Department of Physics and Astronomy
University of Heidelberg

Master Thesis in Physics
submitted by

Till Bungert

born in Lebach (Germany)

2021

WIP

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

WIP

Contents

Chapter 1

Introduction

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 on an intuitive level and then try to highlight a more rigorous approach. 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

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. (LeCun, Y. Bengio, and Hinton, 2015)

As representation-learning methods, neural networks have found use for many discriminative as well as generative tasks. Especially in recent years many advance 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.

We will now briefly introduce two other generative models based on neural networks before introducing invertible neural networks in detail.

Generative Adversarial Networks

Variational Autoencoders

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 determinant the whole forward computation is invertible by composing the inverse layers in reverse order.

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

Dinh et al. (Dinh, Krueger, and Y. Bengio, 2015) describe a family of transformations called coupling layers, that can be used as components in a invertible neural network. In general, coupling layers are defined as follows

Definition 2.2.1 (General Coupling Layer) *Let $x \in \mathcal{X}$, the input to the layer, be a vector 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 $y = (y_{I_1}, y_{I_2})$ as*

$$\begin{aligned} y_{I_1} &= x_{I_1} \\ y_{I_2} &= g(x_{I_2}; m(x_{I_1})) \end{aligned} \tag{2.3}$$

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 (Dinh, Krueger, and Y. Bengio, 2015; Gomez et al., 2017). The coupling layer transformation then becomes

$$\begin{aligned} y_{I_1} &= x_{I_1} \\ y_{I_2} &= x_{I_2} + m(x_{I_1}) \end{aligned} \tag{2.4}$$

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 general affine coupling block introduced by Dinh et al. (Dinh, Sohl-Dickstein, and S.

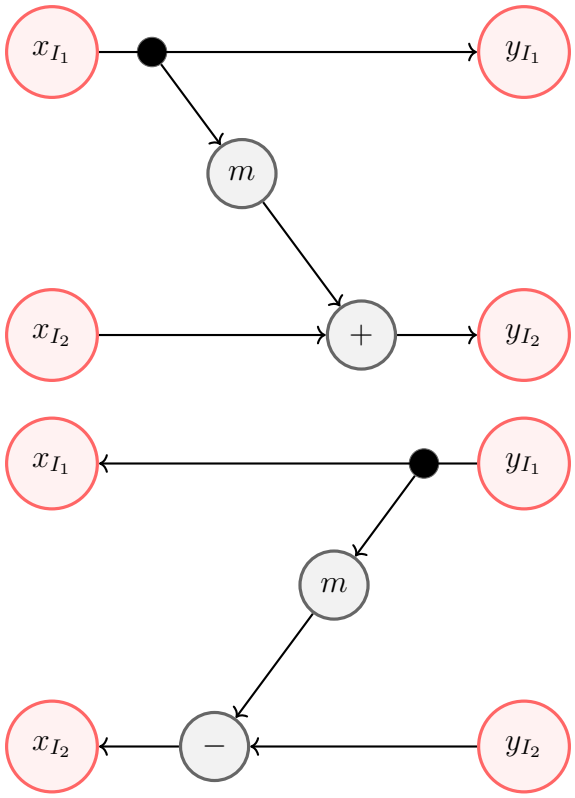

Figure 2.1: Additive coupling block. Left: forward computation, right: inverse computation.

Bengio, 2017) is

$$\begin{aligned} y_{I_1} &= x_{I_1} \\ y_{I_2} &= x_{I_2} \odot \exp(s(x_{I_1})) + t(x_{I_1}) \end{aligned} \tag{2.5}$$

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 Hadarmat or pointwise product. This transformation is again easily invertible, regardless of the complexity of s and t

$$\begin{aligned} x_{I_1} &= y_{I_1} \\ x_{I_2} &= (y_{I_2} - t(y_{I_1})) \odot \exp(-s(y_{I_1})) \end{aligned} \tag{2.6}$$

To ensure all inputs are used we need to use at least two coupling blocks and switch x_{I_1} and 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 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

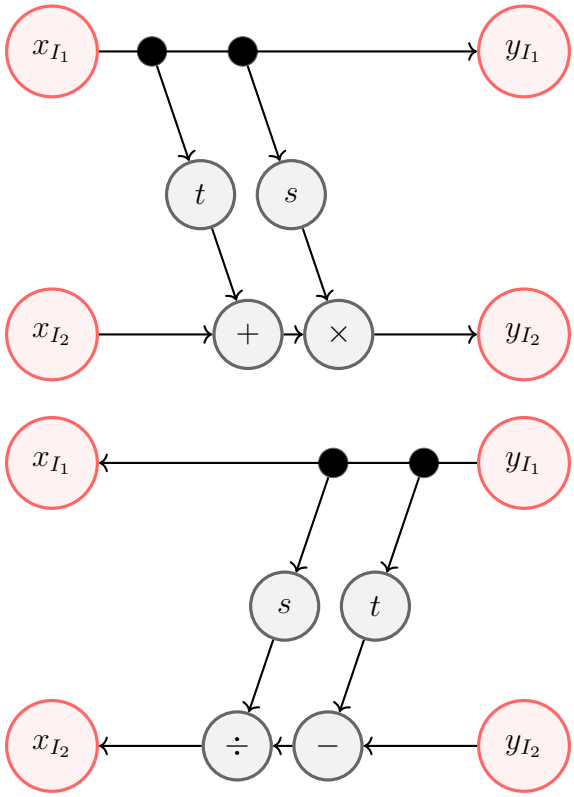

Figure 2.2: Affine coupling block. Left: forward computation, right: inverse computation.

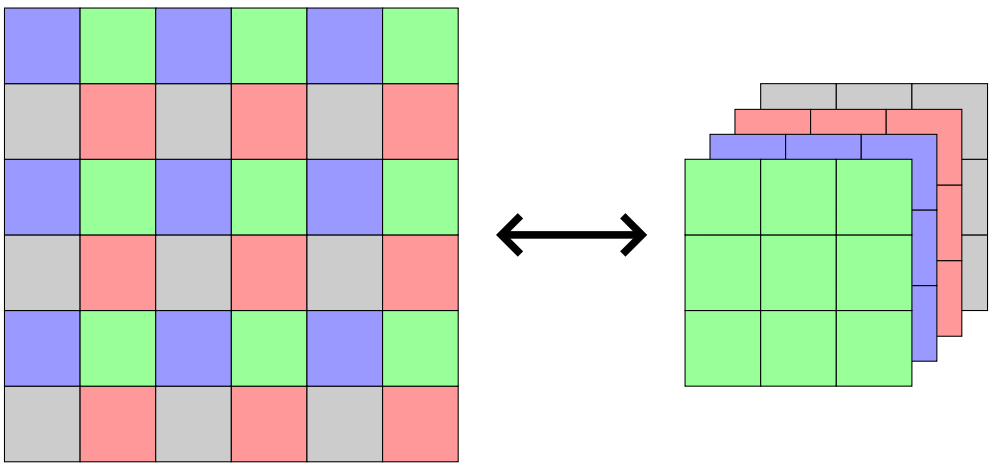

Figure 2.3: IRevNet Downsampling, figure adapted from Gomez et al. (Gomez et al., 2017)

increase field of vision of convolutions and decrease the amount of computations in later layers different strategies can be used. Gomez et al. used a pixel shuffle operation, where spatial dimensions are reduce and shifted into channel dimensions as shown in Figure 2.3 (Gomez et al., 2017).

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

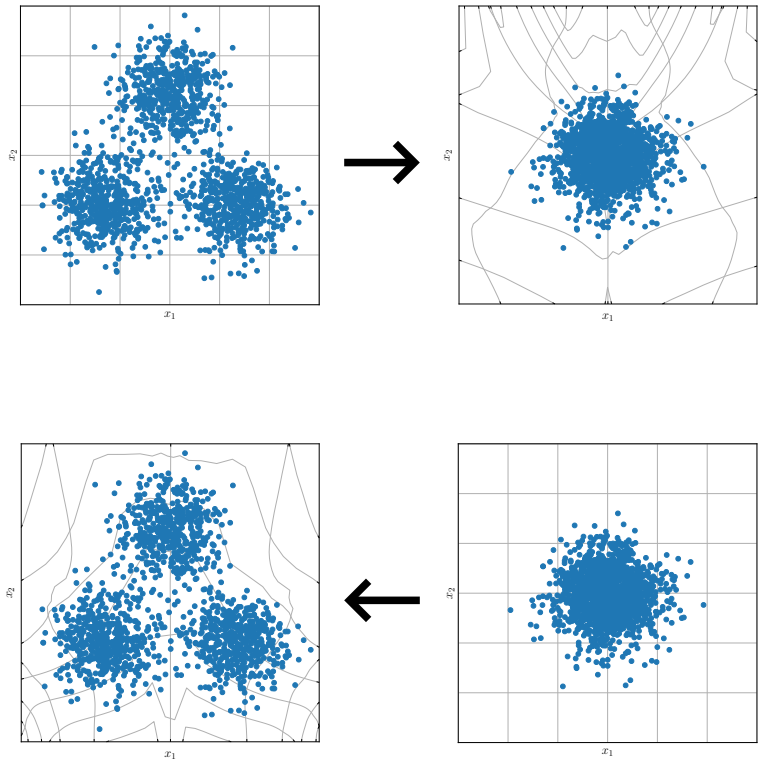

Figure 2.4: Mapping between data and latent space

Normalizing flows transform a simple probability distributions, such as a standard normal distribution, into a complex, often intractable probability distribution (Kobyzev, Prince, and Brubaker, 2020). 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

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

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 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 (Cutler and Breiman, 1994). 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 $\mathbf{z}_j \in \mathbb{R}^d$

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

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

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

Without further constraints this approach does not fulfill the requirements, that the templates are pure archetypes and that 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 convex hull.

Finding the weights a_ij and b_ij can be done by minimizing the square error in the representation

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

Cutler et al. propose an alternating least squares algorithm to solve this optimization (Cutler and Breiman, 1994). Bauckhage et al. introduced a more efficient algorithm based on sub-gradients (Bauckhage et al., 2015), which is also used in FIGURE. Keller et al. build on this framework by applying archetypal analysis to the latent space of a deep generative model (Keller et al., 2020). In their paper they propose using a Deep Variational Information Bottelneck, 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 map the latent vectors to archetype coefficients, the k archetype vectors are initialized to be the vertices of a $(k-1)$-simplex, as there is no absolute frame of reference for the latent space coordinates. The archetype vectors can be fixed when training the flow, or can be allowed to update to accomodate a pretrained flow. The full model will be discussed in the next chapter.

Chapter 3

Methods and Experiments

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 Gumbel Distribution

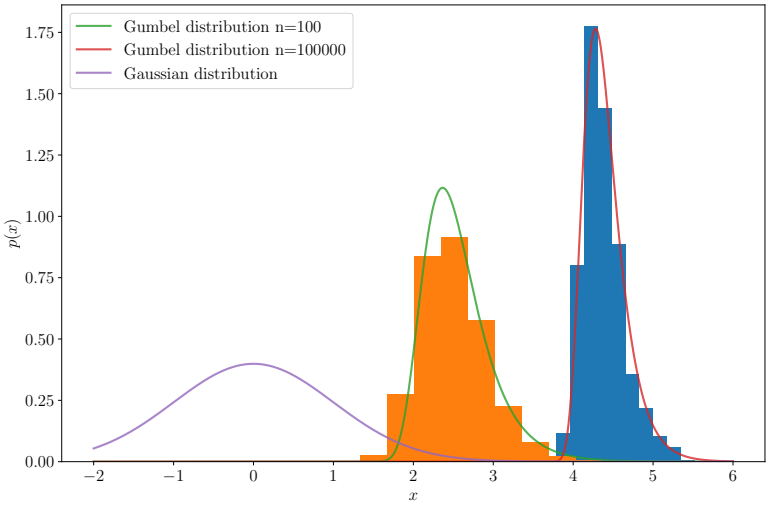

Figure 3.1: Gumbel distribution

3.1.2 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 $X = (X_1, X_2, \ldots, X_d)$ in $\mathbb{R}^d$, where the coordinates X_i are independently distributed with zero mean and unit variance. The squared length of vector X is

$$\|X\|_2^2 = \sum_d X_i^2 \tag{3.1}$$

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

$$\|X\|_2 \sim \chi_d \tag{3.2}$$

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

$$\|X\|_2 \sim \sigma\chi_d \tag{3.3}$$

We can now look at the expected length of the random vector X

$$\begin{aligned} \mathbb{E}\|X\|_2 &= \sqrt{2}\frac{\Gamma(\frac{d+1}{2})}{\Gamma(\frac{d}{2})} \\ \mathrm{Var}\|X\|_2 &= d - (\mathbb{E}\|X\|_2)^2 \end{aligned} \tag{3.4}$$

meaning 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 sample coordinates $X_i \sim \mathcal{N}(0, \sigma^2)$ and choose σ such that a desired radius is reached.

3.2 Geometrical Approach

3.3 Discriminator Training

3.4 Network Architecture

3.5 Datasets

Chapter 4

Results

4.1 Toy Example

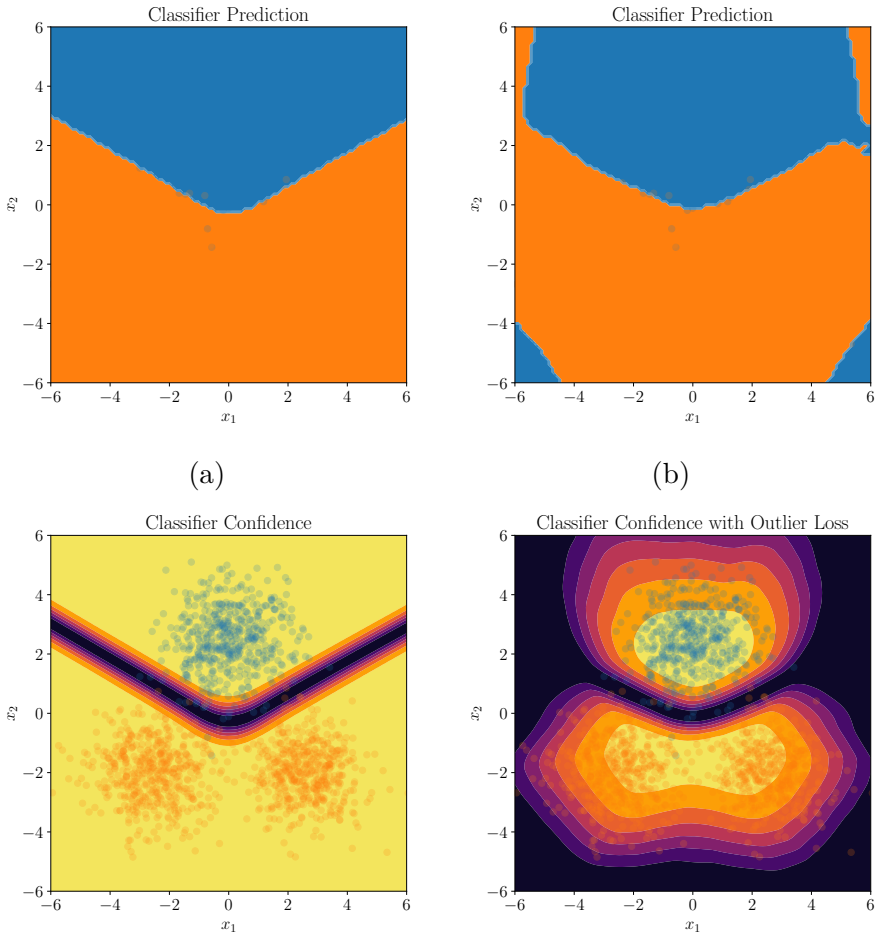

Figure 4.1: Gaussian mixture classifier performance

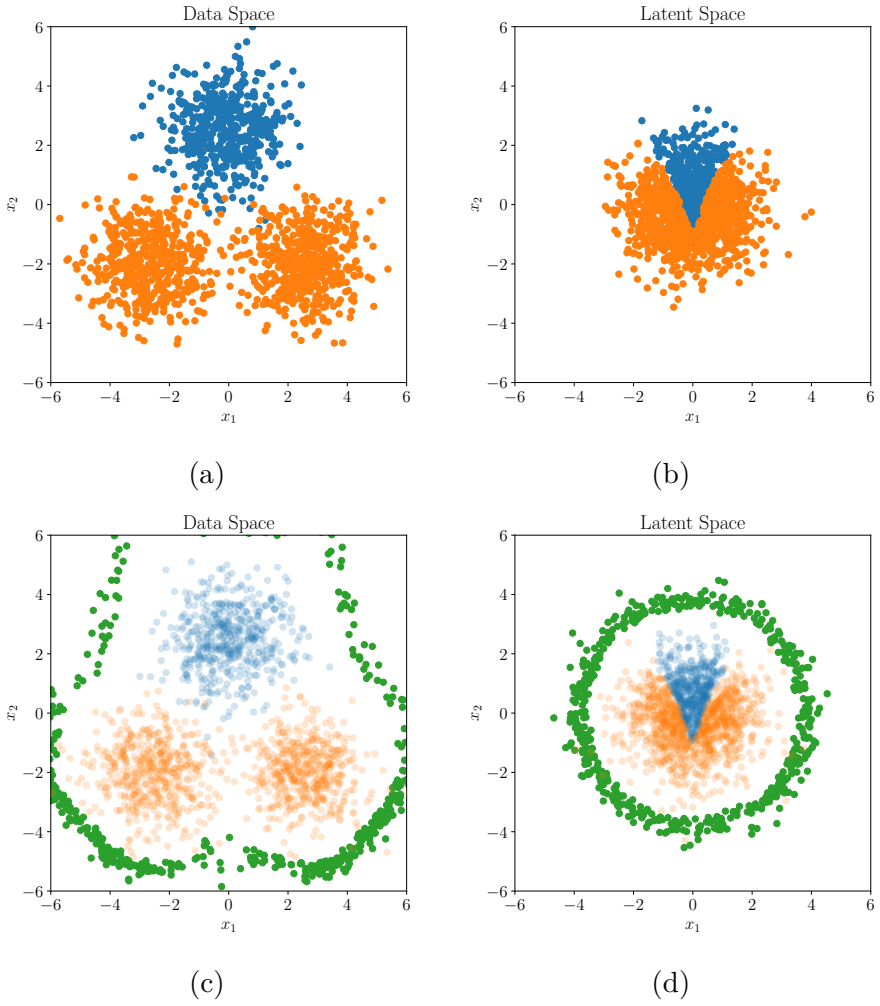

Figure 4.2: Gaussian mixture latent mapping

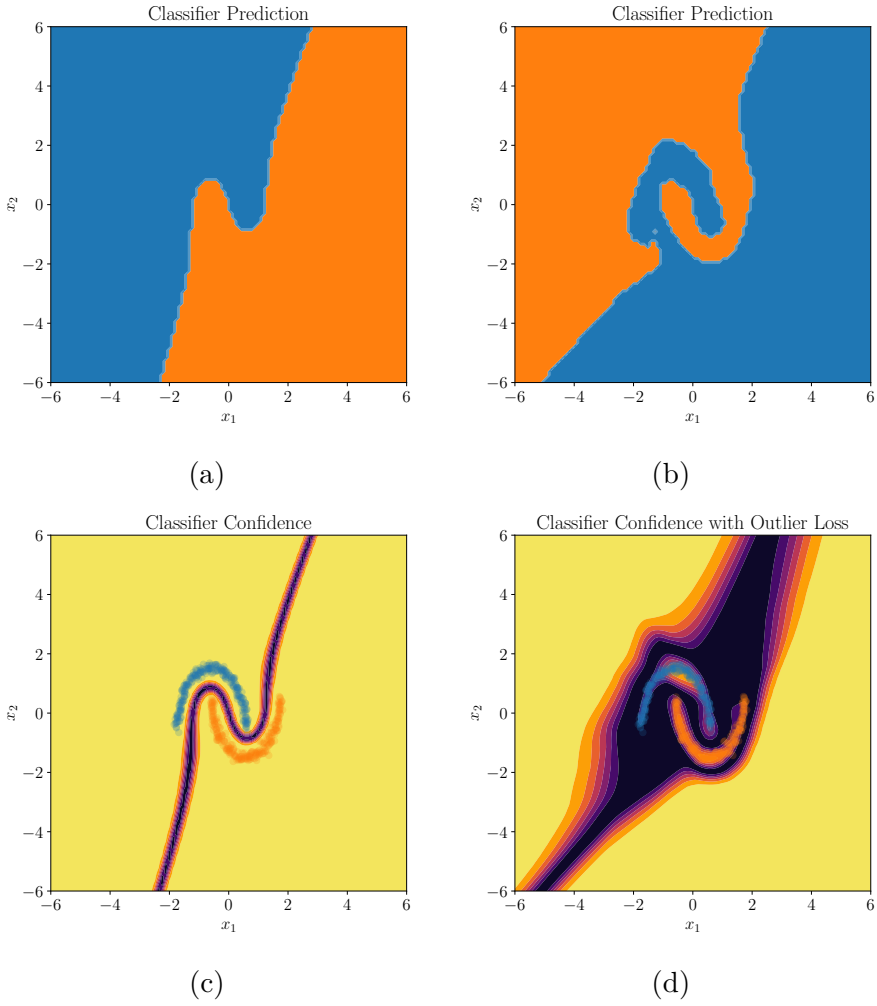

Figure 4.3: Moons classifier performance

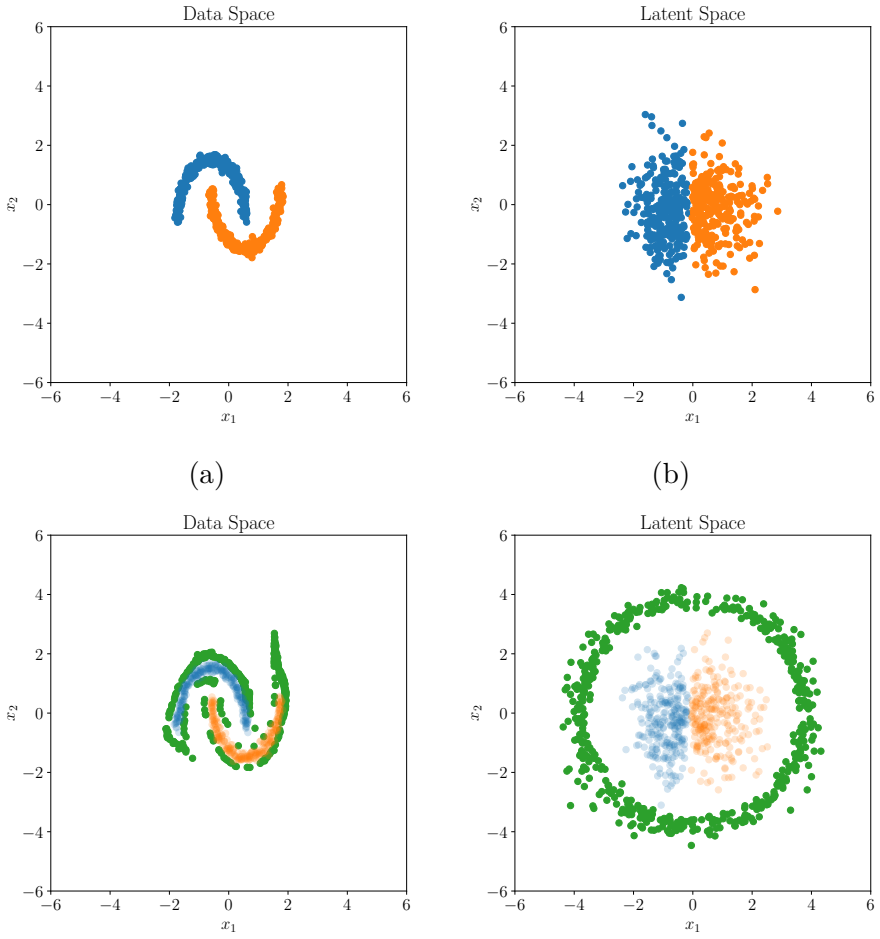

Figure 4.4: Moons latent mapping

4.2 Qualitative Comparison

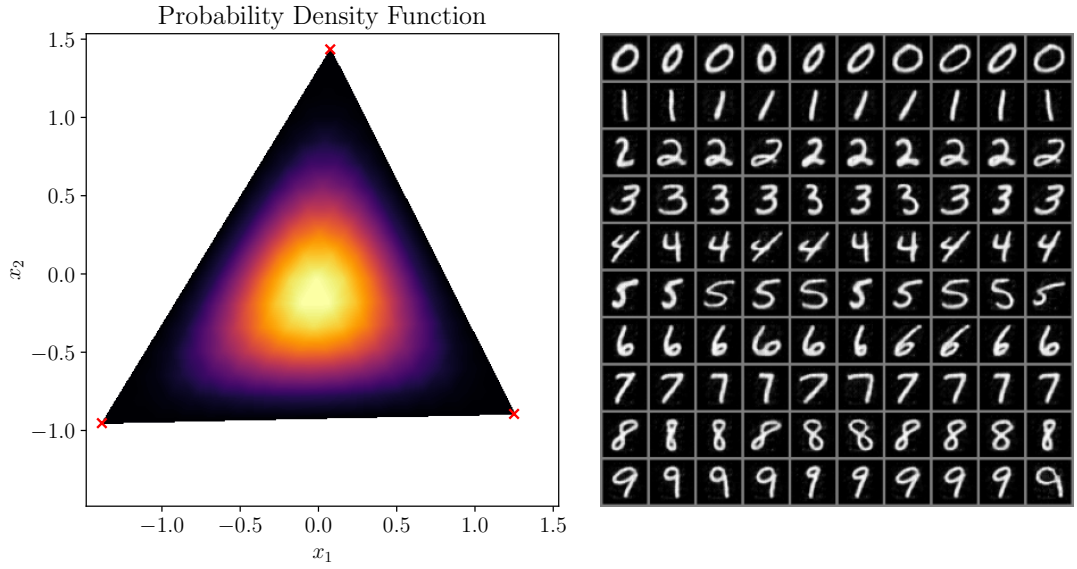

Figure 4.5: AA FERG

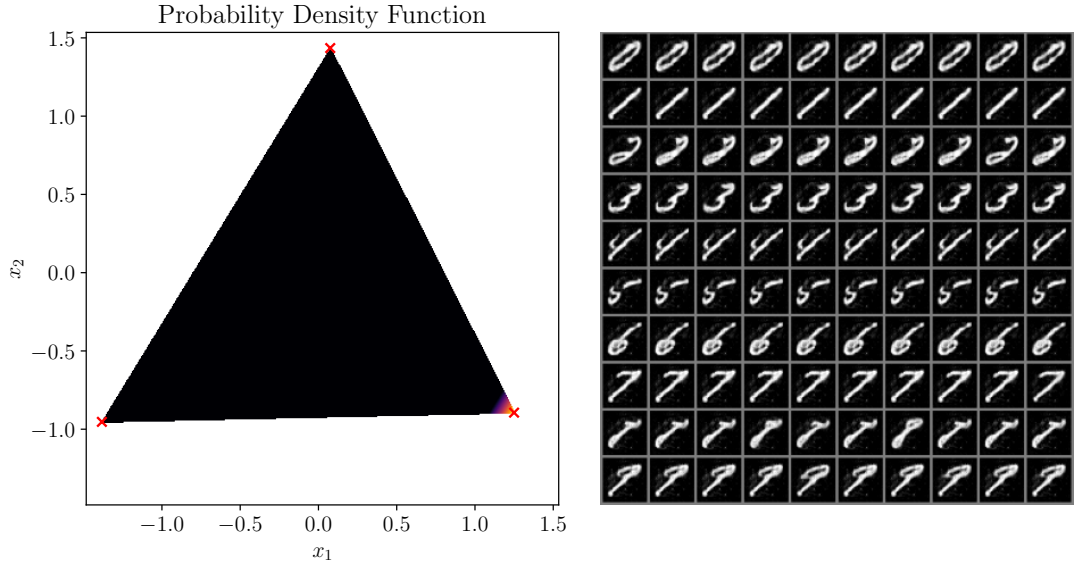

Figure 4.6: AA FERG

4.3 Discriminator Performance

4.4 Analysis of the Latent Space

4.5 General Performance

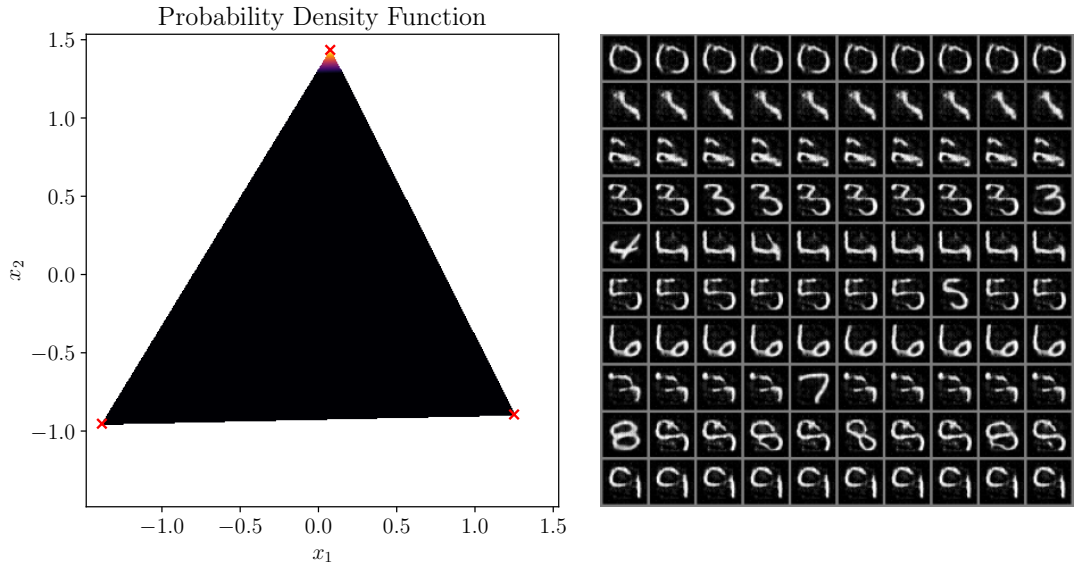

Figure 4.7: AA FERG

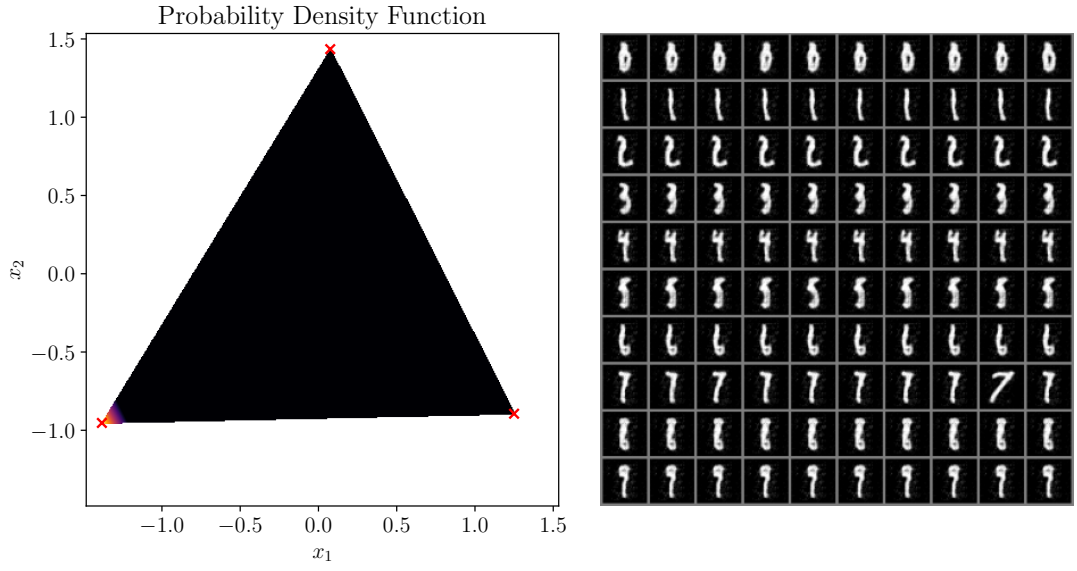

Figure 4.8: AA FERG

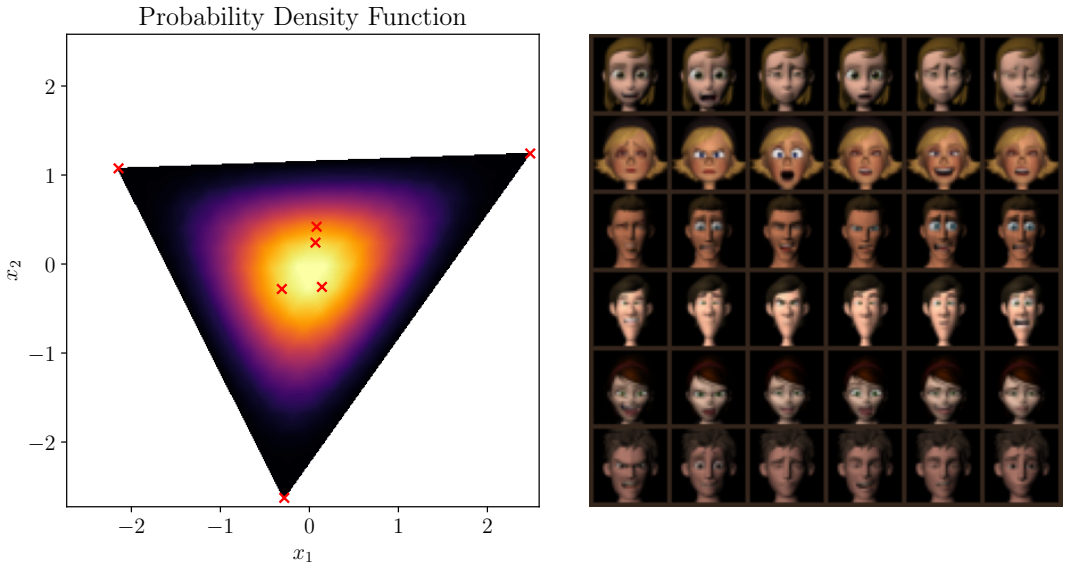

Figure 4.9: AA FERG

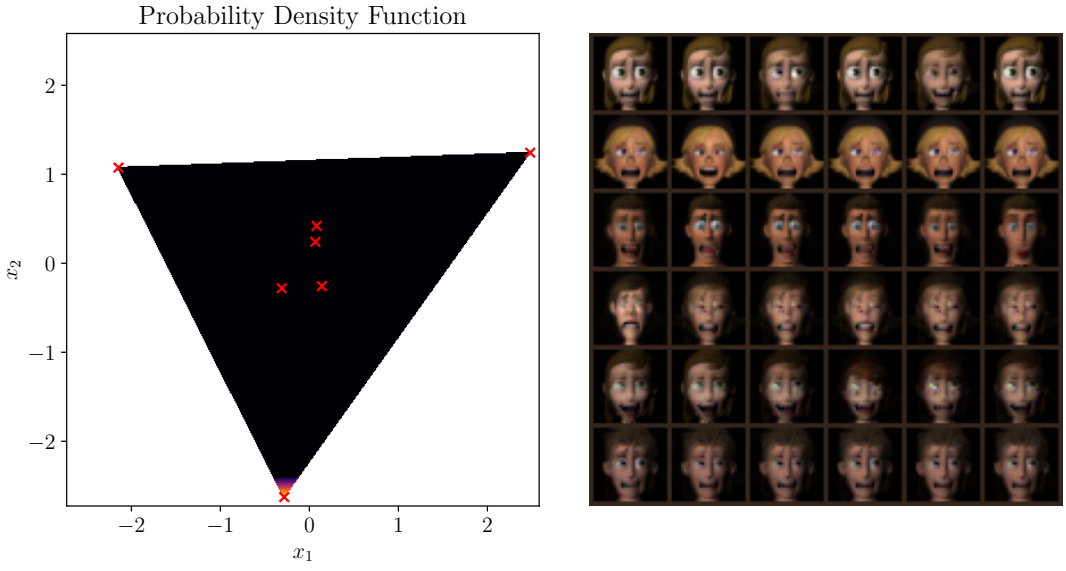

Figure 4.10: AA FERG

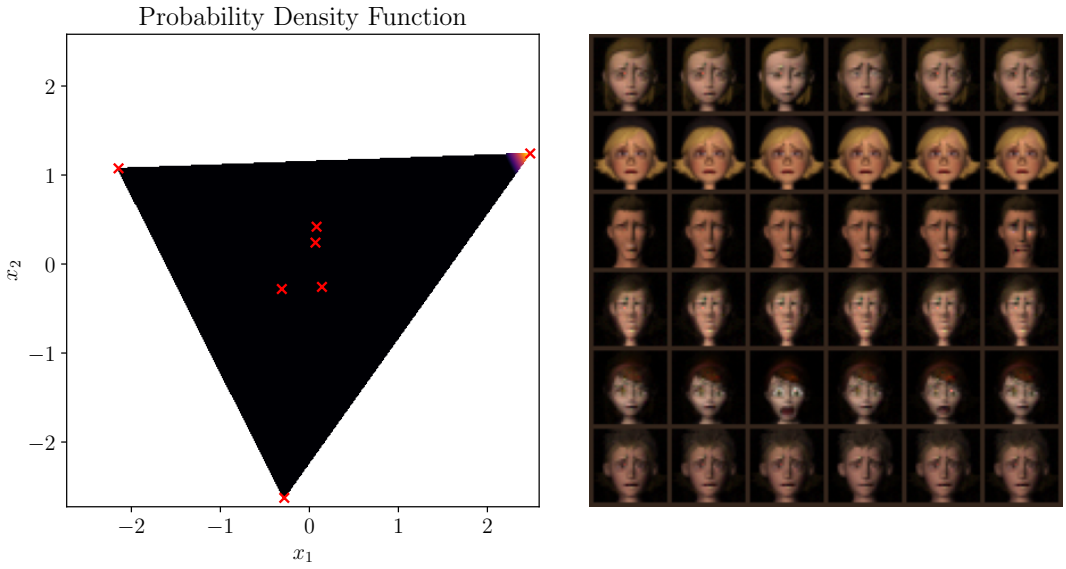

Figure 4.11: AA FERG

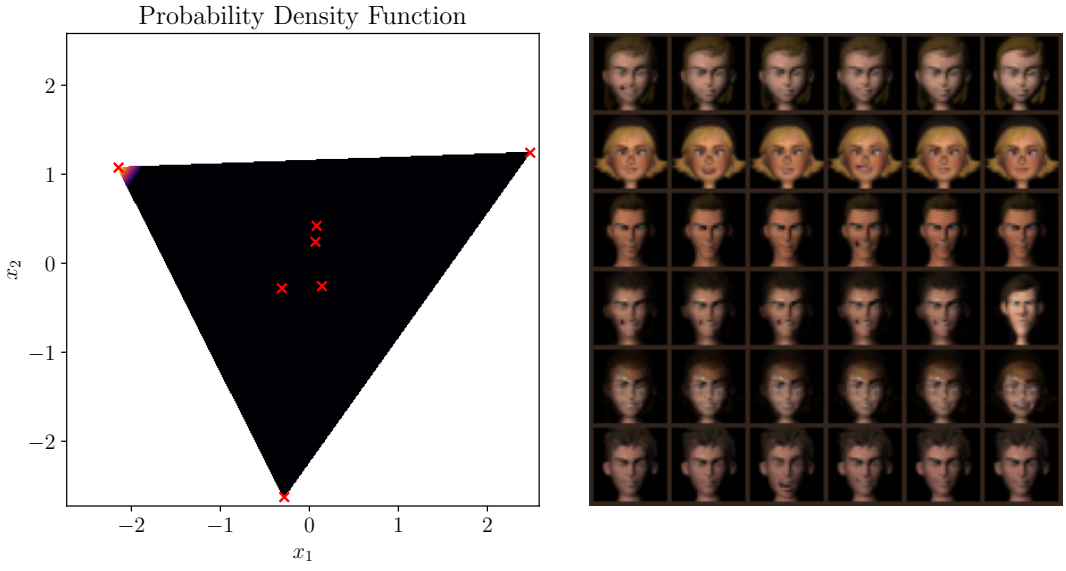

Figure 4.12: AA FERG

4.6 Robustness

Chapter 5

Discussion and Outlook

5.1 Future Work

Bibliography

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

Cutler, Adele and Leo Breiman (Nov. 1, 1994). "Archetypal Analysis". In: *Technometrics* 36.4, 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).

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

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

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

Keller, Sebastian Mathias et al. (Dec. 23, 2020). "Learning Extremal Representations with Deep Archetypal Analysis". In: *International Journal of Computer Vision*. 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).

Kobyzev, Ivan, Simon J. D. Prince, and Marcus A. Brubaker (2020). "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 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).

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