td>5450</td>
<td>5390</td>
</tr>
<tr>
<td>Train Lat.</td>
<td>78.7</td>
<td>33.2</td>
<td>22.7</td>
<td>14.8</td>
<td>9.31</td>
</tr>
<tr>
<td>Valid Lat.</td>
<td>79.1</td>
<td>33.5</td>
<td>24.9</td>
<td>14.9</td>
<td>9.16</td>
</tr>
</tbody>
</table>

Table 5.18: The loss for constrained variational autoencoders trained on Dataset 3. Train stands for training set and Valid stands for validation set. Gen. is the generation loss (or reconstruction loss) and Lat. is the latent loss.
5.2 Discussion

The three different Experiments revealed some interesting results about data complexity and latent dimension. For the first two datasets, with 3 and 9 explaining variables, it was of no use to increase the latent dimension beyond this complexity. First, it did not improve the reconstruction quality. Secondly, the generated samples became worse as the dimensions increased beyond this point. It was interesting to see that constraining the latent loss forced the latent dimension to coincide, or roughly coincide, with the number of explaining variables and complexity. It is possible that this is the reason for improving the generated random samples of VAE where the latent dimension was large in relation to data complexity.

It was surprising to see that the constrained VAE in Experiments 1 and 2 was unsuccessful in disentangling the latent representation while the research paper introducing the method was successful with a very similar dataset [6]. It was again surprising to see Experiment 3 succeed, or somewhat succeed. It is important to note that while Datasets 1 and 2 were generated from uniform distribution, Dataset 3 was not. The initial datasets included radii spanning continuously from 1 to 20 and the same being true for position. Even if Dataset 3 consisted of a large amount of samples, it is not guaranteed that a feature, such as sleeve length, is continuously covered in the examples. The same is true for color, print, length, etc. With that said, the results were impressive.

The differences in the latent dimension within each experiment were large. It was not feasible to perform a more granular study since each training of the network takes a long time. It is also important to note that the number of epochs was fewer than many other implementations. Studying the loss graphs revealed that the loss noticeably plateaued before epoch 50. This is less true for the larger models; ideally, all experiments should have been conducted using 500-1000 epochs and perhaps using step-wise reductions in learning rate. However, since each network is different, it was not feasible to vary and adjust the method for each training, and time/computing power did not allow for more epochs. This is also the reason for concluding that the ideal dimension for Dataset 3, with this particular implementation,
network architecture, image size, etc. is somewhere between 9-20 for the task of disentangling latent representation in tops and t-shirts.

With increasing dimension of the latent space and $\beta$ it seemed as if the features were forced further out on the tails of the Gaussian distribution of $z$. This can be seen as contrast, and colors are seen towards the edges of the grid-plots with larger values. Grey-scale means that the difference between the color channels is small; for example, a pixel is white if all three channels are 255, grey somewhere in the middle, and black if all are 0. Wrongfully reconstructing shape (0 versus 255 across all channels) is in that sense worse than getting one channel wrong (and should therefore be more prioritized in training).
Chapter 6

Conclusion and Future Work

6.1 Conclusion

The thesis set out to investigate to what extent autoencoders could be used for automated extraction of defining features in clothing. After a brief introduction of neural networks and generative models, followed a review of articles and literature regarding CNN implementations and generative methods primarily for unsupervised learning.

The different autoencoders were implemented using a deep CNN architecture capable of encoding features from different datasets with varying complexity. In contrast to recent findings, the constrained variational autoencoder was unable to fully disentangle the latent feature for less complex datasets. It, however, managed to disentangle defining features when trained on a dataset consisting of t-shirts and tops.

After studying the performance, both in terms of reconstruction quality and the ability to generate new realistic samples, it was found that there seems to be a balanced choice of latent dimension in relation to the data complexity. A larger latent vector does not guarantee better reconstruction, and a disproportionately large latent vectors generated inferior samples. The balanced choice or region was empirically found for each dataset by comparing the losses and visually studying the learned features (where superfluous dimension gradually faded as the latent loss was weighted to comprise a larger part of the total loss).
Returning to the retail industry and image-driven analysis, the disentangled latent vector can be used in a number of different applications. For example, comparing clothing items on a high level; t-shirt A being closer, in latent space, to t-shirt B than C. Another, and perhaps more interesting example, is comparisons on feature level. It should also be possible to compare items on a more granular level; t-shirt A and t-shirt B both have similar collars but t-shirt C is more similar in sleeve length. The latter can prove valuable when optimizing inventory.

6.2 Future Work

Unsupervised learning and generative models are two popular fields of machine learning research. Newer and better methods are introduced frequently, one more powerful than the next. It would be interesting to conduct broad and up-to-date reviews of unsupervised disentangling methods on data with varying complexity. By using artificially created benchmark data, it would be possible to label the data with its unique generation parameters, and, in doing so, get more quantifiable disentanglement measures. One especially interesting generative method is the Generative Adversarial Network (GAN). It would be interesting to conduct a similar study using variations of GANs such as InfoGAN [33] or Adversarial Autoencoders [34].

More specifically, to this thesis it would be interesting to apply more advanced training techniques, continue to optimize and find the optimal architecture for the dataset and application. Furthermore, it would be interesting to compare larger datasets with several autoencoders working in parallel. To train separate variational autoencoders on datasets comprised of other product types (limiting the variance within the datasets) in combination with a classifier to slot the image to the correct autoencoder or simply using the one with smallest reconstruction loss.
Bibliography


Appendix A

Plots from Experiment 2

Figure A.1: Grid-plot of VAE9 trained on Dataset 2. Each row represents a latent variables and each column a step-change in that variable.
Figure A.2: Grid-plot of β4-VAE9 trained on Dataset 2. Each row represents a latent variables and each column a step-change in that variable.

Figure A.3: Grid-plot of β16-VAE9 trained on Dataset 2. Each row represents a latent variables and each column a step-change in that variable.
Figure A.4: Grid-plot of β32-VAE9 trained on Dataset 2. Each row represents a latent variable and each column a step-change in that variable.

Figure A.5: Grid-plot of VAE20 trained on Dataset 2. The plot has been rotated 90 degrees anti-clock-wise. Each column represents a latent variable and each row a step-change in that variable.
Figure A.6: Grid-plot of β4-VAE20 trained on Dataset 2. The plot has been rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable.

Figure A.7: Grid-plot of β8-VAE20 trained on Dataset 2. The plot has been rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable.
Figure A.8: Grid-plot of B16VAE20 trained on Dataset 2. The plot has been rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable.

Figure A.9: Grid-plot of β32-VAE20 trained on Dataset 2. The plot has been rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable.
Appendix B

Plots from Experiment 3

Figure B.1: Grid-plot of VAE20 trained on Dataset 3. The plot has been split in two part and rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable. This is part 1 which includes $z_1 - z_{10}$. 
Figure B.2: Grid-plot of VAE20 trained on Dataset 3. The plot has been split in two part and rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable. This is part 2 which includes $z_{11} - z_{20}$.

Figure B.3: Grid-plot of $\beta 4$-VAE20 trained on Dataset 3. The plot has been split in two part and rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable. This is part 1 which includes $z_{1} - z_{10}$. 
Figure B.4: Grid-plot of β4-VAE20 trained on Dataset 3. The plot has been split in two part and rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable. This is part 2 which includes $z_{11} - z_{20}$.

Figure B.5: Grid-plot of β8-VAE20 trained on Dataset 3. The plot has been split in two part and rotated 90 degrees anti-clock-wise. Each column represents a latent variables and each row a step-change in that variable. This is part 1 which includes $z_{1} - z_{10}$. 
Figure B.6: Grid-plot of \( \beta 8 \)-VAE20 trained on Dataset 3. The plot has been split in two parts and rotated 90 degrees anti-clock-wise. Each column represents a latent variable and each row a step-change in that variable. This is part 2 which includes \( z_{11} - z_{20} \).