Department of Physics and Astronomy
Heidelberg University

Bachelor thesis in Physics
submitted by

Richard Häcker

born in Filderstadt (Germany)

2020

Representations of three dimensional Objects using Neural Networks

This Bachelor thesis has been carried out by

Richard Häcker

at the

Heidelberg Collaboratory for Image Processing
at
Heidelberg University

under the supervision of

Prof. Björn Ommer & **Prof. Tilman Plehn**

Representations of three dimensional Objects using Neural Networks

Richard Häcker

Abstract

In this thesis we will investigate the mapping from the image space to the latent space using neural networks. We will focus on image data sets, specifically created to display three-dimensional objects with exact labeled articulations. Using a variational autoencoder in combination with a discriminative network, the aim is to extract and investigate information about articulations from images. This enables us to explore and compare the mapping of specific articulation parameters onto the latent space. The main contribution of this work is the comparison between natural interpolations of articulations and different interpolations in the latent space. Furthermore, we investigate how a metric loss improves the model and how a discriminator helps expand the latent space around observations.

Zusammenfassung

Diese Bachelorarbeit untersucht Abbildungen von Neuronalen Netzen vom Bildraum zum latenten Raum. Wir nutzen Datensätze bestehend aus, von uns erstellten Bildern, die exakt definierte Artikulationen von dreidimensionalen Objekten darstellen. Wir nutzen einen variational autoencoder und einen Diskriminator, um aus den Bildern die Informationen über die Artikulation der Objekte zu extrahieren. Damit können wir die Abbildung spezifischer Artikulationen im latenten Raum untersuchen. Wir werden natürliche Interpolationen von Artikulationen, mit interpolationen im latenten Raum vergleichen. Außerdem werden wir Stichproben aus dem latenten Raum entnehmen und davon neue Bilder generieren.

Contents

Chapter 1.

Introduction

To cope and benefit from the immense data creation happening currently, the machine learning approach proved capable to extract and work with meaningful information [2]. This thesis focuses on image data, specifically we examine the representation of a three-dimensional articulation using machine learning algorithms. In the computer vision environment, there are already many examples ([2],[15]) of machine learning architectures which can learn a representation of a three-dimensional objects from two-dimensional images. Our model consists of two parts: an Variational Autoencoder (VAE)[10], which maps images into the latent space and back to the image space and a part of a General Adversarial Networks (GAN)[4] the Discriminator, that allows us to chart the latent space aside the representations provided by the former.
To be able to better interpret and control our model, we turn to synthetic data sets of cuboid stacks. These data sets are created with a custom tool. It enables us to create image data sets fully parameterized with all relevant labels for each image. The images are rendered in a server application of the game engine Unity, with perfectly specified parameters for articulations and appearances. The images display stacked cuboids which pivot along one edge in a given angle.
We will train a Varational Autoencoder (VAE) and the adversarial discriminator on these data sets. In a first step we run preliminary studies to find a model that can best reproduce our synthetic data and investigate how to optimally train its adversarial components.
In the second step and the main part of the experiments, we focus on the latent space and the embedding of articulations in it. We use a image sequence of an interpolation between two articulations and reconstruct the sequence with the VAE. We will use the latent representation of the first and the last image to then interpolate in the latent space. Comparing the generated images from the interpolation in the latent space with the image sequence which directly interpolates the articulation, will give us an indication how the articulation is embedded in the latent space. Additionally we will use a Uniform Manifold Approximation and Projection (UMAP)[12] for dimension reduction of the latent space. With this projection we can plot the difference of the interpolations directly from the latent space.
In the next experiment we do not only want to interpolate between representations but sample from the latent space to generally create images from latent representations. To further ensure that the images created by samples, are similar to the data set, we will do a Principle Component Analysis (PCA) and sample along a principle component. We will examine the principle component with the highest variance of latent representations from our data set.

Chapter 2.

Fundamentals

This chapter will give you an overview of the methods used in this thesis. With neural networks (NN) as a powerful tool for image and data analysis, we will start from the basic building blocks and move up to established architectures and how to train them.

2.1. Perceptron

A perceptron is a linear regression model which tries to find the relation between an input and output scalar to finaly classify data. It will be explained by the following two-dimensional example shown in Figure (2.1). Every data point is described by their coordinates $\boldsymbol{x} = (x_1, x_2)$ and their corresponding label $y \in \{0, 1\}$. The objective is to create a decision boundary which classifies every point in space to one class. A perceptron achieves this trough a linear decision boundary $f(\boldsymbol{x})$ which is specified with a its normal vector $\boldsymbol{\omega}$ and an offset b. This decision boundary is described with

$$f(\boldsymbol{x}|\boldsymbol{\omega}) = \boldsymbol{\omega}^{\boldsymbol{T}} \cdot \boldsymbol{x} - b \qquad \text{or} \qquad f(\boldsymbol{x}|\boldsymbol{\omega}) = \begin{pmatrix} \omega_1 \\ \omega_2 \\ -b \end{pmatrix}^T \cdot \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} . \tag{2.1}$$

The perceptron will create a positive output for all points above the decision boundary and a negative output for everything below the decision boundary. In the machine learning domain, the parameters $\boldsymbol{\omega}$ are called weights and the parameter b is called the bias. From now on the bias will be integrated into the weight vector as shown in the Equation (2.1). If a heaviside step function Θ also called activation function, takes in the result of $f(\boldsymbol{x}|\boldsymbol{\omega})$, we receive the predicted label $\hat{y}$, leading to

$$\hat{y} = \Theta\Big(f(\boldsymbol{x}|\boldsymbol{\omega})\Big) . \tag{2.2}$$

The activation function introduces a nonlinear part in the prediction, this helps to identify the class and is critical for the multilayer perceptron described in the next section.
In Figure (2.1), we have a look at the decision boundary $\boldsymbol{\omega_0} = (1, -1)$ in green with the parameter $b = -0.7$ in red. Ignoring the color of the data points, or goal is to classify data points to class zero or one displayed as circles and crosses. The perceptron will output positive numbers for all data points of the class one, and negative outputs for class zero. With the heaviside step function as activation function, we will receive the prediction one for elements of class one and the prediction zero for elements from class zero.

2.2. Multilayer Perceptron

One perceptron can only classify very simple linear separable data distributions but a combination of many different perceptrons can reliably classify complex distributions by mapping them nonlinearly into a higher dimensional space which is then again linear separable. This nonlinearity is introduced through activation functions between the perceptrons. Without them, the connected perceptrons would collapse to a simple combination of the linear functions [7]. In the following example, visualized in Figure (2.1), four perceptrons with parameters $\{\boldsymbol{\omega_0}, \boldsymbol{\omega_1}, \boldsymbol{\omega_2}, \boldsymbol{\omega_3}\}$ are used to map the data onto a four-dimensional space. For any point in space every perceptron can produce an output according to Equation(2.2), resulting in a four-dimensional output vector describing areas between the perceptrons. For this example, the color of the data points specifies their predicted class with purple describing class a and blue class b.

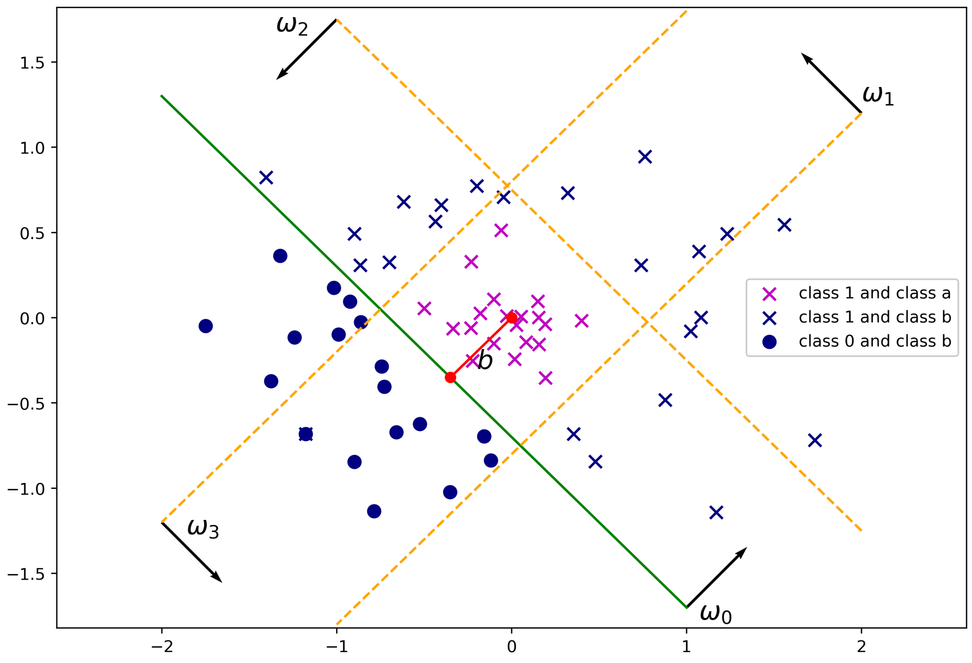

Figure 2.1. Multilayer Perceptron for classification

This output vector in the so called hidden layer, can be processed further with additional perceptrons in the next hidden layer. In this layer the perceptrons use the output vector from the previous layer as an input, multiplying it with their weights and applying again the activation function. A multilayer percepton (MLP) consists of an arbitrary number of perceptrons and hidden layers to reach the output layer and finally predict a class of any point in space. In the machine learning community, an MLP is a subset of neural network with perceptrons as nodes which are also called artificial neurons. Furthermore, a hidden layer of an MLP, is know as a fully connected layer because every node is connected to every input parameter.

The MLP architecture for our presented example is shown in Figure (2.2). Using two hidden layers and only one perceptron in the second hidden layer. The perceptron in the second hidden layer adds up a quarter of every input from the first layer and with the activation function predicts the number one for data points from class a. This is the case if all perceptrons in the first hidden layer activate describing the area enclosed by all perceptrons, displayed in Figure (2.1). If any perceptron in the first layer outputs a zero then the perceptron in the second layer will also output a zero and therefore predict the class b.

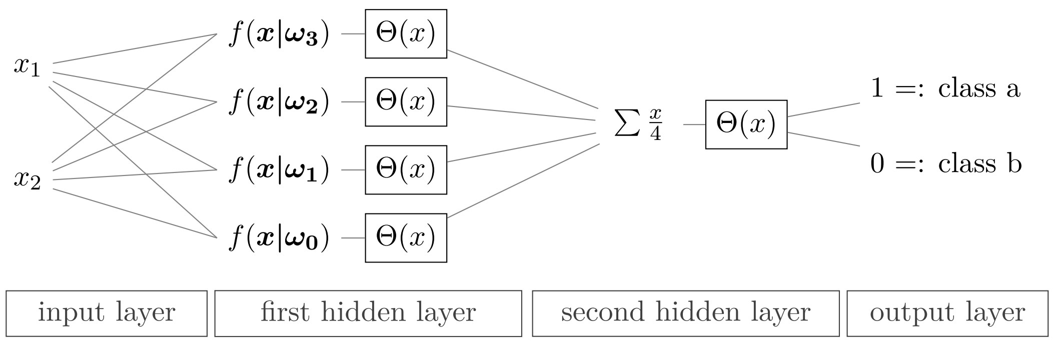
x_1
x_2
$f(\boldsymbol{x}|\boldsymbol{\omega_3})$
$f(\boldsymbol{x}|\boldsymbol{\omega_2})$
$f(\boldsymbol{x}|\boldsymbol{\omega_1})$
$f(\boldsymbol{x}|\boldsymbol{\omega_0})$
$\Theta(x)$
$\sum \frac{x}{4}$
$\Theta(x)$
1 =: class a
0 =: class b
input layer
first hidden layer
second hidden layer
output layer

Figure 2.2. Architecture of a MLP

2.3. Loss Functions

Up until now, we provided the parameters for the examples, however, to change that we require a quantitative metric to evaluate predictions of a network therefor we need a loss function.

The Mean Squared Error (MSE) [5] can quantify those predictions or any comparable input and output. It is also known as the L2 loss

$$\mathcal{L}_2\left(x \,|\, \boldsymbol{\omega}, y\right) = \frac{1}{N} \sum_{i=1}^{N} \Big(y_i - \Theta(f(\boldsymbol{x}|\boldsymbol{\omega}))\Big)^2 \qquad \text{or} \qquad \mathcal{L}_2\left(\hat{y} \,|\, y\right) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \,, \tag{2.3}$$

with N being the number of data points to classify, $y_i \in \{0,1\}$ being the actual label and $\hat{y}_i$ being the prediction for data point $\boldsymbol{x_i}$.
Another similar loss is the mean absolute error (MAE), also known as L1 loss

$$\mathcal{L}_1\left(\hat{y} \,|\, y\right) = \frac{1}{N} \sum_{i=0}^{N} |y_i - \hat{y}_i| \,. \tag{2.4}$$

The Kullback–Leibler Divergence (KLD) [11] is another important loss which quantifies, how similar two given probability distributions P and Q are

$$\mathcal{L}_{KLD}(P \,||\, Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right) . \tag{2.5}$$

One example is, the distribution $P(\mu, \sigma)$ to be a normal probability distribution which will be compared to a normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$ with the mean of 0 and a standard deviation of 1. We can derive from the KLD Formula (2.5)

$$L_{KLD}\Big(P(\mu, \sigma) \,||\, \mathcal{N}(0, 1)\Big) = -\frac{1}{2}\Big(1 + \log\Big(\sigma^2\Big) - \mu^2 - \sigma^2\Big) . \tag{2.6}$$

The Binary Cross Entropy (BCE) [5] Loss measures the error for a classification model predicting a scalar value between zero and one for elements of class zero or one

$$\mathcal{L}_{BCE} = \frac{1}{N} \sum_{i=0}^{N} -\Big(\hat{y}_i \cdot \log(y_i) + (1 - \hat{y}_i) \cdot \log(1 - \hat{y}_i)\Big) . \tag{2.7}$$

The Metric Loss [16] forces a model to create a mapping, which maps similar images to a similar representation in the embedding space. In our case we use a **triplet loss** for the metric loss. It is composed as the name suggests, of triplets. One triplet i includes latent representations of an anchor point $a(i)$, a positive example $p(i)$ and a negative example $n(i)$. In our chase these latent representations are representations of images from an input batch, from which the anchor is randomly chosen. The positive example is described by an image which is similar to the anchor image and the negative examples is dissimilar to the anchor image. This similarity can be defined differently for any specific use case. We focus on the representations of the triplet i and calculate the L2 distance in the embedding space between the anchor and the positive example $D_{p(i)}$ and between the negative example and the anchor $D_{n(i)}$. With these distances we can force that the distance of the positive example has to be smaller, by at least the margin α than the distance of the negative example

$$\mathcal{L} = \frac{1}{N} \sum_{i}^{N} [\, D^2_{p(i)} - D^2_{n(i)} + \alpha \,]_+ \tag{2.8}$$

with $[x]_+ = \max\{0, x\}$ and N being the number of triplets.

2.4. Backpropagation

Now, that we can quantify how well our model works, we want to minimise the loss by adjusting the weights of our neural network. This can be achieved with the algorithm automatic differentiation.
This creates a computational graph with a node for every computation inside the network as well as a function for differentiating the node with respect to the inputs. This essentially means that the chain rule is applied throughout the whole network. Now we can evaluate the model on training data and optimize the parameters by minimizing the the loss function. This minimization process uses the evaluated gradients throughout the network and is also called backpropagation [1].

2.5. Activation Function

If we have a look at the MLP example from Section 2.2 and apply backpropagation, we will not be able to find meaningful gradients because of the non-smooth heaviside step function. To prevent such a cases we should choose the activation function carefully. A convex function is, for example, the rectifier function also called ReLU, and its slight variation LeakyReLU [5].

$$\mathrm{ReLU}(x) = \max(x, 0) \tag{2.9}$$

$$\mathrm{LeakyReLU}(x, m) = \begin{cases} x, & \text{if } x \geqslant 0 \\ x * m, & \text{otherwise} \end{cases} \tag{2.10}$$

With only linear functions between the convex activation function [5], we can find the global minimum from the backpropagation, given some gradient.
Another useful activation function is the Sigmoid function shown in Equation (2.11) which maps all inputs to a range from 0 to 1, it is ideal for an output layer which has to generate pixel values for an image. Similarly the hyperbolic tangent function maps the input to a range from minus 1 to 1 [5].

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{2.11}$$

2.6. Convolution

If we use images as inputs for fully connected neural networks the amount of parameters to store and adjust is immense.For instance, if a Network with one hidden layer and 10 artificial neurons receives a grayscale image with a resolution of 1024 times 1024 pixels as an input, we would need over 20 million parameters for the network. Instead of using only fully connected layers, one popular option is to use convolutional layers to decrease the amount of parameters and extract information efficiently from images.
A convolution uses a kernel to process only a subset of the whole image at a time. A kernel is described by a matrix and every element consists of a weight and a bias. The kernel size determines the dimensions of the matrix. This kernel then moves across the whole input by a given stride value which should be smaller than the kernel size to not skip any input values [5].

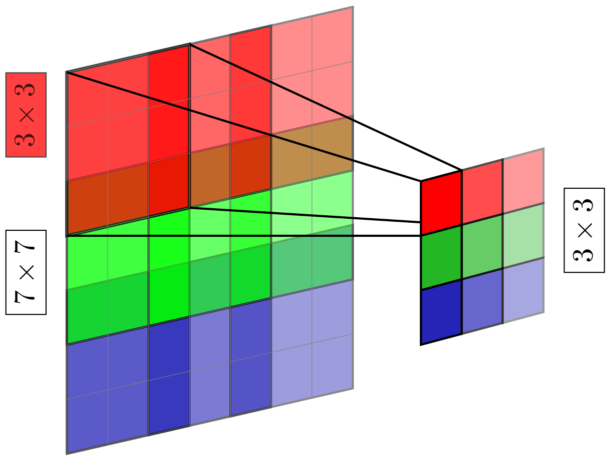

Figure 2.3. Convolutional Layer

In Figure (2.3), we have an example of a two-dimensional convolutional layer. In this thesis, only two-dimensional convolutions are used and from now will be referred to as convolutions. The spatial size of the input image is 7 times 7 and the kernel size is 3 by 3 specified in the red box. The color of the rectangles represent different pixel values. In the first convolutional operation only a square of the kernel size in the left upper corner is used as an input. Now the input is multiplied with the kernel which means every pixel value is multiplied with a weight and a bias. In the next operation the kernel moves with a stride of two to the upper middle of the image and again multiplies the kernel with the new inputs. This is done for the whole image and at last we receive an output of the size three by three with only 18 parameters for this convolutional layer.

A color image does not only have a spatial resolution with pixel values but also encodes the color in three r (red) g (green) b (blue) channels. In this case, the two dimensional kernel takes in all inputs along the channel dimension to create the output. Furthermore a convolutional layer can consist of an arbitrary number of output channels. Each channel then has its own weights and can, hence, filter the image for a interesting property e.g. a horizontal line in an image. Convolutional layers can be successfully used to extract more general information. The more often a convolutional layer is applied i.e. the deeper the network, the more general the extracted information will be [6]. A convolutional operation can be also be interpreted as folding the input space and was prior named Faltung which is german for folding. This means at every convolutional layer, the input space is folded. With many folding projections of the input space we can disentangle, at a high depth in a network, complicated structures in the input space.

Downsampling is used in a layer and describes a process to decrease the spatial size, and with that the spatial information of the input.
This is done to access more general information in a deep network, for example, used for image classification.

In the Figure (2.3) we already used a downsampling layer in form of a convolution with a stride of two which reduces the spatial size. If the spacial resolution of the input is even, and so is the kernel size, a stride of two cuts the resolution in half. If the kernel size is odd then the we can not stride over the whole image we have to introduce padding. Padding will stack a row and column of pixels to each dimension of our input image to address the issue that the input can not be divided by the kernel size without a remainder. Therefore we use zero padding as padding which adds pixels with a value of zero.

Upsampling is the counter part to downsampling and with that is used to increase the spatial size of the input. It is often used to create, from a lower dimensional representation an image.

A convolutional layer can achieve the desired increase in the spatial size when the process is reversed. If we look at the Figure (2.3) from right to left we could interpret it as a transposed convolution with again a kernel size of three by three and a stride of two. It is called transposed convolution because the matrix for the kernel is simply transposed compared to a normal kernel of a convolution. A disadvantage of this process is that images created with transposed convolutions are can sustain checkerboard artifacts due to the overlapping kernels. This would require the network to actively counteract this by adjusting and reducing the weights on the rim of the kernel which not always works perfectly. In Figure (2.3) we can see these checkerboard artifacts where the kernels overlap. To prevent and still use convolutions we can either use the same stride as the kernel size or use normal convolution, preserve the spatial size but create four times as many channels as specified. Afterwards, we can concatenate from the depth the extra channels along the spacial dimension to create an output which has twice the spatial size of the input with the specified amount of channels.

2.7. Artificial Neural Networks

The Autoencoder (AE) [5] is a neural network consisting of an encoder and a decoder which learns to reconstruct images.
The encoder $\mathfrak{E}$ receives an image x from the image space $\mathcal{X}$ and successively uses downsampling methods and activation functions to learn to effectively encode the image into a low dimensional latent representation z in the latent space $\mathcal{Z}$, also called the feature space,

$$\mathfrak{E}: \mathcal{X} \to \mathcal{Z}\,, \quad x \mapsto z \tag{2.12}$$

which should, ideally contain all important features of the image.
The decoder $\mathfrak{D}$ receives the latent representation z from the encoder as an input and successively applies upsampling methods as well as activation functions to generate an image x' of the same size as the original input image

$$\mathfrak{D}: \mathcal{Z} \to \mathcal{X}\,, \quad z \mapsto x'\,. \tag{2.13}$$

To train an autoencoder, we can use a reconstruction loss, for example, the L2 loss which then compares every pixel value of the input image x with the pixel value of the reconstructed output image x'. An optimzer can use this loss to backpropagate through the network and adjust the weights accordingly. If the training data set of input images is sufficiently large and the network trains long enough, the autoencoder will learn a complex encoding of the data distribution and will reconstruct the input images successfully.

The Variational Autoencoder (VAE) [10] is similar to an autoencoder with the key difference that it is possible to sample meaningful latent representations to generate images without an input image.
Instead of directly transferring the encoder output to the decoder, a distribution is created in the latent space. The encoder outputs (z_μ, z_σ) which are interpreted as the expected value and the standard deviation of a normal distribution $\mathcal{N}(z_\mu, z_\sigma)$ from which then a sample z is used to input into the decoder

$$\mathfrak{E}: \mathcal{X} \to \mathcal{N}\,, \quad x \mapsto \mathcal{N}(z_\mu, z_\sigma) \tag{2.14}$$

$$\mathfrak{D}: \mathcal{Z} \to \mathcal{X}\,, \quad z \mapsto x' \qquad \text{with} \quad z \sim \mathcal{N}(z_\mu, z_\sigma)\,. \tag{2.15}$$

To create a meaningful latent representation we use the Kullback–Leibler divergence to quantifies the difference between the latent distribution and a normal distribution around the origin with the standard deviation of one. This quantity helps us to enforce the wanted normal distribution through a loss function.

The Generative Adversarial Networks (GAN) were developed by Goodfellow et al. [4]. The idea is to train a generative network to generate images and a discriminative network to evaluate these images if they are likely to be from a given data distribution. The generator will be trained to at last create images which will be indistinguishable from the original data set while the discriminator tries to learn to distinguish them.

The Discriminator does the classification part which has to predict if input images either belongs to the training data set or are created by the generator network. The network creates a single scalar output in the range between zero and one which classifies a real image and a fake image respectively. We define now that for an output value one the networks predicts the input to be from the training data set and for a zero output the images are generated.

The Generator uses a decoder to creates images from latent representations or sampled noise from the latent space. The goal is to learn a mapping from this latent space to the desired data distribution of the training data and therefore create realistic images, indistinguishable by the discriminator.

2.8. Latent Representation Research

The Uniform Manifold Approximation and Projection (UMAP) for Dimension Reduction can be a helpful tool to visualize high dimensional data from the latent space in a two-dimensional plot. UMAP improved some flaws of the tSNE dimension reduction as for example using a cross entropy loss function instead of a KLD loss to also better preserve global distances. It is still very important to keep in mind that the UMAP reduction can not preserve all information and with that can only be used to some extend for well-founded statements. Nevertheless it can be very helpful to show and display connections and clusters of data with specific parameters.

The Principle Component Analysis (PCA) [13] creates a basis of principle components (PC) along the highest variances of the data. These PC reduce the average squared distance of all data points to them self. They are derived from covariance matrix of the given data. Calculating a eigenvector of the covariance matrix gives us a PC with their eigenvalue describing the amount of variance in the data along this PC.

Chapter 3.

Approach

3.1. Embedding articulations in latent space

We want to examine the latent representation of an image which displays a three-dimensional, rigid but articulated object. An articulated rigid object is defined in this work as a set of cuboids that are connected by one edge which acts as a hinge.
The goal of our research is to visualize and potentially improve how articulations are represented by a deep variational autoencoder model. To implement this, we need an encoder $\mathfrak{E}$ which creates a representation $\mathfrak{E}(x) = z$ in the latent space $z \in \mathcal{Z}^N$ with the latent dimension N, given the input image $x \in \mathcal{X}^{W \times H}$ with image width and height W, H respectively. In order to verify that the encoder representation includes all necessary information of the image, we want to reconstruct an image $x' \in \mathcal{X}$ solely from the latent representation z. This task is solved by the decoder $\mathfrak{D}$ as stated in Equation (2.13) creating an image $\mathfrak{D}(z) = x'$.
Furthermore we want to sample $z_s \in \mathcal{Z}$ and create new, unseen images. In an autoencoder the mapping to the latent space can seem very arbitrary and for inputs, not created by the encoder the decoder will create images with unpredictable artifacts.
A variational autoencoder tackles this problem by creating normal distributions from the encoder outputs z_μ and z_σ as shown in Equation (2.14) and Equation (2.15). The latent representation is created by sampling from this normal distribution which then is fed to the decoder creating the output image. Additionally to the reconstruction loss the Kullback-Leibler divergence enforces the distribution to be roughly around zero with a standard deviation of one, resulting in a robust and continuous mapping of the latent space to the image space. This enables us to create new images from the latent representation using the expected value and the standard deviation as the input. The experiments and results for the ideas up to here can be found in the section Preliminary Studies.

3.2. Creating valid articulations from latent space

Not only do we want to sample $z_s \in \mathcal{Z}$, we also want to ensure that this latent sample, created not by the encoder is a valid representation of an articulation α. For example an image x_α from the training data set displays an articulation α which is embedded in the latent space with $\mathfrak{E}(x_\alpha) = z_\alpha$.
We want that with $\mathfrak{D}(z_s) = x_s$, the generated image x_s should also display a valid articulation α similar to articulations in the training data set. This is why we need a discriminator

network which is a part of The Generative Adversarial Networks (GAN), to ensure generated images are similar to images from the data set.
The classification task of the discriminator is to distinguish input images belonging to the training data set from images created by the generative network. In our case the generative network is not only a decoder but a variational autoencoder (VAE) which has to achieve the image reconstruction task. Additionally the VAE has to learn depending on the discriminator output, how to map the latent space to a similar data distribution as the training data set. If the VAE model operates as planed we can sample $z_s \in \mathcal{Z}$ and create new images x_s alike the images in our training data set with similar articulations α.
Both networks have to be about equally good to maximize their benefit and learning from each other. If the discriminator always knows with a high accuracy, which images are created by the generator, then the generator will have difficulties to adjust its weights that much to fool the discriminator again. The balance of the two opposing network has to be preserved during training. In general the discriminator is more likely to achieve a high accuracy. To reduce that we can control if at every step of the training process the discriminator should be updated. The corresponding experiments are in section Introducing a Discriminator Network.

3.3. Interpolation of articulations

While the quality of the images is an important training signal and a good way to interpret how well our model has learned to represent articulated objects, we are interested in how these images articulations are related to each other. To do this, we will look at representation z_1 and z_2 of two given images x_1 and x_2 from the data set, with articulation $\alpha_1 \neq \alpha_2$ and interpolate z_{int} between z_1 and z_2. With this interpolation we can display the direct embedding of the articulations in the latent space, by creating new images $\mathfrak{D}(z_{int}) = x_{int}$. We can then view and compare x_{int} with images which interpolate naturally the articulation parameter α_1 and α_2. To create these images and interpolate the articulation parameters we need the custom tool used to create images of three-dimensional objects with specified parameters. The chapter Data set explains how the training data is created and with that how to create a image sequence with interpolating parameters.
We investigate the difference of a latent interpolation and the representations of an articulation sequence. To reduce the dimensionality of the latent space we will use UMAP to create a two-dimensional plot. This enables us to view both interpolations in the latent space. The experiments showing the results can be found Interpolation in latent space.
To further optimize the results of the latent interpolation, we will enforce a similar latent representation for images with the same articulation using a triplet metric loss. The metric loss compares distances of representations in the latent space of images with similar and

dissimilar articulations. This loss will be used during the training process to create a new model with similar representations of images with related articulations. The similarity is defined by a threshold value for the different articulation parameters. The following criteria have to be met by two articulations to be similar.

1. The same amount of cuboids.
2. The difference of the horizontal rotation $\Delta\Phi < 30°$.
3. The difference of the angle between two cuboids $\Delta\Theta < 30°$.
4. The difference of the scale $\Delta\Lambda < 0,5$.

Using this similarity metric we can assign positives and negatives for a given anchor. This aims to create better results for interpolations in the latent space. These results can be found in second part of section Interpolation in latent space.

3.4. Bases in the latent space

We want to investigate different bases of the latent space and find out how good they can describe the embedding of the data set in the latent space. To do that we will construct vectors in the latent space and use the decoder to generate new images. The interesting question is how we will construct the latent vectors.
The first approach is to use a vector of the size of the latent dimension containing only ones for every element. We can now multiply a scalar with this vector to generate images and gain a small overview of this high dimensional euclidean space. In theory the trained model should use a mapping around zero to one which was enforced during the training with the KLD loss. Nevertheless the representations of the data set do not necessarily have to be an multiple of the vector we are using. To tackles this problem we will try to find a better base in the next approach.
Using the The Principle Component Analysis (PCA) we will calculate a base along the maximum variances of all representations of the validation data set. We can sort the principle components by their corresponding eigenvalues. Now the first principle component will describe the axis on which the highest variance of the data set representation is. We can normalize this vector and multiply it with a scalar value to generated images just like in the first approach. But now we should receive images which are more likely to be similar to the original data set.

3.5. Technical implementation

We used a variational autoencoder and a discriminative network. For resource reasons, we always resized the images of the used data sets to a square resolution of 64 by 64 with three RGB channels. The VAE used six convolutional blocks to predict the 128 dimensional mean and standard deviation of the normal distribution in the latent space. One convolutional block contains a convolution with a kernel size three and a stride of two as well as a zero padding of the size one. This convolutional operation uses the convolutional module from the VUNet paper [2] implemented in pytorch. This module involves a normalized convolution with a scaling factor and an offset parameter. After the convolutional operation a batch normalization is used to at last apply the ReLU activation function to create an output for the next block. The channel dimension is doubled with every convolution while the spatial dimension of the output is cut in half. The maximum channel size is limited to 128 channels.
The architecture of the model is also shown in Figure (3.1)

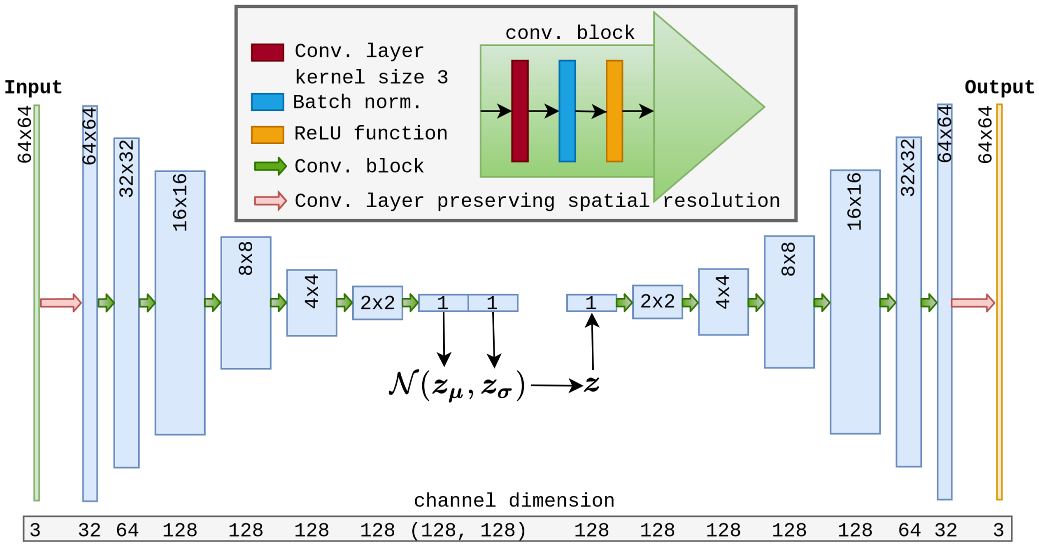

Figure 3.1. VAE Architecture

The decoder uses the similar convolutional blocks but instead of decreasing the spatial size we increase the spatial size. This is done by using a convolution which a keeps the same spatial size but increases the channel size by four. These channels are then concatenated along the spatial dimension. We use a kernel size of three and a stride as well as a zero padding of one. Additional to all these blocks the encoder uses one more convolution which preserving the spatial size. This convolution is applied before any other convolutions block is used. The decoder also uses the same additional convolution right before outputting the final reconstructed image.

Chapter 4.

Data set

To answer the questions asked in the Approach, we need a data set of images which display three-dimensional objects with explicit articulations and a perfect ground truth. Additionally we need image sequences of interpolations of articulations. In this chapter the tools to create such data sets will be explained.

4.1. Overview

The tools used to generate the data sets in this thesis were created beforehand in a project at the research group Computer Vision. The project provided a communication interface between a python client and a Unity 3D server application which can be understood as a mapping

$$A: \Gamma \to \mathcal{X} \times \Gamma,\ q \mapsto (x, q) \equiv x_q \tag{4.1}$$

from the parameter space Γ to the labeled image space. The parameters specify all information for the entire scene with articulations and appearances of the three-dimensional object within. The Unity server receives them through a TCP socket to then render the image x and send it back, creating an image with a corresponding ground truth x_q.
To create a data set D, we can consider a subset $\Gamma_D \subset \Gamma$ of the parameter space by specifying parameter ranges in the python file and requesting a set of N images x_q with random parameters $q \in \Gamma_D$. The data set $D \subset A\left(\Gamma_D\right)$ is described by the set

$$D = \{\ x_q \in A(M) \mid M \subset \Gamma_D \ \wedge\ |M| = N\ \} \tag{4.2}$$

with M being randomly sampled from Γ_D.

4.2. Example and Parameters

The three-dimensional objects displayed in the images are fairly simple. The base is always a cuboid with a given scale and a number of cuboids stacked on top of each other. Between every stacked cuboids there is the defined angle Θ, which is displayed in Figure (4.1). The angles in this example are $\theta_1 = \theta_2 = 61°$, the horizontal rotation of the whole object phi $\varphi = 28°$ and the scale $\Lambda_1 = \Lambda_2 = \Lambda_3 = 1.5$, referring to the length of a cuboid.

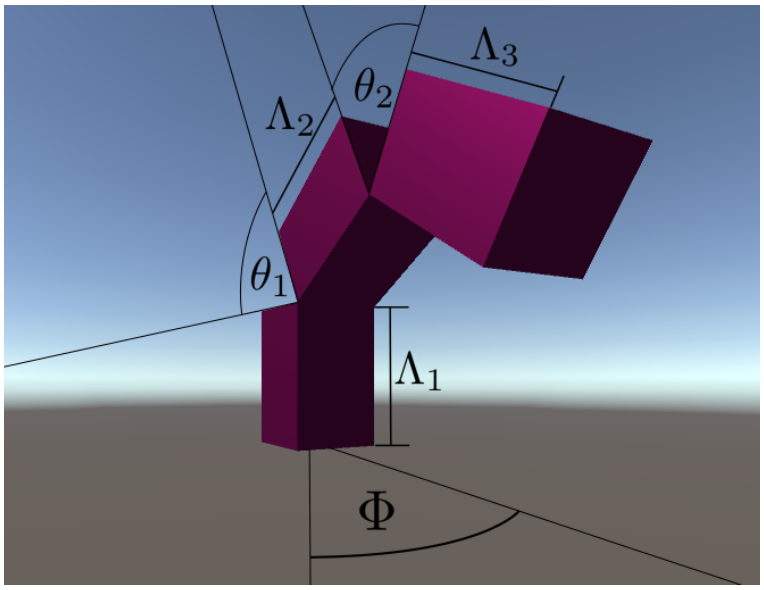

Figure 4.1. Data set example with annotations

The scale and angle does not have to be the same for every instance as it is shown in this example. There are many more parameters which specify the appearance, for example, the color and material settings applied to the cuboids. Also different light scenes can be created with different light types as well as different intensities, directions and colors.
For a detailed look into the data creation, we refer the reader to the git repository which includes a detailed report here[1] or the code documentation here[2].

1. https://github.com/R-Haecker/python_unity_images
2. https://python-unity-images.readthedocs.io/en/latest/

4.3. Different Data sets

Phi Data set consist of $N = 10,000$ samples containing only one articulation of two cuboids shown in Figure (4.2). The scene is defined by one parameter $q \in \Gamma$ with $\Theta_1 = 45°, \Lambda_1 = \Lambda_2 = 1.5$ and a fixed appearance. The parameter space Γ is restricted to Γ_Φ containing only the parameter q and all possible variations of the angle Φ in q. Hence, the articulation is horizontally rotated without any other changes throughout the whole data set varying the angle phi 10,000 times.

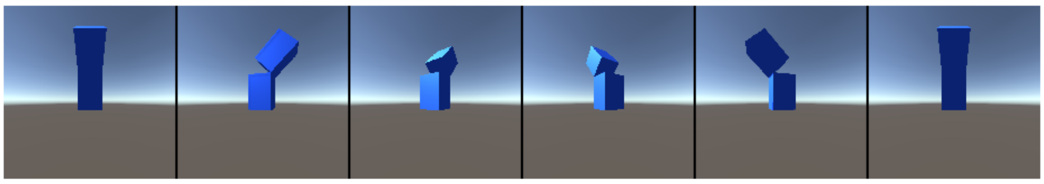

Figure 4.2. Six examples of the Phi Data set

Varied Data set consists of $N = 500,000$ samples displaying between two and four cuboids. To create this diversified data set we vary the articulation parameters Φ, Θ and Λ while allowing different angles between the cuboids but not different scales of cuboids. To introduce even more complexity we will not only vary the articulation parameters but including the appearances and lighting parameters into into the parameter space Γ_{VD}. This enables different colors, directional lights, up to four spotlights and four point lights with different settings to be randomly chosen in every image. Therefore, creating the following examples in Figure (4.3).

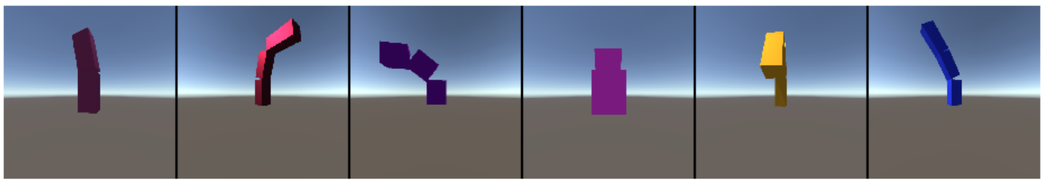

Figure 4.3. Six examples of Varied Data set

Chapter 5.

Experiments

5.1. Preliminary Studies

In this section we will explore the preliminary experiments which form the base for the final experiments.

Encoding and decoding articulations using an autoencoder, which consists of two convolutional layers and a fully connected layer each for the encoder and the decoder. The ReLU activation function was used after every convolutional layer and for the reconstruction loss we use the L2 loss function. We use the simple Phi Data set with only one articulation which is rotated in the scene.

Convolutions

We investigate different convolutional operations with varying stride and kernel size. The dimensionality of the latent space is fixed to 5 for these experiments. In Figure (5.1) we can see the results of training runs with 16.000 training steps. We crop out a patch of the input image and the reconstructed output image, to better display differences.

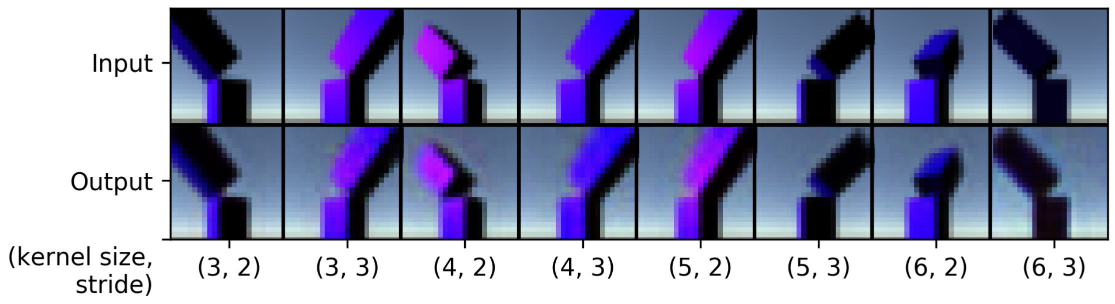

Figure 5.1. Results of different convolutions.

The output images are pretty similar in visual quality but the results of the kernel size 3 and stride 2 stand out a little bit with less artifacts and sharper edges. Additionally a small kernel size has been tested well in the recent years and has proven to be useful [6]. Furthermore, a convolution with the stride two cuts the spatial size of an input in half which introduces simplicity and scalability in the architecture of a model, for images of the resolutions of any power of 2. For these reasons we decide to use a convolution with a kernel size 3 and a stride of 2 with a zero padding of 1.

Latent Dimension

We want to find out how many latent dimensions the AE needs to represent the information given in a image from the Phi Data set as efficently as possible, i.e. with an optimal image quality to latent dimension ratio. In Figure (5.2), we display reconstructed images from the validation data set, created by models, which trained for 7.000 steps with different latent dimensions.

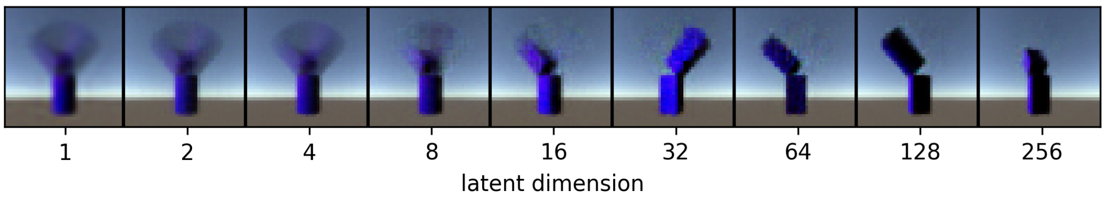

Figure 5.2. Results of different sized latent spaces.

For all latent dimensions smaller than 8 we receive only a mean over all images of the data set instead of a distinct image of the articulation. They fail to reconstruct the image properly. Even the models with a latent dimension of 8 or 16 are not fully capable to recreate the images. Only models with a latent dimension of 32 or higher can reconstruct the images with the specific articulation parameter. In this experiment, we use the simple data set but our model for the main experiments has to be able to cope with the complex Varied Data set. This leads us to the decision to use a latent space with 128 dimensions.

5.2. Introducing a Discriminator Network

Now we move on to a GAN architecture to ensure that images generated from a sample in the latent space will be valid images alike the images in our training data set. The following experiments are conducted with with the Varied data set.

Which Network to update when is an important question to balance the two adversarial networks. We can adjust how often and which of the networks in a GAN will be updated during a training process and investigate three different strategies. The simplest approach is to update **both networks** at every step.
In the second approach we will only update the network which has a higher error regarding the discriminator output. This implies we will update and train always the **inferior network**.
Additionally we add an uncertainty to this decision by also updating the superior network from time to time, which we will call **probabilistic inferior network**. This helps the inferior network to learn more general and robuster weights instead of exploiting specific

patterns in the discriminator network.
Another method to decide which network will be updated is to calculate the accuracy of the predictions the discriminator makes. This means we will update the generator at every step and if the discriminator accuracy is below a given **accuracy threshold** we update the discriminator as well. The accuracy is calculated across the input batch, which works best with a big enough batch size to create a representative sample.
Furthermore, we tested an optimization method which reduces the learning rate linearly at the end of the training process [3]. We can see the effects by comparing the first row with the second. In the Table (5.1) the results of the different updating methods are collected as Fréchet Inception Distances (FID) [8] scores with lower scores indicating better model. Each run performed 100.000 training steps on the Varied data set, using a batch size of 32. We only change the update method for the discriminator and calculate the FID score across 1000 randomly sampled images from the data set.

reducing learning rate	inferior network	probabilistic inferior network	both networks	accuracy threshold
without	50.53	36.39	73.51	39.23
with	52.57	29.14	53.55	28.09

Table 5.1. FID score for models with different updating methods.
Smaller values are better.

According to the results the two best updating methods are the probabilistic inferior network and the method using an accuracy threshold. We decide to use the accuracy threshold which gives us more control over the exact capabilities of the discriminator.

5.3. Charting the latent Space

In this section we try to find out how the articulation parameters are encoded and represented as well as how we can sample from them in the latent space.
We use the Principle Component Analysis to sample in the latent space. At the end of the training we stored the whole latent representation of the validation data set and with that could apply the principle component analysis to examine the distribution in the latent space. With the trained network we could then sample along the principle components with the highest eigenvalues to create new images. These images correspond to an interpolation along the highest variance of the data set which are more likely to represent the parameter Φ and are displayed in the Figure (5.3).

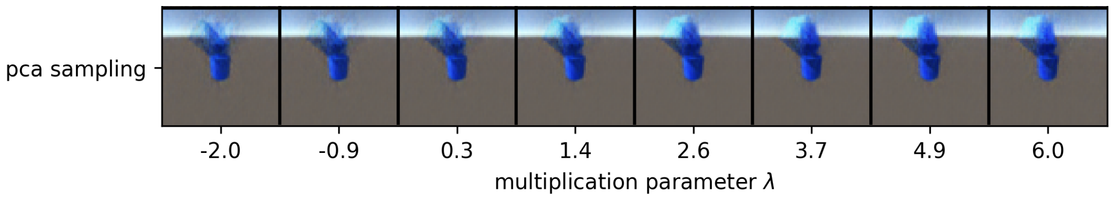

Figure 5.3. Images sampled from the principle components.

We can see that sampling along this principle component created images similar to the images from the data set indicating that on this axis, there are at least some representations of images from the data set. One way to visualize the latent space is to create a UMAP latent representation shown in Figure (5.4). We can see all latent representations of the validation data set as circles with the corresponding parameter Φ visualized by their color map between purple and blue. Additionally the PCA sampling from Figure (5.3) is plotted with triangles as markers and a green color map depending on the scaling factor of the normalised principle component.

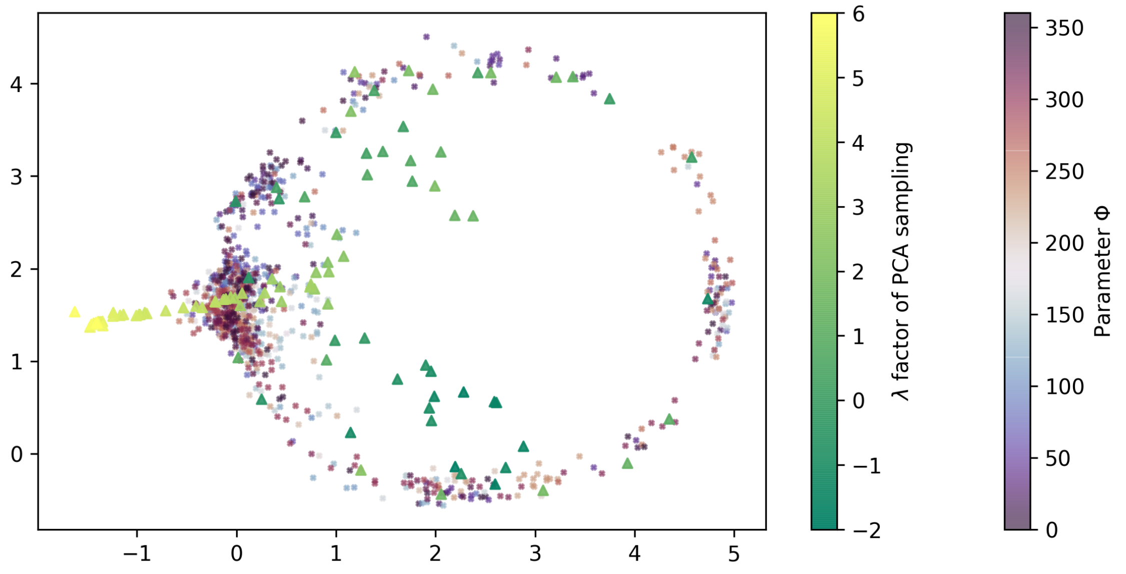

Figure 5.4. UMAP PCA latent representation.

As we can see the representations of samples from the principle components intersect and overlap with a fair amount of the representations from the data set, but does not fully follow all representations, hence can only sample similarly from a subset of the data set.

Adding a Metric Loss will force representations of similar articulations to be close to each other in the latent space. In Figure (5.5) we can view the same sampling along the PCA as in Figure (5.3) but for a model which was trained with a metric loss.

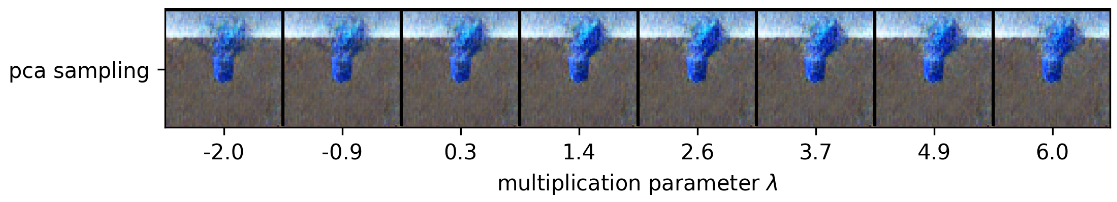

Figure 5.5. Images sampled from direct latent space and the principle components.

As we can see the results in are worse with images containing many artifacts. The metric loss should create a better structure for representations of images regarding the articulation of these images from the data set. For the principle component this can result in a worse samples because the mapping of representations of the data set can evolve to a very complex structure in the latent space. This different mapping can be observed in the Figure (5.6).

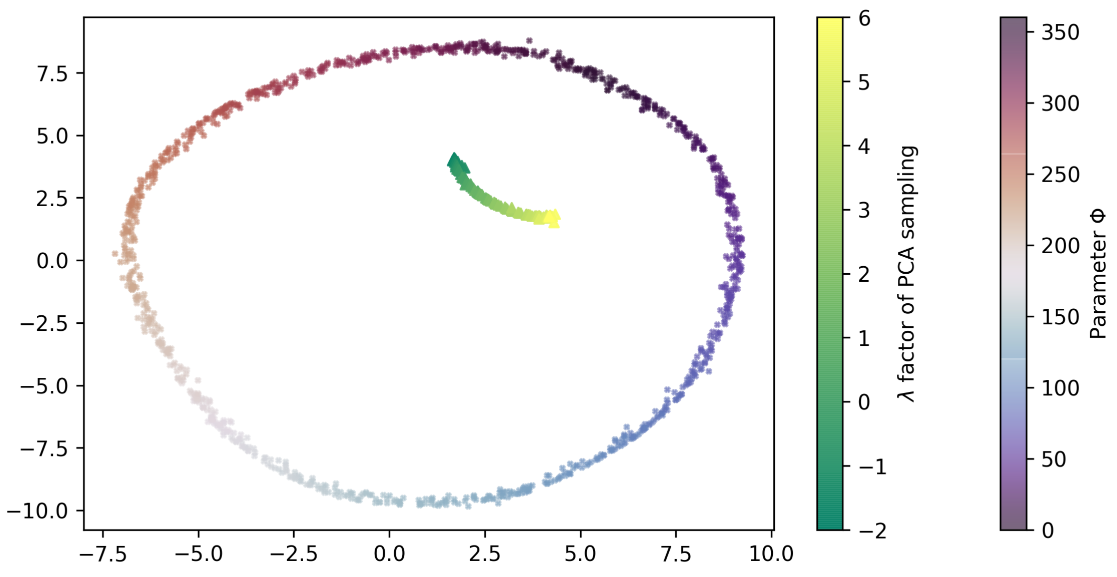

Figure 5.6. UMAP latent representation and euclidean interpolation.

We can see the impact the metric loss on the representation of the data set.

5.4. Interpolation in latent space

From now on we will focus on the latent space and the corresponding representation of the Phi data set in it. By training with this data set we force the network to extract and encode the information of the articulation parameter Φ. To understand how this encoding looks like we will interpolate between different latent representations.
First we interpolate the articulation parameter Φ by creating an image sequence with a

15° difference of the parameter Φ. We feed the input images sequence, in the upper half of Figure (5.8) into our model to create the reconstructed output images. The input images where picked from the Phi data set on which the model was trained on.

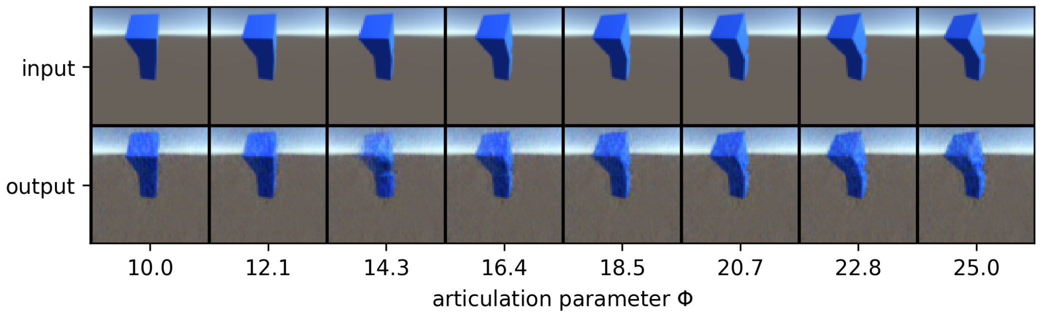

Figure 5.7. Input and output images linear interpolation of the articulation parameter.

We try to recreate the output image sequence by finding out how the articulation parameter is embedded in the latent space.
In Figure (5.8) we interpolate linearly between the first and the last latent representations of the images in Figure (5.7).

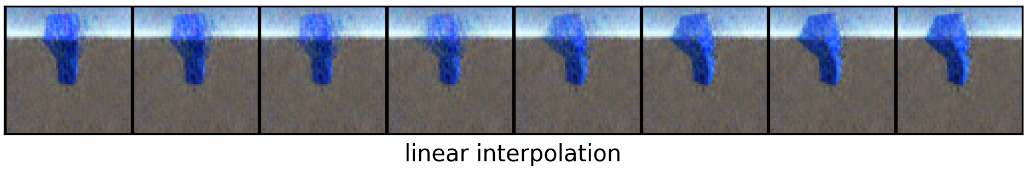

Figure 5.8. Images of a linear latent interpolation from the left, the first data point, to the right, the last data point.

We can tell by the artefacts in the images, that the linear interpolation does not follow exactly the latent representation of the image sequence from Figure (5.7).
Instead of interpolating linearly in a cartesian coordinate system of the latent space we can shift into a base more likely to come natural to the VAE, which would be the spherical coordinate system. The encoder of the VAE network outputs a multivariate normal distribution which creates ellipsoids around zero at equidensities, leading to a more natural representation in spherical coordinates. In Figure (5.9) we can see the images created by interpolating the latent representation along spherical parameters.

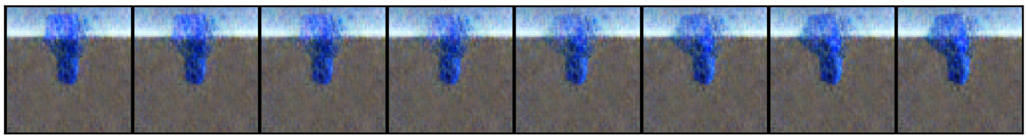

Figure 5.9. Images of a spherical latent interpolation from the left, the first data point, to the right, the last data point.

Instead of receiving images more alike the original image sequence the results show more artefacts indicating that the parameter Φ was more likely embedded linearly in the latent space.

To get an overview of the latent representations and the beforehand created interpolations we can use UMAP for dimension reduction of the latent space to create a two-dimensional plot. In the following Figure (5.10) we can see the start and end points as a red and blue cross. The interpolations use different color maps. The interpolation of the articulation parameter Φ is displayed in red and gets more yellow the higher the parameter Φ. The linear interpolation interpolates from zero, at the first data point to one, at the last data point. We use a blue color map with markers in the shape of triangles which point downwards. The spherical interpolation uses a green color map and markers of triangles, which face up. We used 100 data points for every interpolation.

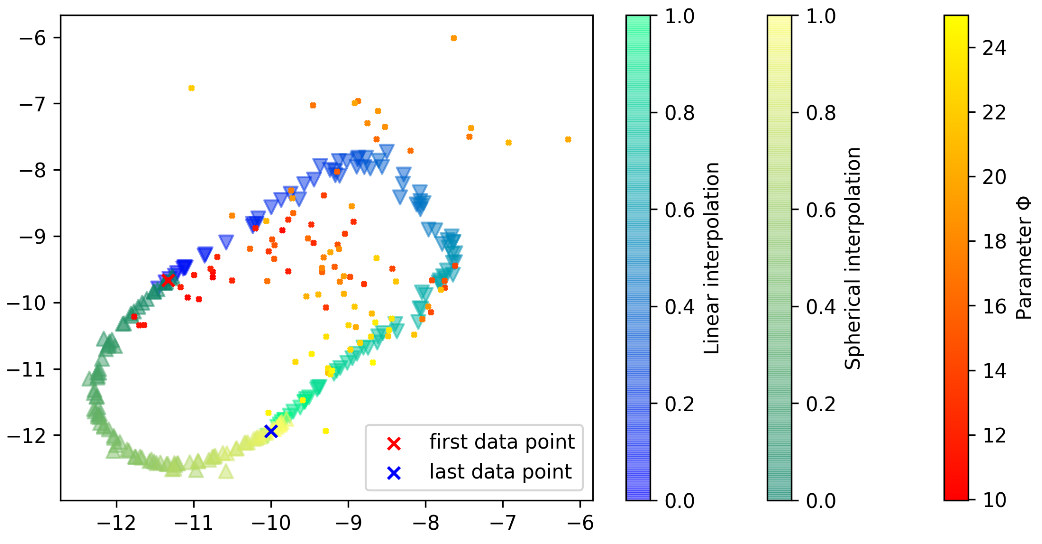

Figure 5.10. UMAP of interpolations in the latent space.

Some example images of these representations are shown in Figure (5.7) to Figure (5.9). In the UMAP plot we can verify the previous results. We can see that the linear latent interpolation overlaps with the representations of the image sequence which interpolates Φ. The spherical interpolation on the other hand does not overlap with these representations and with that clearly does not describe the embedding of the parameter Φ. In general the mapping follows roughly a linear interpolation.

Adding a Metric Loss to further enhance the encoding and reproduction of the articulation parameter Φ. This triplet loss should force similar articulations to be similar in their latent representations as well as to push apart representations which have different articulations. With a newly trained model we do the same interpolations from Figure (5.8, 5.9) again in Figure (5.11). If we apply the UMAP dimension reduction again and plot both interpolations as well as the validation data set we receive the Figure (5.11).

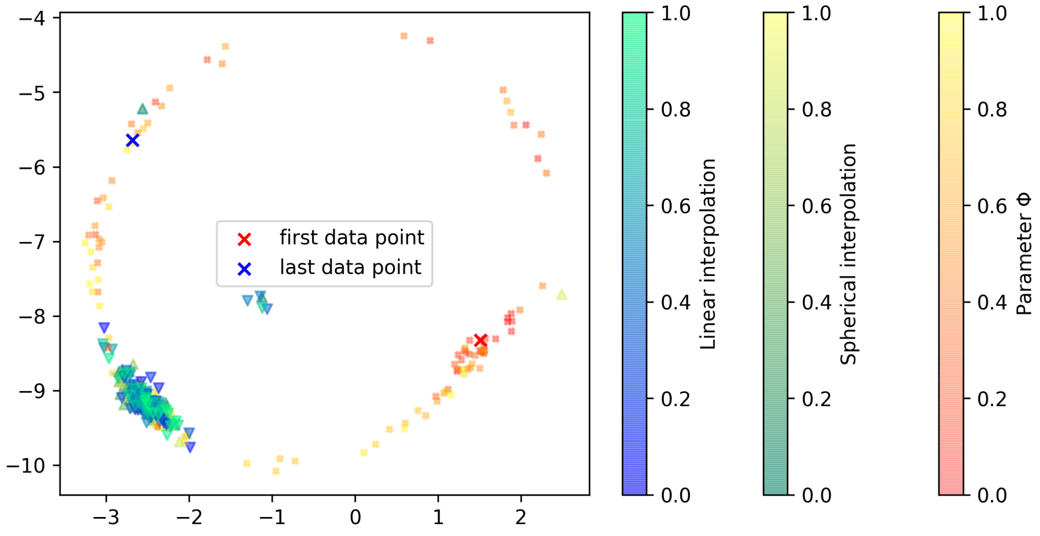

Figure 5.11. Images of linear and spherical latent interpolation.

This UMAP plot shows that as expected, the embedding of the phi sequence does create similar representations to correspond to similar articulations, but it does not improve the representations of the different latent interpolations.

Chapter 6.

Conclusions and Outlook

In this thesis we looked at the mapping of images into the latent space using an autoencoder with an adversarial discriminator. In a preliminary set of experiments we made a parameter search to find a suiting architecture for a subset of the data. Regarding the adversarial nature of our networks, we chose to use a threshold for the accuracy of the discriminator which can benefit both networks in the end.
In the next experiments we examined representations of different articulations displayed in the images in the latent space. Using especially created images with a ground truth for their articulation. We looked at the embedding of a articulation parameter the in the latent space. Furthermore, we conducted a principle component analysis on the data set on which the model was trained. We generated new images from representations along the principle component with the highest eigenvalue. Comparing how this principle component was embedded, in contrast to the articulation parameter. We used a UMAP for dimension reduction, to plot the sampled latent representations along the principle component and the representations created from images with given articulations. We saw that the principle component only intersected with some of the data points in the latent space but did not follow the embedding of articulation parameter. To improve this embedding, we used a metric triplet loss to pull latent representations of similar articulation together and push dissimilar articulations further apart. This resulted in a better structure of the latent representations of data points from the data set. But this did not improve the relation between the articulation parameter and the principle component, in the contrary the difference between them increased.
The last experiment focused on the interpolation in the latent space. We compared the latent representations of an image sequence with an interpolating articulation to the an interpolation in the latent space itself. The generated images of two different interpolations had many artifacts which indicated that they did not following the representations of the input image sequences well. Especially the spherical interpolation did not perform well. These results were again displayed in a UMAP plot confirming that the spherical interpolation was not a good way to interpolate an articulation. In contrast the linear latent interpolation did have many representations overlapping with input sequence and did follow roughly the articulation parameter. Adding again a triplet loss to our model did neither improve the linear interpolation nor the spherical but did add more structure to the representations from the data set.
Future research, focusing on articulations in the latent space should also make more use of the discriminative network by sampling during the training process in the latent space and ensuring real images from these samples. One could try to interpolate between representation during the training.

Appendix A.

First Appendix

A.1. Abbreviations

AE Autoencoder

BCE Binary Cross Entropy

GAN Generative Adversarial Network

KLD Kullback–Leibler Divergence

L1 Mean Absolute Error

L2 Mean Square Error

MAE Mean Absolute Error

MLP Multilayer Perceptron

MSE Mean Square Error

PC Principle Component

PCA Principle Component Analysis

UMAP Uniform Manifold Approximation and Projection

VAE Variational Autoencoder

References

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006 (cit. on p. 6).

[2] Patrick Esser, Ekaterina Sutter, and Björn Ommer. “A Variational U-Net for Conditional Appearance and Shape Generation”. In: (2018) (cit. on pp. 1, 14).

[3] Rong Ge et al. “The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure”. In: *CoRR* abs/1904.12838 (2019). arXiv: 1904.12838 (cit. on p. 20).

[4] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. eprint: arXiv:1406.2661 (cit. on pp. 1, 10).

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 4, 5, 6, 7, 9).

[6] Fred Hamprecht. *Machine Learning*. Lecture Notes. 2020 (cit. on pp. 8, 18).

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009 (cit. on p. 3).

[8] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium”. In: *CoRR* abs/1706.08500 (2017). arXiv: 1706.08500 (cit. on p. 20).

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. eprint: arXiv:1412.6980.

[10] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. eprint: arXiv:1312.6114 (cit. on pp. 1, 9).

[11] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. In: *Ann. Math. Statist.* 22.1 (1951), pp. 79–86 (cit. on p. 5).

[12] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. eprint: arXiv:1802.03426 (cit. on p. 1).

[13] Björn Ommer. *Deep Vision*. Lecture Notes. 2019 (cit. on p. 10).

[14] Prof Dr Björn Ommer. *Lecture notes in Artificial Intelligence*.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. eprint: arXiv:1505.04597 (cit. on p. 1).

[16] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A Unified Embedding for Face Recognition and Clustering”. In: *CoRR* abs/1503.03832 (2015). arXiv: 1503.03832 (cit. on p. 5).

List of Figures

Acknowledgements

First of all I want to thank Prof. Ommer and Prof. Plehn for the opportunity, to freely engage in the research field of machine learning. A special thanks goes to Johannes Haux, who always provided me with advice and motivation, even outside the fields of his of responsibilities. Thank you to Ritvik Marwaha for proof reading and thanks to all members of the Computer Vision research group for the welcoming atmosphere. Furthermore, I want to thank my parents for supporting me throughout my bachelor studies.

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise.

Heidelberg, 06.05.2020

Richard Häcker