Department of Physics and Astronomy
University of Heidelberg

Master Thesis in Physics
submitted by

Till Bungert

born in Lebach (Germany)

2021

An Exploration of Normalizing Flows for Outlier Generation

This Master Thesis has been carried out by Till Bungert at the
Interdisciplinary Research Center for Scientific Computing
under the supervision of
Prof. Dr. Ullrich Köthe

Abstract

Outliers are an important topic in statistics and machine learning. They can be hard to identify and what constitutes an outlier might even be subjective. In recent years, work has been done to use different generative models for outliers. This can not only allow one to investigate what potential outliers might look like directly, it can also be used to improve classification models, as particularly deep neural network classifiers using rectified linear units as activations have been shown to yield high-confidence predictions even far away from the training data. We propose using a normalizing flow to model the probability density of the training data and generate outliers by sampling from an outlier distribution in latent space. We improve interpretability and control over the generated samples by applying archetypal analysis to the latent space of the normalizing flow. We show how this leads to outliers with varying similarity to the inlier data. These outliers can also be used for subsequent tasks such as training confidence-calibrated classifiers.

Zusammenfassung

Ausreißer sind ein wichtiges Problem in der Statistik und maschinellem Lernen. Sie sind oft schwer zu identifizieren, und die Entscheidung, ob ein Messwert als Ausreißer gewertet wird ist meist subjektiv. In den letzten Jahren wurden Methoden entwickelt um mit generativen Modellen an die Ausreißerproblematik heranzugehen. Generative Modelle erlauben nicht nur die direkte Untersuchung von Ausreißern, sie können auch benutzt werden um Klassifikationsmodelle zu verbessern. Klassifikationsmodelle, die auf künstlichen neuronalen Netzen mit rectified linear units als Aktivierungsfunktion beruhen, weisen Klassen hohe Wahrscheinlichkeit zu, auch wenn Daten fern von den Trainingsdaten betrachtet werden. Wir schlagen vor, einen normalizing flow zu verwenden um die Wahrscheinlichkeitsverteilung der Trainingsdaten zu modellieren und Ausreißer zu generieren, indem im Repräsentationsraum von einer Ausreißerverteilung gezogen wird. Wir zeigen außerdem, wie die Interpretierbarkeit und Kontrolle der generierten Ausreißer verbessert werden kann, indem Archetypanalyse im Repräsentationsraum angewendet wird. So können Ausreißer mit variierbarer Ähnlichkeit zu den Trainingsdaten generiert werden. Diese Ausreißer können dann für weitere Aufgaben genutzt werden, z.B. das Trainieren eines Klassifikationsmodells mit kalibrierter Konfidenz.

Contents

Chapter 1

Introduction

Outliers and how to detect and handle them has been an important issue in statistics and now machine learning for a long time [32]. Even the question of what constitutes an outlier when considering a specific dataset or task can be hard to answer, especially when potential outliers are similar to the data.

In recent years, work has been done to use deep neural networks to deal with outliers. Nalisnick et al. [26] propose a method of outlier detection using typicality, evaluating the empirical distribution of likelihoods from several deep generative models. Hein et al. [12] show that classifier networks using rectified linear unit activations struggle with outliers as they yield high-confidence predictions even far away from the training data and try to mitigate this using an auxiliary loss term on outliers generated using Gaussian noise and filtering. Lee at al. [23] propose training a classifier jointly with a generative adversarial network to reduce prediction confidence on outliers. Schott et al. [29] estimate the data using a variational autoencoder network and perform classification by evaluating the class-conditional likelihoods of this generative model in such a way, that it converges to a uniform distribution over class labels for outliers. Meinke et al. [24] obtain mathematical guarantees for low confidence predictions far away from the training data by training a classifier jointly with Gaussian mixture models as estimators for the in- and out-distribution density estimation, to again maximize the assigned class-probabilities on in-distribution data and to yield a uniform distribution over class-labels on out-distribution data.

In this work we use a normalizing flow to model the density of in-distribution data and generate outliers from the latent space of the normalizing flow. In particular, we focus on outliers similar to in-distribution data, that might be hard to identify even for humans. These outliers can provide insight into what to expect and what outliers might be hardest to deal with. We show how more insight and control over the generated samples can be gained by using archetypal analysis to represent the latent space of the normalizing flow. We use the same methods to train a confidence-calibrated classifier similar to Lee et al. [23].

In chapter 2 we introduce outliers, normalizing flows and archetypal analysis. Following that, we go into more detail of how we implemented our methods in chapter 3. In particular, we go over how we achieve outlier sampling from the latent space of the normalizing flow. Finally, we present the results of our experiments in chapter 4.

In summary, our contributions are a method of sampling outliers using normalizing flows and a method of gaining more insights and control over generated samples using archetypal analysis.

Chapter 2

Theoretical Background

This chapter will give an overview of the theoretical background of the problem investigated in this thesis, as well as the methods used. We will first introduce the topic of outliers. We will move on to introduce neural network models especially in their invertible variant and how specific training schemes are particularly useful for investigating outliers. Finally, we will introduce archetypal analysis as a general concept and also as an add-on to more complicated models.

2.1 Outliers And Outlier Detection

Outliers are data points in a dataset, that differ significantly from other data points in the dataset in a way that makes them not indicative of other data from the same source. It is therefore necessary to determine what data points are outliers and how they should be handled in any task performed on the dataset [8]. Especially for extreme data points that still somewhat resemble the rest of the dataset it can be difficult to decide whether or not they are an outlier and this decision might ultimately subjective [32].

2.2 Invertible Neural Networks

Neural networks and deep learning methods have become staple methods in many fields that use machine learning. They are typically comprised of a chain of relatively simple, non-linear blocks. [22]

As representation-learning methods, neural networks have found use for many discriminative as well as generative tasks. Especially in recent years many advances were made for generative methods using neural networks. One such advancement was the introduction of invertible neural networks. Neural networks, though not invertible in general, can be made fully invertible by cleverly designing its building

blocks to be invertible.

2.2.1 Coupling Layers

In general, neural networks are composed of multiple layers or composite functions. The function $\mathcal{F}$ a neural network with L layers computes is given by

$$\mathcal{F} = f_L \circ \cdots \circ f_2 \circ f_1 \ . \tag{2.1}$$

If we design all layers to be bijective and have tractable Jacobian determinants the whole forward computation is invertible by composing the inverse layers in reverse order, i.e.

$$\mathcal{F}^{-1} = f_1^{-1} \circ f_2^{-1} \circ \cdots \circ f_L^{-1} \ . \tag{2.2}$$

Dinh et al. [6] describe a family of transformations called coupling layers, that can be used as components in an invertible neural network. In general, coupling layers are defined by Dinh et al. as follows

Definition 2.2.1 (General Coupling Layer) *Let $\mathbf{x} \in \mathbb{R}^D$ and I_1, I_2 a partition of $[1, D]$ such that $d = |I_1|$ and m a function defined on $\mathbb{R}^d$. We can then define $\mathbf{y} = [\mathbf{y}_{I_1}, \mathbf{y}_{I_2}]$ as*

$$\begin{aligned} \mathbf{y}_{I_1} &= \mathbf{x}_{I_1} \\ \mathbf{y}_{I_2} &= g(\mathbf{x}_{I_2}; m(\mathbf{x}_{I_1})) \ , \end{aligned} \tag{2.3}$$

where $g : \mathbb{R}^{D-d} \times m(\mathbb{R}^d) \to \mathbb{R}^{D-d}$ is the coupling law, an invertible map with respect to the first argument given its second. This transformation is called coupling layer with coupling function m.

Note that the choice of m is not constrained to invertible functions but can be arbitrary.

A simple choice for the coupling law would be an additive law [6, 11]. The coupling layer transformation then becomes

$$\begin{aligned} \mathbf{y}_{I_1} &= \mathbf{x}_{I_1} \\ \mathbf{y}_{I_2} &= \mathbf{x}_{I_2} + m(\mathbf{x}_{I_1}) \ . \end{aligned} \tag{2.4}$$

This transformation has unit Jacobian determinant, i.e. is volume preserving. This can give numerical stability especially if m is a rectified linear network. This however comes at the cost of what transformations can be learned.

A more versatile approach is to use affine coupling blocks, that allow for non-volume preserving transformations. The coupling layer transformation for the gen-

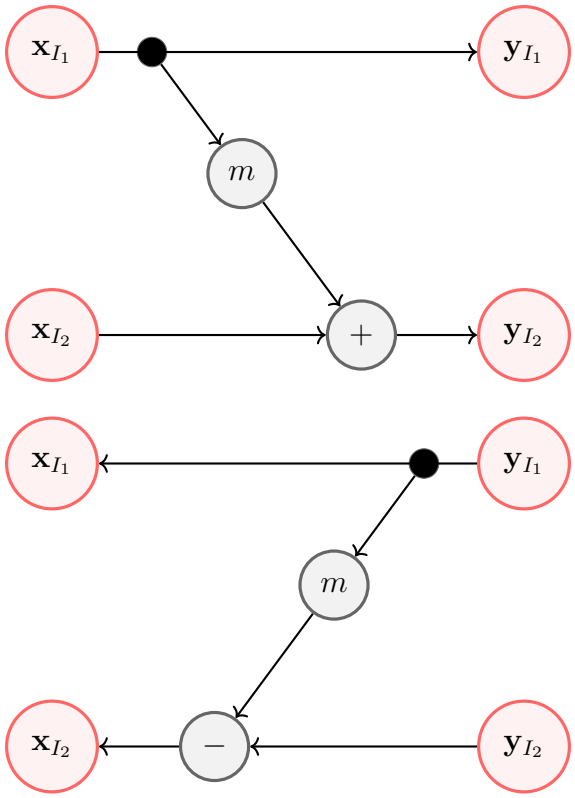

Figure 2.1: Additive coupling block. **Top**: forward computation, **Bottom**: inverse computation.

eral affine coupling block introduced by Dinh et al. [7] is

$$\begin{aligned} \mathbf{y}_{I_1} &= \mathbf{x}_{I_1} \\ \mathbf{y}_{I_2} &= \mathbf{x}_{I_2} \odot \exp(s(\mathbf{x}_{I_1})) + t(\mathbf{x}_{I_1}) \ . \end{aligned} \tag{2.5}$$

where s and t are arbitrary functions that map $\mathbb{R}^{D-d} \to \mathbb{R}^d$ and stand for scale and translation respectively and $\odot$ is the Hadarmard or element-wise product. This transformation is again easily invertible, regardless of the complexity of s and t

$$\begin{aligned} \mathbf{x}_{I_1} &= \mathbf{y}_{I_1} \\ \mathbf{x}_{I_2} &= (\mathbf{y}_{I_2} - t(\mathbf{y}_{I_1})) \odot \exp(-s(\mathbf{y}_{I_1})) \ . \end{aligned} \tag{2.6}$$

To ensure all inputs are used we need to use at least two coupling blocks and switch $\mathbf{x}_{I_1}$ and $\mathbf{x}_{I_2}$ between them.

2.2.2 Downsampling

In typical deep learning architectures, especially in the field of computer vision, the inputs are downsampled in the network. This is usually done with strided convolutions, maximum or average pooling. Since these operations entail a loss

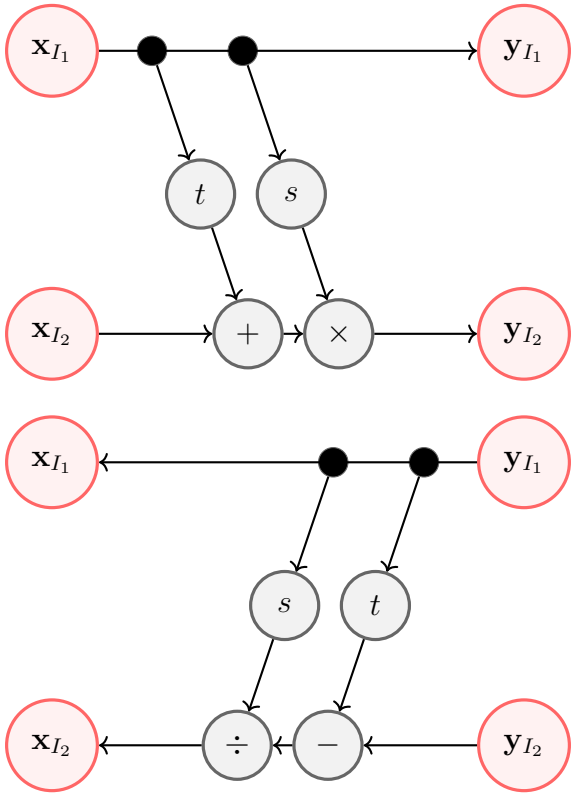

Figure 2.2: Affine coupling block. **Top**: forward computation, **Bottom**: inverse computation.

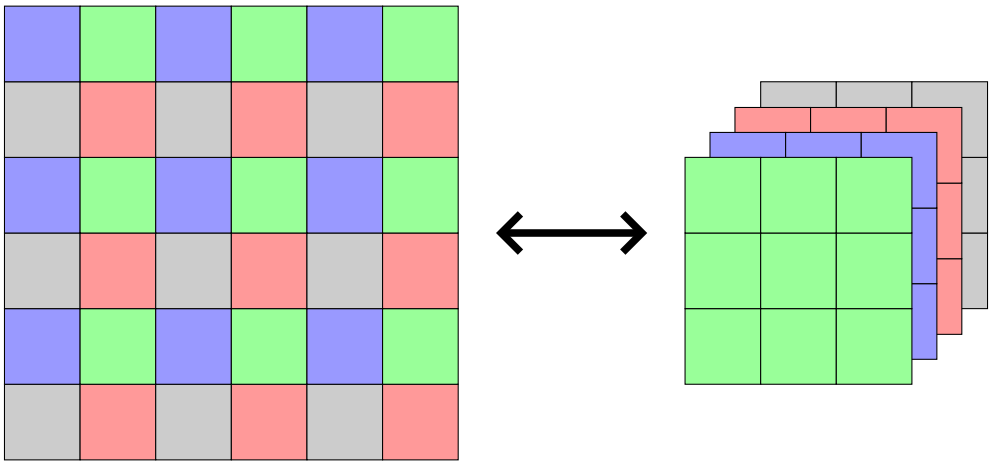

Figure 2.3: IRevNet Downsampling, figure adapted from Gomez et al. [11]. Images are split according to a checkerboard pattern in the spatial dimensions and rearranged along the channel dimension.

of information they are not bijective and cannot be used in an invertible neural network architecture, at least outside of coupling functions. To still be able to increase the field of vision of convolutions and decrease the amount of computations in subsequent layers different strategies can be used. Gomez et al. used a pixel shuffle operation, in which spatial dimensions are reduce and shifted into channel dimensions as shown in Figure 2.3 [11].

As a strategy to reduce the amount of computations done in deeper layers, data can simply be split of. The part that is split of can than be trained to resemble noise while the other part continues through coupling layers. Towards the end of the network the split parts are concatenated back together.

2.2.3 Normalizing Flows

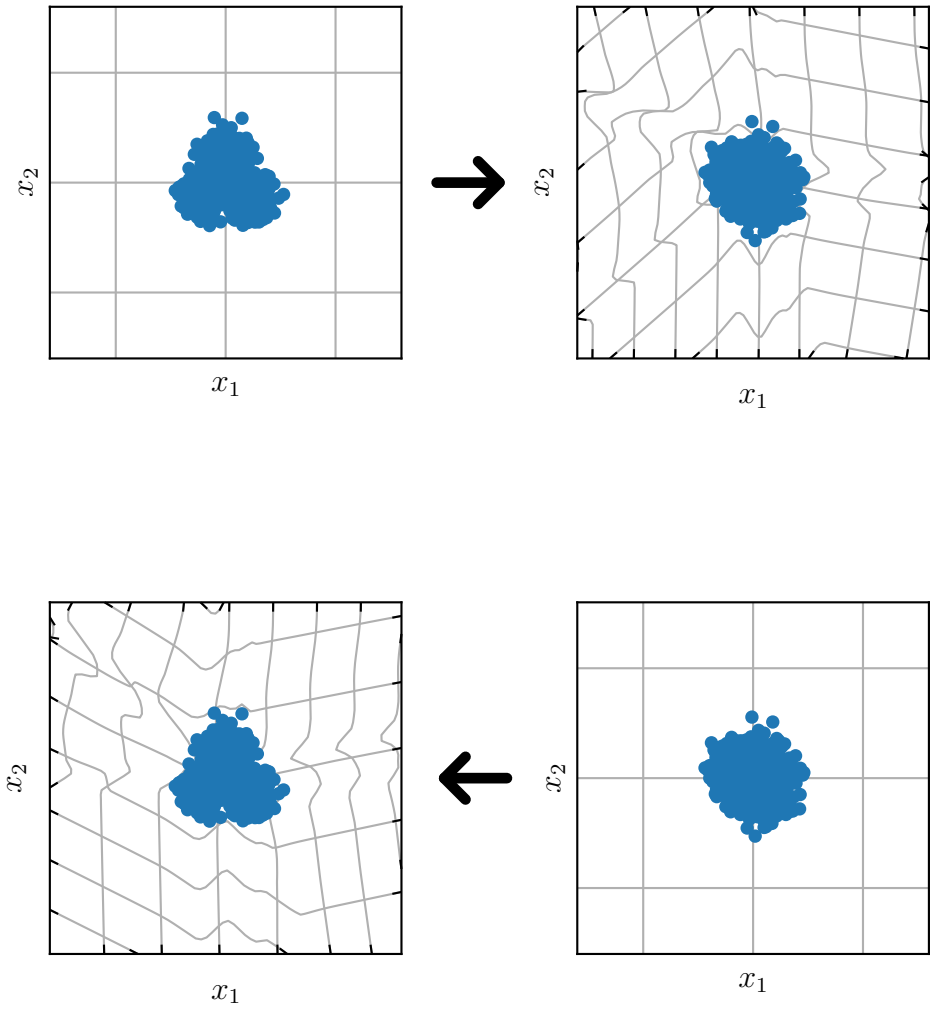

Figure 2.4: Mapping between data and latent space of a normalizing flow from a mixture of three Gaussians to a single normal distribution. The transformation of grid lines is also shown in both directions. Figure adapted from Dinh et al. [7]

Normalizing flows transform a simple probability distributions, such as a stan-

dard normal distribution, into a complex, often intractable probability distribution [18]. They are composed of a series of invertible transformations, e.g., the coupling layers we defined above. The complex probability distribution can be evaluated on a sample by transforming it back to the simple distribution and using the change of variables formula.

In more mathematical terms, consider vectors $\mathbf{z}_i \in \mathbb{R}^d$ that are distributed according to a simple distribution $p_{\mathbf{z}} : \mathbb{R}^d \to \mathbb{R}$. Let $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ be an invertible transformation (e.g. an invertible neural network). The probability density function of a variable $\mathbf{x}_i = \mathcal{F}(\mathbf{z}_i)$ can then be computed according to the change of variables formula

$$p_{\mathbf{x}}(\mathbf{x}_i) = p_{\mathbf{z}}(\mathcal{F}^{-1}(\mathbf{x}_i))|J_{\mathbf{x}}| \ , \tag{2.7}$$

where

$$J_{\mathbf{x}} = \det \frac{\partial \mathcal{F}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \tag{2.8}$$

is the Jacobian determinant of the transformation. Particularly, if $\mathcal{F}$ is defined as in Equation 2.1, the Jacobian is simply

$$J_{\mathbf{x}} = \prod \det \frac{\partial f_i^{-1}(\mathbf{x}')}{\partial \mathbf{x}'} \ . \tag{2.9}$$

To train a neural network as a normalizing flow we can simply maximize the log-likelihood of data samples, or equivalently minimize the negative log-likelihood. If $p_{\mathbf{z}}$ is the standard normal distribution, the negative log-likelihood loss for a mini-batch of size N can be simply calculated up to a constant as

$$\text{NLL} = \frac{1}{N} \sum_i \|\mathcal{F}^{-1}(\mathbf{x}_i)\|^2 - \log J_{\mathbf{x}} \ . \tag{2.10}$$

2.3 Archetypal Analysis

A common way to represent data by template examples is by using prototypes, especially in machine learning. Prototypes are typical examples of their class, that ideally entail all properties of their class. An alternative way, first proposed by Cutler et al., is to represent data samples as a mixture of individuals of pure type or archetypes [5]. A simple way to formulate this representation is by a variant of principal components. A data vector $\mathbf{x}_i \in \mathbb{R}^d$ from a dataset of n samples is represented by coefficients a_{ij} and $m \leq \min(n, d)$ archetype vectors $\mathbf{z}_j \in \mathbb{R}^d$, i.e.,

$$\mathbf{x}_i = \sum_j a_{ij} \mathbf{z}_j \ . \tag{2.11}$$

In turn, the archetypes are computed from the data using coefficients b_{ji} according to

$$\mathbf{z}_j = \sum_i b_{ji}\mathbf{x}_i \ . \tag{2.12}$$

Without further constraints this approach does not fulfill the requirements, that the templates are pure archetypes and that $\mathbf{x}_i$ are represented as mixtures of the archetypes. If we subject the coefficient to the following constraints:

$$\begin{aligned} a_{ij} \geq 0 \text{ and } \sum_{j=1}^{m} a_{ij} = 1 \ , \\ b_{ji} \geq 0 \text{ and } \sum_{i=1}^{n} b_{ji} = 1 \ , \end{aligned} \tag{2.13}$$

we obtain the desired representation. The archetypes now represent the data's convex hull.

Finding the weights a_{ij} and b_{ij} can be done by minimizing the square error in the representation

$$\min_{a,b} \sum_i \|\mathbf{x}_i - \sum_j a_{ij}\mathbf{z}_j\|^2 = \min_{a,b} \sum_l \|\mathbf{x}_l - \sum_j a_{lj} \sum_i b_{ji}\mathbf{x}_i\|^2 \tag{2.14}$$

Cutler et al. propose an alternating least squares algorithm to solve this optimization [5]. Bauckhage et al. introduced a more efficient algorithm based on subgradients [2]. Keller et al. build on this framework by applying archetypal analysis to the latent space of a deep generative model [15]. In their paper they propose using a Deep Variational Information Bottleneck. Here we will use a normalizing flow instead. The normalizing flow maps data vector $\mathbf{x}_i \in \mathbb{R}^d$ to a latent vector $\mathbf{t}_i \in \mathbb{R}^d$ that is approximately normally distributed. Additional linear layers with softmax activations map the latent vectors to archetype coefficients. As there is no absolute frame of reference for the latent space coordinates, the k archetype vectors are initialized to be the vertices of a $(k-1)$-simplex. The archetype vectors can be fixed when training the flow, or can be allowed to update to accommodate a pretrained flow. The full model will be discussed in the next chapter.

Chapter 3

Methods and Experiments

In this chapter we introduce our methods in more detail. In particular, we show how we sample outliers from our models, how these models are implemented and what datasets we use to evaluate our methods.

All our experiments are implemented using the PyTorch library [27] and the *Framework for Easily Invertible Architectures* [1] developed by the Visual Learning Lab in Heidelberg.

3.1 Extremal Sampling

Since our method relies on sampling outliers in the latent space we will now examine how this can be accomplished. The latent space is trained to be approximately Gaussian in the case of the normalizing flow, so we can look at two methods of sampling outliers from a Gaussian distribution. Sampling outliers in the Deep Archetypal Analysis case will be investigated in the next section.

3.1.1 High-Dimensional Gaussian Distributions

Since we are working with high-dimensional data, we can investigate the behavior of random vectors in high dimensions. Consider the vector $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ in $\mathbb{R}^d$, where the coordinates x_i are independently distributed with zero mean and unit variance. The squared length of vector $\mathbf{x}$ is

$$\|\mathbf{x}\|_2^2 = \sum_d x_i^2 \,. \tag{3.1}$$

[1] https://github.com/VLL-HD/FrEIA

If we assume the coordinates x_i to be standard normally distributed, the length of the vector is distributed according to a chi distribution with d degrees of freedom

$$\|\mathbf{x}\|_2 \sim \chi_d \ . \tag{3.2}$$

If the coordinates are distributed with non-unit variance, $x_i \sim \mathcal{N}(0, \sigma^2)$, we similarly get

$$\|\mathbf{x}\|_2 \sim \sigma\chi_d \tag{3.3}$$

Assessing the length of the random vector $\mathbf{x}$ via it's expectation value

$$\mathbb{E}\|\mathbf{x}\|_2 = \sqrt{2}\frac{\Gamma(\frac{d+1}{2})}{\Gamma(\frac{d}{2})} \ , \tag{3.4}$$

and variance

$$\mathrm{Var}\|\mathbf{x}\|_2 = d - (\mathbb{E}\|\mathbf{x}\|_2)^2 \ , \tag{3.5}$$

[9] suggests that even though the area of highest probability for a high-dimensional standard normal distribution is close to the origin, most of the mass will be in a thin, hyperspherical shell with radius $\sim \sqrt{d}$.

If we want to sample from a shell around a high-dimensional standard normal distribution, we can simply draw coordinates $x_i \sim \mathcal{N}(0, \sigma^2)$ and choose σ such that a desired radius is reached.

3.1.2 Gumbel Distribution

Since the latent space is approximately standard normally distributed one way to approach sampling from outliers is extreme value theory — the theory of the distribution of sequence maxima of independent, identically distributed (i.i.d.) random variables. Let $z_1, \ldots, z_n$ be a sequence of i.i.d. standard normally distributed random variables, then the asymptotic distribution of the maxima $M_n = \max(z_1, \ldots, z_n)$ is

$$P(a_n(M_n - b_n) < x) \to e^{-e^{-x}} \ , \tag{3.6}$$

where

$$\begin{aligned} a_n &= (2\log n)^{1/2} \ , \\ b_n &= (2\log n)^{1/2} - \frac{1}{2}(2\log n)^{1/2}(\log\log n + \log 4\pi) \ . \end{aligned} \tag{3.7}$$

[20].

Figure 3.1 shows an exemplary distribution of the maxima of sequences of length $n = 100$ and $n = 100000$ drawn i.i.d. from a standard normal distribution in comparison.

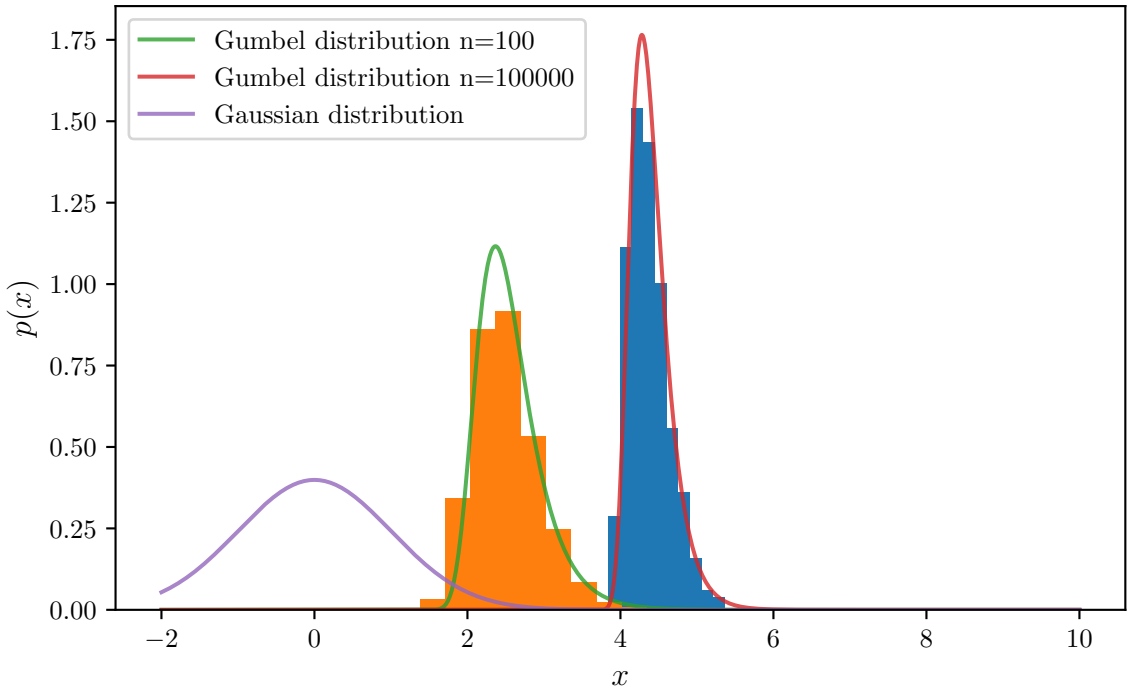

Figure 3.1: Comparison of a standard normal distribution in one dimension with Gumbel distributions for sequences of length 100 and 100000.

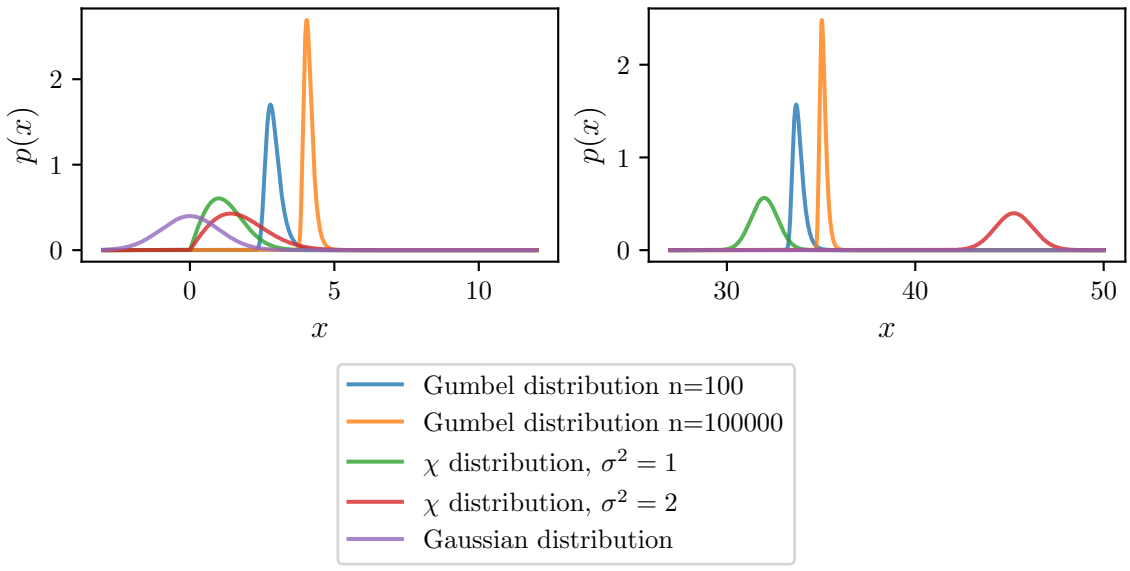

Figure 3.2: Comparison of the distribution of the length random vectors from a multivariat normal distribution to the Gumbel distribution in low and high dimensions. Gumbel distributions are shifted so the mean of the underlying normal distribution would be the mean of the χ distribution.

This method of sampling outliers is especially useful in lower dimensions, as we cannot use the methods established in subsection 3.1.1. Figure 3.2 illustrates, that for $d = 2$ the distribution of the lengths of the random vectors is still rather close to a standard normal distribution, and indeed doubling the variance does not lead to a distribution that is noticeably different from the data distribution. For $d = 1024$, as is the case for the image data used for the experiments, we can use the previously introduced method of sampling from a latent distribution with higher variance, as this leads to a significantly distinct distribution.

3.2 Geometrical Approach

In order to make use of the archetype representation of our data we need to sample archetype coefficients. Sampling from our deep archetypal architecture can be done as follows. The archetypes, whether they are updated during training or not, are taken to be fixed and saved. For a fixed configuration of archetypes the coefficients a_{ij} can be sampled from a Dirichlet distribution, a generalization of the beta distribution for the multivariate case. For k archetypes and shape parameters $c_i > 0, i = 1, \ldots, k$ the probability density function of the Dirichlet distribution is

$$p(a_i) = \frac{\Gamma(\sum_j c_j)}{\prod_j \Gamma(c_j)} \prod_j a_{ij}^{c_j - 1} \ . \tag{3.8}$$

[10]

Since the archetypes are already extreme examples, we can control how atypical the samples should be and furthermore what features we want to be extreme by controlling the shape parameters.

Once N samples have been drawn we can compute samples from the $(k-1)$-dimensional latent space via

$$\mathbf{t}'_i = \sum_j a_{ij} \mathbf{z}_{\mathbf{j},\text{fixed}} \ . \tag{3.9}$$

In addition to the layers that map data from the latent space of the invertible neural network to the archetype space we need to train a layer f_C that approximates the inverse to this, as a mapping to the $(k-1)$-dimensional subspace is in general not invertible. The layer is chosen to be a linear layer with no activation function, details on how this layer is trained can be found in section 3.4

$$\mathbf{t}_i = f_C(\mathbf{t}'_i) \tag{3.10}$$

Finally we can use the neural network to get samples in data space:

$$\mathbf{x}_i = \mathrm{INN}^{-1}(\mathbf{t}_i) \; . \tag{3.11}$$

Additional modifications can be applied: We can add Gaussian noise to the samples in any of the latent spaces to get more variation or more extreme samples. Another idea is to change the constraint on the coefficients to be quadratic, but initial tests have proven unsuccessful so this approach was not investigated further.

Another modification is to add a random sample to $\mathbf{t}_i$ from the nullspace of the mapping to the archetype space to mitigate the loss of information that occurs. Ignoring the additional softmax activation the mapping of the data onto archetype coefficients is linear. If M_A is the matrix of the weights of this linear network layer, then the mapping can be described as

$$\mathbf{a}_i = M_A \mathbf{t}_i \; , \tag{3.12}$$

where $\mathbf{a}_i$ are the resulting archetype coefficients for the data vector $\mathbf{t}_i$. Mapping archetype coefficients back into the latent space of the network can then be thought of as solving this linear equation for $\mathbf{t}_i$. However, in general a solution $\mathbf{t}'_i$ to this linear equation is not unique and

$$\mathbf{t}''_i = \mathbf{t}'_i + \mathbf{k} \tag{3.13}$$

is another solution to the equation. Here, $\mathbf{k}$ is a vector from the kernel or nullspace of M_A, i.e. the space of all vectors that are solutions to $M_A\mathbf{x} = \mathbf{0}$. All solutions to Equation 3.12 are given by

$$\mathbf{t}_i = M_A^+ \mathbf{a}_i + [I - M_A^+ M_A]\mathbf{w} \tag{3.14}$$

where $\mathbf{w}$ is an arbitrary d-dimensional vector and $M_A^+ = (M_A^\top M_A)^{-1} M_A^\top$ is the Moore-Penrose inverse of M_A [14]. As in our case the mapping is not linear but has an additional softmax activation we cannot use this result as is. However, since we train this whole setup to be invertible via the linear mapping f_C the approximation is close enough to still improve the results of the sampling, as shown in subsection 4.3.1. We simply augment the result of f_C similarly to Equation 3.14 using M_A as if no softmax was applied

$$\mathbf{t}'_i = f_C(\mathbf{a}_i) + [I - M_A^+ M_A]\mathbf{w} \tag{3.15}$$

where $\mathbf{w}$ is sampled from the latent space distribution of the normalizing flow.

3.3 Discriminator and Classifier Training

To investigate if our methods of generating outliers can be used in practice we train a discriminator to distinguish inlier samples from outliers and test it on other outlier data. The availability of other outlier data that resembles the inlier data is one of the reasons we chose EMNIST as one of our datasets, as discussed in section 3.5.

The architecture of the discriminator neural network is modeled after the DCGAN architecture [28] with additional inputs to make the discriminator class conditional [25]. In detail, the discriminator consists of two small networks of one strided convolution each with leaky rectified linear unit activations that process the image input and the condition respectively. The outputs of these two networks are concatenated and fed into two convolutional layers with leaky rectified linear unit activations and batch normalization [13]. Finally, a last convolutional layer with the sigmoid function as activation is applied. For samples from the data distribution $\mathbf{x} \sim p_{\text{in}}$ with class labels y and some outlier distribution (e.g. from using the change of variables formula for an outlier distribution of a normalizing flow) $\mathbf{z} \sim p_{\text{out}}$ the discriminator D is trained to maximize

$$\mathbb{E}_{\hat{\mathbf{x}},\hat{y}\sim p_{\text{in}}} \log D(\mathbf{x}, y) + \mathbb{E}_{\mathbf{z}\sim p_{\text{out}}} \log(1 - D(\mathbf{z}, y)) \,, \tag{3.16}$$

where the class labels for the outliers are sampled from a uniform distribution over all class labels.

Another application is to use the generated outliers to train a classifier to have high confidence on inliers and show low confidence otherwise. Lee et al. propose training a generative adversarial network jointly with such a classifier and use the generator network to create outlier samples [23]. Since we do not need to train a generator jointly with the classifier we simply use our methods of outlier generation and train the classifier similarly to the classifier update step of Lee et al. The classifier shows low confidence when the distribution over class labels is close to a uniform distribution. We can compute the Kullback-Leibler divergence to get a measure for how close these two distributions are. For samples from the data distribution $\mathbf{x} \sim p_{\text{in}}$ with class labels y and some outlier distribution $\mathbf{z} \sim p_{\text{out}}$ the classifier is trained to maximize

$$\mathbb{E}_{\hat{\mathbf{x}},\hat{y}\sim p_{\text{in}}} \log p(y = \hat{y}|\hat{\mathbf{x}}) - \beta \mathbb{E}_{\mathbf{z}\sim p_{\text{out}}} \mathrm{KL}(\mathcal{U}(y)|p(y|\mathbf{z})) \,, \tag{3.17}$$

where β is a hyperparameter that weights the two loss terms. Except for the toy example we use a VGG11 model as the classification network [30].

Both, the discriminator and the classifier network are trained using the ADAM optimizer [16].

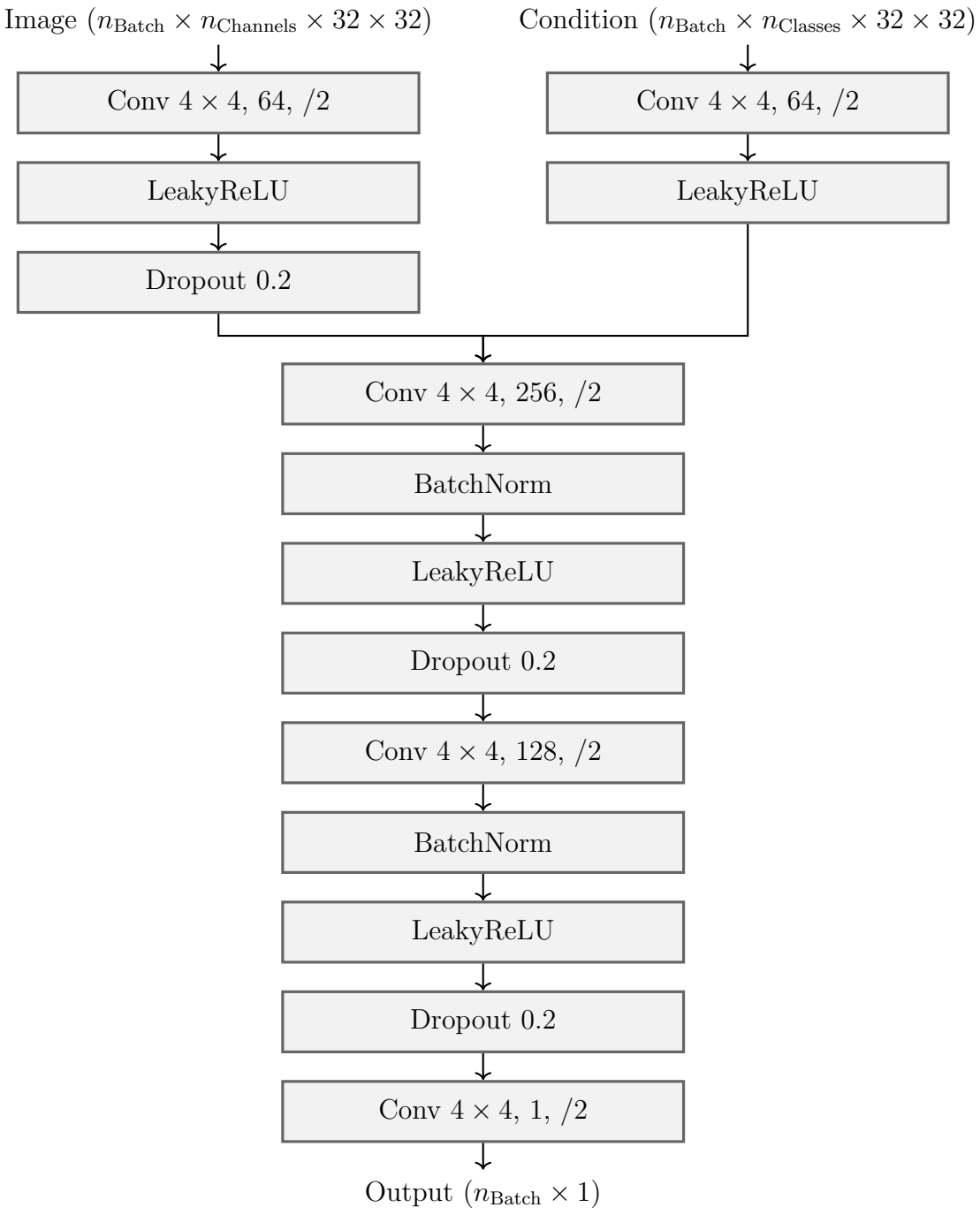

Figure 3.3: Architecture of the discriminator network.

3.4 Network Architecture

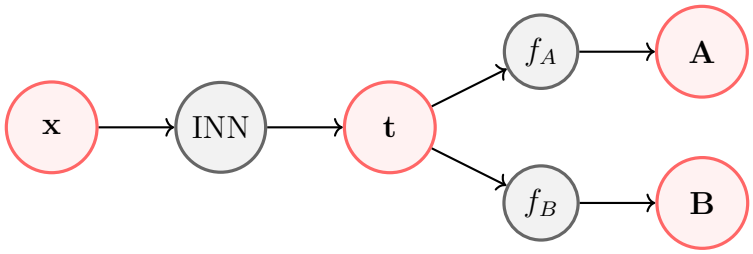

Figure 3.4: Forward pass of the normalizing flow with additional layers for the mapping to archetype coefficients.

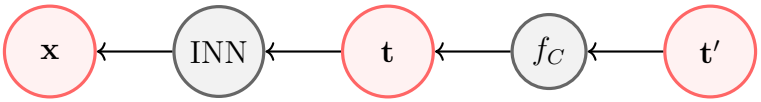

Figure 3.5: Inverse computation from lower dimensional sample created from archetype coefficients to data space.

The architecture of the invertible neural network consists of a series of affine coupling blocks as discussed in subsection 2.2.1. The coupling functions are a series of three layers of either convolutions or linear mappings with rectified linear activation functions. Between sets of coupling layers downsampling is applied. The actual configuration of these components is specific to the experiments and mentioned in the respective subsections.

For the coupling functions s and t we decided to follow the GLOW design [17] and use a single network to jointly compute the scaling and translation. Note, that in the low-resolution convolutional coupling blocks we use 1×1 convolutions in every other block to improve image quality.

As described in subsection 2.2.3 the network is trained to minimize the negative log-likelihood of the training data.

To map from the latent space to the archetype coefficient a_{ij} and b_{ji} one linear layer with softmax activation each is used, denoted as f_A and f_B respectively. Since this is in general not invertible, we additionally train a linear layer without activation function f_C to map back from this low dimensional space to the latent space of the normalizing flow. For this, the whole setup is trained to minimize the mean squared reconstruction error in addition to the optimization problem shown in Equation 2.14.

All optimizations are done using the ADAM optimizer [16].

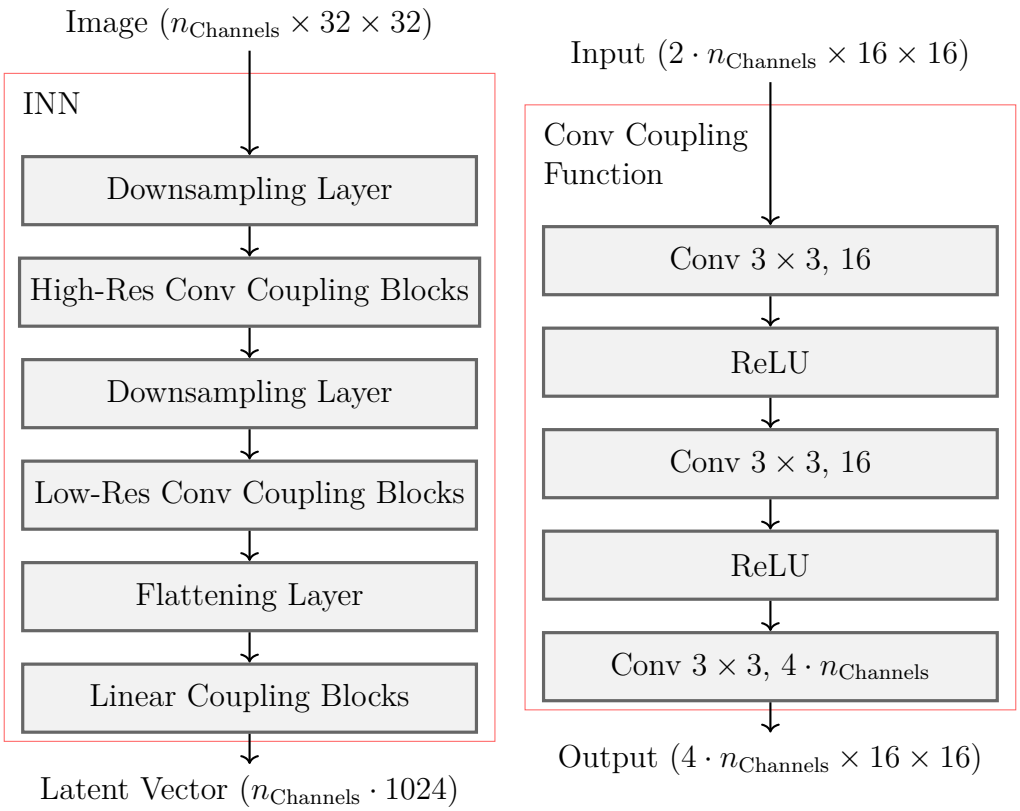

Figure 3.6: **Left**: Architecture of the invertible neural network. The number of coupling blocks in each section is specific to the experiments and mentioned there. **Right**: Coupling function of the high-resolution coupling blocks. Low-resolution coupling blocks are similar, but have one more downsampling in front of them.

3.5 Datasets

In our experiments we use multiple different datasets. The first dataset is the EMNIST dataset [4], an extension to the MNIST dataset of handwritten digits [21]. It consists of greyscale images of handwritten digits and letters that are 28×28 pixels in size. For easier use in the experiments and consistency across all datasets the images are zero-padded to a size of 32×32 pixels. In the rest of the work, we will refer to the set of images of digits as the digits dataset and to the set of letters as the letters dataset. As an additional source of outliers, we use the Fashion-MNIST [31] and Kuzushiji-MNIST [3] datasets, referred to as the fashion dataset and the KMNIST dataset in the rest of this work. The fashion dataset consists of greyscale images of clothing that are 28×28 pixels in size and again zero-padded to 32×32 pixels. The KMNIST dataset contains greyscale images of cursive Japanese characters, again they are 28×28 pixels in size and zero-padded. To investigate our methods on slightly more complex images, we use the Facial Expression Research Group-Database [1], a set of images of six animated, stylized characters with seven different facial expressions. The 256×256 pixel images are resized to 32×32 pixels. In the rest of the work, we refer to this dataset, using the characters as classes, as the people dataset. Since there is no dataset that provides equally closely related outliers for the people dataset as for the digits dataset, we use the CIFAR10 dataset [19] of color images as a substitute. For all datasets, the number of samples in the training and test split as well as the number of classes is listed in Table 3.1.

Table 3.1: Number of classes and samples in each dataset.

Dataset	Classes	Training	Test
digits	10	240000	40000
letters	26	124800	20800
fashion	10	60000	10000
KMNIST	10	60000	10000
people	6	55766	
CIFAR10	10	50000	10000

Chapter 4

Results

4.1 Toy Example

Let us first investigate our methods on some simple toy data. For this we generate two toy datasets. One dataset consists of three two-dimensional Gaussian distributions, each being its own class. The other dataset has two classes shaped like interlocked moons. Both datasets are normalized. On each dataset we train a simple version of the normalizing flow introduced above. It consists of three additive coupling blocks, where the coupling functions are neural networks with five linear layers with rectified linear units as activation functions. As described in section 3.4, it is trained to maximize the likelihood of the training data. Here, we use additive coupling blocks, as all our classes have the same volume so preserving the volume works well.

Samples from the Gaussian mixture toy dataset and the learned latent space is shown in Figure 4.1. As we can see the latent space approximates a standard normal distribution and each class gets mapped to a region of this distribution.

For the moons toy dataset, samples and the learned latent space are shown in Figure 4.2. Again, we observe a reasonable approximation of a standard normal distribution in the latent space.

We can now take a look at the generation of outliers. Since the latent space is low-dimensional, we use the Gumbel distribution sampling as described in subsection 3.1.2. The sampled ring in the latent space can be seen in the bottom right of Figure 4.1 for the Gaussian mixture dataset and in the bottom right of Figure 4.2 for the moons dataset. Using the normalizing flow to transform the outlier samples back to the data space, we get samples that encase the inlier data. For the Gaussian mixture dataset this can be seen in the bottom left of Figure 4.1 and in the bottom left of Figure 4.2 for the moons dataset.

To show how the generated outliers can be of use, we can train a classifier on

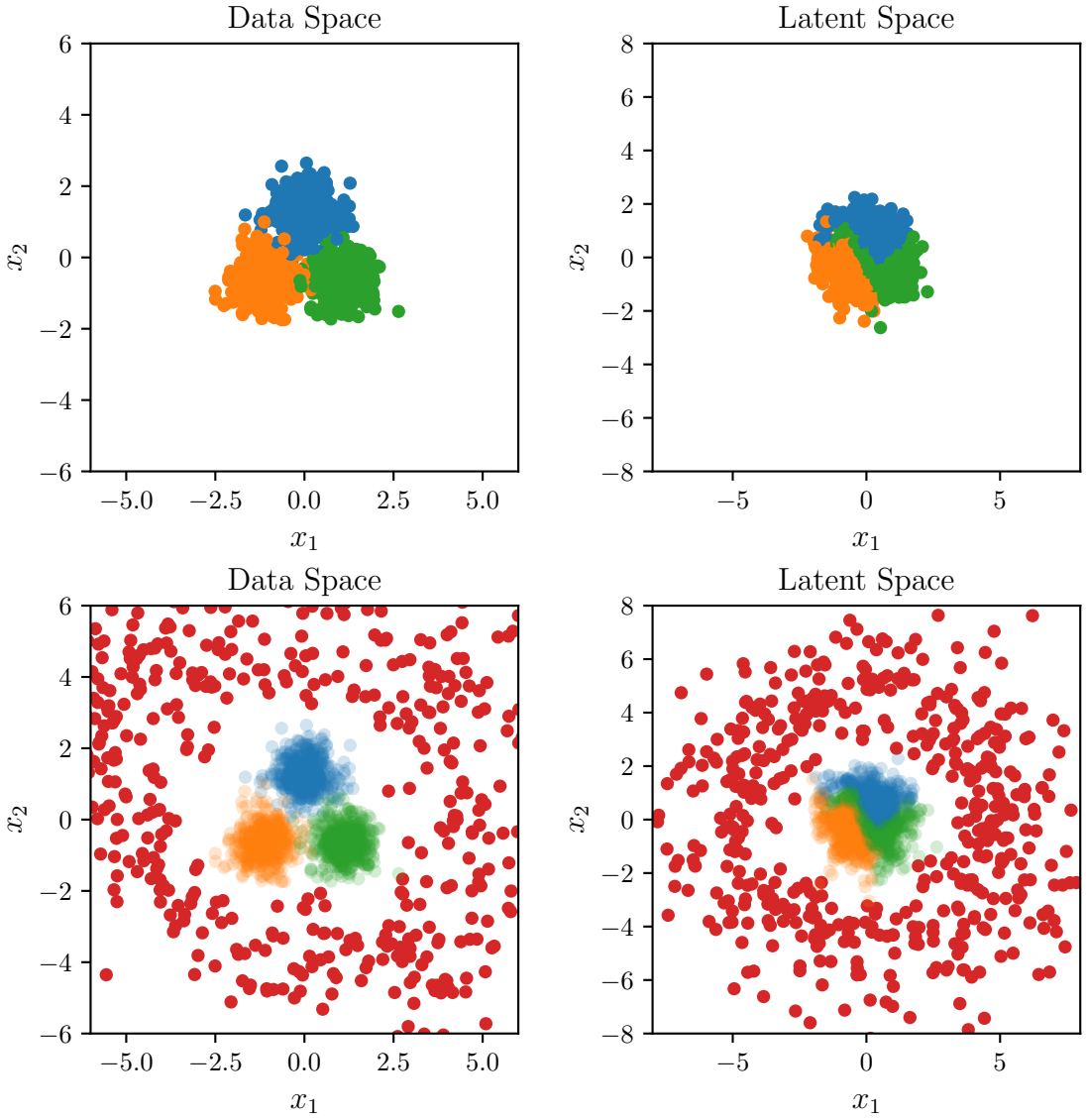

Figure 4.1: **Top**: Data space and latent space of the Gaussian mixture toy dataset. **Bottom**: The red points are sampled from a ring around the latent distribution in latent space and mapped back into data space with the normalizing flow.

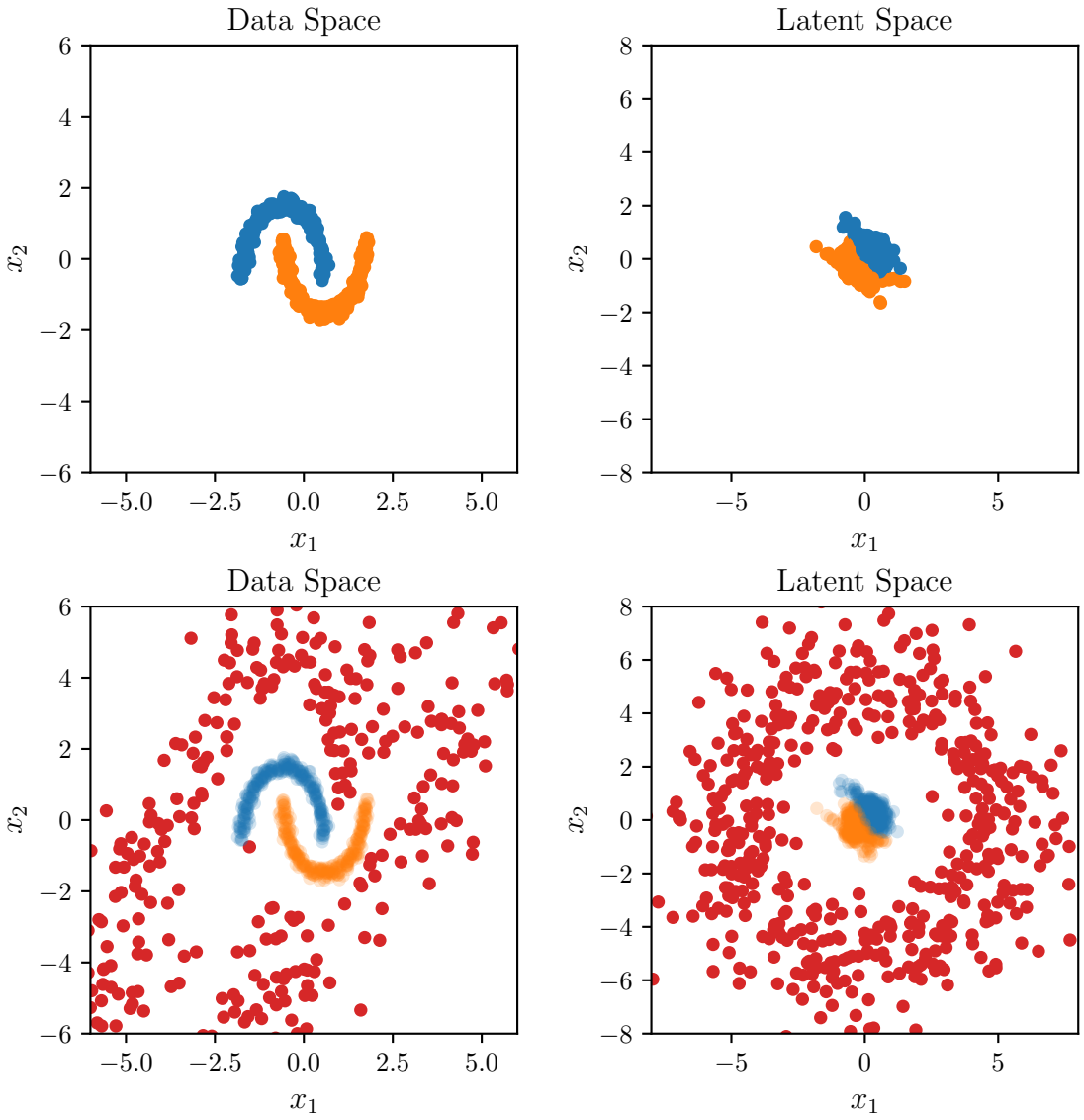

Figure 4.2: **Top**: Data space and latent space of the Moons toy dataset. **Bottom**: The red points are sampled from a ring around the latent distribution in latent space and mapped back into data space with the normalizing flow.

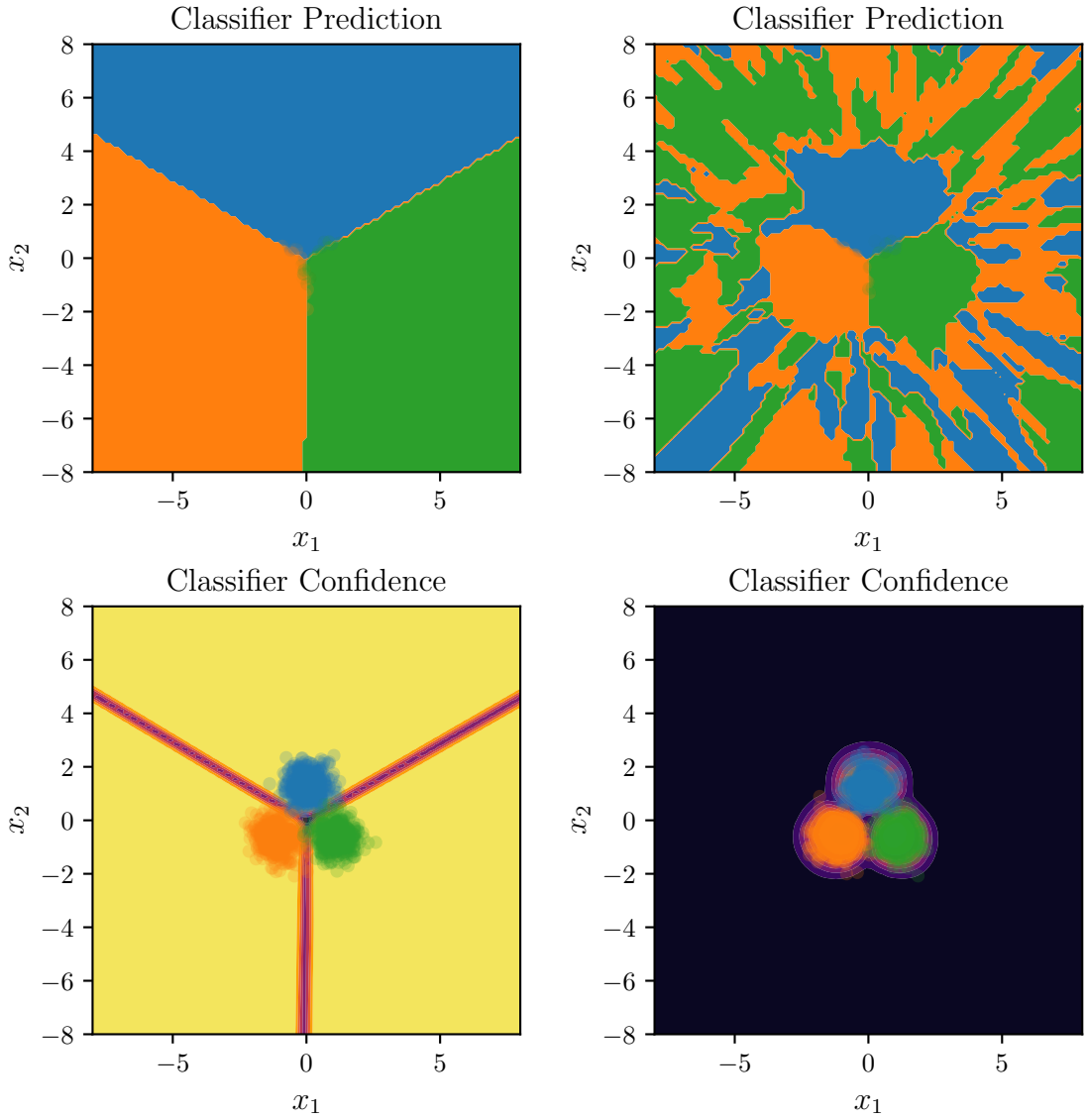

Figure 4.3: **Left**: Decision regions and confidence of a classifier trained on the Gaussian mixture toy dataset. **Right**: Decision regions and confidence of a classifier trained additionally with the Kullback-Leibler divergence loss on outliers. In the confidence plots higher brightness means higher confidence.

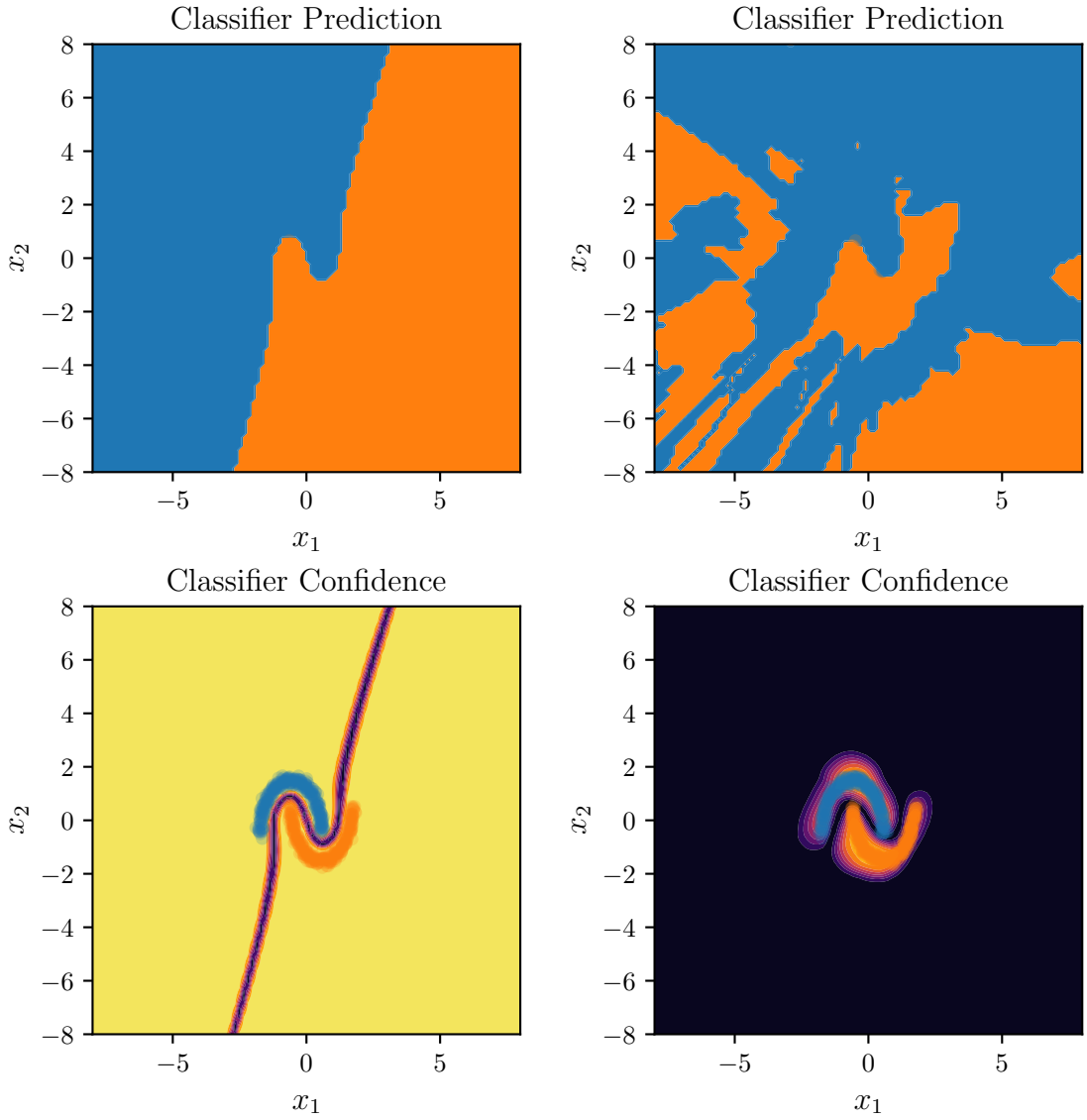

Figure 4.4: **Left**: Decision regions and confidence of a classifier trained on the Moons toy dataset. **Right**: Decision regions and confidence of a classifier trained additionally with the Kullback-Leibler divergence loss on outliers. In the confidence plots higher brightness means higher confidence.

the toy datasets. The classifier is a simple neural network consisting of three linear layers with rectified linear units as activations. It is trained on the generated data to minimize the cross-entropy between the ground truth labels and the predicted labels. This simple setup leads to a classifier that performs well on both datasets. The decision regions can be seen in the top right of Figure 4.3 and Figure 4.4 respectively. To get a measure of the confidence of the classifier, we can inspect the entropy of the predicted class distribution. We can see in the bottom left of Figure 4.3 and Figure 4.4, that this classifier has high confidence everywhere except in a small boundary region between the classes. This is often undesired as this means outliers remain undetected and receive an oftentimes wrong class prediction.

This can be mitigated by augmenting the loss of the classifier as described in section 3.3. We train the classifier to minimize the Kullback-Leibler divergence between the predicted class distribution and a uniform class distribution over the generated outliers. As we can see in the bottom right of Figure 4.3 and Figure 4.4 this leads to regions of low confidence all around the data regions while keeping good decision boundaries in the region of high confidence, shown in the top left of Figure 4.3 and Figure 4.4.

4.2 Qualitative Comparison

After testing our methods on simple data, we can now move on to more complex data. We now consider the digits dataset as described in section 3.5. Again we first train the normalizing flow. In this case the flow consists of four high-resolution convolutional coupling blocks, four low-resolution convolutional coupling blocks and two linear coupling blocks. The flow is trained for 2000 epochs with a learning rate of 10^{-5} that gets reduced by a factor of ten every 600 epochs.

Now we can inspect samples generated from the normalizing flow. Samples are generated by sampling from a multivariate normal distribution where the variance has been multiplied by some coefficient as described in section 3.1. The samples with their variance coefficient are shown in Figure 4.5. Evidently, the images become increasingly distorted without accumulating much noise, exactly as expected. Note that the difference in the kind of distortion is that no particular weight has been placed on structuring the latent space in a meaningful way and the direction in the latent space is completely random, only the variance and thereby the radius of the hyperspherical shell has been fixed. We can also see that images close to the origin are smoother, enabling the generation of images that could be viewed as prototypical of each class. Following the discussion in subsection 3.1.1, we can easily see how those images are smoother than the majority of the training data and therefore do not make up the majority of the mass of the distribution.

0.5σ 0.6σ 0.8σ 0.9σ 1.1σ 1.2σ 1.4σ 1.5σ 1.6σ 1.8σ 1.9σ 2.1σ 2.2σ 2.4σ 2.5σ

Figure 4.5: Samples drawn from a multivariate normal distribution and mapped back to the data space of the digits dataset. The standard deviation in terms of the standard deviation of the data in latent space is shown below the images. Each row represents a class, each column one sampling variance.

1.0σ 1.6σ 2.3σ 2.9σ 3.6σ 4.2σ 4.9σ 5.5σ 6.1σ 6.8σ 7.4σ 8.1σ 8.7σ 9.4σ 10.0σ

Figure 4.6: Samples drawn from a multivariate normal distribution and mapped back to the data space of the people dataset. The standard deviation in terms of the standard deviation of the data in latent space is shown below the images. Each row represents a class, each column one sampling variance.

To consider more complex data, we can repeat this investigation on the people dataset introduced in section 3.5. As described above, the dataset can be ordered by character or by emotion, we choose the character as the class here. The normalizing flow trained on the people dataset consists of four high-resolution convolutional coupling blocks followed by twelve low-resolution convolutional coupling blocks and twelve linear coupling blocks. It is trained for 1500 epochs with a learning rate of 10^{-4} reduced by a factor of ten every 500 epochs. Samples are again generated as in the case of the digits dataset. We can inspect the samples with their variance coefficients in Figure 4.6. Apparently one has to sample much further from the origin in terms of the variance than in the digits case, suggesting the data is more spread and is approximated worse by a normal distribution in latent space. We can again see different extreme images that are unordered, there is again no attention paid to creating meaningful latent space axes and the sampling direction is random. An interesting effect to note is, that due to the conditional training different classes can be noticed to appear as outliers of other classes. We will investigate the position of classes in the latent space in the next section.

4.3 Archetypal Analysis

Let us now take a look at sample generation when we additionally add the archetype mapping to the normalizing flow as described in section 3.2. We first have to consider the choice of the number of archetypes, for that we follow Keller et al. [15] and train multiple models with different numbers of archetypes. All models are trained with the pretrained flow from the previous section. The flow parameters are fixed while the positions of the archetypes in latent space and the parameters of the linear mapping layers are trained. We then compute the mean squared error for the reconstruction of all samples in the training set. The mean squared error as a function of the number of archetypes for the digits dataset can be seen in Figure 4.7. A reasonable choice is 14 archetypes, achieving a good reconstruction error while still keeping the number of archetypes low. For the people dataset the mean squared error as a function of the number of archetypes can be seen in Figure 4.8, the ideal choice are seven archetypes. This is notable, because it lines up with the seven expressions present in the dataset. Indeed, when showing the sample from each class in the training data that is closest to the archetypes in Figure 4.10, the archetypes can clearly be identified with the expressions.

Note that with an increasing number of archetypes, the clear interpretability of the individual archetypes is reduced and clearly distinguishable features become redundant with only subtle differences. This can be seen when we investigate the images from the training dataset and each class closest to the archetypes as in

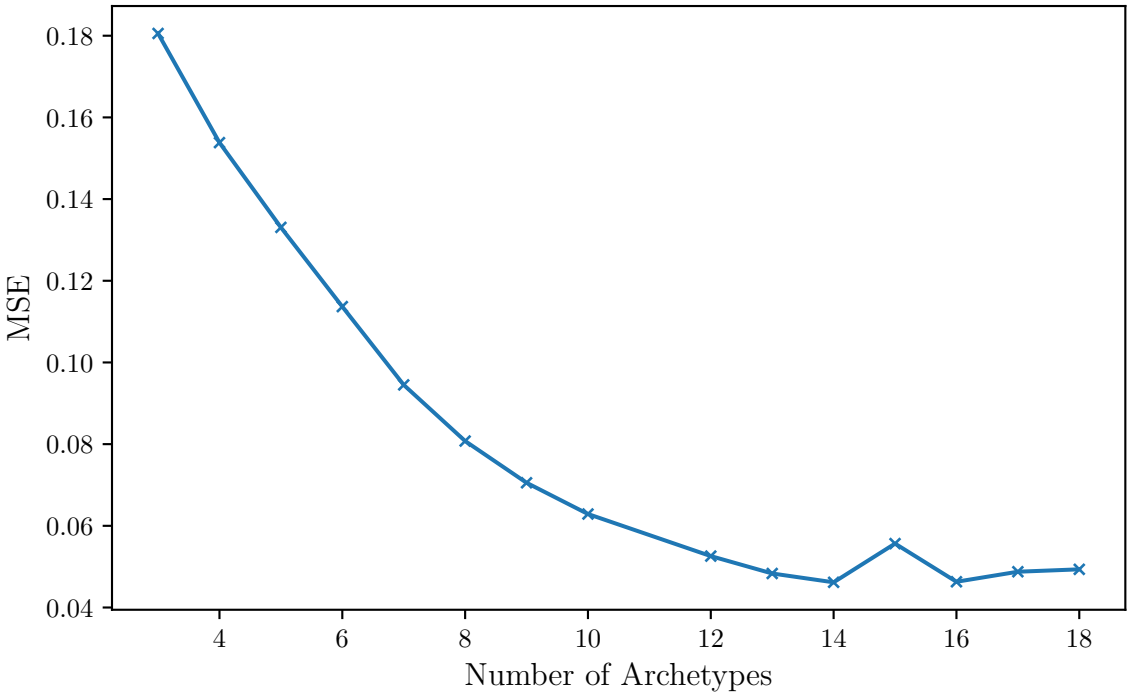

Figure 4.7: Mean squared error when reconstructing the digits training dataset as a function of the number of archetypes

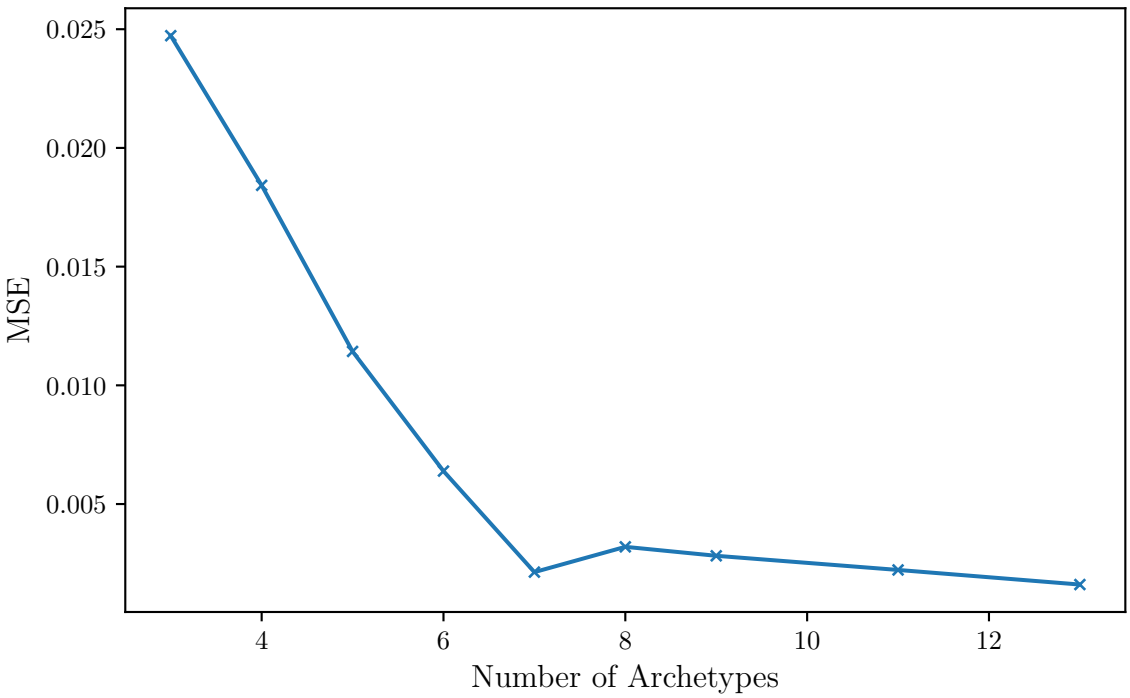

Figure 4.8: Mean squared error when reconstructing the people training dataset as a function of the number of archetypes

Figure 4.9.

Figure 4.9: The images from the digits dataset closest to each of the archetypes of the model with three and the model with 14 archetypes respectively.

Figure 4.11 depicts samples generated by sampling from a Dirichlet distribution and mapping those samples back into the data space of the digits dataset. To identify extreme samples, or outliers, we can change the shape parameters of the Dirichlet distribution to draw samples close to one of the archetypes. Figure 4.12 shows such samples as well as how the images are distorted in specific ways related to the archetype. This could allow, for example, to investigate what extreme points in the data may look like and obtain an intuition for outliers that could act as adversarial data points. The same experiment can be done with the model trained on the people dataset.

Turning to Figure 4.13 and Figure 4.14, we are again able to generate in-distribution samples and also extreme samples, where the distorted features are again controlled by the archetypes via the shape parameters of the Dirichlet distribution. Notably, extreme samples generated are also more distorted than the samples closest to the archetypes from the dataset shown in Figure 4.9 and Figure 4.10.

4.3.1 Nullspace Sampling

For the digits dataset, the number of archetypes needed to achieve a low reconstruction error is surprisingly high. Another way to improve reconstructions, and thereby the quality of generated samples, is augmenting the samples with samples from the nullspace of the mapping to archetype space. This works especially well when using no bias in the mapping. A model with three archetypes and no bias thereby achieves a mean squared reconstruction error of 0.022 on the digits dataset, lower

Figure 4.10: The images from the people dataset closest to each of the archetypes of the model.

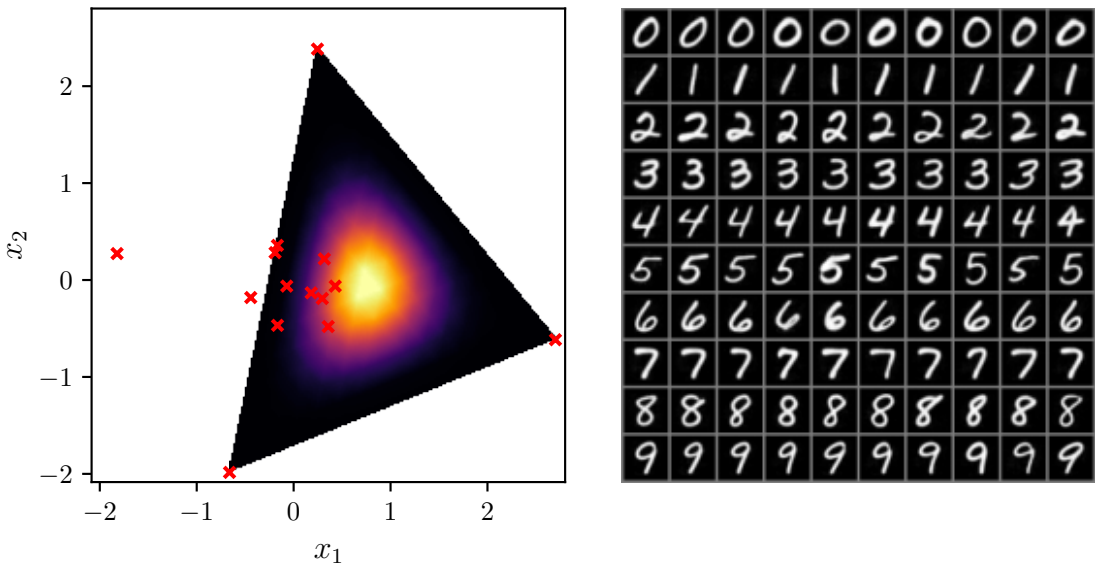

Figure 4.11: Samples drawn from a dirichlet distribution and mapped back to the data space of the digits dataset. On the **left** hand side the probability density function of the latent sampling distribution is shown, on the **right** the resulting samples are shown. Each row corresponds to one class.

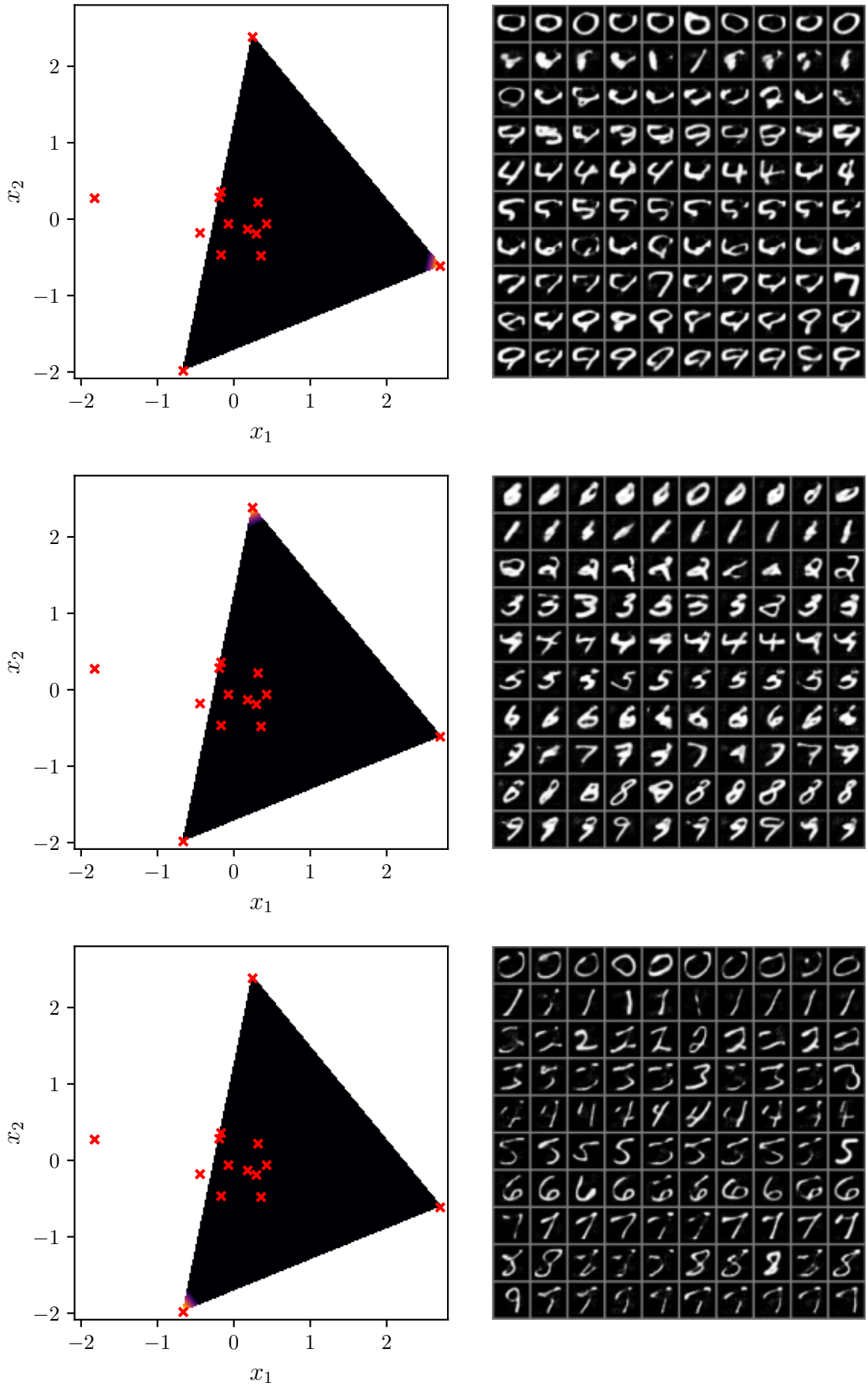

Figure 4.12: Samples drawn from a dirichlet distribution and mapped back to the data space of the digits dataset. **Top**, **mid** and **bottom** show sampling close to one archetype each.

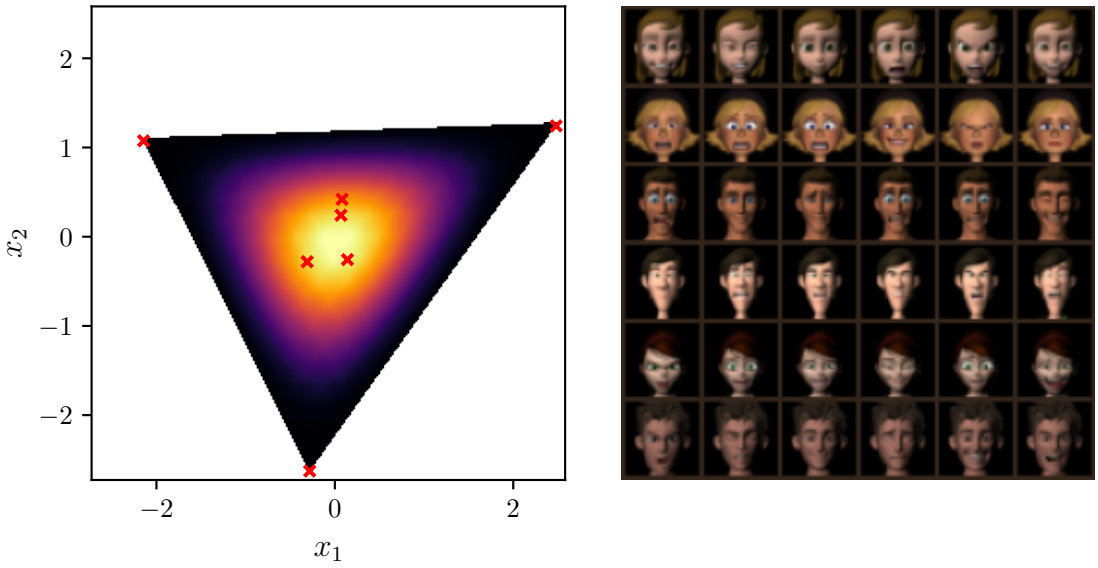

Figure 4.13: Samples drawn from a dirichlet distribution and mapped back to the data space of the people dataset. On the **left** hand side a projection of the probability density function of the latent sampling distribution is shown, on the **right** the resulting samples are shown. Each row corresponds to one class.

than any of the models achieved without augmentation (see Figure 4.7). A model with bias in the mapping can still achieve a mean squared error on the digits dataset of 0.068, which is an improvement equivalent to tripling the number of archetypes. In Figure 4.15 we can see how this affects samples from the center of the simplex and each of the corners.

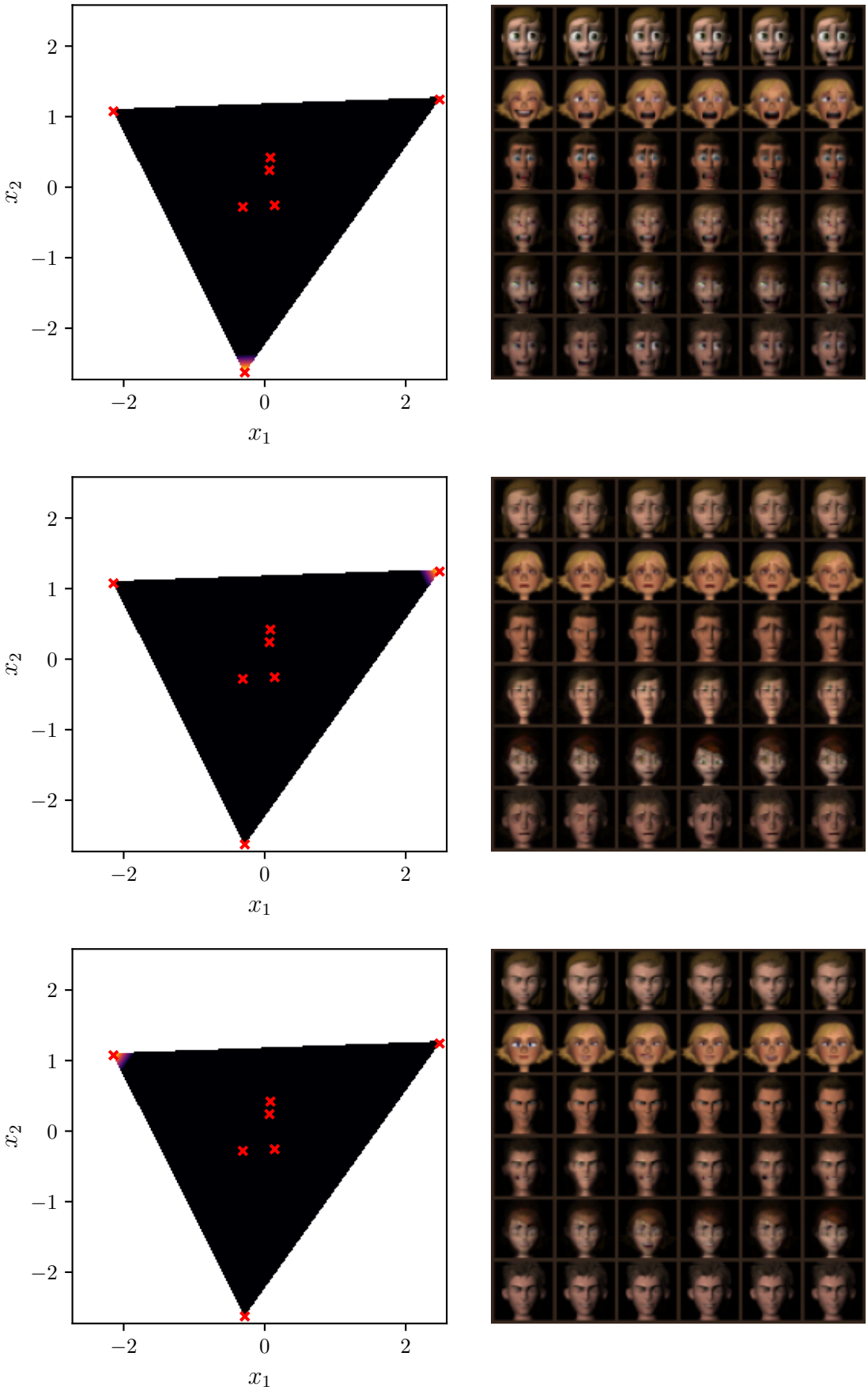

Figure 4.14: Samples drawn from a dirichlet distribution and mapped back to the data space of the people dataset. **Top**, **mid** and **bottom** show sampling close to one archetype each.

Sampling with Nullspace Reconstruction MSE = 0.022

DeepAA Sampling

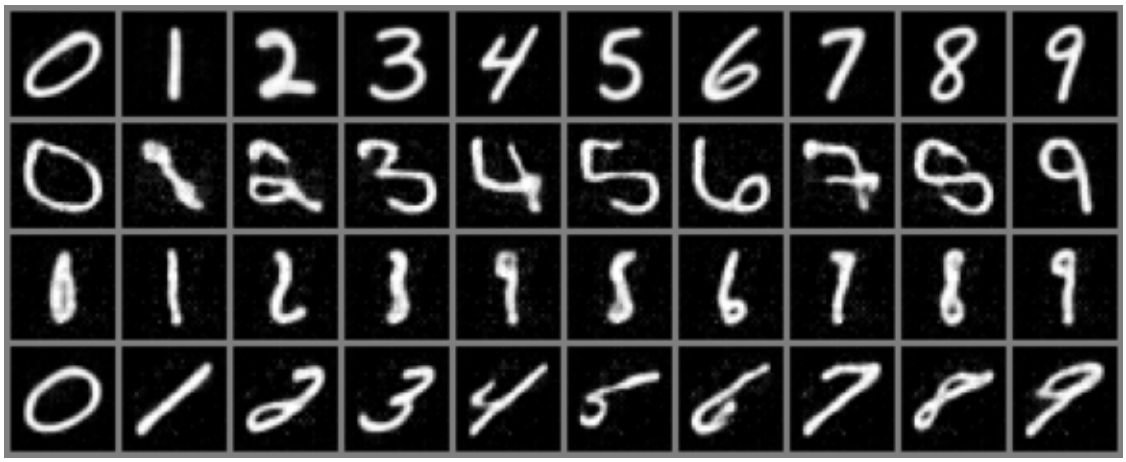

Figure 4.15: Samples from the center of the Dirichlet distribution and each of the corners. Each column is one class. **Top**: Model without bias in the mapping to archetype space. **Bottom**: Model with bias in the mapping to archetype space.

Sampling with Nullspace Reconstruction MSE = 0.068

DeepAA Sampling

Figure 4.16: Samples from the center of the Dirichlet distribution and each of the corners. Each column is one class. **Top**: Model without bias in the mapping to archetype space. **Bottom**: Model with bias in the mapping to archetype space.

4.4 Analysis of the Latent Space

After investigating the models qualitatively, we will now look more closely into the latent space and how different samples and classes relate to each other. Our investigation begins with the digits dataset.

To check our intuition, that classes that are similar to each other visually should be placed close to each other in latent space we can check cluster distances between classes. For that we compute the latent vectors of the data in question conditioned on each of the classes in the digits dataset. To ease computation and visualization we perform a linear discriminant analysis on each of the class pairs and project the data on the first two components of the linear discriminant analysis. Next, we compute the mean euclidean distance of 1000 sample pairs from the two classes. The result is shown as a colored matrix in Figure 4.17 for a comparison of the digits dataset to itself and in Figure 4.18 for a comparison to the letters dataset. As expected, classes that are visually similar are close together: Consider, for example, how the class `6` is close to class `0` when the network is conditioned on class `0`, but also more unexpected relations are revealed, such as class `9` being close to class `4` when conditioned on `4` and class `2` being close to class `8` when conditioned on `8`. Also notable is the asymmetry of the matrix. This is due to the fact that the class conditioning makes the flow maximize the likelihood of samples of the class matching the condition, while the combination of a class condition with a foreign class sample is unseen during training and therefore does not influence the latent space layout.

Our intuition on the placement of visually similar classes is also confirmed when looking at samples from a dataset of outliers, in this case the letters dataset. Notably, the class `O` from the letters dataset is close to the class `0` when conditioned on `0`, as one would expect. Similarly, `i` and `1`, and `p` and `8` or `9` are close.

For the people dataset relations between classes are less intuitive than for the digits dataset. Still, we can look at the distances of samples from test classes to a conditioning class in Figure 4.19. Most notable here is, that although all images are somewhat similar since they are computer generated, the classes `jules` and `ray` are clustered close to all other classes. In contrast to the digits dataset we do not have a dataset of close outliers for the people dataset. To still make some measurements with models trained on the people dataset we use CIFAR10 as a source of outliers. The matrix of pairwise distances can be seen in Figure 4.20. It is again interesting to see, that different classes are closer or further away not to specific other classes but to the dataset as a whole.

Since our method of sampling outliers relies on sampling on a shell around the inlier distribution as a whole and not the position of specific types of outliers, we will now look at the distribution of the lengths of latent vectors. For the digits

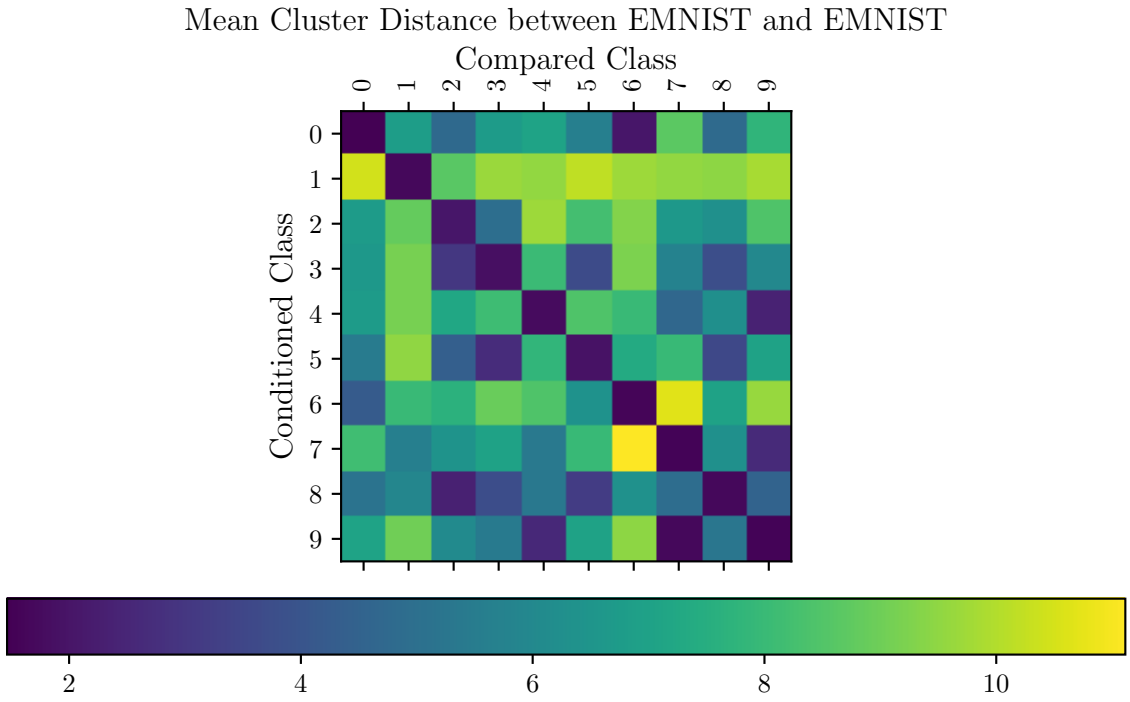

Figure 4.17: Mean distance in LDA projection space between samples from the conditioning class (y axis) and a test class (x axis). Classes and samples are from the digits dataset. Darker means closer together.

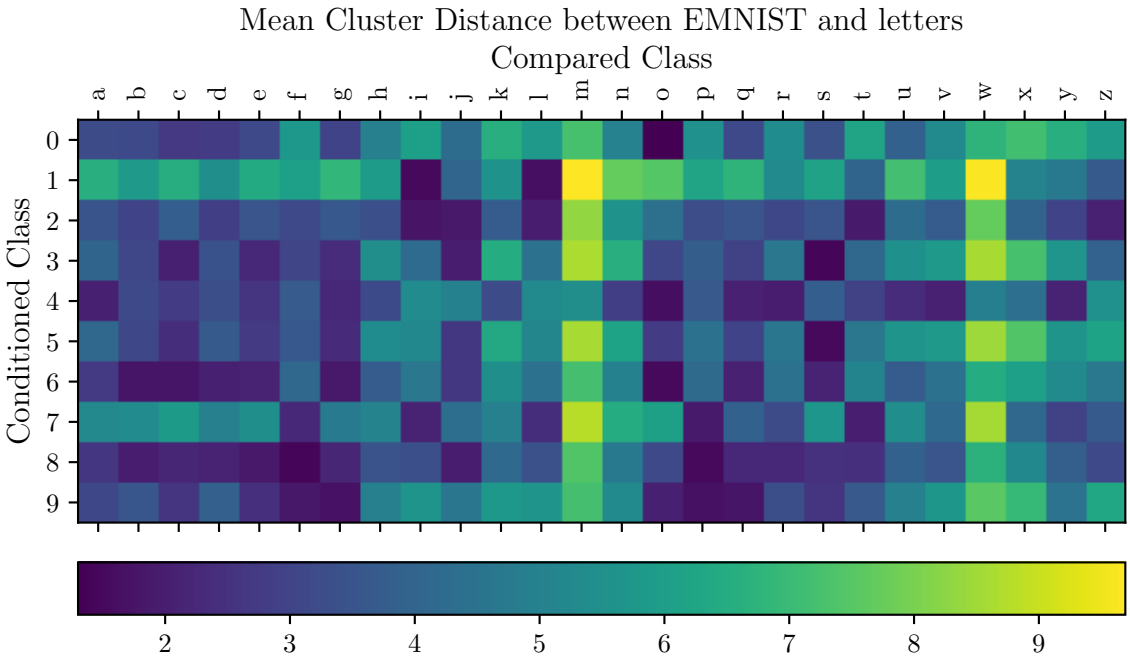

Figure 4.18: Mean distance in LDA projection space between samples from the conditioning class (y axis) and a test class (x axis). Samples from the conditioning class are from the digits dataset, test samples are from the letters dataset. Darker means closer together.

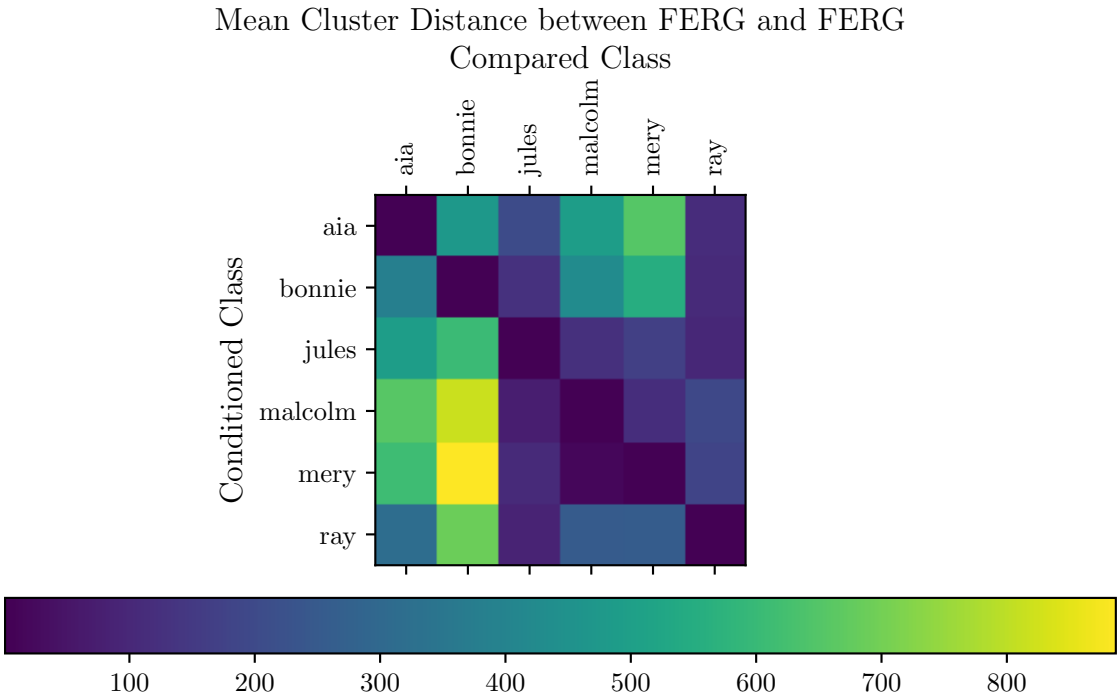

Figure 4.19: Mean distance in LDA projection space between samples from the conditioning class (y axis) and a test class (x axis). Samples are from the people dataset. Darker means closer together.

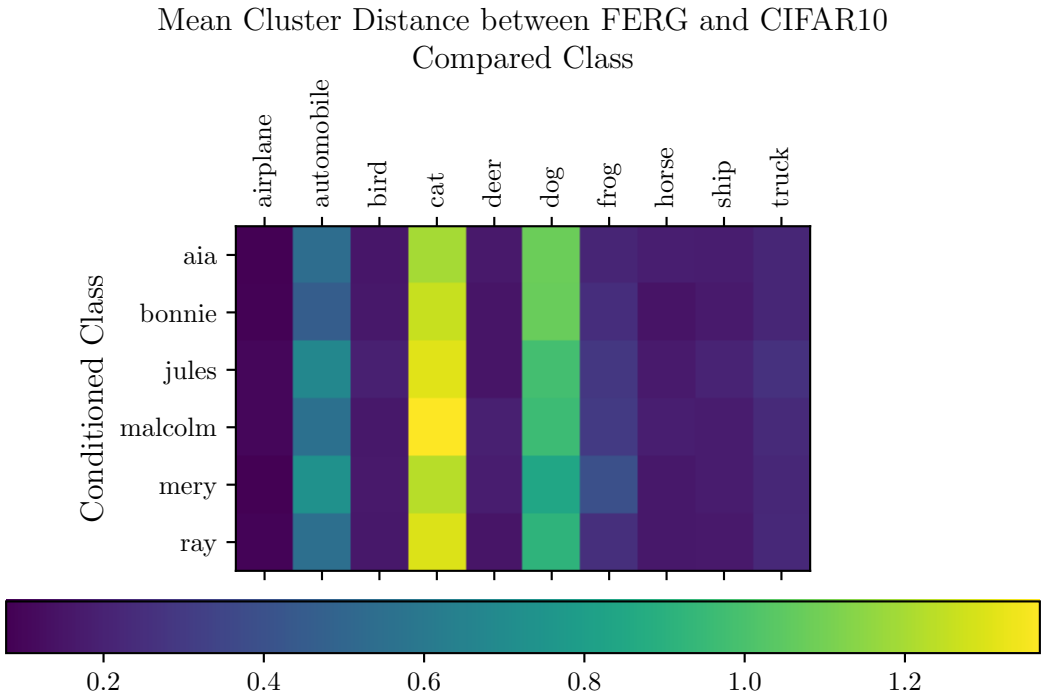

Figure 4.20: Mean distance in LDA projection space between samples from the conditioning class (y axis) and a test class (x axis). Samples from the conditioning class are from the people dataset, test samples are from the CIFAR10 dataset. Darker means closer together.

dataset, the distribution of latent vector lengths of inlier data and from the outlier datasets letters, FashionMNIST and KMNIST are shown in Figure 4.21. We can see that the letters dataset is close to the digits dataset and even has some overlap, while the other two datasets are further away. To show where the overlap might come from we again look at the distribution of latent vector lengths in Figure 4.22 but constrain the data to the classes `o` and `x`. Following our intuition the letters class `o` has a similar latent vector length distribution as the digits class `0` while `x` is more clearly an outlier class by our measure.

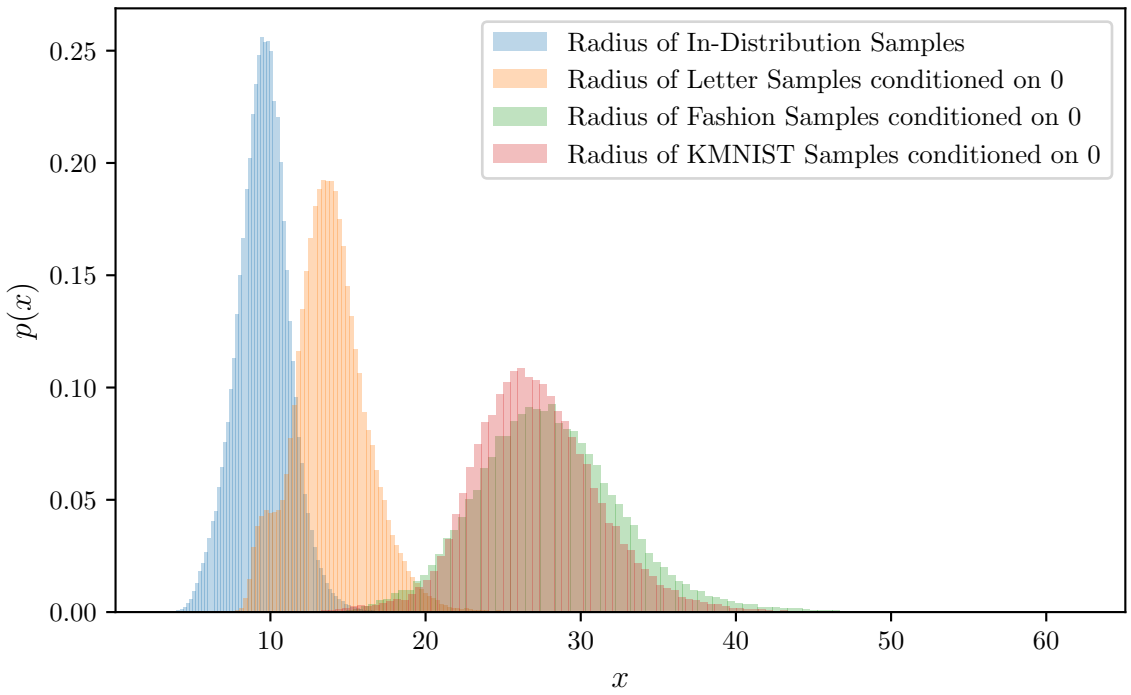

Figure 4.21: Distribution of the length of latent space vectors for the digits dataset and its outlier datasets.

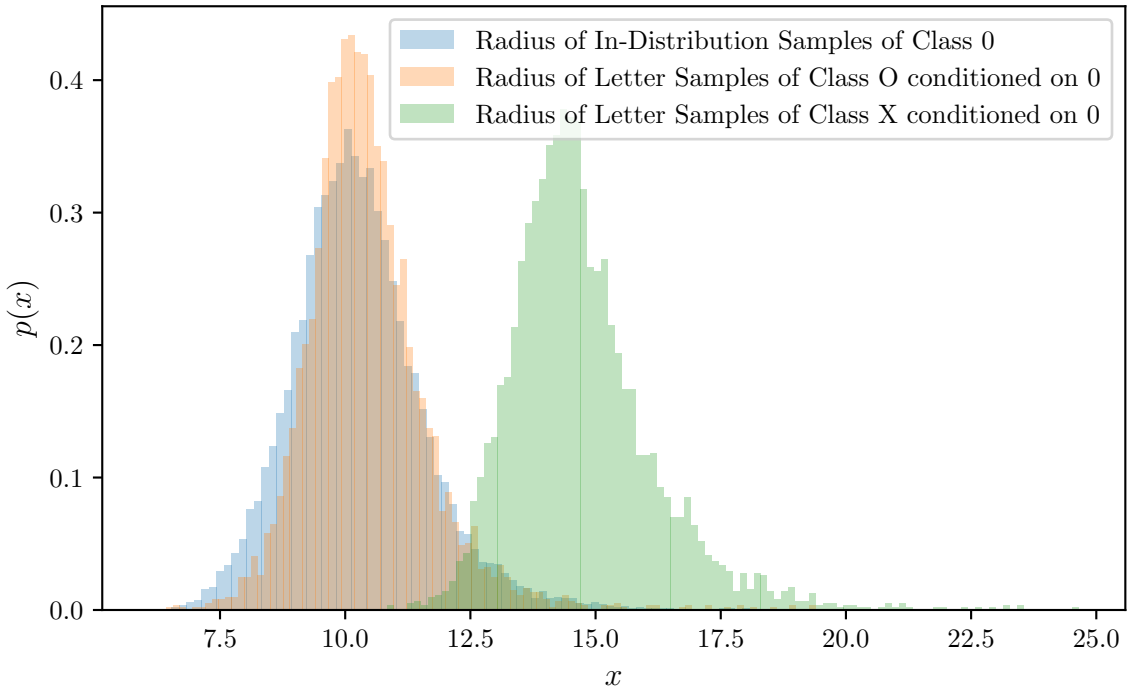

Figure 4.22: Distribution of the length of latent space vectors for `0` compared with `o` and `x`.

4.5 Discriminator and Classifier Performance

To show a potential application of the outlier generation we train discriminator and classification networks on using the models we investigated above. The architecture and training is described in section 3.3. As the source of outliers we use different settings for our normalizing flow network, sampling with various standard deviations, sampling from a Gumbel distribution and sampling from a Dirichlet distribution with various shape parameters. We compare this to using the letters and fashion datasets directly as outliers when taking the digits dataset as inliers. Similarly, we compare our networks to using CIFAR10 as a source of outliers when using the people dataset as inliers. To compare the results we plot the precision-recall (PR) and receiver operating characteristic (ROC) curves for each of the models and compute the average precision and area under the ROC curve.

Taking the digits dataset as the source of inliers we train the discriminator and classification models for ten epochs. The curves as described above can be seen in Figure 4.23 for the discriminators and in Figure 4.24 for the classifiers. For the normalizing flow (INN), p denotes the coefficient with which the standard deviation in latent space is multiplied, or a multiplicative factor controlling the mean and standard deviation of the Gumbel distribution. The models with archetypal analysis (DeepAA) receive samples from a Dirichlet distribution where all shape parameters are equal $c_1 = \cdots = c_k = p$. Especially interesting is seeing the performance

when the letters dataset is used, since this dataset is the most similar to the digits dataset and thereby the most challenging to distinguish. Both the normalizing flow and the archetype model achieve good performance on this task that is close to the discriminator trained directly on the letters dataset. It is also interesting to see, that performance greatly depends on the parameters that control the outlier distribution in latent space, where the performance suffers when the distribution is too close to the inliers as well as when it is too far away. With the classifier we can also note, that classification performance is not impacted by the addition of the outlier loss. We observe similar performance when using the people dataset as inliers, although this measurement is not as meaningful as the previous one, since there is no good source of outliers to compare against. This is most notable when looking at the classifier performance in Table 4.4, where all classifiers perform extremely well against the CIFAR10 dataset, except the classifier with no outlier loss at all.

Table 4.1: Area under the ROC curve (AUC) and average precision (AP) of discriminators trained on different outlier sources tested on different outlier datasets.

outliers	letters		fashion		kmnist	
metric	AUC	AP	AUC	AP	AUC	AP
DeepAA, p = 0.1	0.64	0.33	0.88	0.77	0.67	0.47
DeepAA, p = 0.3	0.57	0.27	0.52	0.37	0.46	0.34
DeepAA, p = 0.5	0.48	0.22	0.38	0.31	0.42	0.33
DeepAA, p = 0.6	0.47	0.22	0.39	0.32	0.37	0.31
DeepAA, p = 0.8	0.43	0.20	0.48	0.35	0.39	0.32
DeepAA, p = 1.2	0.45	0.21	0.74	0.51	0.63	0.42
INN, Gumbel, p = 2.5	0.68	0.35	0.45	0.34	0.47	0.35
INN, Normal, p = 1.5	0.58	0.27	0.12	0.25	0.38	0.32
INN, Normal, p = 2.0	0.65	0.32	0.20	0.27	0.48	0.35
INN, Normal, p = 2.5	0.69	0.39	0.80	0.61	0.72	0.50
INN, Normal, p = 3.0	0.52	0.24	0.42	0.33	0.15	0.25
fashion	0.73	0.44	**1.00**	**1.00**	**0.97**	**0.95**
letters	**0.81**	**0.56**	0.50	0.36	0.48	0.35

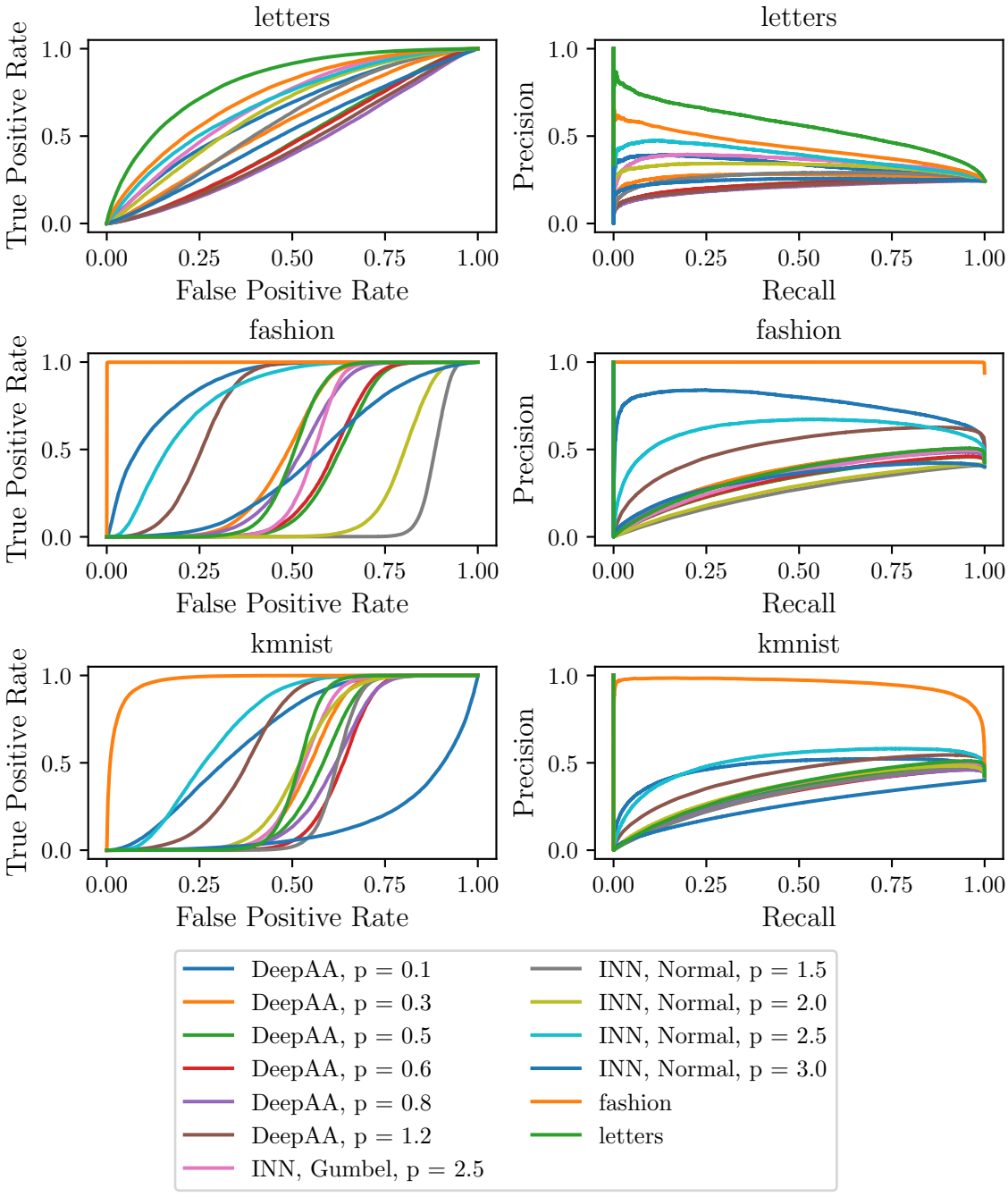

Figure 4.23: ROC curves (**left**) and precision-recall curve (**right**) for discriminators trained on different sources of outliers tested on different sources of outliers.

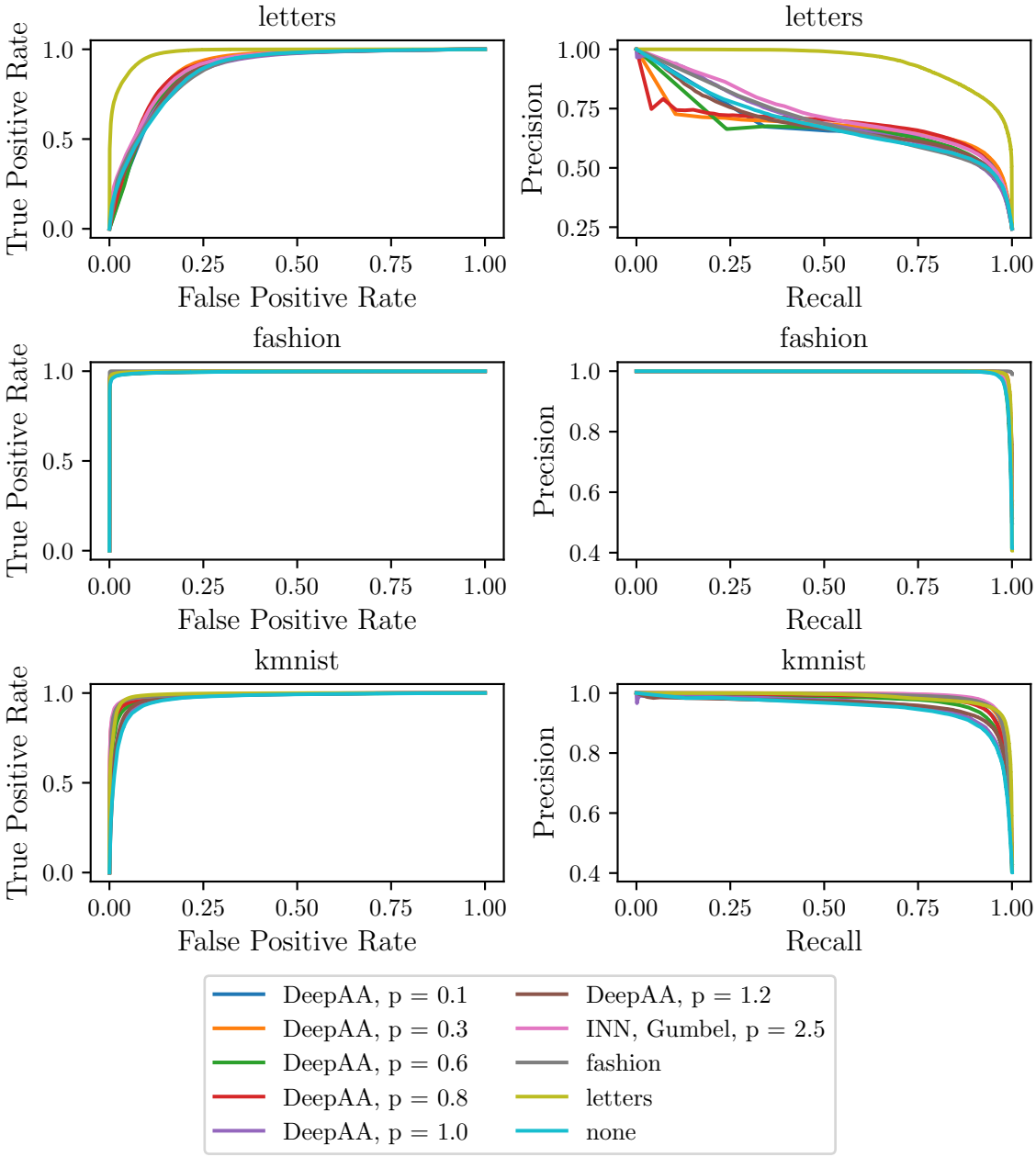

Figure 4.24: ROC curves (**left**) and precision-recall curve (**right**) for classifiers trained on different sources of outliers tested on different sources of outliers.

Table 4.2: Area under the ROC curve (AUC), average precision (AP) and test set classification accuracy (ACC) of classifiers trained on different outlier sources tested on different outlier datasets.

outliers	letters			fashion			kmnist		
metric	AUC	AP	ACC	AUC	AP	ACC	AUC	AP	ACC
DeepAA, p = 0.1	0.89	0.64	1.00	1.00	1.00	1.00	0.99	0.98	1.00
DeepAA, p = 0.3	0.90	0.66	**1.00**	1.00	1.00	**1.00**	0.99	0.99	**1.00**
DeepAA, p = 0.6	0.89	0.63	1.00	1.00	1.00	1.00	0.98	0.97	1.00
DeepAA, p = 0.8	0.90	0.68	1.00	1.00	1.00	1.00	0.98	0.98	1.00
DeepAA, p = 1.0	0.89	0.70	0.99	1.00	1.00	0.99	0.97	0.96	0.99
DeepAA, p = 1.2	0.89	0.68	1.00	1.00	1.00	1.00	0.98	0.96	1.00
INN, Gumbel, p = 2.5	0.90	0.72	1.00	1.00	1.00	1.00	**0.99**	**0.99**	1.00
fashion	0.89	0.67	1.00	**1.00**	**1.00**	1.00	0.99	0.99	1.00
letters	**0.98**	**0.95**	1.00	1.00	1.00	1.00	0.99	0.99	1.00
none	0.89	0.68	0.99	1.00	1.00	0.99	0.97	0.95	0.99

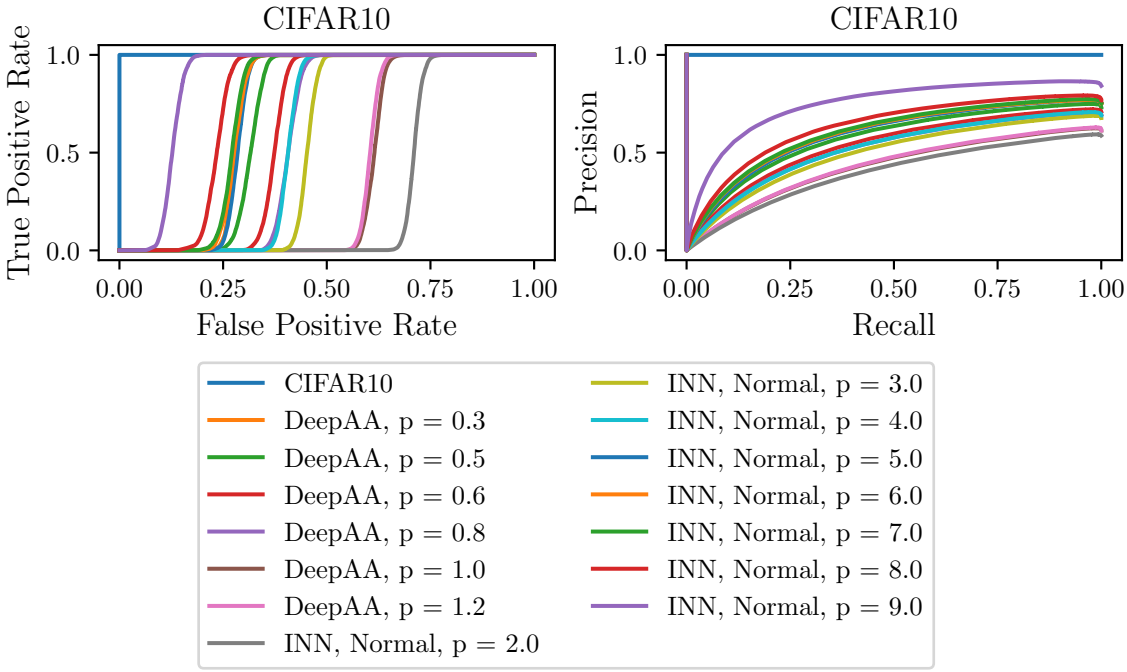

Figure 4.25: ROC curve (**left**) and precision-recall curve (**right**) for discriminators trained on different sources of outliers tested on different sources of outliers.

Table 4.3: Area under the ROC curve (AUC) and average precision (AP) of discriminators trained on different outlier sources tested on different outlier datasets.

outliers	CIFAR10	
metric	AUC	AP
CIFAR10	**1.00**	**1.00**
DeepAA, p = 0.3	0.72	0.60
DeepAA, p = 0.5	0.68	0.57
DeepAA, p = 0.6	0.63	0.54
DeepAA, p = 0.8	0.59	0.52
DeepAA, p = 1.0	0.39	0.43
DeepAA, p = 1.2	0.40	0.43
INN, Normal, p = 2.0	0.29	0.40
INN, Normal, p = 3.0	0.55	0.49
INN, Normal, p = 4.0	0.59	0.52
INN, Normal, p = 5.0	0.72	0.59
INN, Normal, p = 6.0	0.72	0.60
INN, Normal, p = 7.0	0.73	0.60
INN, Normal, p = 8.0	0.77	0.63
INN, Normal, p = 9.0	0.87	0.75

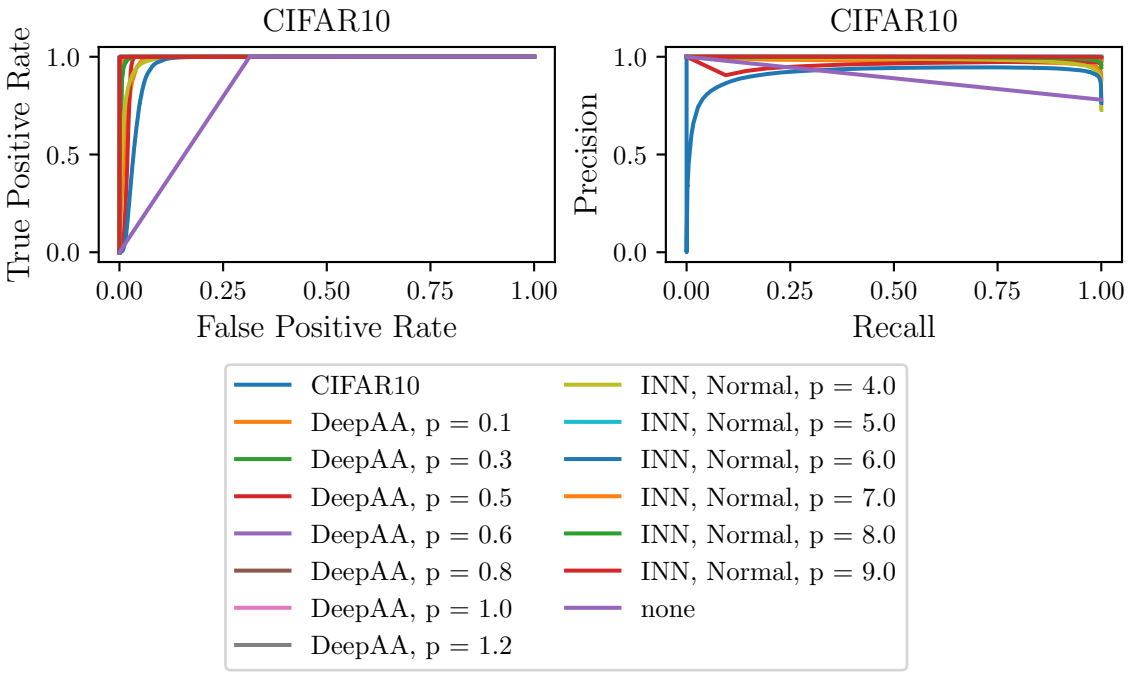

Figure 4.26: ROC curve (**left**) and precision-recall curve (**right**) for classifiers trained on different sources of outliers tested on different sources of outliers.

Table 4.4: Area under the ROC curve (AUC), average precision (AP) and test set classification accuracy (ACC) of classifiers trained on different outlier sources tested on different outlier datasets.

outliers	CIFAR10		
metric	AUC	AP	ACC
CIFAR10	**1.00**	**1.00**	**1.00**
DeepAA, p = 0.1	0.98	0.98	**1.00**
DeepAA, p = 0.3	1.00	0.99	**1.00**
DeepAA, p = 0.5	0.98	0.96	**1.00**
DeepAA, p = 0.6	1.00	1.00	**1.00**
DeepAA, p = 0.8	1.00	1.00	**1.00**
DeepAA, p = 1.0	1.00	1.00	**1.00**
DeepAA, p = 1.2	1.00	1.00	**1.00**
INN, Normal, p = 4.0	0.99	0.99	**1.00**
INN, Normal, p = 5.0	1.00	1.00	**1.00**
INN, Normal, p = 6.0	0.96	0.92	**1.00**
INN, Normal, p = 7.0	1.00	1.00	**1.00**
INN, Normal, p = 8.0	1.00	1.00	**1.00**
INN, Normal, p = 9.0	1.00	1.00	**1.00**
none	0.84	0.78	**1.00**

Chapter 5

Discussion and Outlook

Outliers and outlier detection have been an important topic in statistics for centuries [32]. Especially in recent years work has been done to develop and use generative models to improve the performance of classifiers on outlier data [24, 29, 23]. In this work we introduced normalizing flows for this task, with and without the addition of archetypal analysis.

We first investigated sampling from an outlier distribution in the latent space of a normalizing flow on toy data. In section 4.1 we see, that outlier data sampled around the normal distribution in latent space, e.g. by using a Gumbel distribution, encircles the data well when mapped back into data space. There, we also showed that using these samples we can train a classifier to give low confidence scores to outlier data while keeping up the classification performance.

Moving on to image data we showed how samples drawn from a spherical shell of larger radius around the inlier data in latent space led to sensible images and outliers similar to inlier images. We were able to show this on both, the digits dataset as well as the people dataset in section 4.2. On the people dataset we also observed images from different classes being generated as outliers of other classes. This again confirmed that the latent space is arranged well for sampling from outlier distributions.

Indeed, investigating the latent space more closely in section 4.4 showed this too. Classes that are expected to be close to each other or even hardly distinguishable, such as `0` and `o`, are close to each other in latent space.

Archetypal analysis used in conjunction with a normalizing flow has been shown to be not only an interesting way of structuring the latent space but also to exert more control over the kind of outlier that is generated. After showing that deep archetypal analysis can find meaningful extremes, such as the different emotional expressions in the people dataset, see Figure 4.8 and Figure 4.10, we showed how generating new samples from archetypes creates inlier samples as well as new outliers with control over what visual features are extreme for the digits dataset as well as

the people dataset, see Figure 4.11 to Figure 4.14. We further showed how sampling from the nullspace of the archetype mapping creates better reconstructions and more variation in generated images in Figure 4.15 and Figure 4.16 while allowing a lower number of archetypes, which allows the archetypes to be more interpretable.

Finally, in section 4.5 we were able to show that using our methods of outlier generation can be used to train a cheaper discriminator model and also to create classifiers, that give less false high-confidence scores to outlier samples.

In conclusion, we were able to show that normalizing flows are able to generate outliers close to the in-distribution. These outliers can provide insights into what to expect when working with the data at hand and are also able to be used for subsequent tasks, such as training confidence-calibrated classifiers. We showed how archetypal analysis in conjunction with normalizing flows can provide even more insights by identifying meaningful latent space representations. From these representations both inliers and outliers can be generated and used in subsequent tasks.

5.1 Future Work

In the future our methods should be investigated on even more complex data like higher resolution images. More direct comparisons to competing methods should also be implemented. Another interesting line of research is to try to get rid of the class conditioning and thereby create a fully unsupervised method of generating outliers.

List of Figures

Bibliography

[1] Deepali Aneja et al. "Modeling Stylized Character Expressions via Deep Learning". In: *Computer Vision – ACCV 2016*. Ed. by Shang-Hong Lai et al. Vol. 10112. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 136–153. ISBN: 978-3-319-54183-9 978-3-319-54184-6. DOI: 10.1007/978-3-319-54184-6_9. URL: http://link.springer.com/10.1007/978-3-319-54184-6_9 (visited on 04/19/2021).

[2] C. Bauckhage et al. "Archetypal Analysis as an Autoencoder". In: *Workshop New Challenges in Neural Computation 2015* (2015), pp. 8–16.

[3] Tarin Clanuwat et al. *Deep Learning for Classical Japanese Literature*. 2018-12-03, 2018. arXiv: 1812.01718 [cs.CV].

[4] Gregory Cohen et al. *EMNIST: An Extension of MNIST to Handwritten Letters*. Mar. 1, 2017. arXiv: 1702.05373 [cs]. URL: http://arxiv.org/abs/1702.05373 (visited on 04/19/2021).

[5] Adele Cutler and Leo Breiman. "Archetypal Analysis". In: *Technometrics* 36.4 (Nov. 1, 1994), pp. 338–347. ISSN: 0040-1706. DOI: 10.1080/00401706.1994.10485840. URL: https://www.tandfonline.com/doi/abs/10.1080/00401706.1994.10485840 (visited on 06/21/2021).

[6] Laurent Dinh, David Krueger, and Yoshua Bengio. *NICE: Non-Linear Independent Components Estimation*. Apr. 10, 2015. arXiv: 1410.8516 [cs]. URL: http://arxiv.org/abs/1410.8516 (visited on 05/22/2021).

[7] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. *Density Estimation Using Real NVP*. Feb. 27, 2017. arXiv: 1605.08803 [cs, stat]. URL: http://arxiv.org/abs/1605.08803 (visited on 04/04/2021).

[8] European Mathematical Society. *Outlier - Encyclopedia of Mathematics*. URL: https://encyclopediaofmath.org/index.php?title=Outlier (visited on 07/05/2021).

[9] Catherine Forbes et al. “Chi-Squared Distribution”. In: *Statistical Distributions*. John Wiley & Sons, Ltd, 2010, pp. 69–73. ISBN: 978-0-470-62724-2. DOI: 10.1002/9780470627242.ch11. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470627242.ch11 (visited on 06/18/2021).

[10] Catherine Forbes et al. “Dirichlet Distribution”. In: *Statistical Distributions*. John Wiley & Sons, Ltd, 2010, pp. 77–78. ISBN: 978-0-470-62724-2. DOI: 10.1002/9780470627242.ch13. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470627242.ch13 (visited on 06/18/2021).

[11] Aidan N. Gomez et al. *The Reversible Residual Network: Backpropagation Without Storing Activations*. July 13, 2017. arXiv: 1707.04585 [cs]. URL: http://arxiv.org/abs/1707.04585 (visited on 05/28/2021).

[12] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. *Why ReLU Networks Yield High-Confidence Predictions Far Away from the Training Data and How to Mitigate the Problem*. May 7, 2019. arXiv: 1812.05720 [cs, stat]. URL: http://arxiv.org/abs/1812.05720 (visited on 04/08/2021).

[13] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: (Feb. 11, 2015). URL: https://arxiv.org/abs/1502.03167v3 (visited on 07/07/2021).

[14] M. James. “The Generalised Inverse”. In: *The Mathematical Gazette* 62.420 (June 1978), pp. 109–114. ISSN: 0025-5572, 2056-6328. DOI: 10.1017/S0025557200086460. URL: https://www.cambridge.org/core/journals/mathematical-gazette/article/abs/generalised-inverse/8B58A1EC63AD8C38BE2AB74A5175028F (visited on 07/06/2021).

[15] Sebastian Mathias Keller et al. “Learning Extremal Representations with Deep Archetypal Analysis”. In: *International Journal of Computer Vision* (Dec. 23, 2020). ISSN: 1573-1405. DOI: 10.1007/s11263-020-01390-3. URL: https://doi.org/10.1007/s11263-020-01390-3 (visited on 03/16/2021).

[16] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 05/22/2021).

[17] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. July 10, 2018. arXiv: 1807.03039 [cs, stat]. URL: http://arxiv.org/abs/1807.03039 (visited on 05/22/2021).

[18] Ivan Kobyzev, Simon J. D. Prince, and Marcus A. Brubaker. “Normalizing Flows: An Introduction and Review of Current Methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. ISSN:

0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.2992934. arXiv: 1908.09257. URL: http://arxiv.org/abs/1908.09257 (visited on 04/22/2021).

[19] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009.

[20] M. R. Leadbetter, Georg Lindgren, and Holger Rootzén. "Asymptotic Distributions of Extremes". In: *Extremes and Related Properties of Random Sequences and Processes*. Ed. by M. R. Leadbetter, Georg Lindgren, and Holger Rootzén. Springer Series in Statistics. New York, NY: Springer, 1983, pp. 3–30. ISBN: 978-1-4612-5449-2. DOI: 10.1007/978-1-4612-5449-2_1. URL: https://doi.org/10.1007/978-1-4612-5449-2_1 (visited on 06/28/2021).

[21] Y. Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (7553 May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539 (visited on 05/22/2021).

[23] Kimin Lee et al. *Training Confidence-Calibrated Classifiers for Detecting Out-of-Distribution Samples*. Feb. 23, 2018. arXiv: 1711.09325 [cs, stat]. URL: http://arxiv.org/abs/1711.09325 (visited on 04/08/2021).

[24] Alexander Meinke and Matthias Hein. "Towards Neural Networks That Provably Know When They Don't Know". In: International Conference on Learning Representations. Sept. 25, 2019. URL: https://openreview.net/forum?id=ByxGkySKwH (visited on 04/08/2021).

[25] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. Nov. 6, 2014. arXiv: 1411.1784 [cs, stat]. URL: http://arxiv.org/abs/1411.1784 (visited on 07/07/2021).

[26] Eric Nalisnick et al. *Detecting Out-of-Distribution Inputs to Deep Generative Models Using Typicality*. Oct. 16, 2019. arXiv: 1906.02994 [cs, stat]. URL: http://arxiv.org/abs/1906.02994 (visited on 04/08/2021).

[27] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[28] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. Jan. 7, 2016. arXiv: 1511.06434 [cs]. URL: http://arxiv.org/abs/1511.06434 (visited on 07/07/2021).

[29] Lukas Schott et al. *Towards the First Adversarially Robust Neural Network Model on MNIST*. Version 3. Sept. 20, 2018. arXiv: 1805.09190 [cs]. URL: http://arxiv.org/abs/1805.09190 (visited on 04/08/2021).

[30] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Apr. 10, 2015. arXiv: 1409.1556 [cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 07/07/2021).

[31] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-Mnist: A Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017-08-28, 2017. arXiv: 1708.07747 [cs.LG].

[32] Arthur Zimek and Peter Filzmoser. "There and Back Again: Outlier Detection between Statistical Reasoning and Data Mining Algorithms". In: *WIREs Data Mining and Knowledge Discovery* 8.6 (2018), e1280. ISSN: 1942-4795. DOI: 10.1002/widm.1280. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1280 (visited on 07/08/2021).

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 21.07.2021 .....