technische universität
dortmund

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science

Anomaly Detection in IceCube Waveforms Using Machine Learning and Monte Carlo Methods

Max Pernklau
born in Herdecke

2020

Lehrstuhl für Experimentelle Physik 5b
Fakultät Physik
Technische Universität Dortmund

First Supervisor:	Prof. Dr. Dr. Wolfgang Rhode
Second Supervisor:	Prof. Dr. Kevin Kröninger
Submission Date:	April 17, 2020

Abstract

IceCube is a neutrino detector that instruments $1\,\mathrm{km}^3$ of Antarctic ice. It detects the Cherenkov light emitted by particles created when neutrinos interact inside the detector. Describing the signal that neutrino interactions produce is complicated and relies on MC simulations. There is, however, a small measurement–simulation mismatch that can influence analyses.

Unsupervised machine learning automatically finds patterns in big datasets while not relying on simulations as heavily. So far, little research about its application to IceCube data has been published.

Autoencoders are a class of unsupervised neural networks. This work develops an autoencoder that is applied to waveforms, a low-level type of IceCube data. It is used to find outliers, i.e. waveforms that appear seldomly and differ from more common waveforms. Futhermore, a simplified waveform MC simulation is developed and used to run a hyperparameter optimization, comparing different autoencoders. The autoencoder's stability is analyzed and improved by using an ensemble method.

Contents

Part I

Introduction

1 Preamble

Neutrinos open a unique window into the cosmos: Unaffected by inter- and extra-galactic magnetic fields, the Greisen–Zatsepin–Kuzmin cut-off and clouds of dust, they offer deep views into the universe at the highest energies [17, Chapter 17-18] [5]. They are produced in most hadronic processes and therefore carry information about the inner workings of cosmic accelerators like active galactic nuclei [23] and supernovae [22].

But the same properties that make neutrinos excellent cosmic messengers make them difficult to detect: Only interacting via the weak force, large detector volumes are required to measure astrophysical neutrinos [29]. With $1\,\mathrm{km}^3$ of Antarctic ice, the IceCube Neutrino Observatory is the largest neutrino telescope to date. It measures the Cherenkov light emitted by particles that are produced when a neutrino interacts inside the detector.

Due to the detector's size, the indirect detection mechanism, and a cosmic ray background that is hard to suppress, the detector response of such telescopes is difficult to calibrate with traditional methods. It is therefore modeled in extensive Monte Carlo (MC) simulations [8]. Due to the complexity of the task, building a general model describing the entire detector and possible neutrino interactions well is quite difficult; even describing the data–MC mismatch is not straightforward [6]. Therefore, many analyses focus on a small selection of the available data, such as using only high-energy events that started within the detector. This selection rejects events that are hard to model or do not match other a priori criteria (*quality cuts*). Such an approach allows for better modeling of already known processes (*exploitation*) at the expense of lowering the chance to find new physics or problems with the existing detector models (*exploration*).

This work focuses on developing a machine learning method to find unusual events without requiring an event selection and with a minimal amount of assumptions about the available data. Unsupervised machine learning offers the ability to efficiently check big datasets for anomalies. Autoencoders in particular are a promising way to detect outliers [20] [21]. But unlike supervised learning, where reliable methods exist to assess the quality of the models used, it is much harder to verify that an autoencoder is functioning correctly.

This work, therefore, proposes a method to estimate the performance of autoencoders

on IceCube waveforms. It is exemplified by applying it to an autoencoder model that was selected during pilot testing.

2 IceCube and Neutrino Astronomy

IceCube is a Cherenkov detector utilizing $1\,\mathrm{km}^3$ of Antarctic glacier as a detector medium. It has been taking data since 2009. 5160 Digital optical modules (DOMs), arranged on long strings as shown in Figure 2.1, are used to detect Cherenkov photons. These are emitted by secondary particles when a neutrino interacts with hadrons or electrons inside the detector.

IceCube was constructed to find astrophysical neutrinos, i.e. neutrinos originating beyond our solar system. It successfully detected an astrophysical diffuse neutrino flux [19], albeit the processes generating it are not fully understood. A neutrino point source was likely observed in 2017 using a multi-messenger approach [23], but a highly significant point source has yet to be found. Other scientific goals include the search for neutrino oscillation deviations, dark matter WIMPs and other heavy exotic particles like magnetic monopoles. An in-depth introduction to the IceCube detector is given in [11] while [25] provides a comprehensive introduction to neutrino astronomy.

2.1 Neutrino Interactions, Cherenkov Emission and Photon Propagation

There are multiple ways a neutrino can interact inside the detector: The $\nu_\mu N \to \mu X$ charged current interaction produces track-like event topologies while all other neutral and charged current interactions ($\nu N \to \nu X$ and $\nu_e N \to eX$) produce electromagnetic cascades[1]. About 2/3 of all events are cascades. The different topologies are illustrated in Figure 2.2.

These processes generate relativistic charged particles that emit Cherenkov photons under the angle

$$\Theta_\mathrm{C} = \operatorname{acos} \frac{1}{\beta n\left(\nu\right)} \,, \tag{2.1}$$

[1] N represents a nucleon, X is the hadronic final state.

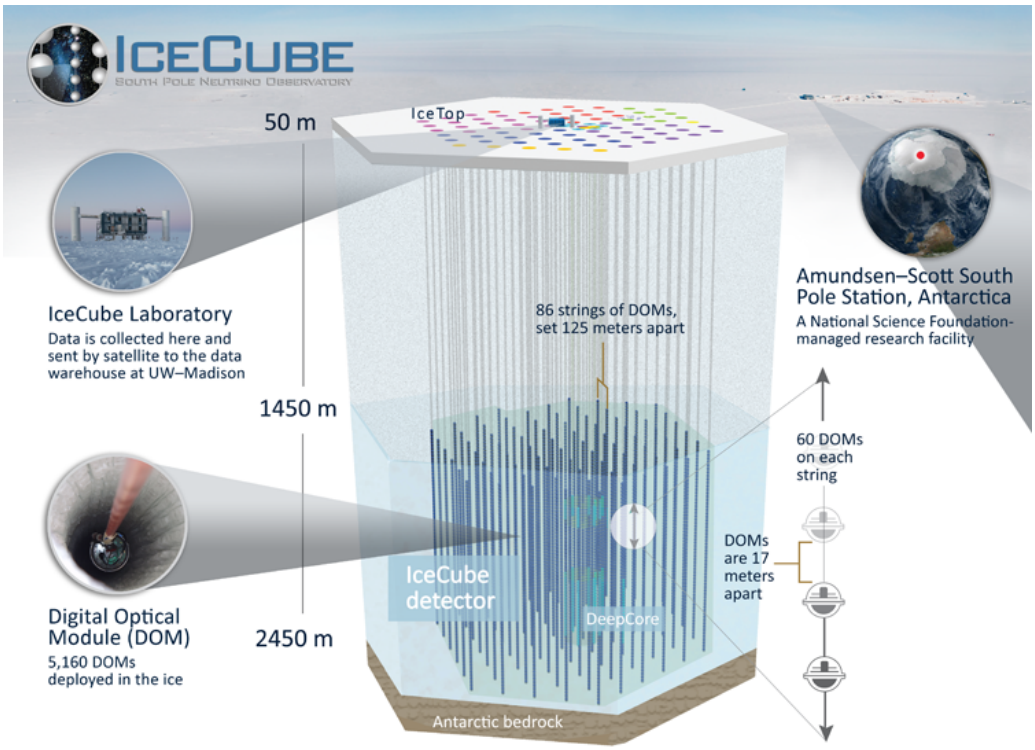

Figure 2.1: Overview of the IceCube detector [15]. IceTop (colored circles), an extensive air shower detector, sits on top of the glacier, surrounding the laboratory. The IceCube detector itself is located under 1.5 km of glacial ice, reducing the cosmic ray background. DeepCore (light blue) is located in the upper and lower center of the instrumented region and has a higher density of DOMs.

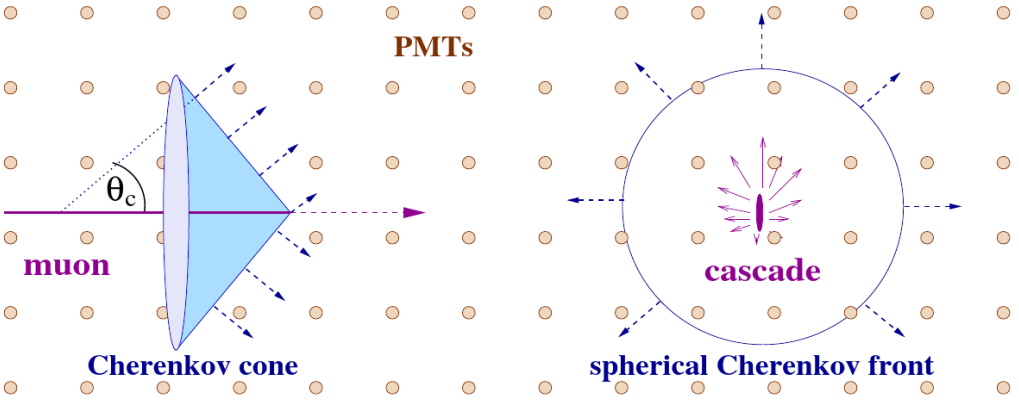

Figure 2.2: Cherenkov photon emission of the two main types of event topologies [25]. Muons produce track-like events where most photons are emitted under the Cherenkov angle Θ_C. In cascade-likes events, many charged particles with different travel directions are created, resulting in an emission of Cherenkov light in all directions. The actual size of the cascade is in the order of 10 cm, basically point-like in comparison to the DOM spacing.

as seen by a resting observer[2]. For typical secondary charged particles in IceCube ($\beta \approx 1$) this produces about 200 photons per centimeter in the relevant transparency window of ice under the angle of $\Theta_C \approx 43°$. As secondary particles lose energy to the surrounding ice, they create additional charged particles that contribute to the Cherenkov light. Pair production and converted bremsstrahlung contribute the most additional Cherenkov light [25].

Atmospheric muons that are generated in extensive air showers when cosmic rays interact with the atmosphere also emit Cherenkov light. Being 10^4 to 10^6 times (depending on depth) more common than neutrino events, they are the most important background for neutrino-induced events.

The Cherenkov photons can be detected far away from their emission site as the ice is mostly clear (the average absorption length is about 100 m in ice). There are, however, dust particles in the ice that scatter photons (the average effective scattering length is about 20 m) [3]. This makes the reconstruction of the emission's origin more difficult. The scattering coefficient varies with depth in ice as the amount of dust deposited on the glacier changed over the centuries. Most notably, around a depth of 2000 m the so-called *dust layer* decreases the number of observed photons significantly [9].

[2] β denotes the emitting particles velocity in natural units, $n(\nu)$ is the frequency-dependent refractive index of the medium.

2.2 Photons and Waveforms

Only Cherenkov photons that hit a DOM can be detected. The photomultiplier tube (PMT) of the DOM produces a photocurrent depending on the number of photons that hit the PMT. This current is captured on four sets of 128 capacitors over 420 ns. The first set is then digitized by the Analog Transient Waveform Digitizer (ATWD), a specialized analog-to-digital converter, with a high gain. If too many photons hit the PMT, the ATWD saturates and the second set of capacitors, containing the same waveform, is digitized with a lower gain. This process repeats once more if the second digitalization also shows ATWD saturation. If certain trigger conditions [12, Chapter 4] are met, the waveform is then sent to the surface. [12] describes the whole process in great detail.

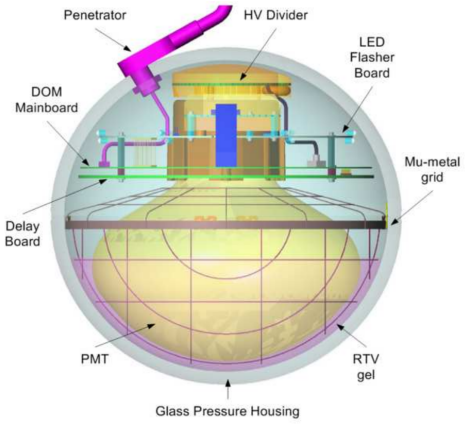

Figure 2.3: Front view of a DOM, showing the PMT and electronics [12]. The PMT's photocathode faces downwards (to the north pole) to suppress the detection of Cherenkov light of atmospheric muons.

This means that a typical waveform in IceCube is a vector of 128 consecutive 3.3 ns slices. The jitter of the internal DOM clock causes a spread of photon arrival times by about the same amount, as does the spreading of arrival times by the PMT [13]. This means that not all waveforms are captured in their entirety[3]: Depending on the distance between the DOM and the charged particle emitting the Cherenkov photons, waveforms have a width of 50 ns to 1 μs (for distances between 1 m to 160 m). Nonetheless, bright waveforms with sources close to the DOM contain the most useful information and can mostly be captured whole.

Figure 2.4 summarizes the most important effects that influence the shape of a waveform: A single charged particle would be observed by a perfect detector in a perfectly clear medium as very narrow peak of Cherenkov light, corresponding to the slightly different angles the photons of different wavelengths are emitted under ($n(\nu)$ in Equation 2.1). Clock jitter smears the arrival times into a broader peak that is symmetric around the actual arrival times. Dark count and DOM electronics add a symmetric noise pedestal. As the original particle loses energy to the medium,

[3] There is, however, a pipelined PMT analog-digital-converter for continuous digitalization that can capture events with a lower time resolution over 6.4 μs. This kind of data is not used in this work in order to focus on the waveforms with a better time resolution.

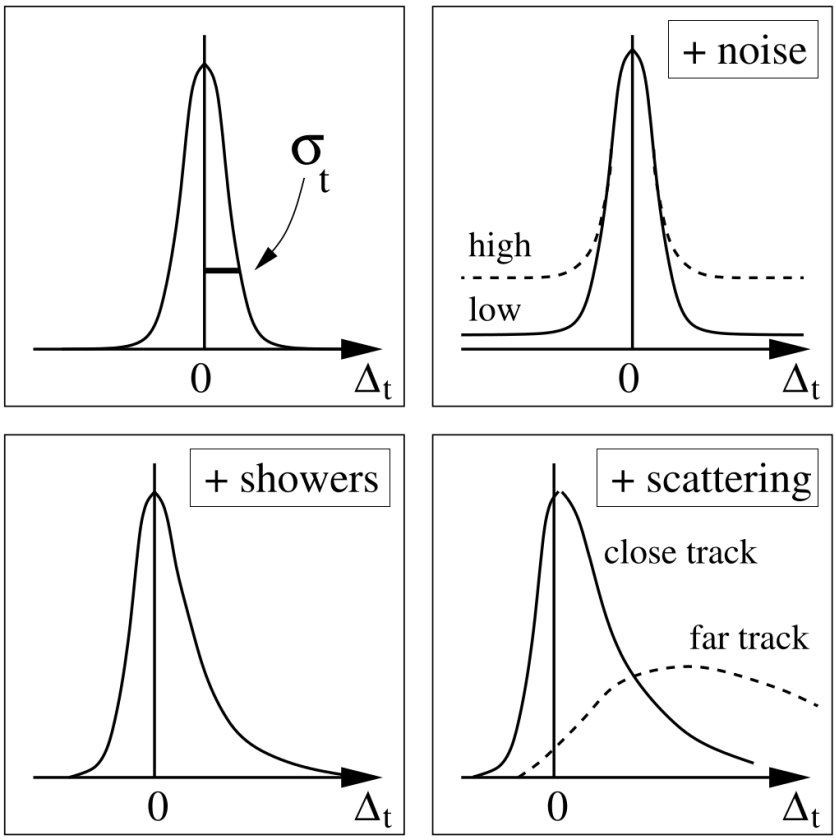

Figure 2.4: Contributions to the shape of a waveform [25]. From left-to-right and top-to-bottom more and more effects contribute to the waveform: PMT with clock jitter σ_t; noise due to DOM electronics and dark count; contributions by secondary particles; transit time spread due to scattering.

additional charged particles emit Cherenkov light, too. This light can only hit the DOM after the photons emitted by the original particle, so an asymmetric tail to higher arrival times develops. Scattering due to dust also increases the amount of photons that arrive later, as scattered photons take longer path lengths. The whole distribution is therefore influenced by the distance to the emission site.

Chapter 4 describes how these insights are used to model waveforms in MC simulations.

3 Machine Learning

Machine learning is an automated extraction of patterns from data in order to make predictions and decisions [27, Chapter 1]. The common problem machine learning is applied to is modeling a real-world phenomenon that has no readily-available model describing the observations.

Instead of employing a fixed set of rules, i.e. an *algorithm* that describes causal relations in the data, machine learning uses stochastic methods to automatically find correlations in the data. As large amounts of data are needed to extract meaningful correlations, machine learning is typically done on computers. Models with great predictive power can be build even without detailed understanding of the underlying processes [7]. For problems involving many parameters with complicated relationships between them, it is often easier to employ a machine learning model than writing an algorithm with the same predictive power [34, pp. 1-4].

This chapter gives a short outline of the basic algorithms and methods used in this work. An elaborate introduction is given in [18]. Why these methods were chosen and how they are applied is explained in chapter 5.

3.1 Supervised and Unsupervised Learning

Machine learning can be divided into three subfields with distinct goals:

Supervised learning uses two datasets to model problems: A set of independent variables ($X = \{\vec{x_1}, \vec{x_2}, ...\}$) that will be called *input* and a set of dependent variables ($Y = \{\vec{y_1}, \vec{y_2}, ...\}$) that will be called *target*. The goal is to use the input dataset to calculate, or *predict*, the target dataset:

$$f\left(\vec{x_i}\right) \approx \vec{y_i} \quad \forall i \; . \tag{3.1}$$

To find a suitable model f, the chosen machine learning method uses both datasets to extract relationships between X and Y. This process is called *training* or *learning*. After obtaining such a model, the machine learning method can be used to *predict* Y for new datasets where only X is known.

On the other hand, **Unsupervised learning** only considers X, which is useful in cases when the target Y is unknown. The goal here is to find patterns like clusters in the data. A common approach is *data reduction*, where an entry $(\vec{x})_t$ of $\vec{x}$ can be described as a dependent variable using a set of other entries:

$$\vec{x} \in X \tag{3.2}$$

$$\vec{k} = [(\vec{x})_a, (\vec{x})_b, ...] \quad \text{with } (\vec{x})_t \notin \vec{k} \tag{3.3}$$

$$r\left(\vec{k}\right) \approx (\vec{x})_t \ . \tag{3.4}$$

In many cases, multiple dependent variables can be found, so that the resulting relations $R = \{r_1, r_2, ...\}$ can be used to parameterize the input as a smaller vector $\vec{k}$:

$$\dim \vec{k} < \dim \vec{x} \tag{3.5}$$

$$\vec{x_i} \approx [r_1\left(\vec{k}\right), r_2\left(\vec{k}\right), \dots] \ . \tag{3.6}$$

In this work, a method of data reduction, *autoencoder*, is used to model a dataset. The primary focus is, however, not on the model itself but on the performance of this model: Most data points can be described by R, but on some $\vec{x}$ these relations do not hold. This kind of pattern recognition is called *outlier detection*.

A third subfield is **reinforcement learning**[1]: It deals with dynamic problems that need to be solved in multiple steps, where each partial solution can change the parameters of the problem. This changes the nature of the task substantially as the quality of each partial solution needs to be judged by a heuristic—a simple loss function would be insufficient [7, Chapter 9]. Good examples are games like chess: Moves of one player influence the moves of other players and vice versa.

3.2 Neural Networks

Feed-forward neural networks are a type of machine learning method that can be used to approximate almost any function [30]. They are composed of so-called layers;

[1] It is not used in this work and only listed here for the sake of completeness.

each layer represents a differentiable function that is applied to the output of the previous layer[2]:

$$\vec{y} \approx f\left(\vec{x}\right) = \left(\ldots f_3 \circ f_2 \circ f_1\right)\left(\vec{x}\right) \ . \tag{3.7}$$

Each layer f_i can have multiple adjustable parameters called *weights* or *learnable parameters* $\Theta_i = \{\theta_j | \theta_j \in \mathbb{R}\}$. The weights of all layers are called $\vec{\theta}$. Given a composition f, the goal of the *optimizer* is to find a $\vec{\theta}$ so that the distance between the target $\vec{y}$ and the output of the neural network $f\left(\vec{x}\right)$ is minimized. This distance is called *loss* and the choice of metric, or *loss function*

$$d : f(\vec{x}), \vec{y} \mapsto l \tag{3.8}$$

$$\text{with } l \in \mathbb{R} \tag{3.9}$$

is problem-specific, but has to be differentiable [28].

To find good weights, the optimizer increases or decreases the entries of $\vec{\theta}$ according to the signs of partial derivate $\frac{\partial d\left(f(\vec{x}), \vec{y}\right)}{\partial \vec{\theta}}$ in a process called *statistical gradient descent*. This process is repeated for each element in X and Y until the loss is sufficiently small: The training is said to *converge*[3]. There are a multitude of approaches of how to adjust $\vec{\theta}$ exactly in order to find a global minimum or a sufficiently small local minimum of the loss function, as the shape of d can be quite complex in the multidimensional space of $\vec{\theta}$.

3.2.1 Layer Types

The most commonly used layer is called the fully-connected or **dense layer**. It maps an n-dimensional input space to an m-dimensional output space via an affine transformation:

[2] The term *layer* is sometimes used ambiguously in the literature. Sometimes it represents the transformation (the layer "applies [a function] to the input" [2]) while denoting the vector the transformation is happening to at other occurrences ("output layer", "hidden layer" [18]). Here, *layer* is used in the former sense.

[3] Convergence is used in this work with the loose meaning of "the loss no longer reducing in a useful way", as no practical approach to reliably detecting convergence neural network training has not been found yet.

$$f'_{\text{dense}} : \mathbb{R}^n \to \mathbb{R}^m \tag{3.10}$$

$$\vec{q} = f'_{\text{dense}}(\vec{p}) = M\vec{p} + \vec{b} \tag{3.11}$$

$$\text{with } M \in \mathbb{R}^m \times \mathbb{R}^n \,. \tag{3.12}$$

To describe non-linear relations, it is sufficient [30] to apply a simple non-linear function, the so-called *activation function* σ, to each component of the affine transformation:

$$f_{\text{dense}} = [\sigma((\vec{q})_1), \sigma((\vec{q})_2), \dots] \,. \tag{3.13}$$

Common choices for σ are the sigmoid function and the ReLU function ($x \mapsto \max(0, x)$). See [36] for a detailed description.

Concatenating two or more dense layers allows a neural network to approximate any continuos function (Universal Approximation Theorem, [30]); using even more layers allows the network to build internal representations of the input data [39].

The main drawback of dense layers is their size and the resulting computational effort and memory usage: Each layer has $(n+1)\,m$ weights, but to achieve low losses, high-dimensional M are needed. There are multiple alternative layers that try to solve this problem; this work uses **convolutional layers**: Instead of using a $\mathbb{R}^{m\times n}$ matrix, the input vector is discretely convolved over a smaller vector $\vec{k} \in \mathbb{R}^w$ called the *kernel*:

$$f'_{\text{conv}} : \mathbb{R}^n \to \mathbb{R}^m \tag{3.14}$$

$$f'_{\text{conv}}(\vec{p}) = \vec{b} + \vec{p} * \vec{k} \tag{3.15}$$

$$\left(f'_{\text{conv}}(\vec{p})\right)_j = \vec{b} + \sum_{i=0}^{w} \left(\vec{k}\right)_i (\vec{p})_{i+j} \,. \tag{3.16}$$

An activation function is applied to each component of f'_{conv} as in the dense case. A constant vector $\vec{b}$ is added as well.

The similarity to a dense layer is striking: Instead of every component of the input influencing every component of the output, only components of the $w/2$-nearest

neighbors are allowed to contribute to each entry in the output. This requires the input data to have a meaningful order, e.g. when the input vector represents a metric space like a time series of measurements (*locality*). Another difference is that every convolution shares the same kernel, so only the relative ordering of the input data should be relevant (*translational invariance*). This reduces the number of *weights* per layer to $(o + n)$, which allows the stacking of many convolutional layers.

Convolutions also reduces the output dimensionality compared to the input dimensionality by $w - 1$. It can be further reduced by the introduction of *striding*: Without striding, every component of the input vector contributes to w output components (except for the first and last components). With a stride of s, every input component only contributes to w/s components[4] of the output:

$$\left(f_{\text{conv}}\left(\vec{p}\right)\right)_j = \vec{b} + \sum_{i=0}^{w} \left(\vec{k}\right)_i \left(\vec{p}\right)_{i+j\cdot s} \ . \tag{3.17}$$

This means that the output dimensionality is much lower than of the input: $\dim f_{\text{conv}}\left(\vec{p}\right) = s^{-1}\left(\dim \vec{p} - (w-1)\right) + 1$.

3.2.2 Preprocessing

Although dense and convolutional layers work independently of the actual values of their weights, the initial weights, the activation functions and the numerical stability of the calculation necessitate some kind of preprocessing. There are several choices of normalizing input and target datasets, such as mean removal and variance scaling, scaling components to a range or scaling vector norms.

A good choice of preprocessing algorithm is an important aspect in creating well-performing neural networks; it has a strong influence on what information from the input data the network actually uses [37].

[1] gives a good overview of the choice and practical application of preprocessors.

3.2.3 Hyperparameters

The choice of f, also called *architecture* of the network, the optimizer, the preprocessing and the loss function are called *hyperparameters*. They differ from learnable

[4] For $w/s \in \mathbb{R} \setminus \mathbb{N}$ not all components contribute equally. All models used in this work have a $w/s \in \mathbb{N}$.

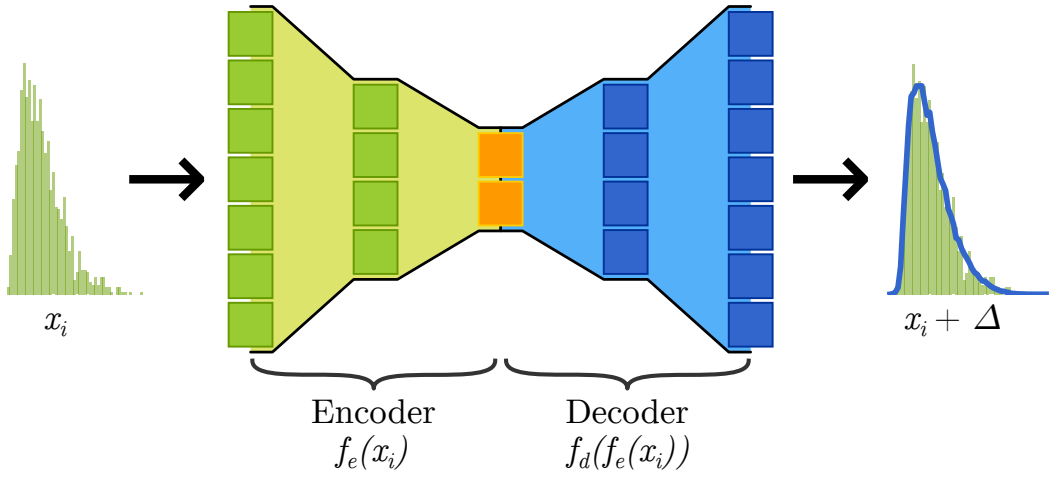

Figure 3.1: Sketch of an autoencoder used to classify a waveform (left-hand histogram). The high-dimensional input is passed though the multiple layers of the encoder (green) that map it to a low-dimensional latent space (orange). The decoder (blue) then reconstructs an approximation of the original input, using only the information contained in the latent space. The resulting output (blue line over the right-hand histogram) matches the original input x_i up to a deviation of Δ.

parameters as they are not differentiable and are not optimized during training. Nonetheless, they have an impact on the neural network's performance and must therefore be carefully chosen. Section 5.4 explains how some of these choices are verified.

3.3 Autoencoder

Neural networks are commonly used in supervised learning, but they can also be adapted to unsupervised learning: An autoencoder is a neural network with identical input and output spaces: $Y := X$ [18].

To force the network to learn a parametrization of the data, the first set of layers f_e, called the *encoder*, is designed to have a low output dimensionality:

$$\text{neural network } f = f_d \circ f_e \tag{3.18}$$

$$\text{encoder } f_e : \vec{x} \mapsto \vec{l} \tag{3.19}$$

$$\text{decoder } f_d : \vec{l} \mapsto \vec{x} + \vec{\Delta} \tag{3.20}$$

$$\text{with } \dim \vec{l} = l_d \ll \dim \vec{x} \ . \tag{3.21}$$

The *latent representation* $\vec{l}$ is then used as the input to a second set of layers, the *decoder* f_d. It maps the *latent space* back to the input space. The objective of the optimizer is then to reduce the *parametrization error* $\vec{\Delta}$.

Figure 3.1 shows this process for a waveform and visualizes the "funnel-like" structure that compels the network to learn an internal representation of the input data. This process can be thought of as a data-specific, lossy compression.
As a consequence, the autoencoder is also able to create parametrizations even when the input data is noisy or has defects. As long as the noise does not completely mask the underlying data, the original data can still be extracted. [43] This ability to denoise inputs is an important building block for outlier detection.

It is important to note that this is also a more general approach than just expressing entries of $\vec{x}$ as a function of other entries $\vec{x}_t \approx r\,(\vec{x}_a, \vec{x}_b)$, as explained in section 3.1. The autoencoder can generate new summary attributes of the data, completely re-parametrizing the input vector.

The form of representation the network learns depends primarily on the latent dimensionality l_d and the loss function. l_d limits the complexity of the description while the loss function determines whether an aberration $\vec{\Delta}$ in the reconstruction is acceptable. The latter is detailed in section 3.4.

3.3.1 Using Autoencoder for Outlier Detection

Describing autoencoders as a method of creating parametrizations makes it easy to see why they can be used for outlier detection: Given a set of many observations and a statistical model to describe them, outliers, also called *anomalies*, are defined as those observations that do not fit the model well [44].

While training an autoencoder, f_e and f_d are adjusted to describe most of the dataset well. With a sufficiently small outlier ratio, the mean loss over all events is mainly determined by the loss over the most common, normal events. As each sample $\vec{x_i}$ contributes equally to this optimization, the optimizer will prefer to decrease

the reconstruction loss of normal events over outliers. The trained autoencoder's reconstruction loss $d(f(\vec{x_i}), \vec{x_i})$ can thus be used as a predictor for outliers.

To describe how this loss looks like for outliers, it is useful to define two types of outliers: If the input data is a normal event with more-than-usual noise and defects, the autoencoder will produce a denoised version of the input. The reconstruction loss can be used as an estimate of the severity of the noise in these cases.
If the input is completely different from normal data, the fitted parameters will not describe the input at all. This leads to a high reconstruction loss as the autoencoder's output will most likely resemble a commonly occurring waveform.

However, using autoencoder for this purpose comes with a caveat: The outlier detection depends on the model's capacity: A model with a sufficiently complex architecture is eventually able to describe both normal data and outliers well, given enough training. This possibility is explored in section 9.2 and section 11.1.

3.4 Loss Function

Depending on the loss function employed to train the network, different aspects of the input data can be reconstructed. It determines which aberrations of the reconstruction are acceptable. For example, using the mean squared error of a waveform and its reconstruction leads to an equal importance of all components of $\vec{x_i}$. Reducing the value of a component that was reconstructed too large takes precedence over improving the overall shape of the waveform or guaranteeing that all components are positive.
Thus, it is of great importance to choose a loss function appropriate to the task.

The loss function used in this work is the Earth Mover's Distance.

3.4.1 Earth Mover's Distance

The earth mover's distance (EMD), also known as the first Wasserstein distance, can be used to measure the similarity between two probability distributions while taking into account the metric space they are defined on. A general definition is given in [31]; here the EMD is defined as it applies to one-dimensional discrete probability distributions.

Let p_x, q_y be two discrete probability distribution defined on the same metric space $x \in X, y \in Y$ and $X, Y \subseteq M$. This can be thought of as piles of dirt with height p_x located at x and holes to be filled with depth q_y at y. Let $z_{x \to y} \cdot ||x - y||$ be the work needed to move an amount of dirt $z_{x \to y} \in \mathbb{R}^+$ from x to y. The EMD is then

the minimal amount of work needed to distribute all earth from piles into holes so that

$$\sum_{y \in Y} z_{x \to y} = p_x \forall x \in X \qquad \text{all piles vanish} \tag{3.22}$$

$$\sum_{x \in X} z_{x \to y} = p_y \forall y \in Y \qquad \text{all holes are filled in} \tag{3.23}$$

For the case of $M = \{0, 1, \ldots, M_{\max}\} \subset \mathbb{N}$, the EMD can be calculated using the following algorithm:

$$z_0 = 0 \tag{3.24}$$

$$z_{x+1} = p_x - q_x + z_x \tag{3.25}$$

$$d_{\mathrm{EMD}}(p, q) = \sum_{x \in M} z_x \ . \tag{3.26}$$

To make this metric comparable between all distributions with a compact support, the EMD can be normed:

$$d'_{\mathrm{EMD}} : M \times M \to [0, 1] \tag{3.27}$$

$$d'_{\mathrm{EMD}}(p, q) = \frac{d_{\mathrm{EMD}}(p, q)}{M_{\max} - 1} \ . \tag{3.28}$$

The EMD is a good choice for comparing fixed-length time series like waveforms: In comparison to other losses like the mean squared error (MSE), the EMD uses the information provided by the underlying metric space: The EMD rewards the autoencoder for putting probability mass into roughly the right time bin, with the loss decreasing the closer the probability mass is to the right bin. In contrast, the MSE punishes a high probability mass in the wrong time bin indifferently with respect to the distance to the correct bin. Therefore, the MSE can only be used to verify that the input waveform matches the output waveform closely, but is useless in determining how closely they match.

Part II

Methods

4 Monte Carlo Simulations

When developing unsupervised models, their performance on real-world data is hard to quantify in a scientifically stringent way. Modifications to the model after applying it to data would bias an analysis towards a certain result, e.g. finding a specific kind of outlier.
At the same time, a method to estimate model performance is needed during development to discard poorly performing models. This is especially true for the development of neural networks, where hyperparameter choices can have non-obvious results.

As a tradeoff, a simple MC simulation for PMT waveforms is developed in this work. This waveform simulation only makes basic assumptions about the detector and the kind of outliers that should be expected, in order to introduce the least amount of bias towards finding a specific kind of outlier.

After developing and analyzing a model on this waveform MC, it is applied to a dataset from the IceCube MC simulation. This is a complex simulation modeling the whole detector. The behavior of the model on these two datasets is then compared to ensure that the lessons learned with the simpler dataset carry over to more complex cases.

4.1 Waveform MC

The autoencoders used in this work describe waveforms as seen by a single DOM. The topology of the whole event is ignored. Treating every waveform's origin as track-like should, therefore, be a good approximation. Based on this, this simulation only models the propagation of the Cherenkov light of a single relativistic, charged particle and its secondary particles, and the most important influences of the detection electronics.

4.1.1 Generating Waveforms

The simulation assumes that the photon arrival times recoded by DOMs are described by the Pandel probability density function (PDF) [38] and that every photon distribution drawn from the Pandel PDF triggers the DOM. The Pandel function

is an empirical description of waveforms that captures the most important effects described in sections 2.1 and 2.2.

The Pandel function, reparametrized for the use in IceCube, is given by [40]:

$$p\left(t, d, \eta\right) = \frac{1}{N\left(d_{\text{eff}}\right)} \frac{\tau^{-d_{\text{eff}}/\lambda}\, t^{d_{\text{eff}}/\lambda - 1}}{\Gamma\left(d_{\text{eff}}/\lambda\right)} e^{-\alpha} \tag{4.1}$$

$$\alpha := t\left(\frac{1}{\tau} + \frac{c_m}{\lambda_a}\right) + \frac{d_{\text{eff}}}{\lambda_a} \tag{4.2}$$

$$d_{\text{eff}} := d a_1 + a_2 - a_3 \cos\eta + a_4 \cos^2\eta \tag{4.3}$$

$$N(d) = e^{-d_{\text{eff}}/\lambda_a} \left(1 + \frac{\tau c_m}{\lambda_a}\right)^{-d_{\text{eff}}/\lambda} . \tag{4.4}$$

There are multiple free parameters: Scattering length λ, absorption length λ_a and parameters τ, a_1, a_2, a_3, a_4 depend on the properties of the ice and the PMT field of view; values given in [40] that match IceCube MC simulations are used here:

$$\begin{aligned}
\lambda &= 33.3\,\text{m} \\
\lambda_a &= 98\,\text{m} \\
\tau &= 557\,\text{ns} \\
a_1 &= 0.84 \\
a_2 &= 3.1\,\text{m} \\
a_3 &= 3.9\,\text{m} \\
a_4 &= 4.6\,\text{m}
\end{aligned}$$

The ice is assumed to be isotropic and of homogenous density, and the speed of light in ice is set to $c_m = \frac{c}{1.31}$. The remaining free parameters are the PMT–light source distance d and the angle η between the up direction and the vector from the PMT to the light source[1]. These parameters are uniformly sampled from the intervals $d \in [5\,\text{m}, 50\,\text{m}]$ and $\eta \in [0, \pi]$. The intervals were chosen because the maximum

[1] i.e. a light source directly below the DOM would have $\eta = \pi$ and directly above $\eta = 0$

distance of a photon emission to the nearest DOM inside the detector[2] is about 60 m.

Due to clock jitter, the arrival times of single photons are spread out over an RMS from 1.2 ns to 2.7 ns [12]. Therefore, the Pandel PDF is convolved with a Gaussian distribution with $\sigma = 2.7\,\text{ns}$:

$$p_f(t, d, \eta) = p\,(t, d, \eta) * \frac{e^{-0.5\left(t/\sigma\right)^2}}{2\pi\sigma} \tag{4.5}$$

The number of photons per waveform $n \in [10, 3000]$ is sampled from a uniform distribution. For each generated waveform, d, η, n are chosen at random and n arrival times are drawn from the $p_f\,(t)$ distribution. The drawn arrival times are histogrammed into 128 uniform bins over 420 ns to mimic the data DOMs record.

4.1.2 Generating Outliers

Outliers are generated following the same procedure as normal waveforms, but use a modified PDF in place of p_f. Two classes of outliers are employed: Gaussian re-weighting of the waveform only makes very broad assumptions of the type of outliers that real data could have. The other class are double-peak outlier, which are created by inserting an additional waveform after the main peak.

Gaussian Re-Weighting

This type of outlier is generated by overlaying p_f with a Gauss distribution:

$$p_{\text{Gauss}} = \frac{1}{1+w}\left(p_f + w\frac{1}{\sigma\sqrt{2\pi}}e^{-0.5\left(\frac{t}{420\,\text{ns}}-\mu\right)/\sigma}\right)\,. \tag{4.6}$$

The distribution has three parameters that affect how the resulting outlier is shaped: The mean μ that defines the location of the peak, the standard deviation σ that defines the width of the peak and the weight w that determines the strength of the re-weighting. They are defined relative to the histogram: An outlier with $\mu = 0$

[2] Outside DeepCore, the horizontal DOM spacing is about 125 m and the vertical spacing about 17 m [25].

Table 4.1: Overview of all parameters that influence outlier generation.

Type	Parameter	Possible Values
Gaussian	μ	$0.200, 0.350, 0.500, 0.650, 0.800$
	σ	$0.100, 0.275, 0.450, 0.625, 0.800$
	w	$0.100, 0.325, 0.550, 0.775, 1.000$
Double Peak	t_{pp}	$0.010, 0.053, 0.097, 0.140, 0.183,$ $0.227, 0.270, 0.313, 0.357, 0.400$
	w	$0.100, 0.157, 0.214, 0.271, 0.329,$ $0.388, 0.442, 0.500, 1.000, 2.000$

peaks in the first bin of the waveform while an outlier with $\mu = 1$ peaks in the last.

The values these parameters take when generating the dataset are listed in Table 4.1. Waveforms are generated for all 125 possible combinations of parameters. Some examples of the PDFs generated by this method are shown in Figure 4.1.

Double Peak

This type of outlier is generated by repeating and shifting p_f:

$$p_{\text{double}}(t) = \frac{1}{1+w}\left(p_f(t) + w \cdot p_f(t + t_{pp} \cdot 420\,\text{ns})\right) . \tag{4.7}$$

This distribution has two parameters: The shift t_{pp} that measures the separation of the peaks relative to the length of the whole waveform and the weight w that determines the size of the second peak relative to the first one.

The values these parameters take when generating the dataset are also listed in Table 4.1. Waveforms are generated for all 100 possible combinations of parameters. Some examples of the PDFs generated by this method are shown in Figure 4.2.

4.1.3 Training and Validation Data

For the purposes of this work a training dataset consisting of 5×10^6 waveforms is generated.
95 % of the data are unmodified, normal waveforms generated from the Pandel PDF as described in subsection 4.1.1. 5 % of generated waveforms are outliers: 2.5 % of

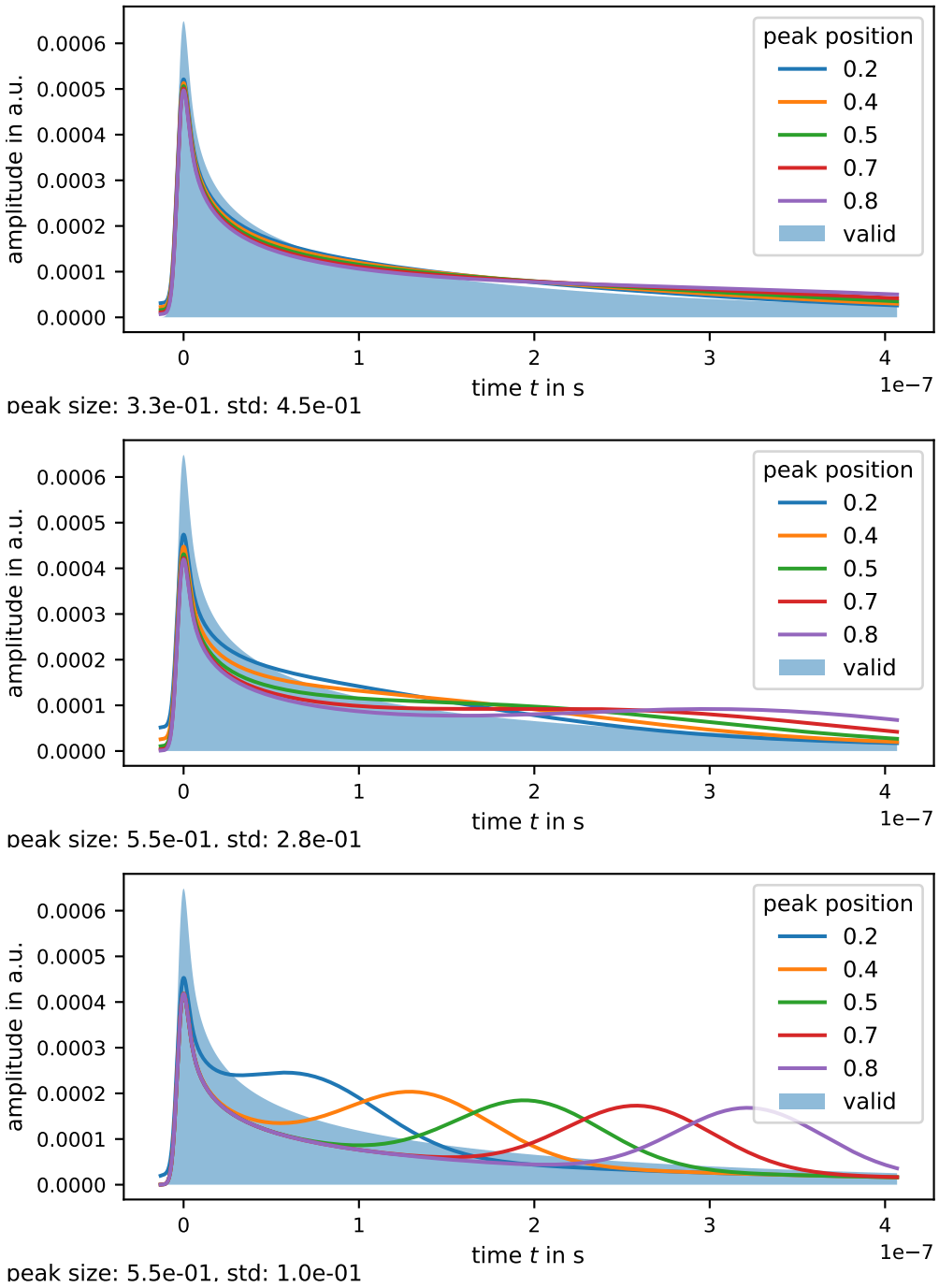

Figure 4.1: Sample PDFs of outliers, used in the Gaussian re-weighting method. Multiple peak positions are drawn in each plot while the peak size and standard deviation changes between plots. The peak positions are relative to the length of the entire waveform.

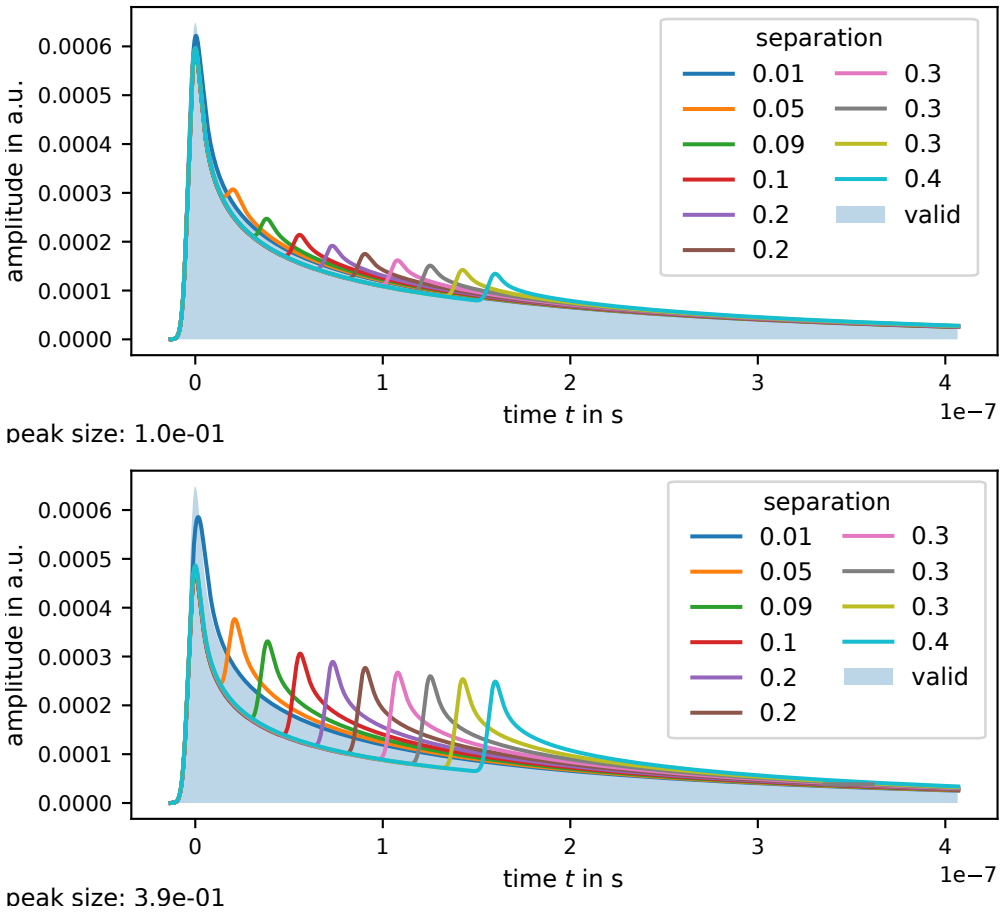

Figure 4.2: Sample PDFs of outliers, used in the double peak method. Multiple different separations are shown in each plot while the size of the peak changes between plots. The separations are relative to the entire length of the waveform.

all waveforms have two peaks and 2.5 % of all waveforms are subjected to Gaussian re-weighting, described in subsection 4.1.2.

The validation set consists of 1.0×10^4 waveforms. In this set, 50 % of all waveforms are unmodified, 25 % have two peaks and 25 % are Gaussian re-weighted.
The class ratios are different to those of the training dataset to increase the execution speed: A validation set with a high outlier ratio can contain more waveforms with unique outlier parameter combinations than a training set with a low outlier ratio. This change in class ratios does not affect the performance measurements used in this work as the area under the ROC curve (AUC) is independent of class ratios.

In situations where losses need to be compared to choose models (most notably in chapter 9), the mean training loss over the last 10 % of training samples is used. Using the validation loss would provide an unfair advantage as datasets with differing outlier ratios are not available for real-world data. Reusing parts of the training dataset instead of using a dedicated second validation set is done to increase execution speed. This is justified, as training losses are only compared when models were trained for no more than one epoch. As each training sample is only used once, the overfitting (that dedicated validation set would safeguard against) cannot appear.

4.2 IceCube MC

The IceCube MC simulation is a complex simulation chain that models a long process of events: Neutrinos are generated and propagated through the earth until they reach the detector. At the same time, non-neutrino cosmic rays and particles created in extended air showers are simulated, as muons created in these processes make up most of the background. Particles from both sources are propagated inside the detector. Different software is used to propagate photons, charged leptons and hadrons. The electronics of the DOMs and their response to photons hitting the PMTs are simulated as well. [35, Chapter 4.1 and Appendix A.1] gives a brief overview over this process.

Due to the complexity of the simulation, there is no straightforward way to identify outliers by using the MC truth. Because of this, the data is treated as unlabeled.

This work uses an existing simulation of tau neutrinos[3]. The simulation contains

[3] `/data/ana/Cscd/StartingEvents/NuGen_new/NuTau/medium_energy/IC86_flasher_p1=0.3_p2=0.0/l2/` together with the detector geometry file `/data/sim/sim-new/downloads/GCD/GeoCalibDetectorStatus_2013.56429_V1.i3.gz`. Both are accessible from the Cobalt server. See Appendix 2 for details.

many different event topologies, especially double-pulse waveforms due to so called double-bang events [14]. From a single waveform perspective, however, most waveforms should be similar to waveforms from a simulation of a normal neutrino mix. For this work, an IceTray [24] module was written that extracts calibrated waveforms from the existing simulation files. It uses an IceTray *segment* provided by Maximilian Meier.

5 Autoencoder Selection and Architecture

A great variety of outlier detection algorithms exists [16, Chapter 3]. This work focuses on using dense and convolutional neural networks inside an architecture called autoencoder for the following reasons:

Generality Dense neural networks are good general approximators; they can be used to map almost arbitrary functions [30]. Convolutional neural networks are good at generating efficient mappings between metric spaces. As a time series, waveforms are defined on a metric space.

Degrees of Freedom It is important to control the complexity of the model used, as explained in subsection 3.3.1. As the number of latent variables that parametrize an observation can be freely chosen when using autoencoders, they are an ideal candidate for limiting model complexity. At the same time, latent space dimensionality is an intuitive measure for the limits of a model: It is easy to see that a function with three parameters will always fit a data sample better than a function with two parameters.

Decomposeability The autoencoder can be decomposed into an encoder which finds a likely parametrization of an observation and a decoder which can predict observations from a parametrization. This simplifies designing the model architecture as encoder and decoder can be build and tested independently.

After pilot testing (chapter 6), one promising preprocessing algorithm, architecture and loss function were selected to be analyzed extensively. These are described here, as well as a range of alternative models and a procedure to compare them.

5.1 Preprocessing

As described in subsection 3.2.2, a good choice of preprocessing algorithm is crucial. This work uses a preprocessor that normalizes each individual sample $\vec{x}$ of the dataset so that the smallest component has zero length and the largest component has unit length:

$$\left(\vec{x}'\right)_j = \frac{\left(\vec{x}\right)_j - \min\left(\vec{x}\right)}{\max\left(\vec{x}\right) - \min\left(\vec{x}\right)} \,. \tag{5.1}$$

This puts all components in the $[0, 1]$ range where most activation functions have the highest gradient.

Many other preprocessors need summary information about the whole dataset; to calculate e.g. the overall mean and variance, the whole dataset must be preprocessed before training can begin. The preprocessor used here scales every sample of the dataset independently, which enables fast on-the-fly preprocessing during training or outlier detection.

This kind of preprocessing, however, removes information from the sample: As the autoencoder does not know the scaling factor, the total number of photons the sample contains is unaccessible to it. This is by design: The total photoelectron charge, i.e. the integral of the waveform, is the best summary statistic. Therefore, the autoencoder focuses heavily on reconstructing the correct magnitude of the input sample, which leads to a worse shape reconstruction when using the same latent space dimensionality. But the goal of this work is to do outlier detection independently of the number of photons involved, so this information is removed.

5.2 Main Model

The main model used in this work, an autoencoder called `conv` with a latent space dimensionality of $l_d = 3$, is given in Table 5.1. The following paragraphs explain the function and purpose of each layer of the model. Figure 5.1 shows a schematic view of the autoencoder.

Encoder

Input The input waveform consists of a 128-dimensional vector representing 3.3 ns windows of photoelectron counts. It is preprocessed as described in the previous section.

Table 5.1: Structure of the convolutional autoencoder mainly used in this work.

	Layer	Output Size	Type	Parameters	Activation
		128	Input		max height to 1
Encoder	1	63	Convolution	kernel=4, stride=2	ReLU
	2	3	Dense	with bias	ReLU
Decoder	3	256	Dense	with bias	ReLU
	4	128	Dense	with bias	Sigmoid

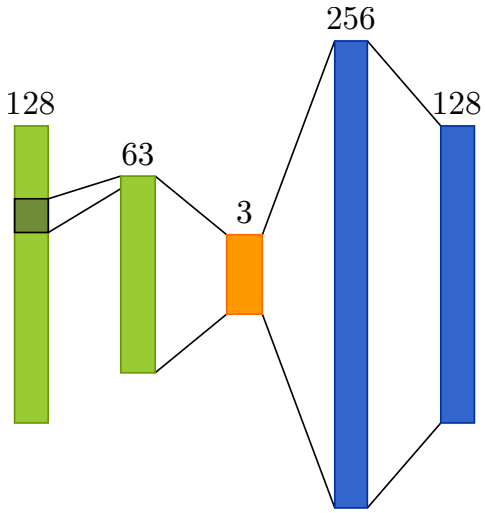

Figure 5.1: Visualization of the convolutional autoencoder mainly used in this work. The numbers at the top signify the output size of each layer. The encoder is rendered in green and the decoder in blue. The input, entering from the left-hand site, passes through a convolutional layer and a dense layer, transforming it into a latent representation (orange box). This representation is mapped to the output space by passing through two other dense layers.

Convolutional layer The time convolution is used to compress and smooth the input data. With a kernel size of four and a stride of two, multiple neighboring bins are combined into one bin.

Many neural network use convolutional layers with multiple channels so each kernel can specialize to detect certain patterns. This is especially advantageous when dealing with large vectors (e.g. images). Here, only one channel is used in order to smooth the data instead of identifying patterns. Increasing the kernel size does not increase the network's performance as far-away components do not correlate strongly with closer ones. Neither does increasing the stride. This forces the network to combine more bins into one loosing too much information in the process.

Dense layer A fully-connected layer is used to map the convolution's output to the latent space. Together with the convolution, this creates a neural network with one hidden unit. Input and latent size are determined by the parameters of the convolution and the target latent dimensionality respectively.

Decoder

Dense layers The decoder consists of a fully-connected neural network with two dense layers. The first layer's large output size is chosen in order to achieve a high reconstruction power.
This should enable the decoder to mimic a wide range of continuos functions, mapping the latent space to the waveform space without needing deeper network architectures [30]. This should also mean that the network's overall performance is not limited by the decoder but rather the encoder.

5.3 Alternative Models

The model described above is tested against alternative architectures. These consist of four dense and one convolutional architecture. To easily identify the different types of dense networks, the following naming scheme is adopted: *Deep* networks have five hidden units (six different f_{dense} are applied to the input) while *shallow* ones have three (four different f_{dense}). *Short* networks have smaller unit sizes (the codomain of f_{dense} is low-dimensional) while *tall* ones have bigger units (f_{dense} with bigger codomains).

The different architectures are listed in Tables 5.2 to 5.6. Each dense network architecture is tested with seven different loss functions for the encoder and seven loss functions for the decoder. This is signified by the $*$ symbol in the tables. Every architecture is also tested with six different latent space dimensionalities. The possible values for loss functions and latent space dimensionalities of these are specified in Table 5.7.

Table 5.2: Alternative model: 'tall, shallow'.

	Unit	Output Size	Type	Parameters	Activation
		128	Input		max height to 1
Encoder	1	256	Dense	with bias	*
	2	1 to 6	Dense	with bias	*
Decoder	3	256	Dense	with bias	*
	4	128	Dense	with bias	*

Table 5.3: Alternative model: 'short, shallow'.

	Layer	Output Size	Type	Parameters	Activation
		128	Input		max height to 1
Encoder	1	20	Dense	with bias	*
	2	1 to 6	Dense	with bias	*
Decoder	3	20	Dense	with bias	*
	4	128	Dense	with bias	*

Table 5.4: Alternative model: 'short, deep'

	Layer	Output Size	Type	Parameters	Activation
		128	Input		max height to 1
Encoder	1	80	Dense	with bias	*
	2	33	Dense	with bias	*
	3	1 to 6	Dense	with bias	*
Decoder	4	33	Dense	with bias	*
	5	80	Dense	with bias	*
	6	128	Dense	with bias	*

Table 5.5: Alternative model: 'tall, deep'

	Layer	Output Size	Type	Parameters	Activation
		128	Input		max height to 1
Encoder	1	200	Dense	with bias	*
	2	100	Dense	with bias	*
	3	1 to 6	Dense	with bias	*
Decoder	4	50	Dense	with bias	*
	5	200	Dense	with bias	*
	6	128	Dense	with bias	*

Table 5.6: Alternative model: 'conv' with different latent space dimensionality.

	Layer	Output Size	Type	Parameters	Activation
		128	Input		max height to 1
Encoder	1	63	Convolution	kernel=4, stride=2	ReLU
	2	$1, 2, 4, 5$ or 6	Dense	with bias	ReLU
Decoder	3	256	Dense	with bias	ReLU
	4	128	Dense	with bias	Sigmoid

5.4 Hyperparameter Optimization

The waveform MC dataset, described in chapter 4, allows to estimate the outlier detection performance of the different models. This allows the usage of a hyperparameter optimization (HPO) to automatically find good alternative models.

5.4.1 Configuration

The HPO is run over a big space of possible hyperparameters, using the state-of-the-art *Hyperband* [32] algorithm. A overview of the hyperparameter space is given in Table 5.7. A practical guide to the optimizer implementation [33] used can be found in the `tune` reference manual [42].

1.0×10^4 hyperparameter configurations are sampled and used to train the same number of models on the waveform MC dataset. This sampling is done by selecting a random configuration of hyperparameters for each model. The EMD reconstruction loss of a waveform is used as predictor of the waveform being an outlier. The model performance[1] is then determined by the area under the ROC curve (AUC) on 1.0×10^4 validation waveforms.
Hyperband stops training on poor-performing models early in order to increase configuration space coverage. During each training iterations, each remaining model is trained on 3.2×10^4 waveforms. The best-performing models are trained for a maximum of 100 iterations.

Parameters of the HPO not explicitly mentioned use the default values given in [42].

[1] Performance measurements on generated datasets strongly depend on the chosen outlier parameters. See subsection 4.1.2 for the parameters of the waveform MC used here.

Table 5.7: Overview of the hyperparameter space. For the definition of the loss and activation functions, see [41]. The special loss functions EMD and MKLD are explained in section 3.4 and Appendix section 3, respectively. Note that if the architecture is `conv` the activation functions are not chosen from this table. Instead, they use the activations given in Table 5.6.

Hyperparameter	Possible Values
architecture	tall, shallow; short, shallow; short, deep; tall, deep; conv
latent dimensionality	$1, 2, 3, 4, 5, 6$
encoder activation function	ReLU, Leaky ReLU, SELU, ELU, Tanh, Hardshrink, Sigmoid
decoder activation function	ReLU, Leaky ReLU, SELU, ELU, Tanh, Hardshrink, Sigmoid
training loss	EMD, mean-squared error (MSE), mean absolute error (L1), cosine similarity, unenforced MKLD, enforced MKLD

5.4.2 Unoptimized Hyperparameters

All models use the optimizer Adam, which is a good general-purpose optimizer [26]. The fact that the optimizer's hyperparameters require no tuning to achieve good results simplifies overall hyperparameter optimization. Different loss function used during training are explored. The EMD as reconstruction loss is always used as an outlier predictor on the validation set.

Every training uses the same batch size of 320. The outlier separation performance is insensitive to the exact training batch size as long as models are trained for the same length of time. Changing the batch size by about a factor of two or more, however, changes the number of waveforms needed to achieve the same performance.

6 Methodology

This chapter outlines the author's research process. It explains which elements of the research are described in detail and which were excluded, as well as the rationale behind these choices. For a description of the methods used in the analysis, see chapter 5 and chapter 4 instead.

6.1 Preparation

After a review of the relevant books and papers, a simple version of the waveform MC simulation was created that could only generate normal waveforms. The author used this simulation to familiarize himself with different methods for outlier detection, and specifically with autoencoders and the impact of loss functions. Then, a filter chain for the IceCube MC data was developed to extract more realistic waveforms.

6.2 Pilot Testing

Using the IceCube MC data, several encoder and decoder designs to classify waveforms were tested separately, as well as whole autoencoder designs. Different loss functions and preprocessing methods were investigated, and the feasibility of outlier detection with autoencoders on Icecube waveforms verified.

This was accompanied by the examination of different software libraries and programming concepts: A switch from the `tensorflow` to the `pytorch` framework was made on the basis of usability. Different hyperparameter optimization algorithms were explored. Multiple possible ways of storing and accessing waveform data were studied. Finally, the possibility of using autoencoders to classify whole IceCube events was explored. Many designs and approaches were investigated: Using 4D convolutions to process whole events; reducing data by combining less important dimensions; engineering summary features such as mapping spacetime coordinates to 1D distances using Hilbert curves; and using parametrization of waveform autoencoders.

However, this task proved to be very intricate, especially because of the sparsity of the event data and the resulting memory requirements when using neural networks.

These approaches required too much time for optimizing code and were difficult to analyze with the scientific rigor necessary.
It was therefore decided to focus on the waveform autoencoder developed during this pilot testing.

6.3 Main Analysis

At the start of the main analysis, a research plan was drafted. It described in advance which properties of the chosen model should be investigated and which methods would be used to accomplish this.

To investigate and compare the `conv` autoencoder with $l_d = 3$, the waveform MC simulation was revisited and improved to enable the generation of outlier waveforms. The performance was also increased in order to generate bigger datasets. The model could then be compared to alternative models in a reproducible way. This also enabled the close examination of the model's behavior and performance under different conditions. The chronology of this analysis is reflected in the order used in Part III.

Finally, the developed model was applied to data from IceCube MC simulations to verify that the model's properties found on waveform MC data would carry over to more realistic data.

Part III

Results

7 Overview

Chapter 8 describes how the **hyperparameter optimization** is used to verify that the selected model is well-performing in **comparison to alternative models**.

The outlier detection performance depends on random variables and the amount of unsupervised training.
Chapter 9, consequently, analyzes the **stability** of the model. A method for blind selection of a well-performing model from an ensemble of trained neural networks is developed to reduce model variability.
Section 9.2 determines **when to stop** the autoencoder's **training** in order to achieve good model performance as well as determining the dependence on the latent dimensionality.

Chapter 10 presents the most important **characteristics of the neural network** chosen from the ensemble such as training loss curve, performance and latent space.

Some additional properties of the model are also explored:
Section 11.1 examines the impact of different **outlier ratios** on the model's outlier detection performance.
Section 11.2 looks at the model's ability to identify **double-peak waveforms** as outliers.

After developing the model and analyzing its properties on the waveform MC, the model is **applied to IceCube MC** waveforms.
Chapter 12 determines whether the model behaves similarly on both types of waveforms. Additionally, training loss curves are presented and the model's stability is estimated.

7.1 Measurement Uncertainties

Uncertainties are only given for values that have high uncertainties. Values without explicitly stated uncertainties are given so that the order of magnitude of the uncertainty is lower than the number of significant digits given. For example, an AUC that is given as AUC $= 0.61$ is guaranteed to have an uncertainty of $\hat{\sigma} \leq 0.005$.

Uncertainties are given as unbiased estimated standard deviations (1σ) for singular values or unbiased estimated standard deviations of the mean for mean values.

All Pearson correlations and Kolmogorov–Smirnov tests in this work are either significant with a p-value of $p < 0.05$ or explicitly state the p-values.

8 Comparison of Alternative Architecture

The hyperparameter optimization explores a large space of different autoencoder. The model and HPO configuration is described in section 5.4.

Figure 8.1 summarizes the results of the hyperparameter optimization. Many models are well-performing ($\text{AUC}_{\text{median}} = 0.79$).

The best-performing models (AUC > 0.80) tend to have a latent dimensionality of $d_l = 2$. The waveform MC used to generate the training data implements the Pandel function, which has two free parameters that affect the shape of waveforms. Therefore, it is expected that models perform better when using latent spaces with the same number of dimensions.
This is, however, not a requirement; as seen in Figure 8.2, there are well-performing models with $d_l \neq 2$. But with increasing model complexity it gets more likely that the model can also describe outlier waveforms well, which leads to a lower AUC.
There are also a very small number of well-performing models with $d_l = 1$. The best ten models are given in Table 8.1.

The exact properties of the best models strongly depend on the exact training data used. As the waveform MC used to generate these results differs from the real data, it is more sensible to look at mean performances of models with good AUCs, which depend less strongly on the detailed parameter choice of the waveform MC.

When considering all models with AUC > 0.8, the differences in model performances are small, see Table 8.2: The maximum mean absolute difference between all tested architectures and latent dimensions is $\Delta\text{AUC} = 0.03$. That implies that neither the architecture nor the latent dimensionality of the autoencoder limit its ability to find outliers. It has to be noted, however, that only the `conv` architecture achieves AUC > 0.875 (see Figure 8.3). Whether this small AUC advantage will carry over to real data is questionable.

[1] Subsequent experiments do not use AUCs from this table and no conclusions influencing the remainder of the analysis are drawn. The values are the high tail of a unknown, highly non-gaussian (cf Figure 8.1) distribution for which errors are difficult to estimate.

Table 8.1: The ten best-performing models after hyperparameter optimization, sorted by AUC. The AUC depends strongly on the number, kind and size of hidden layers (architecture) and the dimensionality of the latent space d_l. The exact configurations of the five mentioned architectures are explained in section 5.2 and section 5.3. No uncertainties were estimated for the AUCs[1].

architecture	AUC	d_l	encoder activation	decoder activation	loss
conv	0.888	2	ReLU	ReLU, Sigmoid	EMD
conv	0.886	4	ReLU	ReLU, Sigmoid	EMD
conv	0.875	1	ReLU	ReLU, Sigmoid	EMD
tall, shallow	0.871	2	SELU	SELU	EMD
tall, deep	0.871	5	Sigmoid	SELU	EMD
short, shallow	0.870	2	SELU	Tanh	EMD
tall, deep	0.869	2	Sigmoid	Tanh	EMD
tall, shallow	0.868	3	Sigmoid	Tanh	EMD
conv	0.866	3	ReLU	ReLU, Sigmoid	EMD
conv	0.863	5	ReLU	ReLU, Sigmoid	EMD

Table 8.2: Mean AUC of models by architecture (horizontal) and latent space dimensionality (vertical). The mean is calculated over all models with AUC > 0.8 (1710 models). The mean standard error of the mean of all AUCs in the table is 0.006.

	1	2	3	4	5	6
conv	0.84	0.85	0.84	0.84	0.82	0.83
short, deep	-	0.83	0.83	0.83	0.83	0.83
short, shallow	-	0.84	0.83	0.84	0.83	0.83
tall, deep	-	0.83	0.83	0.83	0.85	0.83
tall, shallow	-	0.83	0.83	0.83	0.82	0.83

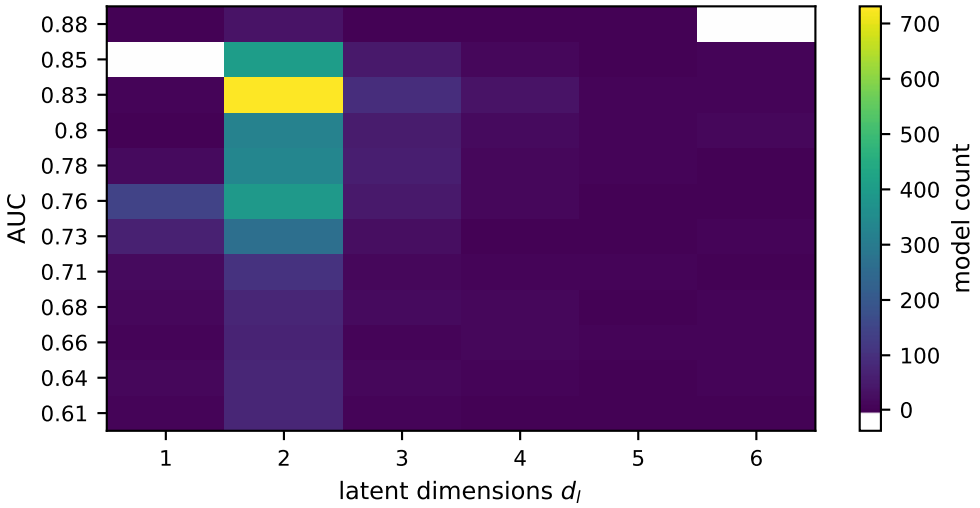

Figure 8.1: Histogram of model outlier separation performance (AUC) versus model capacity (d_l) of all models used in the hyperparameter optimization. White bins indicate that no model was found for these configurations. There are models with AUC < 0.61 which are not shown in this plot.

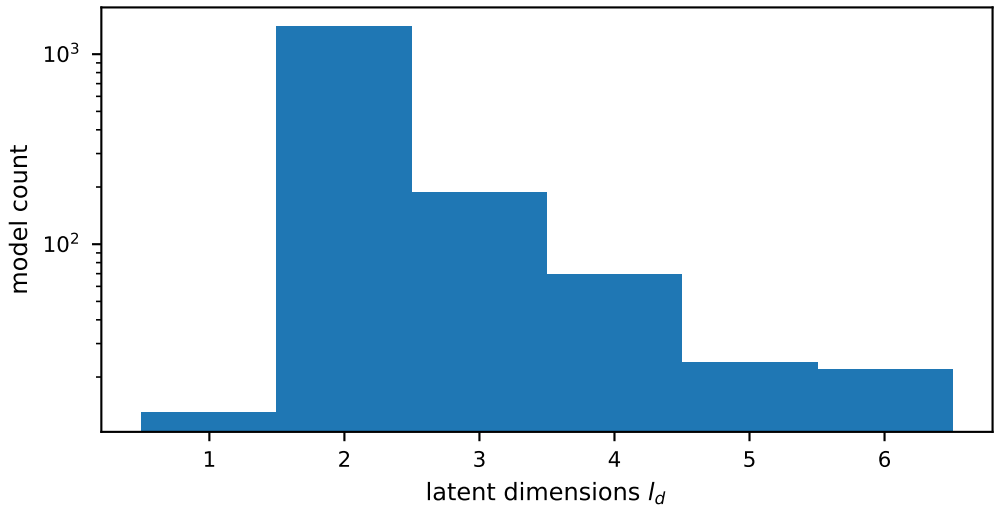

Figure 8.2: Distribution of the number of latent dimensions among models with an AUC > 0.8.

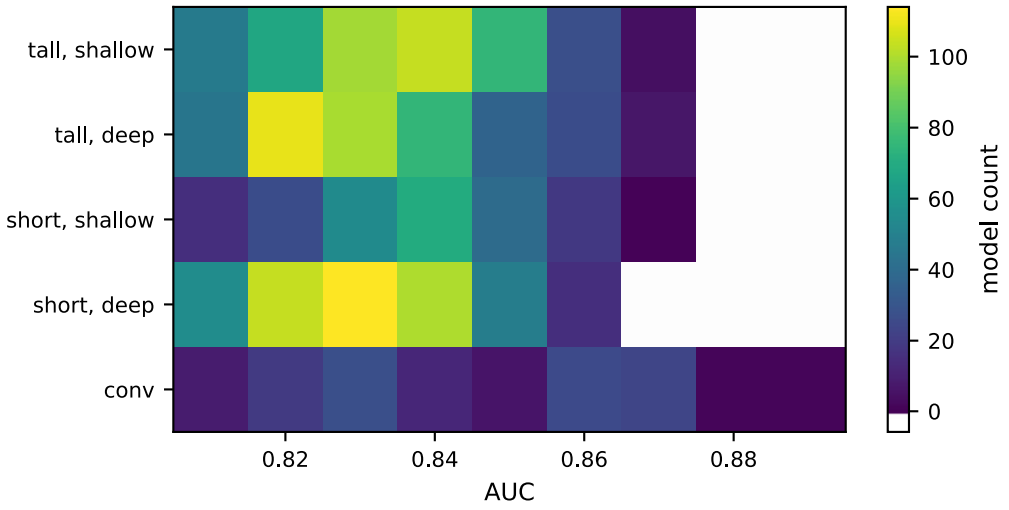

Figure 8.3: Histogram of model outlier detection performance (AUC) depending on the model architecture. A white bin indicates that no model with this AUC was found for the configuration.

8.1 Chosen Model

The `conv` architecture with $d_l = 3$ is used for the remainder of this analysis. Allowing for an additional latent dimension enables the autoencoder to model more complicated, real-world waveforms without sacrificing much performance. Its exact configuration is given in Table 5.1. The same model with $d_l \in \{1, 2, 4\}$ performs slightly better, but these differences are expected to be negligible on real data.

9 Stability

Hyperparameter optimization suggested that training two models with identical configurations on the same data does not necessarily lead to the same AUC[1]. For this reason, the stability of the training process is explored and a method to select well-performing models is developed.

9.1 Ensemble of Models

To estimate the model stability, an ensemble of 8.4×10^3 clones of the model chosen in section 8.1 is trained on 3.2×10^6 waveforms each.

There is a 7.2 % chance that training does not converge within 3.2×10^6 training waveforms. This leads to an outlier detection little better than guessing (AUC < 0.6) in these cases. The underperforming models are easily identified by their high overall training loss, indicated in orange in Figure 9.1.

When training converges, there is no simple relation between the classification power and the training loss, as shown by Figure 9.3. The loss is not decisively correlated with the AUC (Pearson correlation of $\rho_{\mathrm{AUC,loss}} = -0.04$). The most likely performance after training is $\mathrm{AUC}_{\mathrm{mode}} = 0.87$ while the chance of getting a model with a worse performance is $p_{\mathrm{worse}} = 0.68$ (cf Figure 9.1).

Nonetheless, the ensemble can still be used to mitigate the risk of using a suboptimal model for outlier classification. Picking the model with median training loss from the ensemble reduces the chance of picking worse models while keeping the most likely AUC about the same (Figure 9.2).
Using an ensemble of 350 models improves the chance of selecting a model worse than $\mathrm{AUC}'_{\mathrm{mode}} = 0.87$ to $p'_{\mathrm{worse}} = 0.30$. However, there is still a high variability when using an ensemble of this size; p'_{worse} is drawn from a distribution with an estimated standard deviation of $\hat{\sigma} = 0.13$. Notwithstanding the above, picking the model with median training loss is still an improvement over using just one model in almost all ($> 95\,\%$) cases.

[1] This is due to the random weight initialization and the random order of training samples. This randomness during training ensures that the order of training samples does not introduce bias and improves results and training speed [4].

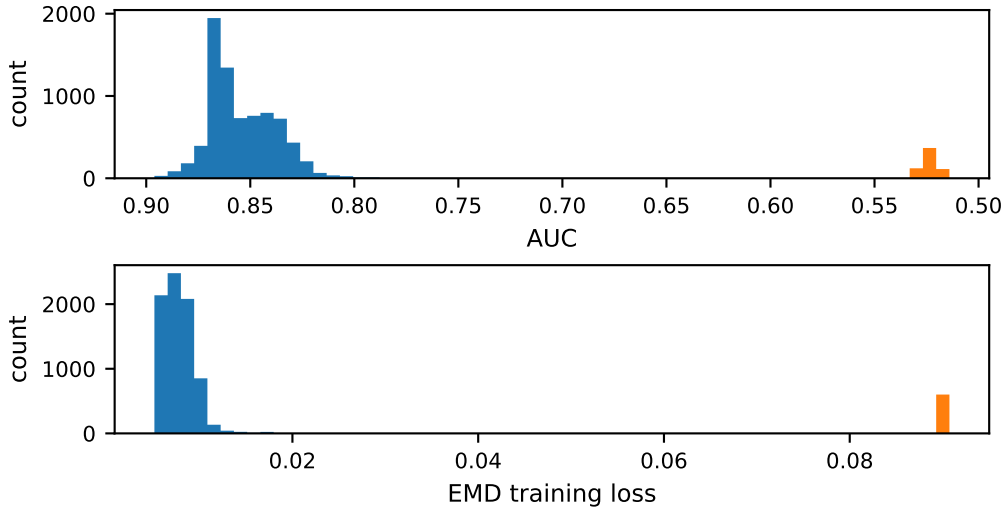

Figure 9.1: Histogram of outlier detection performance (AUC) and mean training loss. Corresponding colors indicate corresponding model populations, i.e. all models with AUC $<$ 0.6 have $\text{loss}_{\text{EMD}} > 0.08$ (orange) and vice versa (blue).

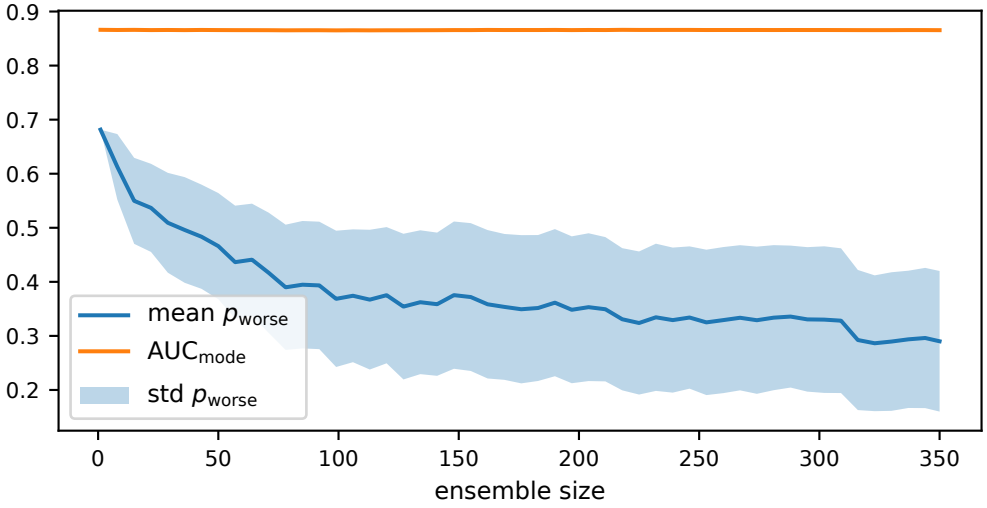

Figure 9.2: Probability of selecting a model worse than the most likely AUC_{mode} using the median training loss when considering an ensemble of trained models. Depicted are the mean probability (solid line) and the estimated standard deviation of the distribution from which the probability is drawn (shaded area). For each probability 1×10^3 ensembles are used to estimate the mean and the standard deviation.

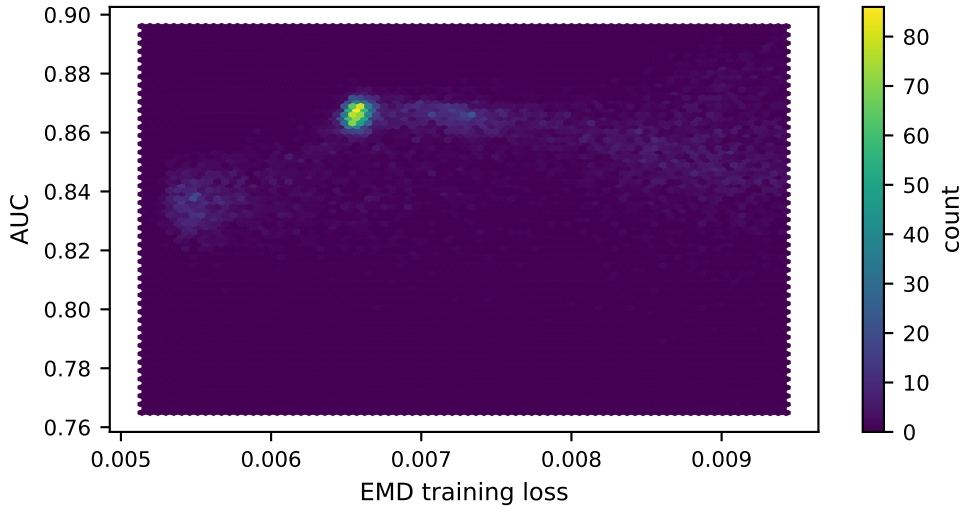

Figure 9.3: Histogram of mean model loss after training and the outlier detection performance (AUC). The plot excludes the highest 20.0 % of models with high loss to show the region of main interest in greater detail.

9.2 Training Time and Network Capacity

An important question when training neural networks is when training converges and how this depends on different hyperparameters. It is also important to quantify if and when overtraining happens.

The `conv` model achieves the best outlier detection performance after training on about 1×10^6 waveforms (see Figure 9.4). This happens independently of the chosen latent dimensionality.

Training beyond this point, the change in performance is compatible with zero for models with few ($d_l = 2, 3$) latent space dimensions, as the correlation (see Table 9.1) between the number of training waveforms and AUC are not significant. For models with higher ($d_l = 4, 6$) latent space dimensionality, there are indications that the AUC decreases when training for longer times. There is a significant inverse correlation between training length and AUC.

This can be understood by looking at the loss histograms of models trained on different numbers n of waveforms (Figure 9.5): After training for 6.0×10^5, the reconstruction loss of waveforms that the model can represent well does not change much. This is indicated by the peak position and height shifting only slowly when increasing n. Only the width of the long-tailed distribution decreases as the model

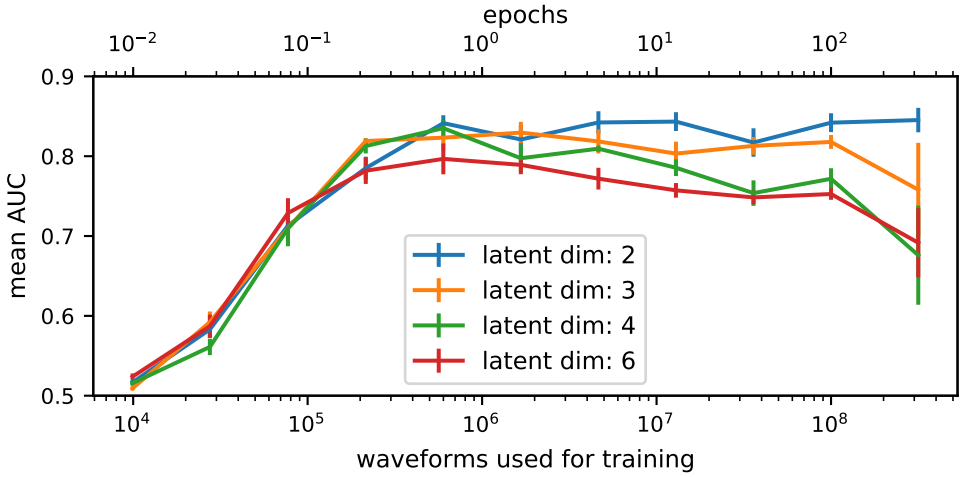

Figure 9.4: Correlation between training length and model performance on validation data. The error bars represent the standard deviation of the mean. Each point represents the mean performance, calculated on 2.7×10^1 independently trained models.

learns to reconstruct the few remaining high-loss waveforms better.

The sudden drop in performance for $n > 1 \times 10^8$ is most likely due to overfitting: The waveform MC dataset used to train models contains 5×10^6 unique waveforms; therefore, when training for longer, the model is trained on each waveform multiple times (i.e. training for multiple epochs). In this setting, overfitting can occur and is characterized by an increasing loss on the validation data. Independently of latent space dimensionality, no signs of overfitting can be detected when training for up to 100 epochs (see Figure 9.6). Only when training for more than 100 epochs, overfitting becomes a problem for models with $d_l > 2$.

In conclusion, the model does not need additional regularization as the training should be stopped around $n \approx 1 \times 10^6$ for best performance, long before overfitting becomes an issue. It was also shown l_d limits the capacity of the model.

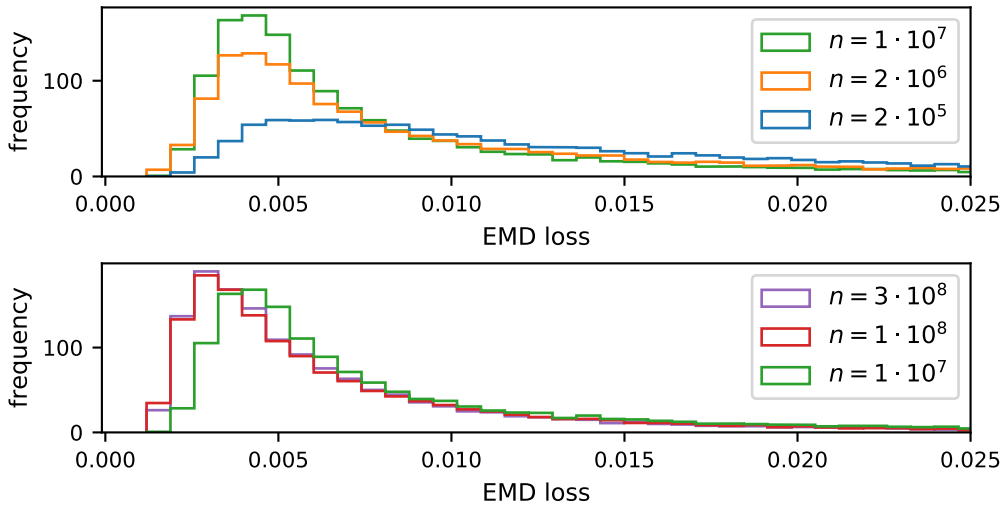

Figure 9.5: Histograms of validation loss distributions for models trained on different number of waveforms n. The `conv` models with $d_l = 3$ were selected from ensembles of 2.7×10^1 models for having the median training loss. All histograms are normed to unit area. The distributions have long tails for high losses, which are not shown.

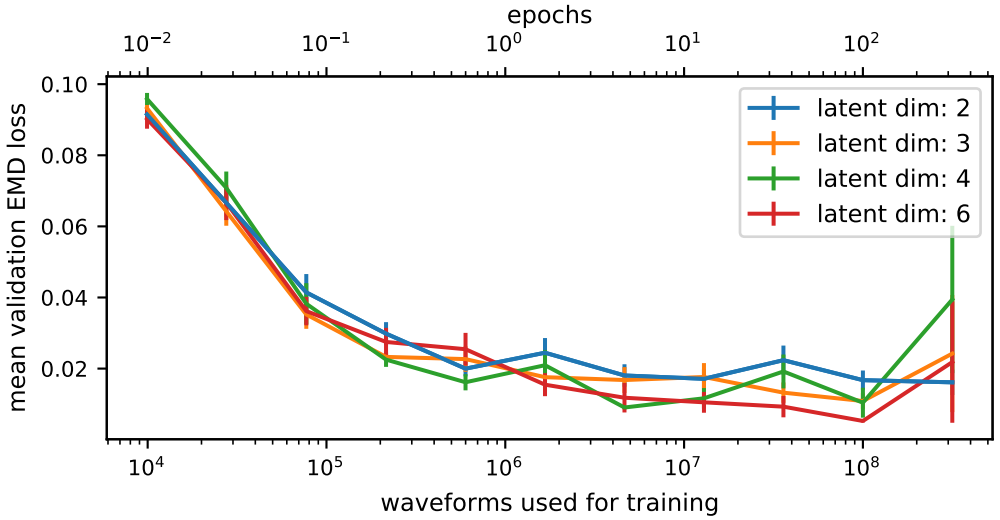

Figure 9.6: Correlation between training length and validation loss. It is important to note that the validation sample contains an outlier ratio of 50 %. Therefore, the validation loss is highly sensitive to overtraining on the outliers. The error bars represent the standard deviation of the mean. Each point represents the mean validation loss, calculated on 2.7×10^1 independently trained models.

Table 9.1: Correlations between the training length and the resulting AUC for training lengths exceeding 6.0×10^5 waveforms. The p-values represent the chance that random uncorrelated data would have the same or a lower r. The correlations for lower $d_l = 2, 3$ are small and compatible with the uncorrelated case ($p \gg 0.05$).

latent dimension d_l	Pearson correlation r	p-value
2	0.039	0.544
3	−0.093	0.167
4	−0.285	0.000
6	−0.226	0.001

10 Neural Network Training

Using the method determined in section 9.1 to identify stable models, the `conv` autoencoder with $d_l = 3$ with median validation loss, chosen from an ensemble of 8.4×10^3 models, is analyzed in detail. Although the exact quantitative results vary within the ensemble, the qualitative results in this chapter apply to most models of the ensemble (for exceptions, see section 9.1).

10.1 Loss and Separation Power

During training, the loss is minimized in a way commonplace in neural network training (see Figure 10.3) and converges after about 1×10^6 waveforms. Extending the training time leads only to minor loss improvements and degrades outlier detection performance (see Figure 9.6, see also section 9.2). The ROC curve, calculated on validation data after training, has a shape commonly seen when using neural networks (see Figure 10.2).

Figure 10.1 shows how the reconstruction loss of a waveform can be used to predict outliers. Normal waveforms are approximated better than outliers: The normal loss distribution peaks sharply at small losses, while the outlier loss distributions peak at higher losses and have long tails.

The peak of the loss for the validation sample, reweighted to 95 % normal and 5 % outlier waveforms, is at $\mu_{\text{peak}} = 3.9 \times 10^{-3}$, the standard deviation of the loss is $\sigma = 7.4 \times 10^{-3}$.

10.2 Latent Space

The `conv` autoencoder with $d_l = 3$ learns to represent waveforms as three-dimensional vectors. These parametrizations of normal and outlier waveforms are distributed roughly similar in latent space: There are two main axes visible in Figure 10.4 around which the parametrizations cluster. However, in comparison the normal waveforms, outliers scatter more broadly around and extend further beyond the axes. This meets the expectation in subsection 3.3.1 that the autoencoder tries to reconstruct the underlying waveform, even if noise is superimposed on the waveform.

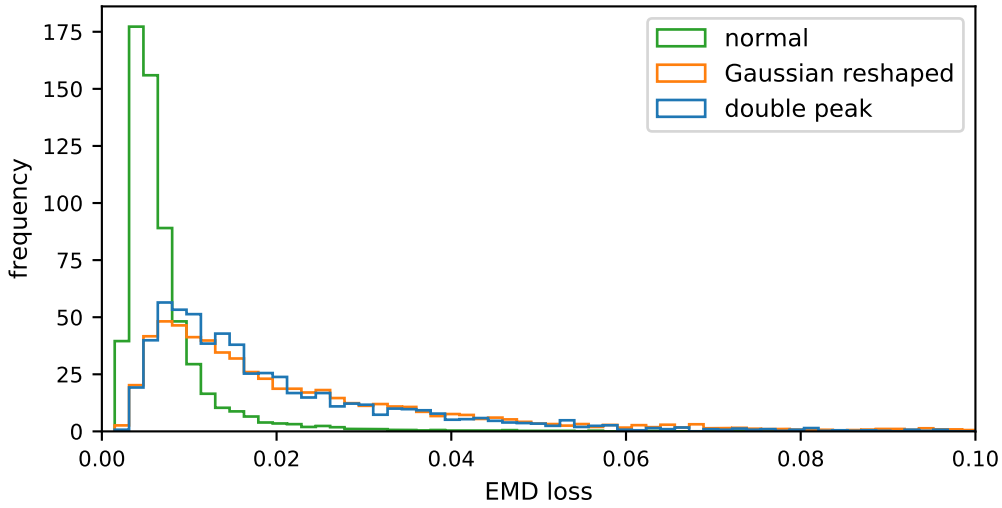

Figure 10.1: Typical loss distribution depending on outlier type for a `conv` model with $d_l = 3$. The distribution is obtained by histogramizing the reconstruction loss of the validation sample after training. All histograms are normed to unit area.

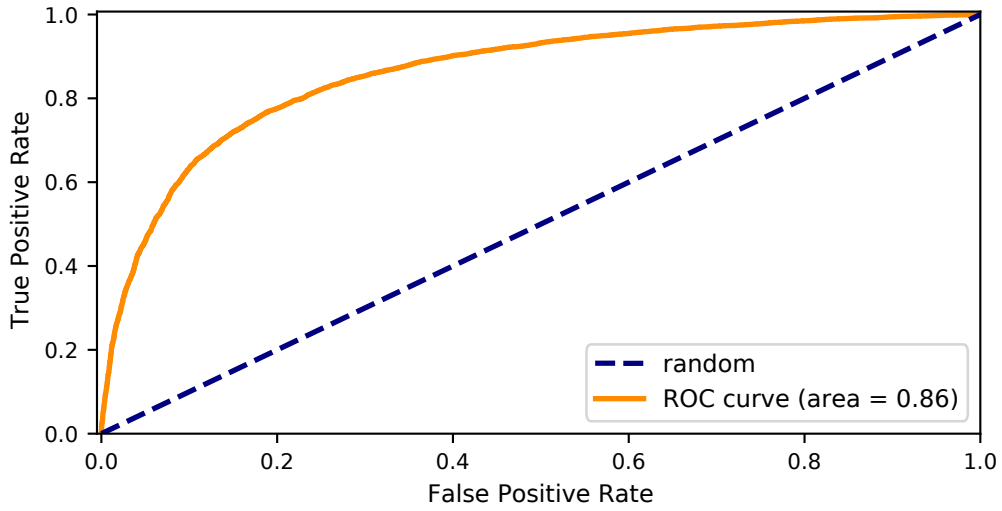

Figure 10.2: Typical ROC curve after training a `conv` model with $d_l = 3$. The ROC curve is calculated on the validation set using the EMD loss of each waveform as a class predictor. The "positive" class consist of outlier waveforms, the "negative" class of normal waveforms. The dashed blue line indicates the ROC curve for a model randomly guessing the waveform's class.

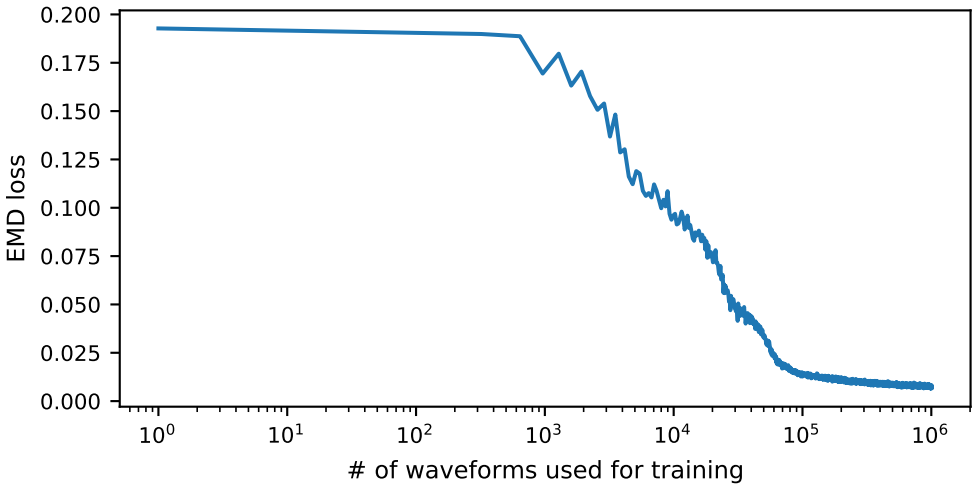

Figure 10.3: Typical loss progression during the training of a `conv` model with $d_l = 3$. The mean EMD loss of each batch of training data is shown. There is no need for an explicit validation set as the waveforms in each batch are unique.

The shape of the parametrization distributions differ from model to model as the autoencoder does not restrain the latent space strongly as e.g. a variational autoencoder would. Nonetheless, most models show one or two central axes around which parametrizations cluster as the underlying Pandel function has two free parameters.

Figure 10.5 shows the three one-dimensional distributions of the latent space after a principal component analysis. The distributions for outlier and normal waveform parametrizations significantly differ in each dimension according to a two-sided Kolmogorov–Smirnov test. Using just one coordinate along any dimension as an outlier predictor leads to an AUC ≤ 0.56. This demonstrates that outliers are not easily identified by looking at the one-dimensional projections alone; the actual multi-dimensional latent space has to be considered.

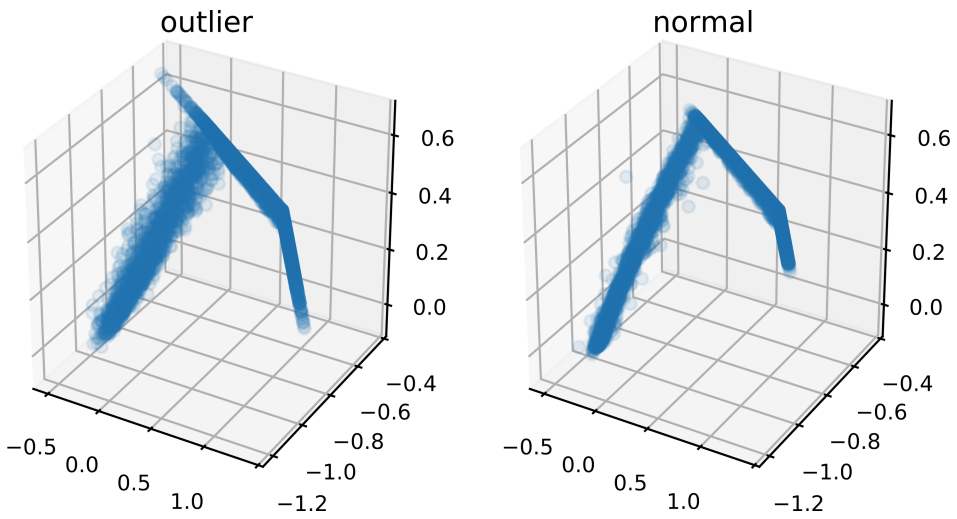

Figure 10.4: Visualization of the latent space for a `conv` model with median training loss, selected from an ensemble of 8.4×10^3 models and trained on 1×10^6 waveforms. Each point represents the latent parametrization of one waveform in the validation set. Normal waveforms (right) cluster tightly around two central axes while outlier waveforms (left) scatter in a broader region from and extend further along the axes.

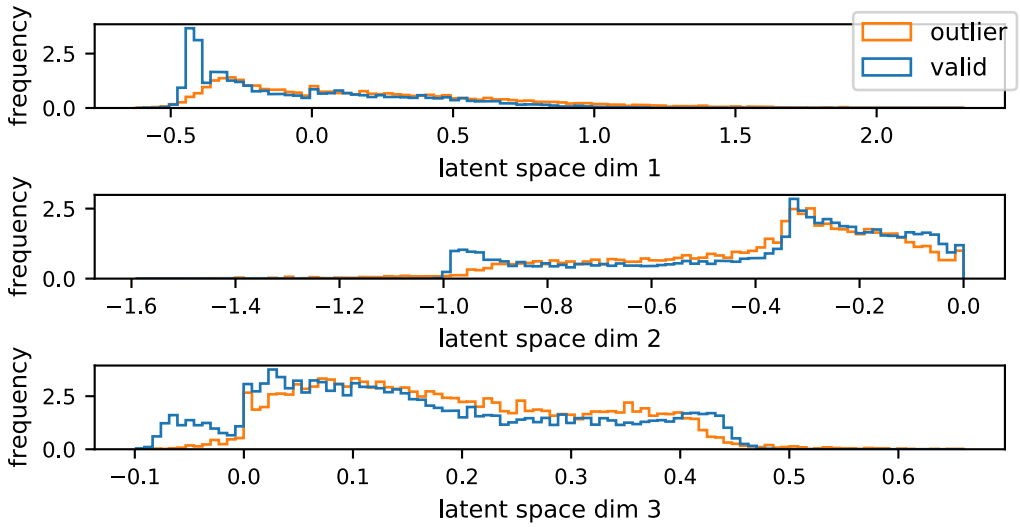

Figure 10.5: Histograms of waveform parametrizations in the latent space of a `conv` model with median training loss, selected from an ensemble of 8.4×10^3 models and trained on 1×10^6 waveforms. The latent space is rotated so that the first and second dimensions maximize variance (principal component analysis). The histograms for normal waveforms (blue) and outliers (orange) are normed to unit area.

11 Model Properties

Having verified that the training process is stable and the latent space behaves as expected, two additional properties of the model can be investigated:
The impact of the number of outliers in the training sample on the outlier detection performance; and the model's ability to identify a physically important kind of outlier.

11.1 Impact of Outlier Ratio on Training

One of the initial assumptions was that outlier waveforms are reconstructed worse than normal waveforms because only a small ratio of the training data consists of outliers. The autoencoder should prioritize reconstructing the most common waveforms well to minimize overall loss.

This assumption is tested here by training two ensembles of autoencoders with identical hyperparameters: The first ensemble is trained on waveform MC data that contains 5 % outliers while the second ensemble's training data contains no outliers.

Both ensembles perform similarly, with identical mean AUCs and mean validation losses. The AUC distributions also look similar (Figure 11.1). Differences between these distributions are, however, significant: The null hypothesis that the AUCs of both ensemble are drawn from the same distribution is rejected by a two-sided Kolmogorov–Smirnov test (K-S statistic $D = 1.0 \times 10^{-1}$ with 8.4×10^3 models trained on the dataset with outliers and 7.6×10^3 models trained on dataset without outliers).

11.2 Double-Peak Separation

Double-peak waveforms, which are one possible predictor for finding tau neutrinos [10], are outliers with two free parameters: The intensity of the second peak and the time separation between the peaks.

To test how sensitive the trained autoencoder is to the time separation, multiple

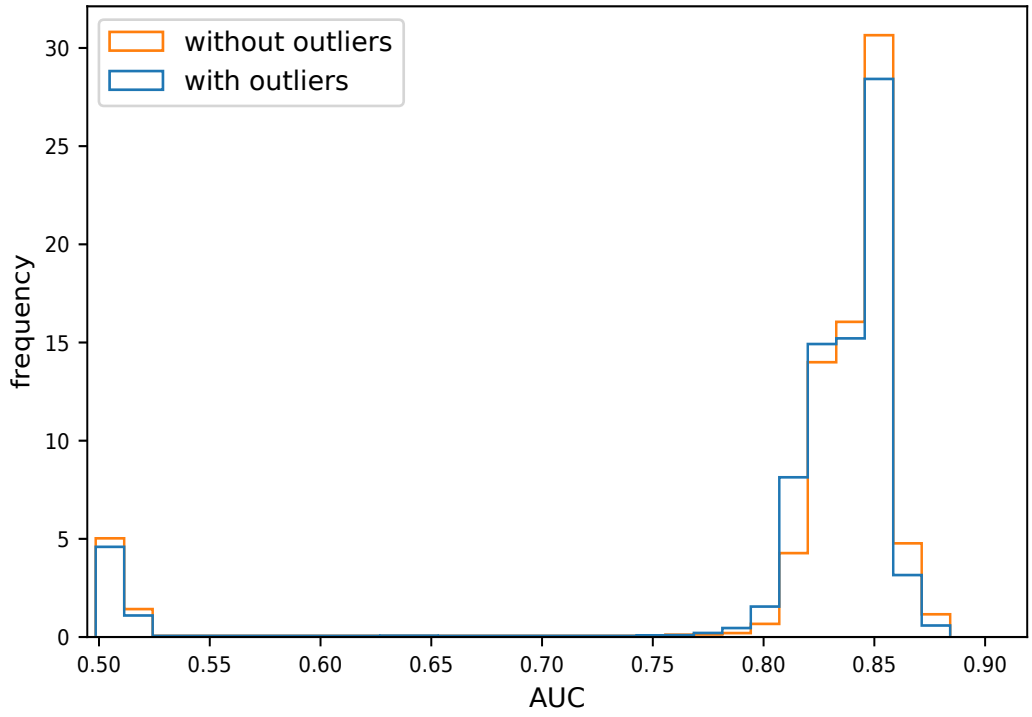

Figure 11.1: Histogram of AUCs after training an ensemble of models on a dataset with 5 % and without outliers. The former dataset contains 8.4×10^3 and the latter 7.6×10^3 models. Both histograms are normed to unit area. All models are trained on 1.0×10^6 waveforms.

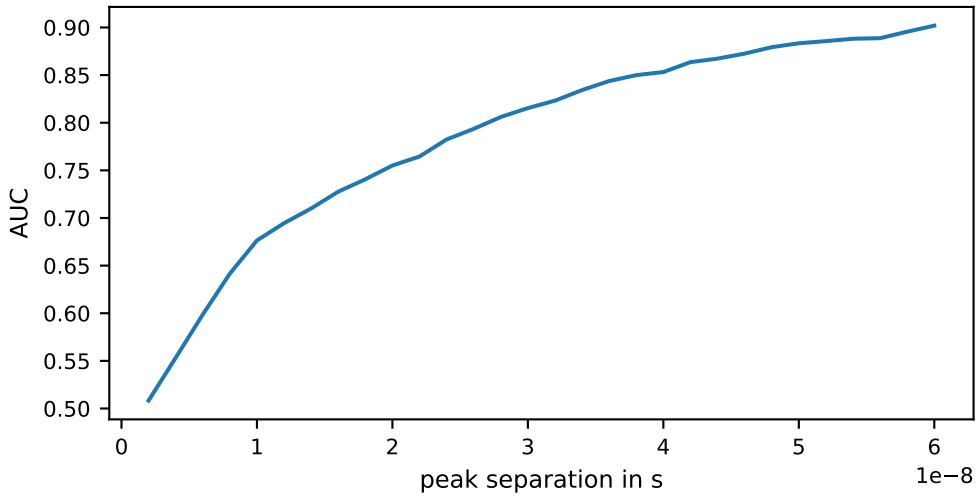

Figure 11.2: AUC of a `conv` model with $d_l = 3$ trained on 1.0×10^6 waveforms. The AUC is calculated on datasets consisting of 50 % normal waveforms and 50 % double-peak waveforms where the peaks have the given time separation.

waveform MC datasets are generated. Each dataset consists of 50 % normal waveforms and 50 % double-peak outliers with the same time separation. The second peak has the same intensity as the first one. Then an already trained autoencoder is used to classify all waveforms in all sets.

The AUC dependence on the time separation can be seen in Figure 11.2; as expected, the performance on double peaks with separations of $< 3.3\,\text{ns}$ (DOM time resolution) is little better than random guessing (AUC < 0.55). The performance increases with increasing time separation, reaching an AUC > 0.90 for a separation of $t_{pp} > 60\,\text{ns}$. This is equivalent to a separation of only 18 bins (out of 128 total bins).

Figure 11.3 shows how the separation influences the loss distributions. There is a significant Pearson correlation between the loss and the peak separation of $r_{\text{loss},t_{pp}} = 0.35$ in the interval of $t_{pp} = [0\,\text{ns}, 60\,\text{ns}]$. This demonstrates that the loss is a predictor for double-peak waveforms, but only gives a rough indication of the actual separation.

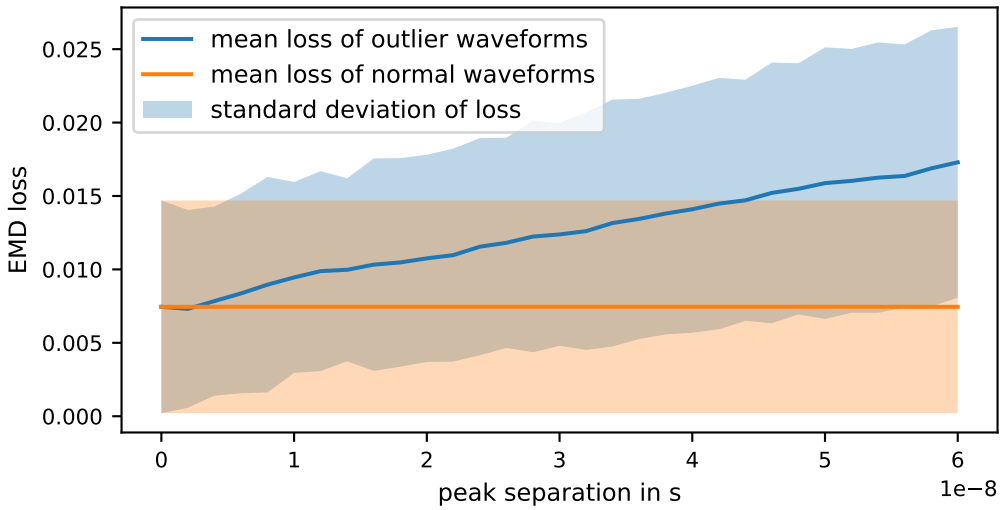

Figure 11.3: Loss of a `conv` model with $d_l = 3$ trained on 1.0×10^6 waveforms. The blue line indicates the mean reconstruction loss of double-peaked waveforms with the given time separation between peaks. As a reference, the orange line depicts the mean reconstruction loss of non-outlier waveforms.

12 Performance on IceCube MC

The previous sections evaluated the `conv` model with $d_l = 3$ for waveform MC data, where outliers were tagged and therefore outlier separation performance could be determined. This section applies the same model to IceCube MC data, where outliers are not tagged. Hence, this section verifies that the model behaves similarly on IceCube MC data as it does on waveform MC data.

12.1 Training Length and Stability

The validation loss distributions after training for n waveforms show a progression similar to the one observed on the waveform MC data (cf Figure 9.5). This is visualized in Figure 12.1. Training models on 1×10^6 or more waveforms only shifts the peak position slightly while reducing the broadness of the distribution appreciably. This is also reflected by the validation loss converging at around 1×10^6 waveforms, as shown in Figure 12.3. Therefore, models trained on 1×10^6 waveforms are examined more closely in the following.

Training on IceCube MC waveforms instead of the waveform MC leads to similar stability results: There is a 9.9 % chance that model training does not converge (loss > 0.08). As with the waveform MC, this is indicated by a high validation loss compared to the median validation loss (see Figure 12.2).

12.2 Neural Network Training

Based on the method developed in section 9.2, the median validation loss model, chosen from an ensemble of 2.2×10^2 models trained for 1×10^6, is analyzed here.

The validation loss distribution generated by this model (see Figure 12.4) peaks at $\mu_{\text{peak}} = 2.4 \times 10^{-2}$ and has a standard deviation of $\sigma = 1.3 \times 10^{-2}$. This distribution is broader and peaks at higher losses compared to the distribution attained by using the waveform MC, where μ_{peak} and σ are about six times smaller (cf section 10.1).

The training loss curve, depicted in Figure 12.5, has an overall shape similar to the training loss curve calculated on waveform MC data (cf Figure 10.3). The most

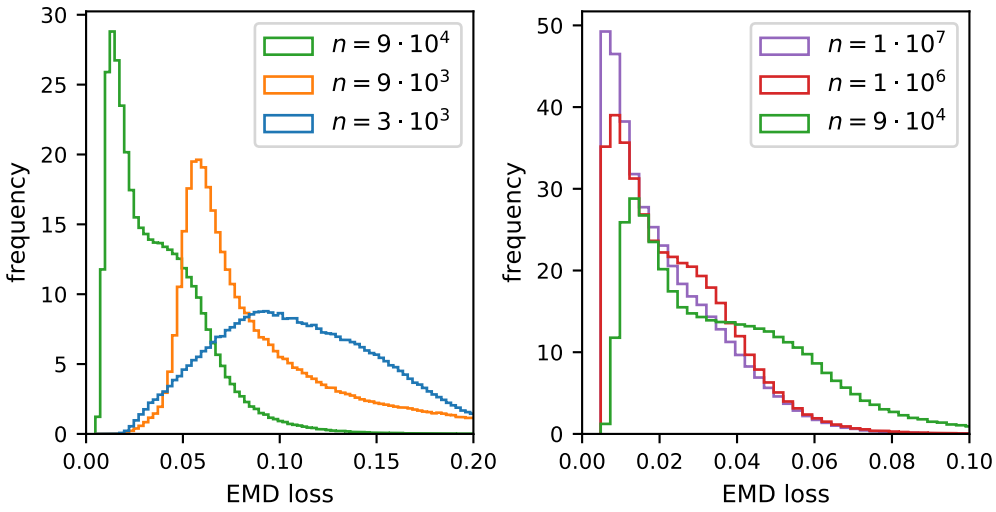

Figure 12.1: Validation loss distributions for models trained on different number of waveforms n. For each n, a `conv` model with $d_l = 3$ was selected from an ensemble of 2.2×10^2 models for having the median training loss. The distributions are obtained by histogramizing the reconstruction losses of 3.7×10^5 random validation waveforms. All histograms are normed to unit area. The distributions have long tails for high losses, which are not shown.

notable differences are that the former converges at higher losses and has a higher variance between batches.

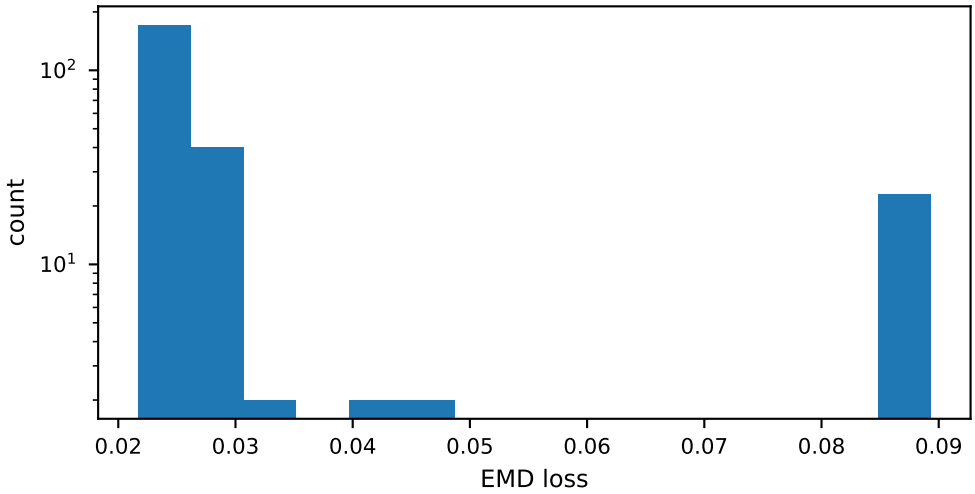

Figure 12.2: Histogram of mean validation losses after training an ensemble of 2.2×10^2 `conv` models with $d_l = 3$ on 1.0×10^6 IceCube MC waveforms.

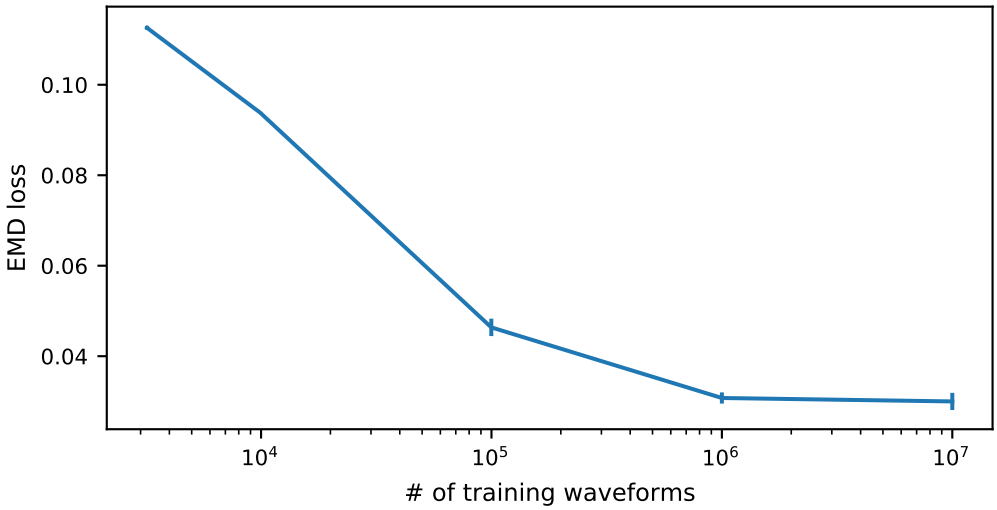

Figure 12.3: Dependency of the mean validation loss of `conv` models with $d_l = 3$ on the number of waveforms used in training. The loss converges around 1×10^6. Each point represents the mean validation loss of 1.3×10^2 independently trained models. The error bars represent the standard deviation of the mean.

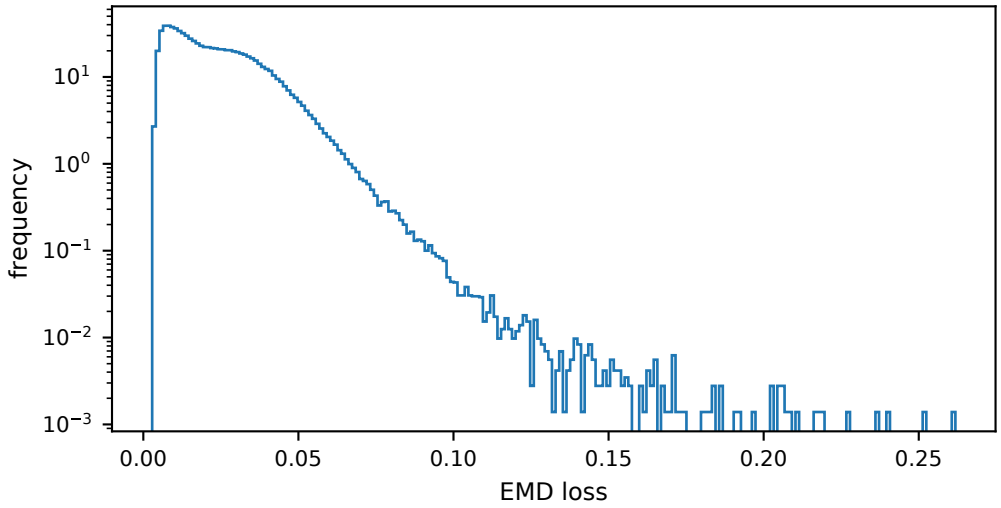

Figure 12.4: Validation loss distributions for the median validation loss model with $d_l = 3$, selected from an ensemble of 1.0×10^6 models. The distribution is obtained by histogramizing the reconstruction losses of 3.7×10^5 random validation waveforms. The histogram is normed to unit area.

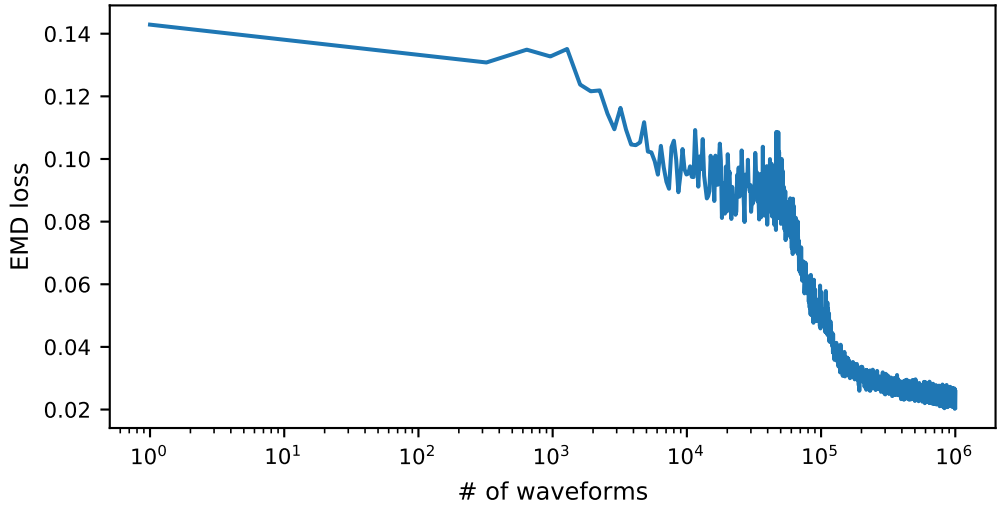

Figure 12.5: Training loss progression of the median validation loss model with $d_l = 3$, selected from an ensemble of 2.2×10^2 models.

13 Conclusion and Outlook

Conclusion

This work proposes and demonstrates an approach to judge the suitability of autoencoders for finding outliers in IceCube waveforms.

A labeled **waveform MC simulation**, based on the Pandel function, is developed to make informed comparisons to alternative models. The simulation is deliberately kept simple while still providing a reasonable approximation of real waveforms. Normal waveforms are described by a marginalized distribution that only retains distance, angle and brightness as independent parameters. Outliers are simulated by modifying normal waveforms with Gaussian distributions or repeating normal waveforms multiple times.

Using this labeled simulation data, it was possible to run an **automated hyperparameter optimization**, finding a range of alternative models. It is shown that the differences in outlier detection performance between the selected model and the best alternative models are small ($\Delta\text{AUC} < 5\,\%$). Thus, having verified that there are no obvious superior alternatives, the selected model is extensively analyzed.

Its **training stability** is assessed using an ensemble of independently trained copies. It becomes apparent that, after training the model, the outlier detection performance is a random variable. Except for 7 % of all cases, when the model is unable to identify outliers at all, it is shown that there is no simple relation between the outlier detection performance and the mean reconstruction loss.
To mitigate this problem, an **algorithm to select a well-performing model** from the ensemble is developed. It reduces the chances of picking a suboptimal model ($\text{AUC} < 0.87$) by a factor of 2 and avoids picking bad models ($\text{AUC} < 0.6$) entirely.

The fully trained model picked by this algorithm rates each waveform with an outlier score. It detects outliers in the waveform MC with an $\text{AUC} = 0.86$. The parametrizations of normal and outlier waveforms show a reasonable distribution in the latent space. It is shown that the presence of outliers in the training sample has a negligible influence on the training process. Additionally, it is verified that limiting the **number of latent dimensions** to 3 or lower **constrains** the model's **capacity**. This ensures that the model cannot reduce the reconstruction loss of

outliers, which would lead to an inability to detect them. Also, the optimal range for **early stopping** is determined to prevent overfitting. The model's ability to **identify double-pulse waveforms** as outliers is demonstrated as well.

Finally, with the model's ability to work on the simplified data sample of the waveform MC, it is trained on unlabeled IceCube MC data. All tests transferable to unlabeled data are repeated; the most important difference is that the overall **reconstruction loss** is about 6 times **higher**. But this is to be expected as the IceCube MC dataset is considerably more complex. Nonetheless, the model shows a **qualitatively similar behavior on IceCube MC** data as it does on waveform MC data.

In summary, an approach to analyze autoencoders for waveform outlier detection has been demonstrated. Albeit simple, the developed autoencoder model was shown to be robust and is likely suited for detecting outliers in actual IceCube data.

Outlook

The neural networks used in this work are rudimentary and only deep enough to demonstrate their applicability. Examining deeper neural networks and more advanced techniques like variational autoencoders and additional layer types might therefore be worthwhile.

The most interesting possibility, however, seems to be the prediction of outliers on real data. Applying the outlier detection model to real data is straightforward, but interpreting the results is likely to be challenging: Like all methods involving neural networks, the model only provides outlier scores but no explanation why waveforms are anomalous. Considering only singular waveforms might be intractable.

Instead, considering the combination of waveforms contained in one IceCube event might prove more useful, as it allows the selection of anomalous events. This can, for example, be done by constructing an "event autoencoder". It would use as its input the parametrization and reconstruction loss that the autoencoder developed in this work provides. Such an autoencoder can also be used for event classification as well as outlier detection: The parametrization of IceCube events should lend itself to a cluster analysis.

In conclusion, this work can be seen as a jumping-off point for unsupervised learning with IceCube.

Bibliography

[1] *6.3. Preprocessing data — scikit-learn 0.22.2 documentation*. URL: https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing (visited on 04/09/2020).

[2] *About Keras layers - Keras Documentation*. URL: https://keras.io/layers/about-keras-layers/ (visited on 04/14/2020).

[3] M. Ackermann et al. "Optical properties of deep glacial ice at the South Pole". In: *Journal of Geophysical Research: Atmospheres* 111 (D13 2006). _eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2005JD006687. ISSN: 2156-2202. DOI: 10.1029/2005JD006687. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2005JD006687 (visited on 04/15/2020).

[4] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *arXiv:1206.5533 [cs]* (Sept. 16, 2012). arXiv: 1206.5533. URL: http://arxiv.org/abs/1206.5533 (visited on 03/16/2020).

[5] J. Bluemer, R. Engel, and J. R. Hoerandel. "Cosmic Rays from the Knee to the Highest Energies". In: *Progress in Particle and Nuclear Physics* 63.2 (Oct. 2009), pp. 293–338. ISSN: 01466410. DOI: 10.1016/j.ppnp.2009.05.002. arXiv: 0904.0725. URL: http://arxiv.org/abs/0904.0725 (visited on 04/10/2020).

[6] Mathis Börner et al. "Measurement/simulation mismatches and multivariate data discretization in the machine learning era". In: *Proc. of the ADASS XXVII* (2017).

[7] Nikhil Buduma. *Fundamentals of Deep Learning*. ISBN: 978-1-4919-2561-4. URL: http://shop.oreilly.com/product/0636920039709.do (visited on 04/07/2020).

[8] IceCube Collaboration et al. "Energy Reconstruction Methods in the IceCube Neutrino Telescope". In: *J. Inst.* 9.3 (Mar. 17, 2014), P03009–P03009. ISSN: 1748-0221. DOI: 10.1088/1748-0221/9/03/P03009. arXiv: 1311.4767. URL: http://arxiv.org/abs/1311.4767 (visited on 04/13/2020).

[9] IceCube Collaboration et al. "Measurement of South Pole ice transparency with the IceCube LED calibration system". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 711 (May 2013), pp. 73–89. ISSN: 01689002. DOI: 10.1016/j.nima.2013.01.054. arXiv: 1301.5361. URL: http://arxiv.org/abs/1301.5361 (visited on 01/28/2020).

[10] IceCube Collaboration et al. "Search for Astrophysical Tau Neutrinos in Three Years of IceCube Data". In: *Phys. Rev. D* 93.2 (Jan. 12, 2016), p. 022001. ISSN: 2470-0010, 2470-0029. DOI: 10.1103/PhysRevD.93.022001. arXiv: 1509.06212. URL: http://arxiv.org/abs/1509.06212 (visited on 03/16/2020).

[11] IceCube Collaboration et al. "The IceCube Neutrino Observatory: Instrumentation and Online Systems". In: *J. Inst.* 12.3 (Mar. 14, 2017), P03012–P03012. ISSN: 1748-0221. DOI: 10.1088/1748-0221/12/03/P03012. arXiv: 1612.05093. URL: http://arxiv.org/abs/1612.05093 (visited on 04/13/2020).

[12] The IceCube Collaboration. "The IceCube Data Acquisition System: Signal Capture, Digitization, and Timestamping". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 601.3 (Apr. 2009). version: 2, pp. 294–316. ISSN: 01689002. DOI: 10.1016/j.nima.2009.01.001. arXiv: 0810.4930. URL: http://arxiv.org/abs/0810.4930 (visited on 01/13/2020).

[13] The IceCube Collaboration et al. "Calibration and Characterization of the IceCube Photomultiplier Tube". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 618.1 (June 2010), pp. 139–152. ISSN: 01689002. DOI: 10.1016/j.nima.2010.03.102. arXiv: 1002.2442. URL: http://arxiv.org/abs/1002.2442 (visited on 01/10/2020).

[14] D F Cowen and the IceCube Collaboration. "Tau Neutrinos in IceCube". In: *J. Phys.: Conf. Ser.* 60 (Mar. 1, 2007), pp. 227–230. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/60/1/048. URL: https://iopscience.iop.org/article/10.1088/1742-6596/60/1/048 (visited on 04/15/2020).

[15] *Detector*. URL: https://icecube.wisc.edu/science/icecube/detector (visited on 04/13/2020).

[16] Clarence Chio Freeman David. *Machine Learning and Security*. ISBN: 978-1-4919-7990-7. URL: http://shop.oreilly.com/product/0636920065555.do (visited on 01/27/2020).

[17] Thomas K. Gaisser, Ralph Engel, and Elisa Resconi. *Cosmic Rays and Particle Physics*. Cambridge Core. ISBN: 9780521016469 9781139192194 Library Catalog: www.cambridge.org Publisher: Cambridge University Press. June 2016. DOI: 10.1017/CBO9781139192194. URL: /core/books/cosmic-rays-and-particle-physics/C81BA71195ADFC89EFCC2C565B617702 (visited on 04/13/2020).

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Google-Books-ID: omivDQAAQBAJ. MIT Press, Nov. 10, 2016. 801 pp. ISBN: 978-0-262-33737-3.

[19] Christian Haack, Christopher Wiebusch, and on behalf of the IceCube Collaboration. "A measurement of the diffuse astrophysical muon neutrino flux using eight years of IceCube data." In: *Proceedings of 35th International Cosmic Ray*

Conference — PoS(ICRC2017). 35th International Cosmic Ray Conference. Vol. 301. SISSA Medialab, Aug. 3, 2018, p. 1005. DOI: 10.22323/1.301.1005. URL: https://pos.sissa.it/cgi-bin/reader/contribution.cgi?id=PoS(ICRC2017)1005 (visited on 04/13/2020).

[20] Simon Hawkins et al. "Outlier detection using replicator neural networks". In: *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2002, pp. 170–180.

[21] Robert Hecht-Nielsen. "Replicator Neural Networks for Universal Optimal Source Coding". In: *Science* 269.5232 (Sept. 29, 1995). Publisher: American Association for the Advancement of Science Section: Reports, pp. 1860–1863. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.269.5232.1860. URL: https://science.sciencemag.org/content/269/5232/1860 (visited on 04/15/2020).

[22] K. S. Hirata et al. "Observation in the Kamiokande-II detector of the neutrino burst from supernova SN1987A". In: *Phys. Rev. D* 38.2 (July 15, 1988). Publisher: American Physical Society, pp. 448–458. DOI: 10.1103/PhysRevD.38.448. URL: https://link.aps.org/doi/10.1103/PhysRevD.38.448 (visited on 04/15/2020).

[23] The IceCube et al. "Multi-messenger observations of a flaring blazar coincident with high-energy neutrino IceCube-170922A". In: *Science* (July 12, 2018), eaat1378. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aat1378. arXiv: 1807.08816. URL: http://arxiv.org/abs/1807.08816 (visited on 04/10/2020).

[24] *Icetray — Metaproject 'combo.trunk' at r177596*. URL: http://software.icecube.wisc.edu/documentation/projects/icetray/index.html (visited on 04/17/2020).

[25] Ulrich F. Katz and Christian Spiering. "High-Energy Neutrino Astrophysics: Status and Perspectives". In: *Progress in Particle and Nuclear Physics* 67.3 (July 2012), pp. 651–704. ISSN: 01466410. DOI: 10.1016/j.ppnp.2011.12.001. arXiv: 1111.0507. URL: http://arxiv.org/abs/1111.0507 (visited on 01/28/2020).

[26] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 29, 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 03/13/2020).

[27] Matthew Kirk. *Thoughtful Machine Learning with Python*. ISBN: 978-1-4919-2413-6. URL: http://shop.oreilly.com/product/0636920039082.do (visited on 04/07/2020).

[28] Lev Klebanov, Teodosii Rachev, and F.J. Fabozzi. "Robust and non-robust models in statistics". In: *Robust and Non-Robust Models in Statistics* (Jan. 1, 2009), pp. 1–317.

[29] Claudio Kopper. "Observation of astrophysical neutrinos in six years of IceCube data". In: *PoS* (2017). Publisher: SISSA, p. 981.

[30] Anastasis Kratsios. "The Universal Approximation Property: Characterizations, Existence, and a Canonical Topology for Deep-Learning". In: *arXiv:1910.03344 [cs, math, stat]* (Feb. 20, 2020). arXiv: 1910.03344. URL: http://arxiv.org/abs/1910.03344 (visited on 04/09/2020).

[31] E. Levina and P. Bickel. "The Earth Mover's distance is the Mallows distance: some insights from statistics". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001.* Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001. Vol. 2. July 2001, 251–256 vol.2. DOI: 10.1109/ICCV.2001.937632.

[32] Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *arXiv:1603.06560 [cs, stat]* (June 18, 2018). arXiv: 1603.06560. URL: http://arxiv.org/abs/1603.06560 (visited on 02/04/2020).

[33] Richard Liaw et al. "Tune: A Research Platform for Distributed Model Selection and Training". In: *arXiv:1807.05118 [cs, stat]* (July 13, 2018). arXiv: 1807.05118. URL: http://arxiv.org/abs/1807.05118 (visited on 02/04/2020).

[34] Stephen Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition*. Google-Books-ID: y_oYCwAAQBAJ. CRC Press, Sept. 15, 2015. 459 pp. ISBN: 978-1-4987-5978-6.

[35] Maximilian Meier. "Search for Astrophysical Tau Neutrinos using 7.5 years of IceCube Data". PhD thesis. TU Dortmund, 2019.

[36] Chigozie Nwankpa et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: (Nov. 8, 2018). URL: https://arxiv.org/abs/1811.03378v1 (visited on 04/09/2020).

[37] Douwe Osinga. *Deep Learning Cookbook*. ISBN: 978-1-4919-9584-6. URL: http://shop.oreilly.com/product/0636920097471.do (visited on 04/11/2020).

[38] Dirk Pandel. "Bestimmung von Wasser-und Detektorparametern und Rekonstruktion von Myonen bis 100 TeV mit dem Baikal-Neutrinoteleskop NT-72". In: *Humboldt-Universitaet Berlin* (1996).

[39] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[40] Kai Schatto. "Stacked searches for high-energy neutrinos from blazars with IceCube". In: (2014). URL: https://core.ac.uk/display/56719291 (visited on 02/18/2020).

[41] *torch.nn — PyTorch master documentation*. URL: https://pytorch.org/docs/stable/nn.html#loss-functions (visited on 04/15/2020).

[42] *Tune Trial Schedulers — Ray 0.9.0.dev0 documentation*. URL: https://ray.readthedocs.io/en/latest/tune-schedulers.html#hyperband (visited on 02/19/2020).

[43] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, July 5, 2008, pp. 1096–1103. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390294. URL: https://doi.org/10.1145/1390156.1390294 (visited on 04/11/2020).

[44] Arthur Zimek and Erich Schubert. "Outlier Detection". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. New York, NY: Springer, 2017, pp. 1–5. ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_80719-1. URL: https://doi.org/10.1007/978-1-4899-7993-3_80719-1 (visited on 01/28/2020).

Appendix

1 Abbreviations

ADC Analog-to-digital converter. An electric circuit that converts e.g. the photocurrent of a PMT into a digital signal.

AE Autoencoder, a type of neural network, see section 3.3.

ATWD Analog transient waveform digitizer, a specialized ADC. See section 2.2.

AUC Area under curve. This is a shorthand for area under the ROC curve, a measure of classifier performance.

DOM Digital optical module, a sensor that detects Cherenkov light. See chapter 2.

EMD Earth mover's distance, see subsection 3.4.1.

HPO Hyperparameter optimization, see section 5.4.

K-S Kolmogorov–Smirnov test. A statistical test to determine whether two probability distributions are equal.

KLD Kullback–Leibler divergence. A function that measures the difference between two probability distributions. See Appendix 3.

L1 See MAE.

MAE Mean absolute error of two vectors x, y is given by $\mathrm{MSE}(x, y) = \sum_i |x_i - y_i|$.

MC Monte Carlo method. Used to i.a. to model and simulate statistical systems too complex for analytical calculations.

MKLD a modified version of the Kullback–Leibler divergence, see Appendix 3.

ML Machine learning.

MSE Mean squared error of two vectors x, y is given by $\mathrm{MSE}(x, y) = \sum_i (x_i - y_i)^2$.

NN (Artificial) Neural network, see section 3.2. All NNs in this work are also feed-forward NN.

PDF Probability density function.

PMT Photomultiplier tube, see section 2.2.

RMS The root mean square of n measurements x_i is given by $\mathrm{RMS} = \sqrt{n^{-1}\left(\sum_i^n x_i^2\right)}$.

ROC Receiver operating characteristic. A graphical tool to illustrate the performance of a classifier.

STD Standard deviation.

WIMP Weakly-interacting massive particle, one possible explanation for dark matter.

2 Software

2.1 Installation:

Python Version

At the time this master thesis was developed, Python's most recent version was 3.8; many important packages had not been ported from Python 3.7 (e.g. `ray`) yet, so Python 3.7 was used. All code by the author should also run in Python 3.8 as soon as all required packages are supported.

The only exception is the code used to extract waveform data from IceCube MC files as a lot of IceCube libraries only support legacy Python. Therefore, Python 2.7 is used to run `01-extractor.py`.

Local Installation

This is a guide for installing the software needed to run code from this work on a Linux system. The process should be similar on a Mac.

Pull the repository containing the code:

```
$ git clone https://github.com/DimensionalScoop/MasterThesisCode.git
$ cd MasterThesisCode
```

It is recommended to use a virtual environment:[1]

```
$ virtualenv -p /usr/bin/python3.7 venv
$ source venv/bin/activate
```

Replace the path to the python interpreter as needed for your system.

Install the package and all dependencies:

```
$ pip install -e .
$ pip install ray[tune]
```

Using the switch `-e` allows you to make changes to the code after installing it as a package.

To get started, open jupyter lab:

```
$ venv/bin/jupyter-lab icae/manual/01-toy-mc.ipynb
```

[1] This means that the path to common python tools also changes. To start e.g. Jupyter Notebook, use `$ venv/bin/jupyter-notebook`.

Installation on Cobalt

Parts of this software package are only executed on *Cobalt*, a server used for software development and testing.[2] It is easier to preprocess some of the MC data there in order to only transfer the data needed for this work.

The IceTray `combo.trunk r163695` release is used in conjunction with Python 2.7.13.

Using The Waveform MC

The code needed to run the waveform MC is located at `icae/toy/waveformMC`. A Jupyter Notebook that demonstrates the basic usage is located at `icae/manual/01-toy-mc.ipynb`. The dataset used in this work is generated with `icae/results/n01_toy/n01*.py`.

2.2 Reproducing This Work

The following steps reproduce the MC simulations, values and plots used in this work:

- Run `icae/notebooks/01-extractor.py` on the Cobalt server. This is the only file that needs to be executed on Cobalt.
- Copy the hdf files generated in the previous step to a local machine with a non-deprecated version of Python.
- Configure the software and filepaths in `config.yml`.
- Run all scripts in `icae/results/n01_toy/*.py` in alphabetical order to reproduce the results on the waveform MC.
- Run all scripts in `icae/results/n02_waveform/*.py` in alphabetical order to reproduce the results on the IceCube MC.

Some scripts that train models produce data indefinitely and should be interrupted manually when the desired amount of MC data has been produced. Please note that the software was developed and used on a computer with 32 GB of RAM and 8 GB of GPU RAM. Running it on a machine with less (GPU) RAM might results in memory issues that might necessitate minor alterations to the code.

Notes on Naming Conventions

Files that solve one problem in multiple steps have the same number (e.g. `n01a...` and `n01b...`). Plots are saved to `icae/results/plots/[name_of_script]/*.pdf`

[2] https://wiki.icecube.wisc.edu/index.php/Cobalt_Nodes

and scalar values to `saved_values.yml`.

The waveform MC was previously named “toy MC”. It was renamed as it became more sophisticated, but this name change is not reflected in the codebase.

3 Modified Kullback–Leibler Divergence

During the hyperparameter optimization, different training loss functions are used. One of them is the modified Kullback–Leibler divergence (MKLD), a loss function developed for this work.

The (unmodified) Kullback–Leibler divergence of two discrete probability distributions q_x and p_x is given by [18, Chapter 3]:

$$D_{\mathrm{KL}}(q,p) = \sum_i p_i \left(\log p_i - \log q_i\right) \ . \tag{I.1}$$

The input and output of the autoencoder can be described as probability distributions. This can be interpreted as the output p being an approximate distribution of the input q. The divergence then describes the information gain achieved when moving from the output to the input [18, Chapter 3].

Applying D_{KL} to the output of neural networks raises two problems, as the output does not necessarily fulfill the following conditions:

$$q_i \neq 0 \quad \forall i | p_i \neq 0 \tag{I.2}$$

$$\sum_i q_i = \sum_i p_i = 1 \ . \tag{I.3}$$

To meet these two conditions, some modifications to D_{KL} are made. Numerically, p and q are described as vectors $\vec{p}$ and $\vec{q}$. First, the sigmoid function is applied to each component of $\vec{p}$ and $\vec{q}$. This guarantees that $(\vec{p})_i, (\vec{q})_i \in [0,1]$. Adding a small number $\delta = 1 \times 10^{-9}$ to every component of both vectors fulfills condition I.2. This also allows the network to compute gradients for output components that would previously been zero.

With

$$n_p = \sum_i (\vec{p})_i \,, \quad n_q = \sum_i (\vec{q})_i \tag{I.4}$$

$$\left(\vec{p}\,'\right)_i = \frac{(\vec{p})_i}{n_p} \tag{I.5}$$

$$\left(\vec{q}\,'\right)_i = \frac{(\vec{q})_i}{n_q} \tag{I.6}$$

$$D_{\mathrm{uMKL}}\left(\vec{p}, \vec{q}\right) = D_{\mathrm{KL}}\left(\vec{p}\,', \vec{q}\,'\right) \,. \tag{I.7}$$

condition I.3 can be satisfied. This is called the *unenforced* MKLD. The unenforced version does not constrain the total area of the unmodified output of the network.

In the *enforced* version, $\vec{q}$ is not normed to unit area. The neural network has to learn to output correctly normed $\vec{q}$. To achieve this, an additional penalty term is added to the loss:

$$\left(\vec{q}\,''\right)_i = \frac{(\vec{q})_i}{n_p} \tag{I.8}$$

$$\Delta = -1 + \exp\left|-1 + \sum_i \left(\vec{q}\,''\right)_i\right| \tag{I.9}$$

$$D_{\mathrm{eMKL}}\left(\vec{p}, \vec{q}\right) = D_{\mathrm{KL}}\left(\vec{p}\,', \vec{q}\,''\right) + \Delta \,. \tag{I.10}$$

Eidesstattliche Versicherung
(Affidavit)

PERNKLAU MAX
Name, Vorname
(Last name, first name)

175276
Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der ~~Bachelor-~~/Masterarbeit*:
(Title of the ~~Bachelor's-~~Master's* thesis):

ANOMALY DETECTION IN ICECUBE WAVEFORMS USING MACHINE LEARNING AND MONTE CARLO METHODS.

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

DORTMUND, 17.04.2020
Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

DORTMUND, 17.04.2020
Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.