Jan Kowalczyk

Lidar Degradation Quantification for Robot Navigation in Hazy Environments

master's thesis

submitted to

Graz University of Technology

Supervisors

Univ.-Prof. Dipl.-Ing. Dr.mont Franz Pernkopf

Signal Processing and Speech Communication Laboratory

in cooperation with

Virtual Vehicle Research GmbH
Graz, Austria

Graz, September, 2025

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

date	(signature)

Acknowledgements

Here you can tell us, how thankful you are for this amazing template ;)

Abstract (English)

Write some fancy abstract here!

Abstract (German)

Hier könnte eine Kurzfassung sein.

Contents

1 Introduction

Goal: *"What should the reader know after reading this section?"*
Context: *"Why is that of interest to the reader at this point?"*
Method: *"How am I achieving the stated goal?"*
Transition: *"How does this lead to the next question or section?"*

Goal: Create interest in topic, introduce main goal of thesis, summarize results
Context: Reader only has abstract as context, need to create interest at beginning
Method: emotional rescue missions, explain why data may be bad, state core research question
Transition: what has and hasn't been done → Scope of Research

Autonomous robots have gained more and more prevailance in search and rescue (SAR) missions due to not endangering another human being and still being able to fulfil the difficult tasks of navigating hazardous environments like collapsed structures, identifying and locating victims and assessing the environment's safety for human rescue teams. To understand the environment, robots employ multiple sensor systems such as lidar, radar, ToF, ultrasound, optical cameras or infrared cameras of which lidar is the most prominently used due to its accuracy. The robots use the sensors' data to map their environments, navigate their surroundings and make decisions like which paths to prioritize. Many of the aforementioned algorithms are deep learning-based algorithms which are trained on large amounts of data whose characteristics are learned by the models.

Environments of search and rescue situations provide challenging conditions for the sensor systems to produce reliable data. One of the most promiment examples are aerosol particles from smoke and dust which can obstruct the view and lead sensors to produce erroneous data. If such degraded data was not present in the robots' algorithms' training data these errors may lead to unexpected outputs and potentially endanger the robot or even human rescue targets. This is especially important for autonomous robots whose decisions are entirely based on their sensor data without any human intervention. To safeguard against these problems, robots need a way to assess the trustworthiness of their sensor systems' data.

For remote controlled robots a human operator can make these decisions but many search and rescue missions do not allow remote control due to environment factors, such as radio signal attenuation or the search area's size and therefore demand autonomous robots. Therefore, during the design for such robots we arrive at the following critical question:

> Can autonomous robots quantify the reliability of lidar sensor data in hazardous environments to make more informed decisions?

In this thesis we aim to answer this question by assessing a deep learning-based anomaly detection method and its performance when quantifying the sensor data's degradation. The employed algorithm is a semi-supervised anomaly detection algorithm which uses manually labeled training data to improve its performance over unsupervised methods. We show how much the introduction of these labeled samples improves the methods performance. The models output is an anomaly score which quantifies the data reliability and can be used by algorithms that rely on the sensor data. These reliant algorithms may decide to for example slow down the robot to collect more data, choose alternative routes, signal for help or rely more heavily on other sensor's input data.

discuss results (we showed X)

1.1 Scope of Research

Goal: clearly state what has and hasn't been researched + explanation why
Context: from intro its clear what thesis wants to achieve, now we explain how we do that
Method: state limit on data domain, sensors, output of method + reasoning for decisions
Transition: clear what we want to achieve → how is thesis structured to show this work

In this thesis, we focus our research on the unique challenges faced by autonomous rescue robots, specifically the degradation of sensor data caused by airborne particles. While degradation in sensor data can also arise from adverse weather, material properties, or dynamic elements such as moving leaves, these factors are considered less relevant to the rescue scenarios targeted by our study and are therefore excluded. Although our method is versatile enough to quantify various types of degradation, our evaluation is limited to degradation from airborne particles, as this is the most prevalent issue in the operational environments of autonomous rescue robots.

While robotic computer vision systems often incorporate a variety of sensors—such as time-of-flight cameras, infrared cameras, and ultrasound sensors—we found that autonomous rescue robots primarily depend on LiDAR data for mapping and navigation. LiDAR sensors offer high accuracy, high resolution, and an extensive field of view (often a full 360° horizontally and a substantial vertical coverage), which are essential for constructing comprehensive environmental maps in challenging scenarios. Furthermore, the cost of LiDAR sensors has decreased significantly in recent decades, driven by their widespread adoption in autonomous driving, drones, and robotics, as well as manufacturing advancements like microelectromechanical systems (MEMS). For these reasons, our research is focused exclusively on LiDAR sensor data—specifically, the point clouds generated within a defined coordinate system. Although sensor fusion techniques are commonly used to enhance data accuracy and confidence, incorporating fused data would not only add significant complexity to our study but also limit our analysis to platforms equipped with all the sensor types involved. Consequently, we concentrate on quantifying sensor degradation solely through LiDAR data.

The method we employ produces an analog score that reflects the confidence in the sensor data, with lower confidence indicating higher degradation. Although we do not investigate the direct applications of this score, potential uses include simple thresholding to decide whether to proceed with a given action as well as dynamically adjusting the robot's speed based on data quality to collect additional data when confidence is low. Importantly, this output score is a snapshot for each LiDAR scan and does not incorporate temporal information. While many LiDAR sensors capture multiple scans per second—enabling the possibility of time-series analyses such as running averages or more advanced statistical evaluations—we focus solely on individual scans without examining the differences between successive scans.

1.2 Structure of the Thesis

Goal: explain how structure will guide reader from zero knowledge to answer of research question
Context: since reader knows what we want to show, an outlook over content is a nice transition
Method: state structure of thesis and explain why specific background is necessary for next section
Transition: reader knows what to expect → necessary background info and related work

brief overview of thesis structure

in section x we discuss anomaly detection, semi-supervised learning since such an algorithm was used as the chosen method, we also discuss how lidar works and the data it produces. then in we discuss in detail the chosen method Deep SAD in section X, in section 4 we discuss the traing and evaluation data, in sec 5 we describe our setup for training and evaluation (whole pipeline). results are presented and discussed in section 6. section 7 contains a conclusion and discusses future work

2 Background and Related Work

Goal: explain which background knowledge is necessary and why + mention related work
Context: reader learns what he needs to know and which related work exists
Method: state which background subsections will follow + why and mention related work
Transition: necessity of knowledge and order of subsections are explained → essential background

This thesis tackles a broad, interdisciplinary challenge at the intersection of robotics, computer vision, and data science. In this chapter, we introduce the background of anomaly detection, which we formulate our degradation quantification problem as. Anomaly detection has its roots in statistical analysis and has been successfully applied in various domains. Recently, the incorporation of learning-based techniques, particularly deep learning, has enabled more efficient and effective analysis of large datasets.

Because anomalies are, by nature, often unpredictable in form and structure, unsupervised learning methods are widely used since they do not require pre-assigned labels—a significant advantage when dealing with unforeseen data patterns. However, these methods can be further refined through the integration of a small amount of labeled data, giving rise to semi-supervised approaches. The method evaluated in this thesis, DeepSAD, is a semi-supervised deep learning approach that also leverages an autoencoder architecture in its design. Autoencoders have gained widespread adoption in deep learning for their ability to extract features from unlabeled data, which is particularly useful for handling complex data types such as LiDAR scans.

LiDAR sensors function by projecting lasers in multiple directions near-simultaneously, measuring the time it takes for each reflected ray to return. Using the angles and travel times, the sensor constructs a point cloud that is often accurate enough to map the sensor's surroundings. In the following sections, we will delve into these technologies, review how they work, how they are generally used, how we employ them in this thesis and explore related work from these backgrounds.

2.1 Anomaly Detection

Goal: explain AD in general, allude to DeepSAD which was core method here
Context: problem is formulated as AD problem, so reader needs to understand AD
Method: give overview of AD goals, categories and challenges. explain we use DeepSAD
Transition: lots of data but few anomalies + hard to label → semi-supervised learning

Anomaly detection refers to the process of detecting unexpected patterns of data, outliers which deviate significantly from the majority of data which is implicitly defined as normal by its prevalence. In classic statistical analysis these techniques have been studied as early as the 19th century [1]. Since then, a multitude of methods and use-cases for them have been proposed and studied. Examples of applications include healthcare, where computer vision algorithms are used to detect anomalies in medical images for diagnostics and early detection of diseases [2], detection of fraud in decentralized financial systems based on block-chain technology [3] as well as fault detection in industrial machinery using acoustic sound data [4].

Figure 2.1 depicts a simple but illustrative example of data which can be classified as ei-

ther normal or anomalous and shows the problem anomaly detection methods try to generally solve. A successful anomaly detection method would somehow learn to differentiate normal from anomalous data, for example by learning the boundaries around the available normal data and classifying it as either normal or anomalous based on its location inside or outside of those boundaries. Another possible approach could calculate an analog value which correlates with the likelihood of an sample being anomalous, for example by using the sample's distance from the closest normal data cluster's center.

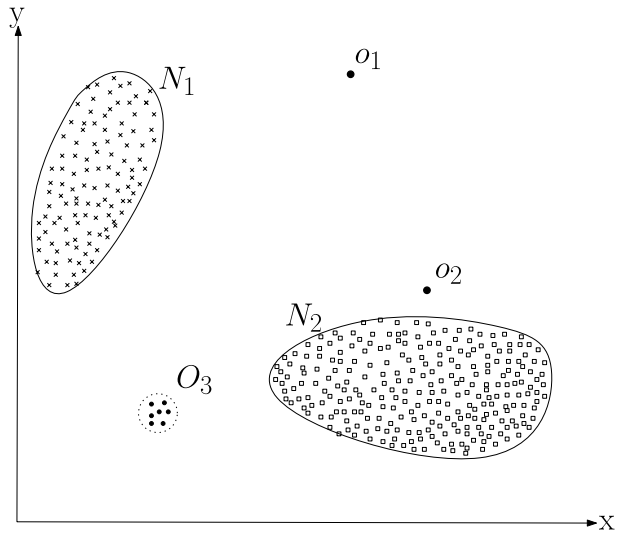


Figure 2.1: An illustrative example of anomalous and normal data containing 2-dimensional data with clusters of normal data N_1 and N_2 as well as two single anomalies o_1 and o_2 and a cluster of anomalies O_3. Reproduced from [5]

By their very nature anomalies are rare occurences and oftentimes unpredictable in nature, which makes it hard to define all possible anomalies in any system. It also makes it very challenging to create an algorithm which is capable of detecting anomalies which may have never occured before and may not have been known to exist during the creation of the detection algorithm. There are many possible approaches to this problem, though they can be roughly grouped into six distinct categories based on the techniques used [5]:

1. **Classification Based** - A classification technique such as an SVM or a fitting neural network is used to classify samples as either normal or anomalous based on labeled training data. Alternatively, if not enough labeled training data is available a one-class classification algorithm can be employed. In that case, the algorithm assumes all training samples to be normal and then learns a boundary around the normal samples to differentiate them from anomalous samples which lie outside the learnt boundary.

2. **Clustering Based** - Clustering techniques such as K-Means clustering or DBSCAN aim to group similar data together into clusters, differentiating it from dissimilar data which may belong to another or no cluster at all. Anomaly detection methods from this category employ such a technique, with the assumption that normal data will assemble into one or more clusters due to their similar properties, while anomalies may create their own smaller clusters, not belong to any cluster at all or at least be an appreciable distance from the closest normal cluster's center.

3. **Nearest Neighbor Based** - Similar to the clustering based category, these techniques assume normal data is more closely clustered than anomalies and therefore utilize either a sample's distance to their k^{th} nearest neighbor or the density of their local neighborhood, to judge wether a sample is anomalous.

4. **Statistical** - These methods try to fit a statistical model of the normal behaviour to the data. After the distribution from which normal data originates is defined, samples can be found to be normal or anomalous based on their likelihood to arise from said distribution.

5. **Information Theoretic** - The main assumption for information theoretic anomaly detection methods, is that anomalies differ somehow in their information content from anomalous data. An information theoretic measure is therefore used to determine iregularities in the data's information content, enabling the detection of anomalous samples.

6. **Spectral** - Spectral approaches assume the possibility to map data into a lower-dimensional space, where normal data appears significantly different from anomalous data. To this end a dimensionality reduction technique such as PCA is used to embed the data into a lower dimensional subspace. Although it may be easier to differentiate normal and anomalous data in that subspace, it is still necessary to employ a technique capable of this feat, so spectral methods are oftentimes used as a pre-processing step followed by an anomaly detection method operating on the data's subspace.

In this thesis we used an anomaly detection method, namely "Deep Semi-Supervised Anomaly Detection" [6] to model our problem -how to quantify the degradation of lidar sensor data- as an anomaly detection problem. We do this by classifying good quality data as normal and degraded data as anomalous and rely on a method which can express each samples likelihood of being anomalous as an analog anomaly score, which enables us to interpret it as the datas degradation quantification value.

Chapter 3 describes DeepSAD in more detail, which shows that it is a clustering based approach with a spectral pre-processing component, in that it uses a neural network to reduce the inputs dimensionality while simultaneously clustering normal data closely around a given centroid. It then produces an anomaly score by calculating the geometric distance between a data sample and the aforementioned cluster centroid, assuming the distance is shorter for normal than for anomalous data. Since our data is high dimensional it makes sense to use a spectral method to reduce the datas dimensionality and an approach which results in an analog value rather than a binary classification is useful for our use-case since we want to quantify not only classify the data degradation.

There is a wide array of problems in domains similar to the one we research in this paper, for which modeling them as anomaly detection problems has been proven successful. The degradation of pointclouds, produced by an industrial 3D sensor, has been modeled as an anomaly detection task in "Anomaly Detection in 3D Point Clouds using Deep Geometric Descriptors" [7]. Bergmann and Sattlegger propose a student-teacher model capable of infering a pointwise anomaly score for degradation in point clouds. The teacher network is trained on an anomaly-free dataset to extract dense features of the point clouds' local geometries, after which an identical student network is trained to emulate the teacher networks' outputs. For degraded pointclouds the regression between the teacher's and student's outputs is calculated and interpreted as the anomaly score, with the rationalization that the student network has not observed features produced by anomalous geometries during training, leaving it incapable of producing a similar output as the teacher for those regions. Another example would be "Automatic Detection and Classification of Pole-Like Objects in Urban Point Cloud Data Using an Anomaly Detection Algorithm" [8], which proposes a method to detect and classify pole-like objects in urban point cloud data, to differentiate between natural and man-made objects such as street signs, for autonomous driving purposes. An anomaly detection method was used to identify the vertical pole-like objects in the point clouds and then the preprocessed objects were grouped by similarity using a clustering algorithm to then classify them as either trees or man-made poles.

As already shortly mentioned at the beginning of this section, anomaly detection methods and their usage are oftentimes challenged by the limited availability of anomalous data, owing to the very nature of anomalies which are rare occurences. Oftentimes the intended use-case is to even

find unknown anomalies in a given dataset which have not yet been identified. In addition, it can be challenging to classify anomalies correctly for complex data, since the very definition of an anomaly is dependent on many factors such as the type of data, the intended use-case or even how the data evolves over time. For these reasons most types of anomaly detection approaches limit their reliance on anomalous data during training and many of them do not differentiate between normal and anomalous data at all. DeepSAD is a semi-supervised method which is characterized by using a mixture of labeled and unlabeled data.

2.2 Semi-Supervised Learning Algorithms

Goal: Give machine learning overview, focus on semi-supervised (what & why)
Context: used method is semi-supervised ML algorithm, reader needs to understand
Method: explain what ML is, how the different approaches work, why to use semi-supervised
Transition: autoencoder special case (un-/self-supervised) used in DeepSAD → explain autoencoder

Machine learning refers to algorithms capable of learning patterns from existing data to perform tasks on previously unseen data, without being explicitely programmed to do so [9]. Central to many approaches is the definition of an objective function that measures how well the model is performing. The model's parameters are then adjusted to optimize this objective. By leveraging these data-driven methods, machine learning can handle complex tasks across a wide range of domains.

Among the techniques employed in machine learning algorithms, neural networks have become especially prominent over the past few decades due to their ability to achieve state-of-the-art results across a wide variety of domains. They are most commonly composed of layers of interconnected artificial neurons. Each neuron computes a weighted sum of its inputs, adds a bias term, and then applies a nonlinear activation function, enabling them to model complex non-linear relationships. These layers are typically organized into three types:

- Input layer, which receives raw data.
- Hidden layers, where the network transforms and extracts complex features by combining signals through successive nonlinear operations. Networks with at least two hidden layers are typically called deep learning networks.
- Output layer, which produces the network's final prediction.

As outlined above, neural network training is formulated as an optimization problem: we define an objective function that measures how well the model is achieving its task and then we adjust the network's parameters to optimize that objective. The most common approach is stochastic gradient descent (SGD) or one of its variants (e.g., Adam). In each training iteration, the network first performs a forward pass to compute its outputs and evaluate the objective, then a backward pass—known as backpropagation—to calculate gradients of the objective with respect to every weight in the network. These gradients indicate the direction in which each weight should change to improve performance, and the weights are updated accordingly. Repeating this process over many iterations (or epochs) allows the network to progressively refine its parameters and better fulfill its task.

Aside from the underlying technique, one can also categorize machine learning algorithms by the type of feedback provided during learning, for the network to improve. Broadly speaking, three main categories-supervised, unsupervised and reinforcement learning-exist, although many other approaches do not exactly fit any of these categories and have spawned less common categories like semi-supervised or self-supervised learning.

In supervised learning, each input sample is paired with a "ground-truth" label representing the desired output. During training, the model makes a prediction and a loss function quantifies

the difference between the prediction and the true label. The learning algorithm then adjusts its parameters to minimize this loss, improving its performance over time. Labels are typically categorical (used for classification tasks, such as distinguishing “cat” from “dog”) or continuous (used for regression tasks, like predicting a temperature or distance).

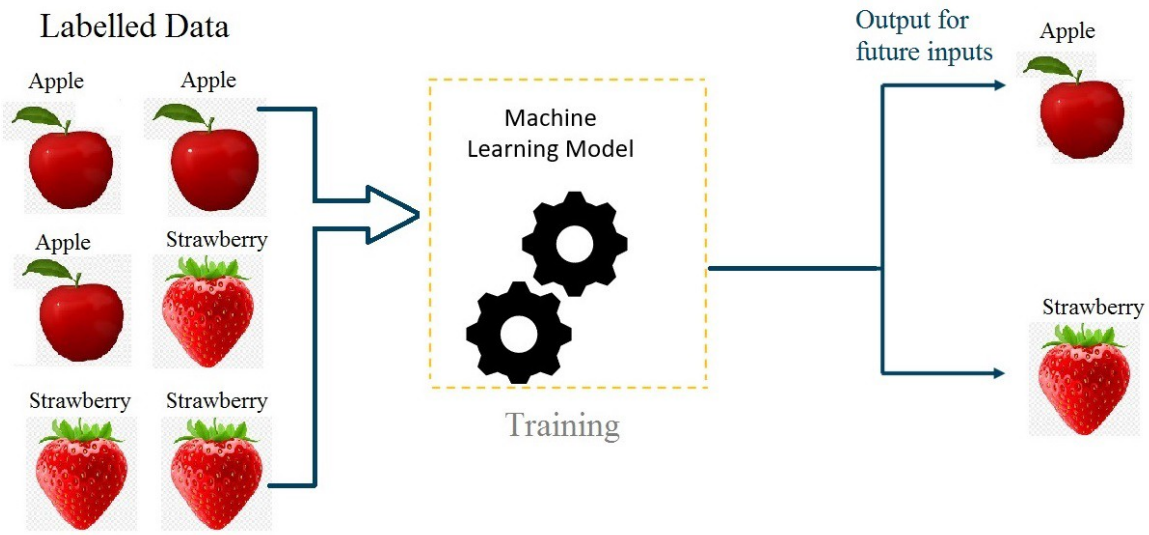


Figure 2.2: PLACEHOLDER - An illustration of supervised learning-the training data is augmented to include the algorithms optimal output for the data sample, called labels.

In unsupervised learning, models work directly with raw data, without any ground-truth labels to guide the learning process. Instead, they optimize an objective that reflects the discovery of useful structure—whether that is grouping similar data points together or finding a compact representation of the data. For example, cluster analysis partitions the dataset into groups so that points within the same cluster are more similar to each other (according to a chosen similarity metric) than to points in other clusters. Dimensionality reduction methods, on the other hand, project high-dimensional data into a lower-dimensional space, optimizing for minimal loss of the original data’s meaningful information. By focusing purely on the data itself, unsupervised algorithms can reveal hidden patterns and relationships that might be difficult to uncover with manual analysis.

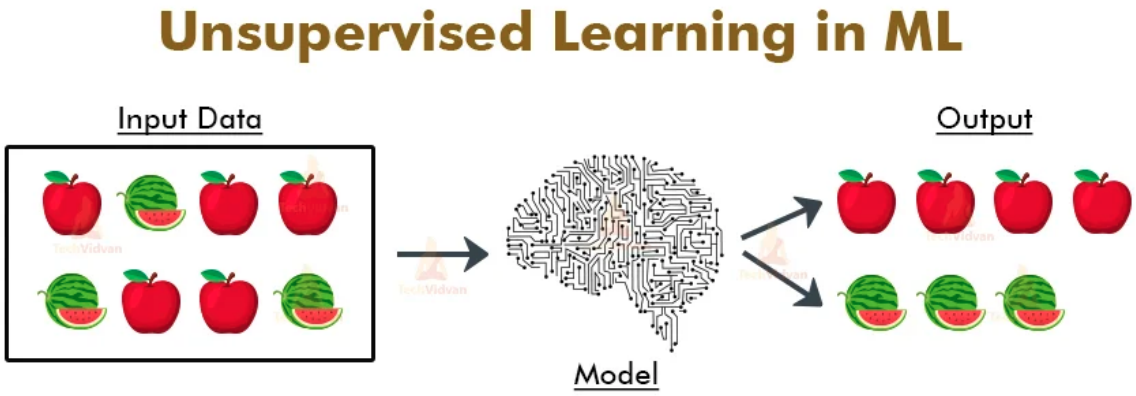


Figure 2.3: PLACEHOLDER - An illustration of unsupervised learning-the training data does not contain any additional information like a label. The algorithm learns to group similar input data together.

In reinforcement learning, the model—often called an agent—learns by interacting with an

environment, that provides feedback in the form of rewards or penalties. At each step, the agent observes the environment's state, selects an action, and an interpreter judges the action's outcome based on how the environment changed, providing a scalar reward or penalty that reflects the desirability of that outcome. The agent's objective is to adjust its decision-making strategy to maximize the cumulative reward over time, balancing exploration of new actions with exploitation of known high-reward behaviors. This trial-and-error approach is well suited to sequential decision problems in complex settings, such as autonomous navigation or robotic control, where each choice affects both the immediate state and future possibilities.

illustration reinforcement learning

Semi-Supervised learning algorithms are an inbetween category of supervised and unsupervised algorithms, in that they use a mixture of labeled and unlabeled data. Typically vastly more unlabeled data is used during training of such algorithms than labeled data, due to the effort and expertise required to label large quantities of data correctly. Semi-supervised methods are oftentimes an effort to improve a machine learning algorithm belonging to either the supervised or unsupervised category. Supervised methods such as classification tasks are enhanced by using large amounts of unlabeled data to augment the supervised training without additional need of labeling work. Alternatively, unsupervised methods like clustering algorithms may not only use unlabeled data but improve their performance by considering some hand-labeled data during training.

Machine learning based anomaly detection methods can utilize techniques from all of the aforementioned categories, although their usability varies depending on the available training data. While supervised anomaly detection methods exist, their suitability depends mostly on the availability of labeled training data as well as a reasonable proportionality between normal and anomalous data. Both requirements can be challenging due to labeling often being labour intensive and anomalies' intrinsic property to occur rarely when compared to normal data, making capture of enough anomalous behaviour a hard problem. Semi-Supervised anomaly detection methods are of special interest in that they may overcome these difficulties inherently present in many anomaly detection tasks [10]. These methods typically have the same goal as unsupervised anomaly detection methods which is to model the normal class behaviour and delimitate it from anomalies, but they can incorporate some hand-labeled examples of normal and/or anomalous behaviour to improve their perfomance over fully unsupervised methods. DeepSAD is a semi-supervised method which extends its unsupervised predecessor Deep SVDD by including some labeled samples during training. Both, DeepSAD and Deep SVDD also utilize an autoencoder in a pre-training step, a machine learning architecture, frequently grouped with unsupervised algorithms, even though that definition can be contested when scrutinizing it in more detail, which we will do next.

2.3 Autoencoder

Goal: Explain how autoencoders work and what they are used for
Context: autoencoder used in deepSAD
Method: explain basic idea, unfixed architecture, infomax, mention usecases, dimension reduction
Transition: dimensionality reduction, useful for high dim data → pointclouds from lidar

Autoencoders are a type of neural network architecture, whose main goal is learning to encode input data into a representative state, from which the same input can be reconstructed, hence the name. They typically consist of two functions, an encoder and a decoder with a latent space inbetween them as depicted in the toy example in figure 2.4. The encoder learns to extract the most significant features from the input and to convert them into the input's latent space representation. The reconstruction goal ensures that the most prominent features of the input get retained during the encoding phase, due to the inherent inability to reconstruct the input if

too much relevant information is missing. The decoder simultaneously learns to reconstruct the original input from its encoded latent space representation, by minimizing the error between the input sample and the autoencoder's output. This optimization goal creates uncertainty when categorizing autoencoders as an unsupervised method, although literature commonly defines them as such. While they do not require any labeling of the input data, their optimization target can still calculate the error between the output and the optimal target, which is typically not available for unsupervised methods. For this reason, they are sometimes proposed to be a case of self-supervised learning, a type of machine learning where the data itself can be used to generate a supervisory signal without the need for a domain expert to provide one.

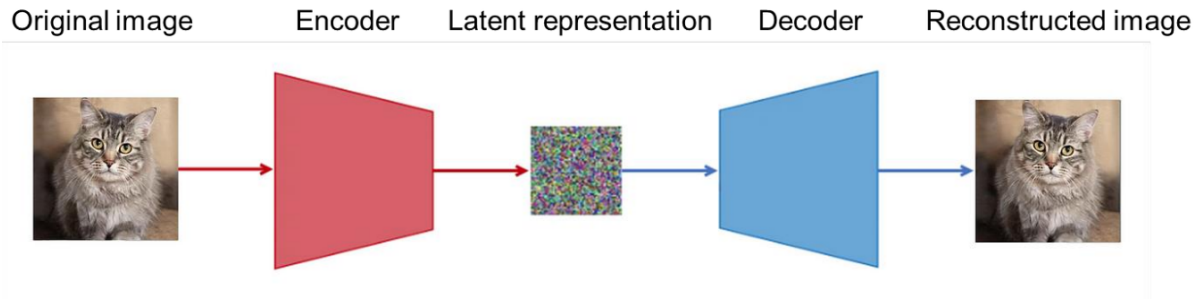


Figure 2.4: PLACEHOLDER - An illustration of autoencoders' general architecture and reconstruction task.

Paragraph about Variational Autoencoders? generative models vs discriminative models, enables other common use cases such as generating new data by changing parameterized generative distribution in latent space - VAES are not really relevant, maybe leave them out and just mention them shortly, with the hint that they are important but too much to explain since they are not key knowledge for this thesis

One key use case of autoencoders is to employ them as a dimensionality reduction technique. In that case, the latent space inbetween the encoder and decoder is of a lower dimensionality than the input data itself. Due to the aforementioned reconstruction goal, the shared information between the input data and its latent space representation is maximized, which is known as following the infomax principle. After training such an autoencoder, it may be used to generate lower-dimensional representations of the given datatype, enabling more performant computations which may have been infeasible to achieve on the original data. DeepSAD uses an autoencoder in a pre-training step to achieve this goal among others.

Autoencoders have been shown to be useful in the anomaly detection domain by assuming that autoencoders trained on more normal than anomalous data are better at reconstructing normal behaviour than anomalous one. This assumption allows methods to utilize the reconstruction error as an anomaly score. Examples of this are the method in "Outlier Detection with Autoencoder Ensembles" [11] or the one in "Memorizing Normality to Detect Anomaly: Memory-Augmented Deep Autoencoder for Unsupervised Anomaly Detection" [12] which both employ an autoencoder and the aforementioned assumption. Autoencoders have also been shown to be a suitable dimensionality reduction technique for lidar data, which is oftentimes high-dimensional and sparse, making feature extraction and dimensionality reduction popular preprocessing steps. As an example, "Deep Learning Approach for Building Detection Using LiDAR–Orthophoto Fusion" [13] shows the feasibility and advantages of using an autoencoder architecture to reduce lidar-orthophoto fused feature's dimensionality for their building detection method, which can recognize buildings in visual data taken from an airplane. Similarly, we can make use of the dimensionality reduction in DeepSAD's pre-training step, since our method is intended to work with high-dimensional lidar data.

2.4 Lidar - Light Detection and Ranging

Goal: Explain how lidars work and what data they produce
Context: understand why data is degraded, and how data looks
Method: explain how radar/lidar works, usecases, output = pointclouds, what errors
Transition: rain degradation paper used deepsad → explained in detail in next chapter

LiDAR (Light Detection and Ranging) measures distance by emitting short laser pulses and timing how long they take to return, an approach many may be familiar with from the more commonly known radar technology, which uses radio-frequency pulses and measures their return time to gauge an object's range. Unlike radar, however, LiDAR operates at much shorter wavelengths and can fire millions of pulses per second, achieving millimeter-level precision and dense, high-resolution 3D point clouds. This fine granularity makes LiDAR ideal for applications such as detailed obstacle mapping, surface reconstruction, and autonomous navigation in complex environments.

Because the speed of light in air is effectively constant, multiplying half the round-trip time by that speed gives the distance between the lidar sensor and the reflecting object, as can be seen visualized in figure 2.5. Modern spinning multi-beam LiDAR systems emit millions of these pulses every second. Each pulse is sent at a known combination of horizontal and vertical angles, creating a regular grid of measurements: for example, 32 vertical channels swept through 360° horizontally at a fixed angular spacing. While newer solid-state designs (flash, MEMS, phased-array) are emerging, spinning multi-beam LiDAR remains the most commonly seen type in autonomous vehicles and robotics because of its proven range, reliability, and mature manufacturing base.

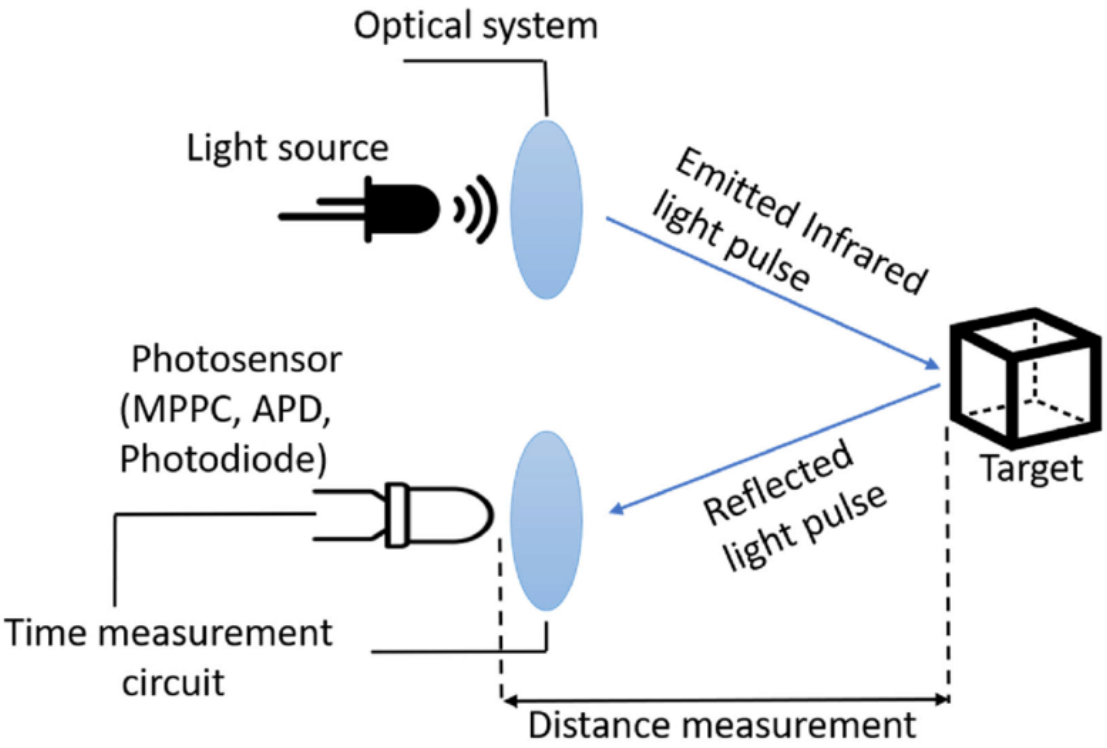


Figure 2.5: PLACEHOLDER - An illustration of lidar sensors' working principle.

Each instance a lidar emits and receives a laser pulse, it can use the ray's direction and the calculated distance to produce a single three-dimensional point. By collecting millions of such points each second, the sensor constructs a "point cloud"—a dense set of 3D coordinates relative

to the LiDAR's own position. In addition to X, Y, and Z, many LiDARs also record the intensity or reflectivity of each return, providing extra information about the surface properties of the object hit by the pulse.

LiDAR's high accuracy, long range, and full-circle field of view make it indispensable for tasks like obstacle detection, simultaneous localization and mapping (SLAM), and terrain modeling in autonomous driving and mobile robotics. While complementary sensors—such as time-of-flight cameras, ultrasonic sensors, and RGB cameras—have their strengths at short range or in particular lighting, only LiDAR delivers the combination of precise 3D measurements over medium to long distances, consistent performance regardless of illumination, and the pointcloud density needed for safe navigation. LiDAR systems do exhibit intrinsic noise (e.g., range quantization or occasional multi-return ambiguities), but in most robotic applications these effects are minor compared to environmental degradation.

In subterranean and rescue domain scenarios, the dominant challenge is airborne particles: dust kicked up by debris or smoke from fires. These aerosols create early returns that can mask real obstacles and cause missing data behind particle clouds, undermining SLAM and perception algorithms designed for cleaner data. This degradation is a type of atmospheric scattering, which can be caused by any kind of airborne particulates (e.g., snowflakes) or liquids (e.g., water droplets). Other kinds of environmental noise exist as well, such as specular reflections caused by smooth surfaces, beam occlusion due to close objects blocking the sensor's field of view or even thermal drift-temperature affecting the sensor's circuits and mechanics, introducing biases in the measurements.

All of these may create unwanted noise in the point cloud created by the lidar, making this domain an important research topic. "LiDAR Denoising Methods in Adverse Environments: A Review" [14] gives an overview about the current state of research into denoising methods for lidar in adverse environments, categorizes them according to their approach (distance-, intensity- or learning-based) and concludes that all approaches have merits but also open challenges to solve, for autonomous systems to safely navigate these adverse environments. The current research is heavily focused on the automotive domain, which can be observed by the vastly higher number of methods filtering noise from adverse weather effects-environmental scattering from rain, snow and fog-than from dust, smoke or other particles occuring rarely in the automotive domain.

A learning-based method to filter dust-caused degradation from lidar is introduced in "Dust De-Filtering in LiDAR Applications With Conventional and CNN Filtering Methods" [15]. The authors employ a convultional neural network to classify dust particles in lidar point clouds as such, enabling the filtering of those points and compare their methods to more conservative approaches, such as various outlier removal algorithms. Another relevant example would be the filtering method proposed in "Efficient Real-time Smoke Filtration with 3D LiDAR for Search and Rescue with Autonomous Heterogeneous Robotic Systems" [16], which enables the filtration of pointclouds degraded by smoke or dust in subterranean environments, with a focus on the search and rescue domain. To achieve this, they formulated a filtration framework that relies on dynamic onboard statistical cluster outlier removal, to classify and remove dust particles in point clouds.

Our method does not aim to remove the noise or degraded points in the lidar data, but quantify its degradation to inform other systems of the autonomous robot about the data's quality, enabling more informed decisions. One such approach, though from the autonomous driving and not from the search and rescue domain can be found in "LiDAR Degradation Quantification for Autonomous Driving in Rain" [17]. A learning-based method to quantify the lidar's sensor data degradation caused by adverse weather-effects was proposed, implemented by posing the problem as an anomaly detection task and utilizing DeepSAD to learn degraded data to be an anomaly and high quality data to be normal behaviour. DeepSAD's anomaly score was used as the degradation's quantification score. From this example we decided to imitate this method and adapt it for the search and rescue domain, although this proved challenging

due to the more limited data availability. Since it was effective for the closely related "LiDAR Degradation Quantification for Autonomous Driving in Rain" [17], we also employed DeepSAD, whose detailed workings we present in the following chapter.

3

Deep SAD: Semi-Supervised Anomaly Detection

Goal: Introduce DeepSAD, how and why do we use it
Context: let reader know why they need to know about Deepsad in detail
Method: explain use-case, similar use-case worked, allude to core features
Transition: interest/curiosity created → wants to learn about DeepSAD

In this chapter, we explore the method "Deep Semi-Supervised Anomaly Detection" (Deep SAD) [6], which we employ to quantify the degradation of LiDAR scans caused by airborne particles in the form of artificially introduced water vapor from a theater smoke machine. A similar approach—modeling degradation quantification as an anomaly detection task—was successfully applied in "LiDAR Degradation Quantification for Autonomous Driving in Rain" [17] to assess the impact of adverse weather conditions on LiDAR data for autonomous driving applications. Deep SAD leverages deep learning to capture complex anomalous patterns that classical statistical methods might miss. Furthermore, by incorporating a limited amount of hand-labeled data (both normal and anomalous), it can more effectively differentiate between known anomalies and normal data compared to purely unsupervised methods, which typically learn only the most prevalent patterns in the dataset [6].

3.1 Algorithm Description

Goal: give general overview about how it works
Context: overview helps reader understand method, then go into detail
Method: how clustering AD generally works, how it does in DeepSAD
Transition: since the reader knows the general idea → what is the step-by-step?

Deep SAD's overall mechanics are similar to clustering-based anomaly detection methods, which according to "Anomaly detection: A survey" [5] typically follow a two-step approach. First, a clustering algorithm groups data points around a centroid; then, the distances of individual data points from this centroid are calculated and used as an anomaly score. In Deep SAD, these concepts are implemented by employing a neural network, which is jointly trained to map input data onto a latent space and to minimize the volume of an data-encompassing hypersphere, whose center is the aforementioned centroid. The data's geometric distance in the latent space to the hypersphere center is used as the anomaly score, where a larger distance between data and centroid corresponds to a higher probability of a sample being anomalous. This is achieved by shrinking the data-encompassing hypersphere during training, proportionally to all training data, of which is required that there is significantly more normal than anomalous data present. The outcome of this approach is that normal data gets clustered more closely around the centroid, while anomalies appear further away from it as can be seen in the toy example depicted in figure 3.1.

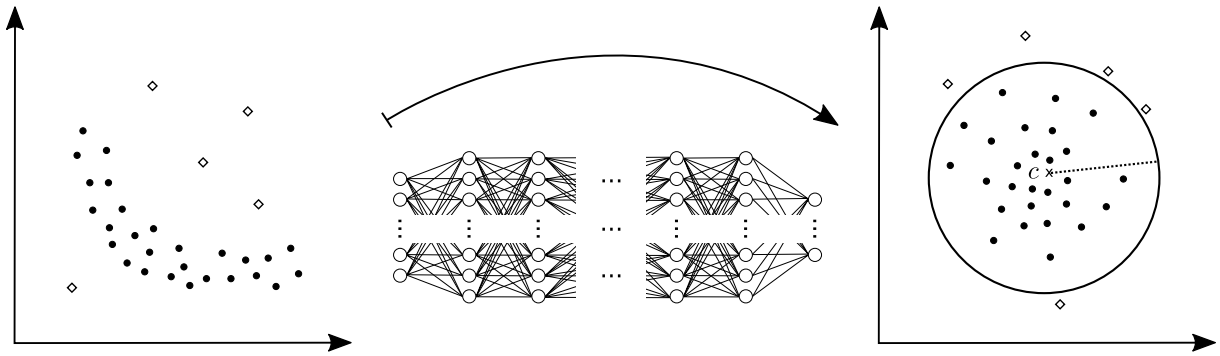


Figure 3.1: DeepSAD teaches a neural network to transform data into a latent space and minimize the volume of an data-encompassing hypersphere centered around a predetermined centroid $\mathbf{c}$. *Reproduced from [18].*

Goal: first step is pre-training, why does it use an autoencoder?
Context: go chronologically over how the algorithm works. starting at pre-training
Method: pre-training is autoencoder, self-supervised, dimensionality reduction
Transition: pre-training done → how are the pre-training results used?

Before DeepSAD's training can begin, a pre-training step is required, during which an autoencoder is trained on all available input data. One of DeepSAD's goals is to map input data onto a lower dimensional latent space, in which the separation between normal and anomalous data can be achieved. To this end DeepSAD and its predecessor Deep SVDD make use of the autoencoder's reconstruction goal, whose successful training ensures confidence in the encoder architecture's suitability for extracting the input datas' most prominent information to the latent space inbetween the encoder and decoder. DeepSAD goes on to use just the encoder as its main network architecture, discarding the decoder at this step, since reconstruction of the input is unnecessary.

Goal: what is pre-training output used for, how is centroid calculated and why
Context: reader knows about pre-training, what are next steps and how is it used
Method: pre-training weights used to init main network, c is mean of forward pass, collapse
Transition: network built and initialized, centroid fixed → start main training

The pre-training results are used in two more key ways. First, the encoder weights obtained from the autoencoder pre-training initialize DeepSAD's network for the main training phase. Second, we perform an initial forward pass through the encoder on all training samples, and the mean of these latent representations is set as the hypersphere center, $\mathbf{c}$. According to Ruff et al., this initialization method leads to faster convergence during the main training phase compared to using a randomly selected centroid. An alternative would be to compute $\mathbf{c}$ using only the labeled normal examples, which would prevent the center from being influenced by anomalous samples; however, this requires a sufficient number of labeled normal samples. Once defined, the hypersphere center $\mathbf{c}$ remains fixed, as allowing it to be optimized freely could in the unsupervised case lead to a hypersphere collapse-a trivial solution where the network learns to map all inputs directly onto the centroid $\mathbf{c}$.

Goal: how does the main training work, what data is used, what is the optimization target
Context: main training is next step since all preconditions are met
Method: main training is SGD backpropagation, minimizing volume, un-/labeled data used
Transition: network is trained → how does one use it for AD?

In the main training step, DeepSAD's network is trained using SGD backpropagation. The unlabeled training data is used with the goal to minimize an data-encompassing hypersphere. Since one of the pre-conditions of training was the significant prevelance of normal data over anomalies in the training set, normal samples collectively cluster more tightly around the centroid, while the rarer anomalous samples do not contribute as significantly to the optimization,

resulting in them staying further from the hypersphere center. The labeled data includes binary class labels signifying their status as either normal or anomalous samples. Labeled anomalies are pushed away from the center by defining their optimization target as maximizing the distance between them and **c**. Labeled normal samples are treated similar to unlabeled samples with the difference that DeepSAD includes a hyperparameter capable of controling the proportion with which labeled and unlabeled data contribute to the overall optimization. The resulting network has learned to map normal data samples closer to **c** in the latent space and anomalies further away.

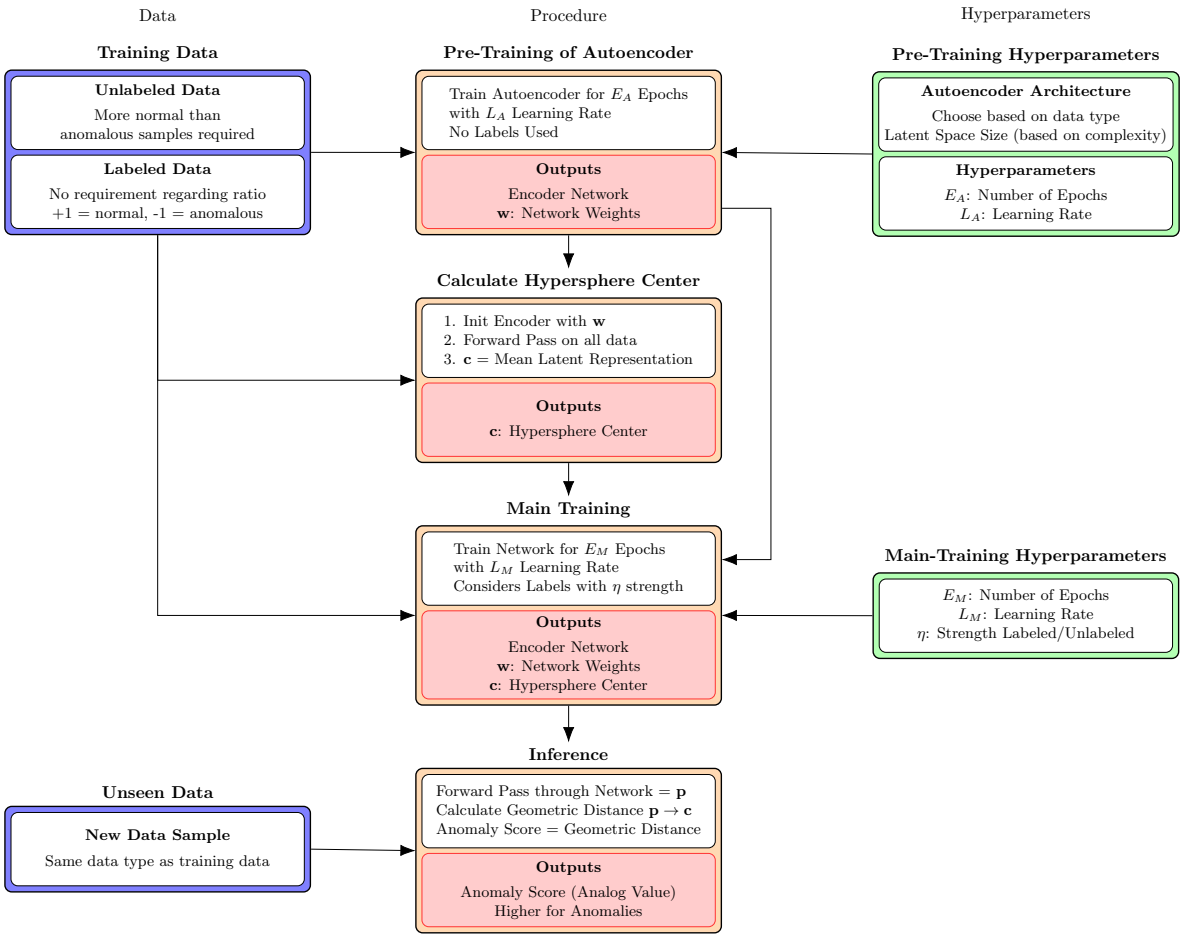


Figure 3.2: (WORK IN PROGRESS) Depiction of DeepSAD's training procedure, including data flows and tweakable hyperparameters.

Goal: how to use the trained network?
Context: since we finished training, we need to know how to utilize it
Method: forward pass, calculate distance from c = anomaly score, analog, unknown magnitude
Transition: General knowledge of the algorithm achieved → go into more detail

To infer if a previously unknown data sample is normal or anomalous, the sample is fed in a forward-pass through the fully trained network. During inference, the centroid **c** needs to be known, to calculate the geometric distance of the samples latent representation to **c**. This distance is tantamount to an anomaly score, which correlates with the likelihood of the sample being anomalous. Due to differences in input data type, training success and latent space dimensionality, the anomaly score's magnitude has to be judged on an individual basis for each trained network. This means, scores produced by one network that signify normal data, may very well clearly indicate an anomaly for another network. The geometric distance between two

points in space is a scalar analog value, therefore post-processing of the score is necessary to achieve a binary classification of normal and anomalous if desired.

DeepSAD's full training and inference procedure is visualized in figure 3.2, which gives a comprehensive overview of the dataflows, tuneable hyperparameters and individual steps involved.

3.2 Algorithm Details and Hyperparameters

Since Deep SAD is heavily based on its predecessor "Deep One-Class Classification" (Deep SVDD) [18] it is helpful to first understand Deep SVDD's optimization objective, so we start with explaining it here. For input space $\mathcal{X} \subseteq \mathbb{R}^D$, output space $\mathcal{Z} \subseteq \mathbb{R}^d$ and a neural network $\phi(\cdot; \mathcal{W}) : \mathcal{X} \to \mathcal{Z}$ where $\mathcal{W}$ depicts the neural networks' weights with L layers $\{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$, n the number of unlabeled training samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $\mathbf{c}$ the center of the hypersphere in the latent space, Deep SVDD teaches the neural network to cluster normal data closely together in the latent space by defining its optimization objective as seen in 3.1.

$$\min_{\mathcal{W}} \quad \frac{1}{n} \sum_{i=1}^{n} \|\phi(\mathbf{x}_i; \mathcal{W}) - \mathbf{c}\|^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} \|\mathbf{W}^\ell\|_F^2. \tag{3.1}$$

As can be seen from 3.1, Deep SVDD is an unsupervised method which does not rely on labeled data to train the network to differentiate between normal and anomalous data. The first term of the optimization objective depicts the shrinking of the data-encompassing hypersphere around the given center $\mathbf{c}$. For each data sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, its geometric distance to $\mathbf{c}$ in the latent space produced by the neural network $\phi(\cdot; \mathcal{W})$ is minimized proportionally to the amount of data samples n. The second term is a standard L2 regularization term which prevents overfitting with hyperparameter $\lambda > 0$ and $\|\cdot\|_F$ denoting the Frobenius norm.

Ruff et al. argue that the pre-training step employing an autoencoder—originally introduced in Deep SVDD—not only allows a geometric interpretation of the method as minimum volume estimation i.e., the shrinking of the data encompassing hypersphere but also a probabilistic one as entropy minimization over the latent distribution. The autoencoding objective during pre-training implicitly maximizes the mutual information between the data and its latent representation, aligning the approach with the Infomax principle while encouraging a latent space with minimal entropy. This insight enabled Ruff et al. to introduce an additional term in DeepSAD's objective, beyond that of its predecessor Deep SVDD, which incorporates labeled data to better capture the characteristics of normal and anomalous data. They demonstrate that DeepSAD's objective effectively models the latent distribution of normal data as having low entropy, while that of anomalous data is characterized by higher entropy. In this framework, anomalies are interpreted as being generated from an infinite mixture of distributions that differ from the normal data distribution.

The introduction of the aforementioned term in Deep SAD's objective allows it to learn in a semi-supervised way, though it can operate in a fully unsupervised mode—effectively reverting to its predecessor, Deep SVDD—when no labeled data are available. This additional supervision helps the model better position known normal samples near the hypersphere center and push known anomalies farther away, thereby enhancing its ability to differentiate between normal and anomalous data.

From 3.1 it is easy to understand Deep SAD's optimization objective seen in 3.2 which additionally defines m number of labeled data samples $\{(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_m, \tilde{y}_1)\} \in \mathcal{X} \times \mathcal{Y}$ and $\mathcal{Y} = \{-1, +1\}$ for which $\tilde{y} = +1$ denotes normal and $\tilde{y} = -1$ anomalous samples as well as a new hyperparameter $\eta > 0$ which can be used to balance the strength with which labeled and unlabeled samples contribute to the training.

$$\min_{\mathcal{W}} \quad \frac{1}{n+m}\sum_{i=1}^{n}\|\phi(\mathbf{x}_i;\mathcal{W})-\mathbf{c}\|^2 + \frac{\eta}{n+m}\sum_{j=1}^{m}\left(\|\phi(\tilde{\mathbf{x}}_j;\mathcal{W})-\mathbf{c}\|^2\right)^{\tilde{y}_j} + \frac{\lambda}{2}\sum_{\ell=1}^{L}\|\mathbf{W}^\ell\|_F^2. \quad (3.2)$$

The first term of 3.2 stays mostly the same, differing only in its consideration of the introduced m labeled datasamples for its proportionality. The second term is newly introduced to incorporate the labeled data samples with hyperparameter η's strength, by either minimizing or maximizing the distance between the samples latent represenation and $\mathbf{c}$ depending on each data samples label $\tilde{y}$. The third term, is kept identical compared to Deep SVDD as standard L2 regularization. It can also be observed that in case of $m = 0$ labeled samples, Deep SAD falls back to the same optimization objective of Deep SVDD and can therefore be used in a completely unsupervised fashion as well.

Hyperparameters

The neural network architecture of DeepSAD is not fixed but rather dependent on the data type the algorithm is supposed to operate on. This is due to the way it employs an autoencoder for pretraining and the encoder part of the network for its main training step. This makes the adaption of an autoencoder architecture suitable to the specific application necessary but also allows for flexibility in choosing a fitting architecture depending on the application's requirements. For this reason the specific architecture employed, may be considered a hyperparameter of the Deep SAD algorithm.

latent space size, talk about auto encoder performance, trying out sensible dimensionalities and find reconstruction elbow, choose smallest possible, but as large as necessary

latent space size for AE shows that most likely all of the important data may be captured inside this dim (since recons;truction is possible) but we may only require some of the encoded patterns to differentiate normal from anomaly so smaller may still be possible? should this be discussed here or not? maybe only discuss; AE considerations and then move this discussion to discussion / results

eta, think of possible important scenarios, learning rate, epochs

4 Data and Preprocessing

Goal: Introduce data chapter, what will be covered here, incite interest
Context: all background covered, deepsad explained, data natural next step
Method: emotional why data scarce, lot of data necessary, what will be covered
Transition: what will we talk about next → requirements

Situations such as earthquakes, structural failures, and other emergencies that require rescue robots are fortunately rare. When these operations do occur, the primary focus is on the rapid and safe rescue of survivors rather than on data collection. Consequently, there is a scarcity of publicly available data from such scenarios. To improve any method, however, a large, diverse, and high-quality dataset is essential for comprehensive evaluation. This challenge is further compounded in our work, as we evaluate a training-based approach that imposes even higher demands on the data, especially requiring a great deal of diverse training samples, making it difficult to find a suitable dataset.

In this chapter, we outline the specific requirements we established for the data, describe the dataset selected for this task—including key statistics and notable features—and explain the preprocessing steps applied for training and evaluating the methods.

4.1 Data Requirements and Challenges

Goal: list requirements we had for data
Context: what were our requirements for choosing a dataset
Method: list from basic to more complex with explanations
Transition: ground truth for evaluation → ground truth/labeling challenges

To ensure our chosen dataset meets the needs of reliable degradation quantification in subterranean rescue scenarios, we imposed the following requirements:

1. **Data Modalities:**
   The dataset must include LiDAR sensor data, since we decided to train and evaluate our method on what should be the most universally used sensor type in the given domain. To keep our method as generalized as possible, we chose to only require range-based point cloud data and forego sensor-specific data such as intensity or reflectivity, though it may be of interest for future work. It is also desirable to have complementary visual data such as camera images, for better context, manual verification and understanding of the data.

2. **Context & Collection Method:**
   To mirror the real-world conditions of autonomous rescue robots, the data should originate from locations such as subterranean environments (tunnels, caves, collapsed structures), which closely reflect what would be encountered during rescue missions. Ideally, it should be captured from a ground-based, self-driving robot platform in motion instead of aerial, handheld, or stationary collection, to ensure similar circumstances to the target domain.

3. **Degradation Characteristics:**
   Because our goal is to quantify the degradation of lidar data encountered by rescue robots,

the dataset must exhibit significant degradation of LiDAR returns from aerosols (i.e., dust or smoke particles), which should be the most frequent and challenging degradation encountered. This requirement is key to evaluating how well our method detects and measures the severity of such challenging conditions.

4. **Volume & Class Balance:**
   The dataset must be large enough to train deep learning models effectively. Since our semi-supervised approach depends on learning a robust model of "normal" data, the majority of samples should be high-quality, degradation-free scans. Simultaneously, there must be a sufficient number of degraded (anomalous) scans to permit a comprehensive evaluation of quantification performance.

5. **Ground-Truth Labels:**
   Finally, to evaluate and tune our method, we need some form of ground truth indicating which scans are degraded. Obtaining reliable labels in this domain is challenging. Manual annotation of degradation levels is laborious and somewhat subjective. We address these challenges and issues of labeling data for evaluation next.

Goal: What are the challenges for correctly labeling the data
Context: we alluded to labels being challenging before this
Method: difficult to define degradation, difficult to capture, objective leads to puppet
Transition: with all requirements and challenges know → what dataset did we choose

Quantitative benchmarking of degradation quantification requires a degradation label for every scan. Ideally that label would be a continuous degradation score, although a binary label would still enable meaningful comparison. As the rest of this section shows, producing any reliable label is already challenging and assigning meaningful analog scores may not be feasible at all. Compounding the problem, no public search-and-rescue (SAR) LiDAR data set offers such ground truth as far as we know. To understand the challenges around labeling lidar data degradation, we will look at what constitutes degradation in this context.

In section 2.4 we discussed some internal and environmental error causes of lidar sensors, such as multi-return ambiguities or atmospheric scattering respectively. While we are aware of research into singular failure modes, such as "Modelling of LIDAR sensor disturbances by solid airborne particles" [19] or research trying to model the totality of error souces occuring in other domains, such as ."A Survey on Sensor Failures in Autonomous Vehicles: Challenges and Solutions" [20], there appears to be no such model for the search and rescue domain and its unique environmental circumstances. Although, scientific consensus appears to be, that airborne particles are the biggest contributor to degradation in SAR [21], we think that a more versatile definition is required to ensure confidence during critical SAR missions, which are often of a volatile nature. We are left with an ambiguous definition of what constitutes lidar point cloud degradation in the SAR domain.

We considered which types of objective measurements may be available to produce ground-truth labels, such as particulate matter sensors, lidar point clouds' inherent properties such as range-dropout rate and others, but we fear that using purely objective measures to label the data, would limit our learning based method to imitating the labels' sources instead of differentiating all possible degradation patterns from high quality data. Due to the incomplete error model in this domain, there may be novel or compound error sources that would not be captured using such an approach. As an example, we did observe dense smoke reflecting enough rays to produce phantom objects, which may fool SLAM algorithms. Such a case may even be labeleled incorrectly as normal by one of the aforementioned objective measurement labeling options, if the surroundings do not exhibit enough dispersed smoke particles already.

To mitigate the aforementioned risks we adopt a human-centric, binary labelling strategy. We judged analog and multi-level discrete rating scales to be too subjective for human consideration, which only left us with the simplistic, but hopefully more reliable binary choice. We used

two labeling approaches, producing two evaluation sets, whose motivation and details will be discussed in more detail in section 4.3. Rationale for the exact labeling procedures requires knowledge of the actual dataset we ended up choosing, which we will present in the next section.

4.2 Chosen Dataset

Goal: give a comprehensive overview about chosen dataset
Context: all requirements/challenges are clear, now reader wants to know about subter dataset
Method: overview, domain, sensors, lidar, experiments, volume, statistics
Transition: statistics about degradation, not full picture → how did we preprocess and label

Based on the previously discussed requirements and the challenges of obtaining reliable labels, we selected the "Multimodal Dataset from Harsh Sub-Terranean Environment with Aerosol Particles for Frontier Exploration" [22] for training and evaluation. This dataset comprises multimodal sensor data collected from a robotic platform navigating tunnels and rooms in a subterranean environment, an underground tunnel in Luleå, Sweden. Notably, some experiments incorporated an artificial smoke machine to simulate heavy degradation from aerosol particles, making the dataset particularly well-suited to our use case. A Pioneer 3-AT2 robotic platform, which can be seen in figure 4.1(a), was used to mount a multitude of sensors that are described in table 4.1 and whose mounting locations are depicted in figure 4.1(b).

todo: check table for accuracy/errors

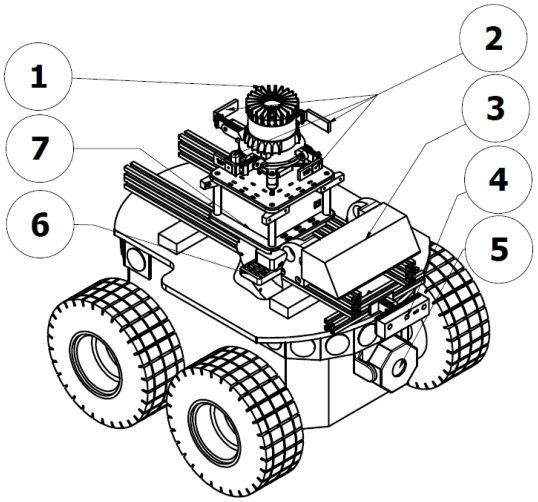
Table 4.1: On–board sensors recorded in the "Multimodal Dataset from Harsh Sub-Terranean Environment with Aerosol Particles for Frontier Exploration" dataset. Numbers match the labels in Fig. 4.1; only the most salient details are shown for quick reference.

#	Sensor	Recorded Data	Key Specs
1	Spinning 3-D LiDAR *Ouster OS1-32*	3-D cloud, reflectivity	10 Hz, 32 ch, 360° × 42.4°, ≤ 120 m
2	mm-wave RADAR (×4) *TI IWR6843AoP*	4 × 60° RADAR point clouds	30 Hz, 60 GHz, 9 m max, 0.05 m res.
3	Solid-state LiDAR *Velodyne Velarray M1600*	Forward LiDAR cloud	10 Hz, 160 ch, 120° × 32°, 0.1–30 m
4	RGB-D / stereo cam *Luxonis OAK-D Pro*	stereo b/w images, depth map	15 fps, 75 mm baseline, active IR 930 nm
5	LED flood-light *RS PRO WL28R*	Illumination for stereo cam	7 W, 650 lm (no data stream)
6	IMU *Pixhawk 2.1 Cube Orange*	Accel, gyro, mag, baro	190 Hz, 9-DoF, vibration-damped
7	On-board PC *Intel NUC i7*	Time-synced logging	Quad-core i7, 16 GB RAM, 500 GB SSD

We use data from the *Ouster OS1-32* LiDAR sensor, which was configured to capture 10 frames per second with a resolution of 32 vertical channels and 2048 measurements per channel. These settings yield equiangular measurements across a vertical field of view of 42.4° and a complete 360° horizontal field of view. Consequently, every LiDAR scan can generate up to 65,536 points. Each point contains the X, Y, and Z coordinates (in meters, with the sensor location as the origin) along with values for *range*, *intensity*, and *reflectivity*—typical metrics measured by LiDAR sensors. The datasets' point clouds are saved in a dense format, meaning each of the 65,536 measurements is present in the data, although fields for missing measurements contain zeroes.

During the measurement campaign, a total of 14 experiments were conducted—10 prior to operating the artificial smoke machine (hereafter referred to as normal experiments) and 4 after it has already been running for some time (anomalous experiments). In 13 of these experiments,

(a) Pioneer 3-AT2 mobile base carrying the sensor tower. The four-wheel, skid-steered platform supports up to 30 kg payload and can negotiate rough terrain—providing the mobility required for subterranean data collection.



(b) Sensor layout and numbering. Components: 1 OS1-32 LiDAR, 2 mm-wave RADARs, 3 M1600 LiDAR, 4 OAK-D Pro camera, 5 LED flood-light, 6 IMU, 7 Intel NUC. See Table 4.1 for detailed specifications.

Figure 4.1: Robotic platform and sensor configuration used to record the dataset.

the sensor platform was in near-constant motion (either translating at roughly 1m/s or rotating), with only one anomalous experiment conducted while the platform remained stationary. Although this means we do not have two stationary experiments from the same exact position for a direct comparison between normal and anomalous conditions, the overall experiments are similar enough to allow for meaningful comparisons. In addition to the presence of water vapor from the smoke machine, the experiments vary in illumination conditions, the presence of humans on the measurement grounds, and additional static artifacts. For our purposes, only the artificial smoke is relevant; differences in lighting or incidental static objects do not affect our analysis. Regardless of illumination, the LiDAR sensor consistently produces comparable point clouds, and the presence of static objects does not influence our quantification of point cloud degradation.

In the anomalous experiments, the artificial smoke machine appears to have been running for some time before data collection began, as evidenced by both camera images and LiDAR data showing an even distribution of water vapor around the machine. The stationary experiment is particularly unique: the smoke machine was positioned very close to the sensor platform and was actively generating new, dense smoke, to the extent that the LiDAR registered the surface of the fresh water vapor as if it were a solid object.

The figures 4.2 and 4.3 show an representative depiction of the environment of the experiments as a camera image of the IR camera and the point cloud created by the OS1 lidar sensor at practically the same time.

Regarding the dataset volume, the 10 normal experiments ranged from 88.7 to 363.1 seconds, with an average duration of 157.65 seconds. At a capture rate of 10 frames per second, these experiments yield 15,765 non-degraded point clouds. In contrast, the 4 anomalous experiments, including one stationary experiment lasting 11.7 seconds and another extending to 62.1 seconds, averaged 47.33 seconds, resulting in 1,893 degraded point clouds. In total, the dataset comprises 17,658 point clouds, with approximately 89.28% classified as non-degraded (normal) and 10.72% as degraded (anomalous). The distribution of experimental data is visualized in figure 4.4.

The artificial smoke introduces measurable changes that clearly separate the *anomalous* runs from the *normal* baseline. One change is a larger share of missing points per scan: smoke parti-

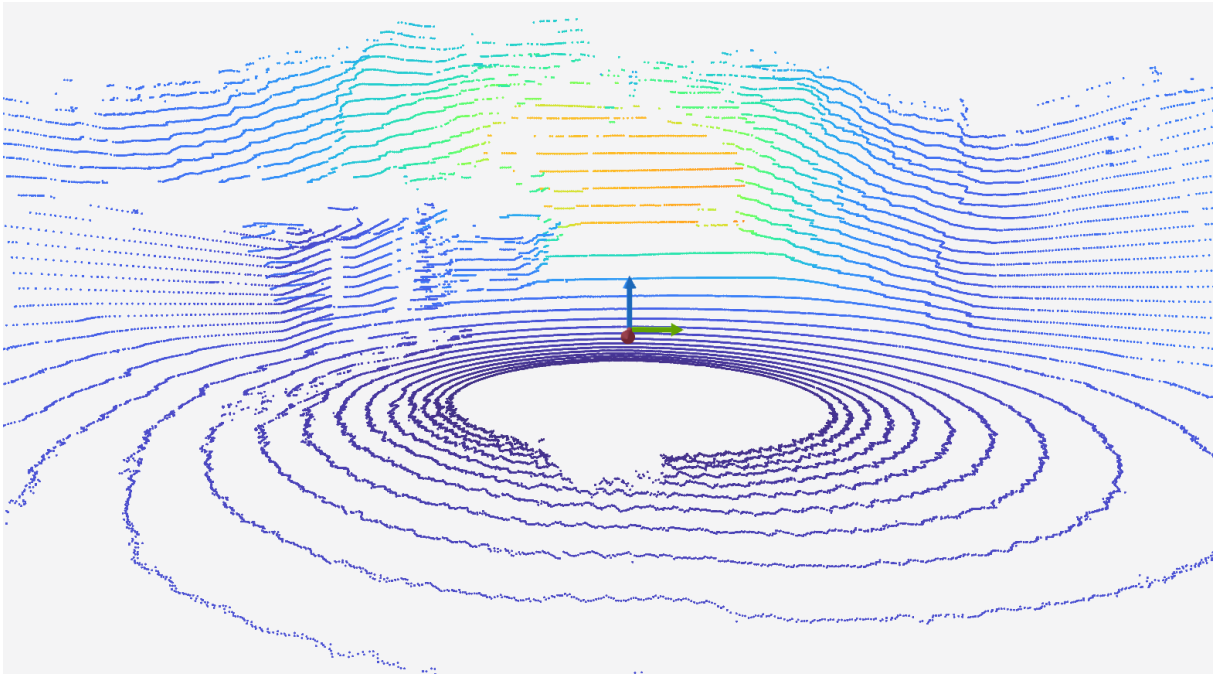


Figure 4.2: Screenshot of 3D rendering of an experiment without smoke and with illumination (same frame and roughly same alignment as figure 4.3). Point color corresponds to measurement range and axis in center of figure is the lidar's position.

Figure 4.3: Screenshot of IR camera output of an experiment without smoke and with illumination (same frame and roughly same alignment as figure 4.2)

cles scatter or absorb the laser beam before it reaches a solid target, so the sensor reports an error instead of a distance. Figure 4.5 shows the resulting right–shift of the missing-point histogram, a known effect for lidar sensors in aerosol-filled environments [23]. Another demonstrative effect is the appearance of many spurious returns very close to the sensor; these near-field points arise when back-scatter from the aerosol itself is mistaken for a surface echo. The box-plot in Fig. 4.6 confirms a pronounced increase in sub-50 cm hits under smoke, consistent with the behaviour

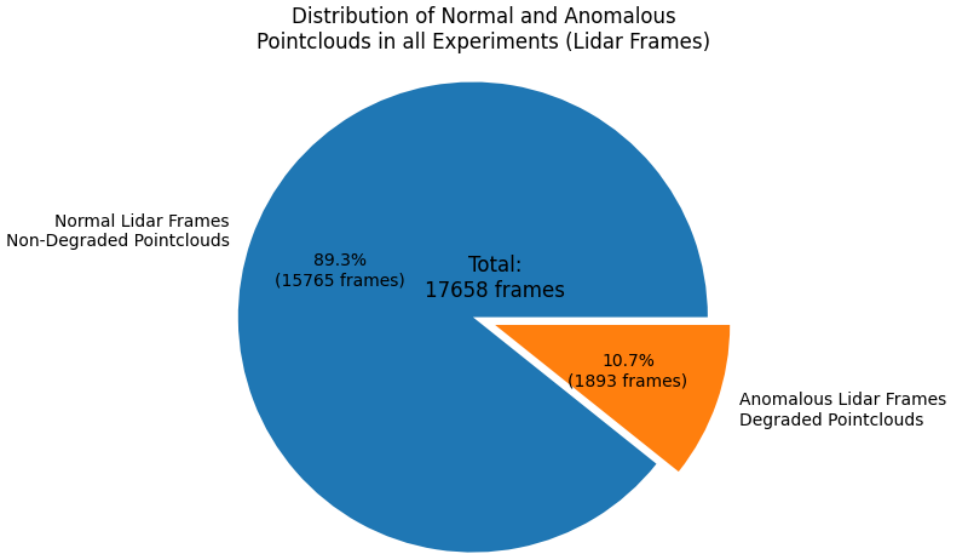


Figure 4.4: Pie chart visualizing the amount and distribution of normal and anomalous point clouds in [22]

reported in "When the Dust Settles: The Four Behaviors of LiDAR in the Presence of Fine Airborne Particulates" [23].

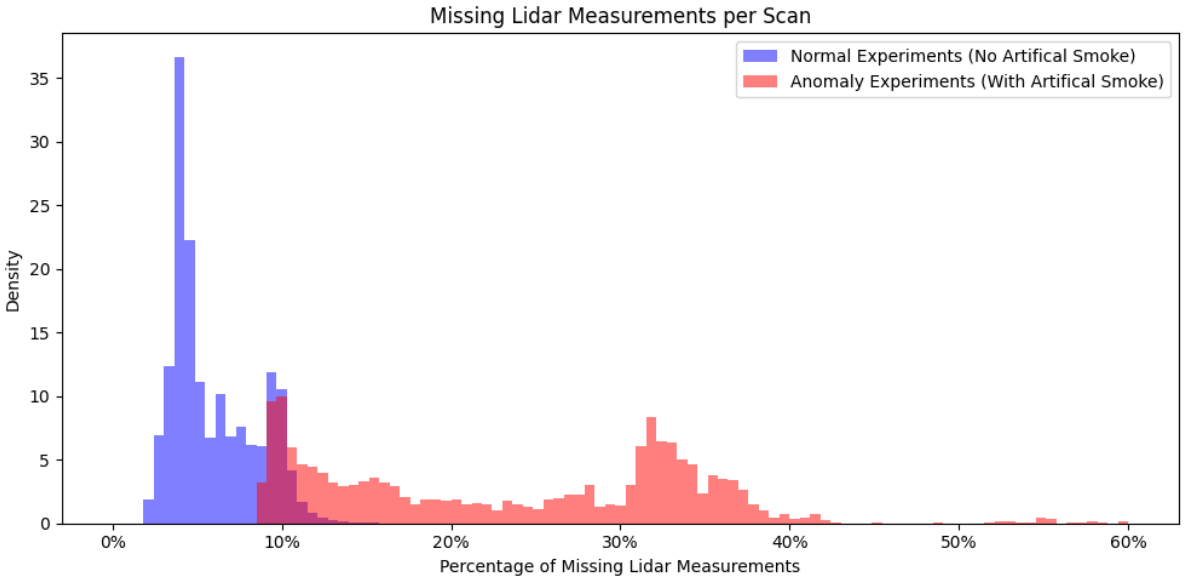


Figure 4.5: Density histogram showing the percentage of missing measurements per scan for normal experiments without degradation and anomalous experiments with artifical smoke introduced as degradation.

Taken together, the percentage of missing points and the proportion of near-sensor returns provide a concise indication of how strongly the smoke degrades our scans—capturing the two most prominent aerosol effects, drop-outs and back-scatter spikes. They do not, however, reveal the full error landscape discussed earlier (compound errors, temperature drift, multipath, ...), so they should be read as an easily computed synopsis rather than an exhaustive measure of LiDAR quality. Next we will discuss how the lidar scans were preprocessed before use and how we actually assigned ground-truth labels to each scan, so we could train and evaluate our

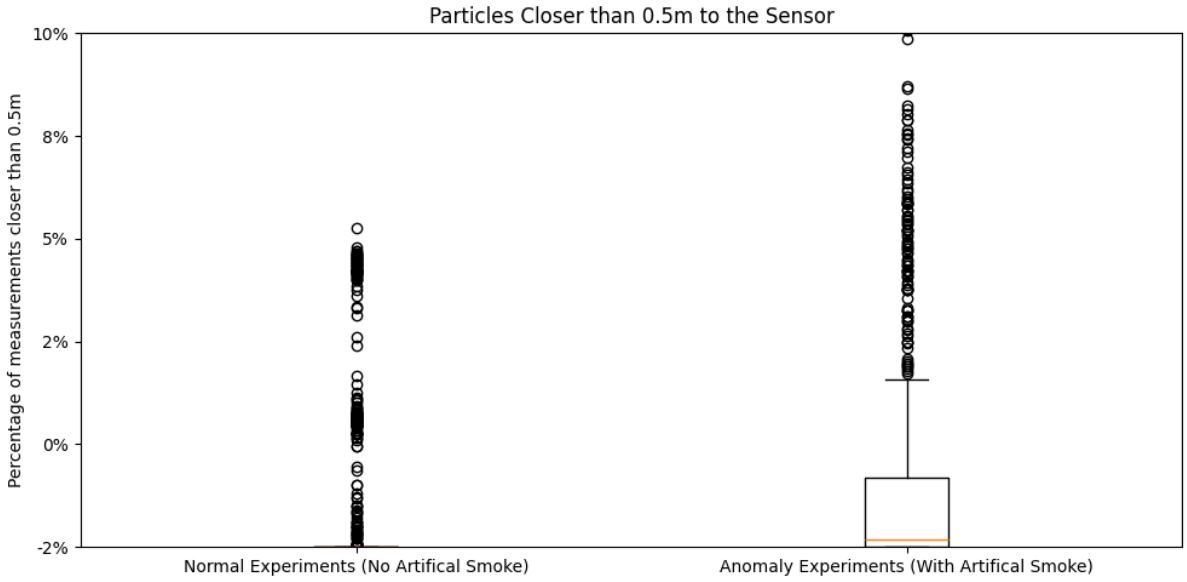


Figure 4.6: Box diagram depicting the percentage of measurements closer than 50 centimeters to the sensor for normal and anomalous experiments

quantification degradation methods.

4.3 Preprocessing Steps and Labeling

Goal: explain preprocessing and rationale
Context: raw dataset has been explained, how did we change the data before use
Method: projection because easier autoencoder, how does projection work, index-based
Transition: preprocessing known → how did we label the data for use

As described in Section 3.1, the method under evaluation is data type agnostic and can be adapted to work with any kind of data by choosing a suitable autoencoder architecture. In our case, the input data are point clouds produced by a lidar sensor. Each point cloud contains up to 65,536 points, with each point represented by its X, Y, and Z coordinates. To tailor the Deep SAD architecture to this specific data type, we would need to design an autoencoder suitable for processing three-dimensional point clouds. Although autoencoders can be developed for various data types, "A comprehensive survey on design and application of autoencoder in deep learning" [24] noted that over 60% of recent research on autoencoders focuses on two-dimensional image classification and reconstruction. Consequently, there is a more established understanding of autoencoder architectures for images compared to those for three-dimensional point clouds.

For this reason and to simplify the architecture, we converted the point clouds into two-dimensional grayscale images using a spherical projection. This approach—proven sucessful in related work [17]—encodes each LiDAR measurement as a single pixel, where the pixel's grayscale value is determined by the reciprocal range, calculated as $v = \frac{1}{\sqrt{X^2+Y^2+Z^2}}$. Given the LiDAR sensor's configuration, the resulting images have a resolution of 2048 pixels in width and 32 pixels in height. Missing measurements in the point cloud are mapped to pixels with a brightness value of $v = 0$.

To create this mapping, we leveraged the available measurement indices and channel information inherent in the dense point clouds, which are ordered from 0 to 65,535 in a horizontally ascending, channel-by-channel manner. For sparser point clouds without such indices, one would need to rely on the pitch and yaw angles relative to the sensor's origin to correctly map each point to its corresponding pixel.

Figure 4.7 displays two examples of LiDAR point cloud projections to aid in the reader's understanding. Although the original point clouds were converted into grayscale images with a resolution of 2048×32 pixels, these raw images can be challenging to interpret. To enhance human readability, we applied the viridis colormap and vertically stretched the images so that each measurement occupies multiple pixels in height. The top projection is derived from a scan without artificial smoke—and therefore minimal degradation—while the lower projection comes from an experiment where artificial smoke introduced significant degradation.

add same projections as they are used in training? grayscale without vertical scaling

Goal: explain labeling techniques and rationales
Context: raw dataset has been explained, how did we label the data before use
Method: experiment-based labeling, problems, manual labeling for traing, both for evaluation
Transition: method and labeled preprocessed data known → explain experimental setup

The remaining challenge, was labeling a large enough portion of the dataset in a reasonably accurate manner, whose difficulties and general approach we described in section 4.1. Since, to our knowledge, neither our chosen dataset nor any other publicly available dataset provide objective labels for LiDAR data degradation in the SAR domain, we had to define our own labeling approach. With objective measures of degradation unavailable, we explored alternative labeling methods—such as using the datas' statistical properties like the number of missing measurements per point cloud or the higher incidence of erroneous measurements near the sensor we described in section 4.1. Ultimately, we were concerned that these statistical approaches might lead the method to simply mimic the statistical evaluation rather than to quantify degradation in a generalized and robust manner. After considering these options, we decided to label all

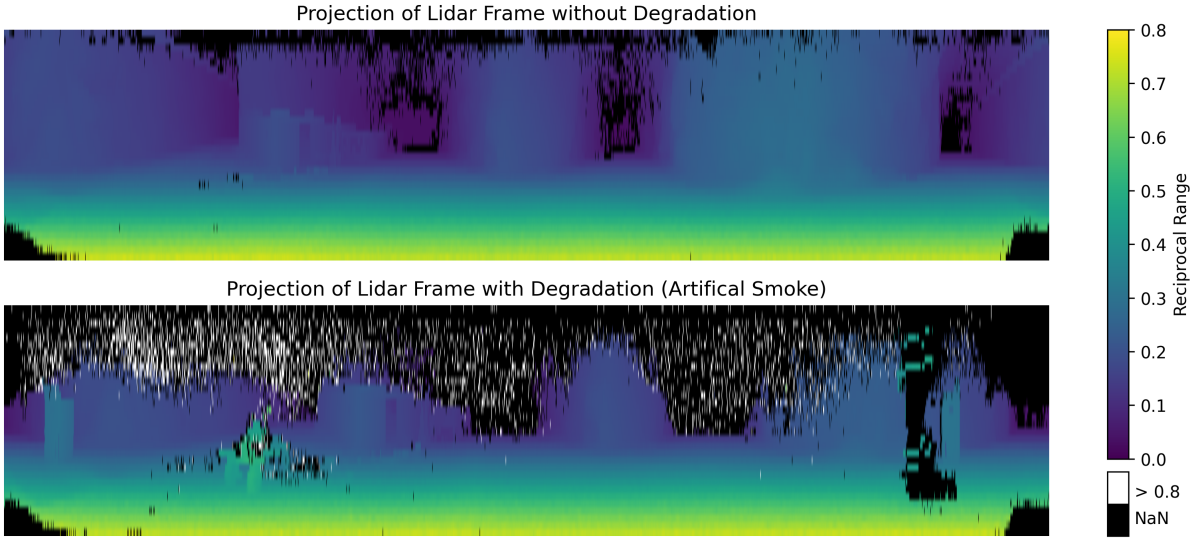


Figure 4.7: Two-dimensional projections of two pointclouds, one from an experiment without degradation and one from an experiment with artifical smoke as degradation. To aid the readers perception, the images are vertically stretched and a colormap has been applied to the pixels' reciprocal range values, while the actual training data is grayscale.

point clouds from experiments with artificial smoke as anomalies, while point clouds from experiments without smoke were labeled as normal data. This labeling strategy—based on the presence or absence of smoke—is fundamentally an environmental indicator, independent of the intrinsic data properties recorded during the experiments.

The simplicity of this labeling approach has both advantages and disadvantages. On the positive side, it is easy to implement and creates a clear distinction between normal and anomalous data. However, its simplicity is also its drawback: some point clouds from experiments with artificial smoke do not exhibit perceptible degradation, yet they are still labeled as anomalies. The reason for this, is that during the three non-static anomalous experiments the sensor platform starts recording in a tunnel roughly 20 meters from the smoke machine's location. It starts by approaching the smoke machine, navigates close to the machine for some time and then leaves its perimeter once again. Since the artificical smoke's density is far larger near the machine it originates from, the time the sensor platform spent close to it produced highly degraded point clouds, whereas the temporal beginnings and ends of the anomalous experiments capture point clouds which are subjectively not degraded and appear similar to ones from the normal experiments. This effect is clearly illustrated by the degradation indicators which we talked about earlier-the proportion of missing points and the amount of erroneous points close to the sensor per pointcloud-as can be seen in figure 4.8.

Afraid that the incorrectly labeled data may negatively impact DeepSAD's semi-supervised training, we chose to manually remove the anomalous labels from the beginning and end of the anomalous experiments, for training purposes. This refinement gave us more confidence in the training signal but reduced the number of labeled anomalies. For evaluation, we therefore report results under both schemes:

1. **Experiment-based labels:** All scans from anomalous experiments marked anomalous, including border cases—yielding conservative performance metrics that reflect real-world label noise.

2. **Manually-refined labels:** Only unequivocally degraded scans marked anomalous—producing near-ideal separation in a lot of cases.

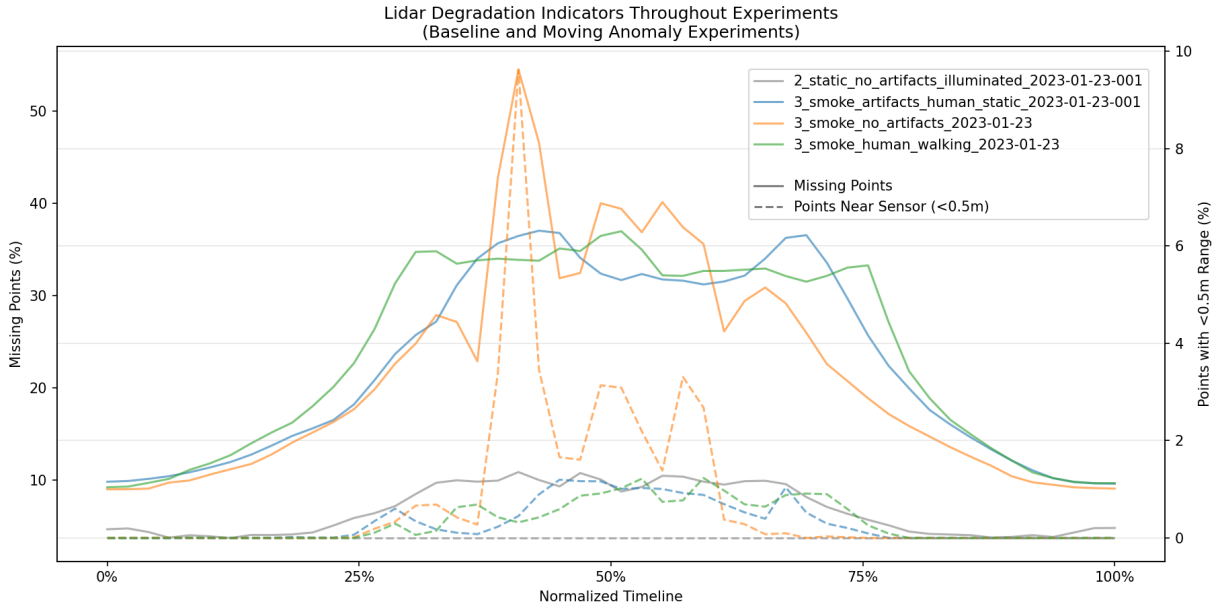


Figure 4.8: Missing points and points with a measured range smaller than 50cm per point cloud over a normalized timeline of the individual experiments. This illustrates the rise, plateau and fall of degradation intensity during the anomalous experiments, owed to the spacial proximity to the degradation source (smoke machine). One of the normal experiments (without artifical smoke) is included as a baseline.

By evaluating and comparing both approaches, we hope to demonstrate a more thorough performance investigatation than with only one of the two.

5 Experimental Setup

Goal: introduce experimental setup, give overview of what will be covered
Context: motivation, bg, method and data is know and understood, how was it used
Method: codebase, hardware description overview of training setup, details of deepsad setup
Transition: overview of chapter given → give sequential setup overview

We built our experiments on the official DeepSAD PyTorch implementation and evaluation framework, available at `https://github.com/lukasruff/Deep-SAD-PyTorch`. This codebase provides routines for loading standard datasets, training DeepSAD and several baseline models, and evaluating their performance.

In the following sections, we detail our adaptations to this framework:

- Data integration: preprocessing and loading the dataset from "Multimodal Dataset from Harsh Sub-Terranean Environment with Aerosol Particles for Frontier Exploration."
- Model architecture: configuring DeepSAD's encoder to match our pointcloud input format, contrasting two distinct neural network architectures to investigate their impact on the method's output.
- Training & evaluation: training DeepSAD alongside two classical baselines—Isolation Forest and one-class SVM—and comparing their degradation-quantification performance.
- Experimental environment: the hardware and software stack used, with typical training and inference runtimes.

Together, these components define the full experimental pipeline, from data preprocessing to the evaluation metrics we use to compare methods.

5.1 Framework & Data Preparation

DeepSAD PyTorch codebase and our adaptations

Goal: Explain deepsad codebase as starting point
Context: what is the starting point?
Method: codebase, github, dataloading, training, testing, baselines
Transition: codebase understood → how was it adapted

DeepSAD's PyTorch implementation includes standardized datasets such as MNIST, CIFAR-10 and datasets from *ODDS Library* [25], as well as suitable network architectures for the corresponding datatypes. The framework can train and test DeepSAD as well as a number of baseline algorithms, namely SSAD, OCSVM, Isolation Forest, KDE and SemiDGM with the loaded data and evaluate their performance by calculating the ROC area under curve for all given algorithms. We adapted this implementation which was originally developed for Python 3.7 to work with Python 3.12 and changed or added functionality for dataloading our chosen dataset, added DeepSAD models that work with the lidar projections datatype, added more evaluation methods and an inference module.

SubTER dataset preprocessing, train/test splits, and label strategy

Goal: explain how dataloading was adapted
Context: loading data first point step to new training
Method: preprocessed numpy (script), load, labels/meta, split, k-fold
Transition: k-fold → also adapted in training/testing

The raw SubTER dataset is provided as one ROS bag file per experiment, each containing a dense 3D point cloud from the Ouster OS1-32 LiDAR. To streamline training and avoid repeated heavy computation, we project these point clouds offline into 2D "range images" and save them as NumPy arrays. We apply a spherical projection that maps each LiDAR measurement to a pixel in a 2D image of size Height × Width, where Height = number of vertical channels (32) and Width = measurements per rotation (2048). Instead of computing per-point azimuth and elevation angles at runtime, we exploit the sensor's metadata:

- **Channel index:** directly gives the row (vertical position) of each measurement.
- **Measurement index:** by taking the measurement index modulo Width, we obtain the column (horizontal position) in the 360° sweep.

The measurement index is only available because the SubTER data is dense—every possible channel × measurement pair appears in the bag, even if the LiDAR did not record a return. We can perform a direct 1:1 mapping without collision or missing entries. This avoids the ambiguities we previously encountered when reconstructing the projection via angle computations alone, which sometimes mapped multiple points to the same pixel due to numerical errors in angle estimation.

For each projected pixel, we compute $\mathbf{v}_i = \sqrt{{x_i}^2 + {y_i}^2 + {z_i}^2}$ where $\mathbf{v}_i$ is the reciprocal range value assigned to each pixel in the projection and x_i, y_i and z_i are the corresponding measurement's 3d coordinates. This transformation both compresses the dynamic range and emphasizes close-range returns—critical for detecting near-sensor degradation. We then save the resulting tensor of shape (Number of Frames, Height, Width) using NumPy's save function. Storing precomputed projections allows rapid data loading during training and evaluation.

Many modern LiDARs can be configured to output range images directly which would bypass the need for post-hoc projection. When available, such native range-image streams can further simplify preprocessing or even allow skipping this step completely.

Any implementation challenges or custom data loaders

We extended the DeepSAD framework's PyTorch `DataLoader` by implementing a custom `Dataset` class that ingests our precomputed NumPy range-image files and attaches appropriate evaluation labels.

Each experiment's frames are stored as a single `.npy` file of shape (Number of Frames, H, W), containing the reciprocal range values described in Section 4.3. Our `Dataset` initializer scans a directory of these files, loads the NumPy arrays from file into memory, transforms them into PyTorch tensors and assigns evaluation and training labels accordingly.

The first labeling scheme, called *experiment-based labels*, assigns

$$y_{\text{exp}} = \begin{cases} -1 & \text{if the filename contains "smoke", signifying anomalous/degraded data,} \\ +1 & \text{otherwise, signifying normal data.} \end{cases}$$

At load time, any file with "smoke" in its name is treated as anomalous (label -1), and all others (normal experiments) are labeled $+1$.

To obtain a second source of ground truth, we also support *manually-defined labels*. A companion JSON file specifies a start and end frame index for each of the four smoke experiments—defining the interval of unequivocal degradation. During loading the second label y_{man} is assigned as follows:

$$y_{\text{man}} = \begin{cases} -1 & \text{Frames within the manually selected window from smoke experiments} \\ +1 & \text{All frames from non-smoke experiments} \\ 0 & \text{Frames outside the manually selected window from smoke experiments} \end{cases}$$

We pass instances of this `Dataset` to PyTorch's `DataLoader`, enabling batch sampling, shuffling, and multi-worker loading. The dataloader returns the preprocessed lidar projection, both evaluation labels and a semi-supervised training label. This modular design lets us train and evaluate DeepSAD under both labeling regimes without duplicating data-handling code.

DeepSAD supports both unsupervised and semi-supervised training by optionally incorporating a small number of labeled samples. To control this, our custom PyTorch `Dataset` accepts two integer parameters, `num_labelled_normal` and `num_labelled_anomalous`, which specify how many samples of each class should retain their labels during training. All other samples are assigned a label of 0 ("unknown") and treated as unlabeled.

When using semi-supervised mode, we begin with the manually-defined evaluation labels. We then randomly un-label (set to 0) enough samples of each class until exactly `num_labelled_normal` normals and `num_labelled_anomalous` anomalies remain labeled. This mechanism allows us to systematically compare unsupervised mode, where `num_labelled_normal` = `num_labelled_anomalous` = 0, and Semi-supervised modes with varying label budgets.

To obtain robust performance estimates on our relatively small dataset, we implement k-fold cross-validation. A single integer parameter, `num_folds`, controls the number of splits. We use scikit-learn's `KFold` (from `sklearn.model_selection`) with `shuffle=True` and a fixed random seed to partition each experiment's frames into `num_folds` disjoint folds. Training then proceeds across k rounds, each time training on $(k-1)/k$ of the data and evaluating on the remaining $1/k$. In our experiments, we set `num_folds=5`, yielding an 80/20 train/evaluation split per fold.

For inference (i.e. model validation on held-out experiments), we provide a second `Dataset` class that loads a single experiment's NumPy file (no k-fold splitting), does not assign any labels to the frames nor does it shuffle frames, preserving temporal order. This setup enables seamless, frame-by-frame scoring of complete runs—crucial for analyzing degradation dynamics over an entire traversal.

5.2 Model Configuration & Evaluation Protocol

Since the neural network architecture trained in the deepsad method is not fixed as described in section 3.2 but rather chosen based on the input data, we also had to choose an autoencoder architecture befitting our preprocessed lidar data projections. Since "LiDAR Degradation Quantification for Autonomous Driving in Rain" [17] reported success in training DeepSAD on similar data we firstly adapted the network architecture utilized by them for our usecase, which is based on the simple and well understood LeNet architecture [26]. Additionally we were interested in evaluating the importance and impact of a well-suited network architecture for DeepSAD's performance and therefore designed a second network architecture henceforth reffered to as "efficient architecture" to incorporate a few modern techniques, befitting our usecase.

Network architectures (LeNet variant, custom encoder) and how they suit the point-cloud input

The LeNet-inspired autoencoder can be split into an encoder network (figure 5.1) and a decoder network (figure 5.2) with a latent space inbetween the two parts. Such an arrangement is typical for autoencoder architectures as we discussed in section 2.3. The encoder network is simultaneously DeepSAD's main training architecture which is used to infer the degradation quantification in our use-case, once trained.

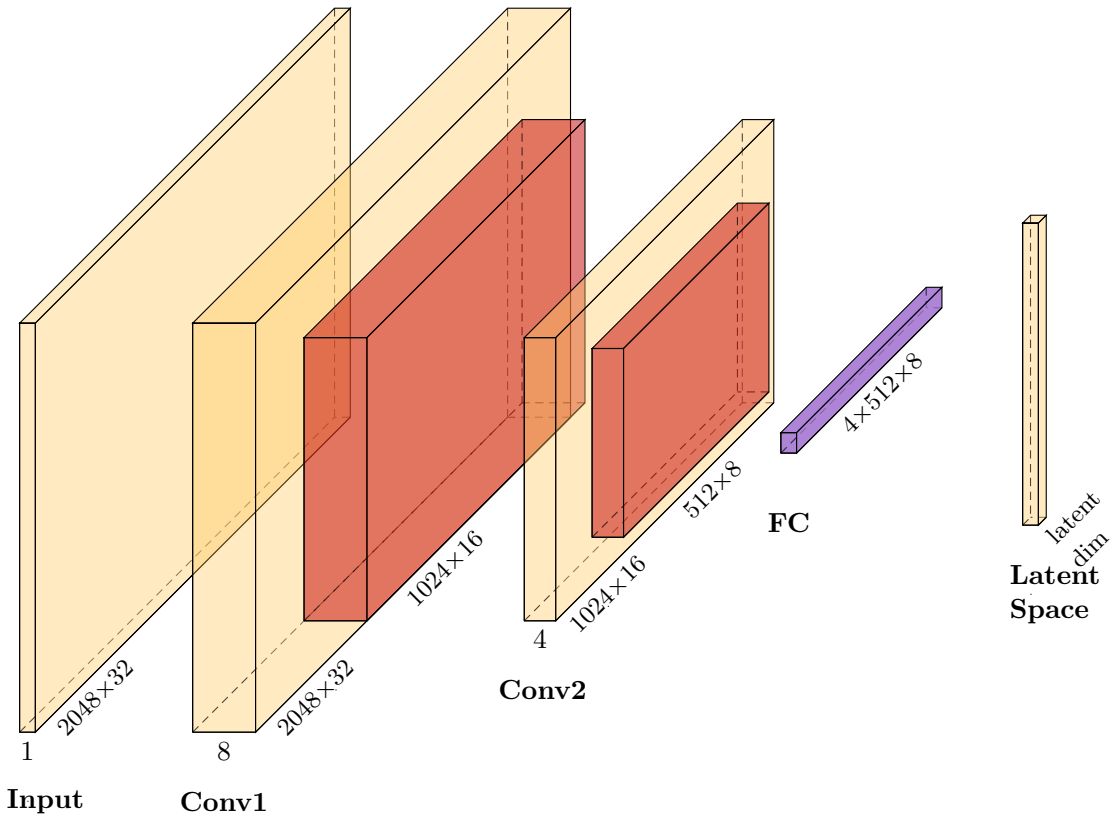


Figure 5.1: Architecture of the LeNet-inspired encoder. The input is a LiDAR range image of size $1 \times 2048 \times 32$ *(channels × width × height). The first block (Conv1) applies a* 5×5 *convolution with 8 output channels, followed by batch normalization, LeakyReLU activation, and* 2×2 *max pooling, resulting in a feature map of size* $8 \times 1024 \times 16$*. The second block (Conv2) applies another* 5×5 *convolution with 4 output channels, again followed by normalization, activation, and* 2×2 *max pooling, producing* $4 \times 512 \times 8$*. This feature map is flattened and passed through a fully connected (FC) layer of size* $4 \cdot 512 \cdot 8 = 16384$*, which maps to the latent space of dimensionality* d*, where* d *is a tunable hyperparameter (*$32 \leq d \leq 1024$ *in our experiments). The latent space serves as the compact representation used by DeepSAD for anomaly detection.*

The LeNet-inspired encoder network (see figure 5.1) is a compact convolutional neural network that reduces image data into a lower-dimensional latent space. It consists of two stages of convolution, normalization, non-linear activation, and pooling, followed by a dense layer that defines the latent representation. Conceptually, the convolutional layers learn small filters that detect visual patterns in the input (such as edges or textures). Batch normalization ensures that these learned signals remain numerically stable during training, while a LeakyReLU activation introduces non-linearity, allowing the network to capture more complex relationships. Pooling operations then downsample the feature maps, which reduces the spatial size of the data and emphasizes the most important features. Finally, a dense layer transforms the extracted

feature maps into the latent space, which serves as the datas' representation in the reduced dimensionality latent space.

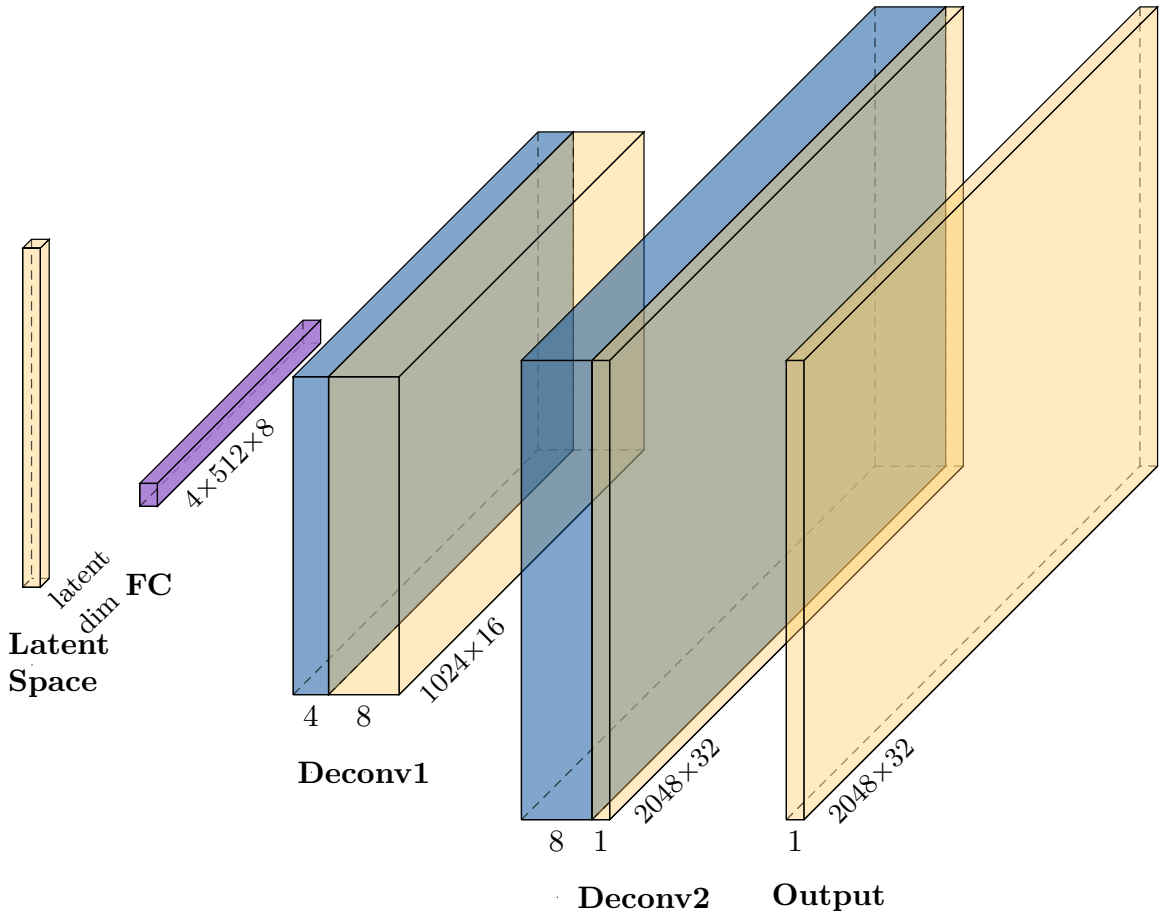


Figure 5.2: Architecture of the LeNet-inspired decoder. The input is a latent vector of dimension d, where d is the tunable latent space size ($32 \leq d \leq 1024$ in our experiments). A fully connected (FC) layer first expands this vector into a feature map of size $4 \times 512 \times 8$ (channels × width × height). The first upsampling stage applies interpolation with scale factor 2, followed by a transpose convolution with 8 output channels, batch normalization, and LeakyReLU activation, yielding $8 \times 1024 \times 16$. The second stage again upsamples by factor 2 and applies a transpose convolution, reducing the channels to 1. This produces the reconstructed output of size $1 \times 2048 \times 32$, which matches the original input dimensionality required for the autoencoding objective.

The decoder network (see figure 5.2) mirrors the encoder and reconstructs the input from its latent representation. A dense layer first expands the latent vector into a feature map of shape $4 \times 512 \times 8$, which is then upsampled and refined in two successive stages. Each stage consists of an interpolation step that doubles the spatial resolution, followed by a transpose convolution that learns how to add structural detail. The first stage operates on 4 channels, and the second on 8 channels, with the final transpose convolution reducing the output to a single channel. The result is a reconstructed output of size $1 \times 2048 \times 32$, matching the original input dimensionality required for the autoencoding objective.

Even though the LeNet-inspired encoder proved capable of achieving our degradation quantification objective in initial experiments, we identified several shortcomings that motivated the design of a second, more efficient architecture. The most important issue concerns the shape of the CNN's receptive field (RF) which describes the region of the input that influences a single output activation. Its size and aspect ratio determine which structures the network can effec-

tively capture: if the RF is too small, larger patterns cannot be detected, while an excessively large RF may hinder the network from learning to recognize fine details. For standard image data, the RF is often expressed as a symmetric $n \times n$ region, but in principle it can be computed independently per axis.

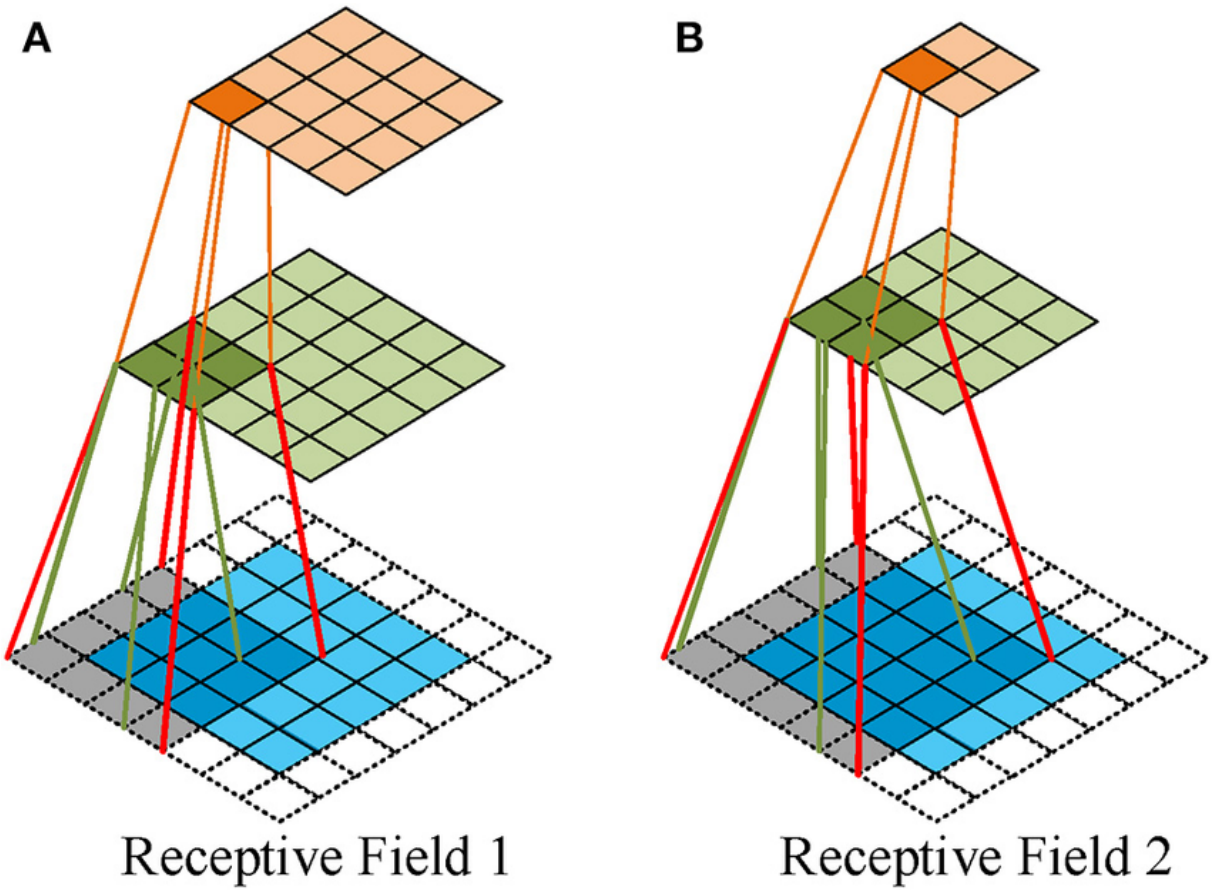


Figure 5.3: Receptive fields in a CNN. Each output activation aggregates information from a region of the input; stacking layers expands this region, while kernel size, stride, and padding control how quickly it grows and what shape it takes. (A) illustrates slower, fine-grained growth; (B) shows faster expansion, producing a larger—potentially anisotropic—receptive field and highlighting the trade-off between detail and context. Reproduced from [27]

The RF shape's issue arises from the fact that spinning multi-beam LiDAR oftentimes produce point clouds posessing dense horizontal but limited vertical resolution. In our case this, this results in a pixel-per-degree resolution of approximately $5.69\,^{pixel}/_{deg}$ vertically and $1.01\,^{pixel}/_{deg}$ horizontally. Consequently, the LeNet-inspired encoder's calculated receptive field of 16×16 pixels translates to an angular size of $15.88° \times 2.81°$, which is highly rectangular in angular space. Such a mismatch risks limiting the network's ability to capture degradation patterns that extend differently across the two axes.

To adjust for this, we decided to modify the network architecture and included further modificatons to improve the method's performance. The encoder (see figure 5.4) follows the same general idea as the LeNet-inspired encoder, but incorporates the following modificatons:

- **Non-square convolution kernels.** Depthwise-separable convolutions with kernel size 3×17 are used instead of square kernels, resulting in an RF of 10×52 pixels, corresponding to $9.93° \times 9.14°$, substantially more balanced than the LeNet-inspired network's RF.
- **Circular padding along azimuth.** The horizontal axis is circularly padded to respect the wrap-around of 360° LiDAR data, preventing artificial seams at the image boundaries.

- **Aggressive horizontal pooling.** A 1×4 pooling operation is applied early in the network, which reduces the over-sampled horizontal resolution (2048 px to 512 px) while keeping vertical detail intact.
- **Depthwise-separable convolutions with channel shuffle.** Inspired by MobileNet and ShuffleNet, this reduces the number of parameters and computations while retaining representational capacity, making the network more suitable for embedded platforms, while simultaneously allowing more learnable channels without increasing computational demand.
- **Max pooling.** Standard max pooling is used instead of average pooling, since it preserves sharp activations that are often indicative of localized degradation.
- **Channel compression before latent mapping.** After feature extraction, a 1×1 convolution reduces the number of channels before flattening, which lowers the parameter count of the final fully connected layer without sacrificing feature richness.

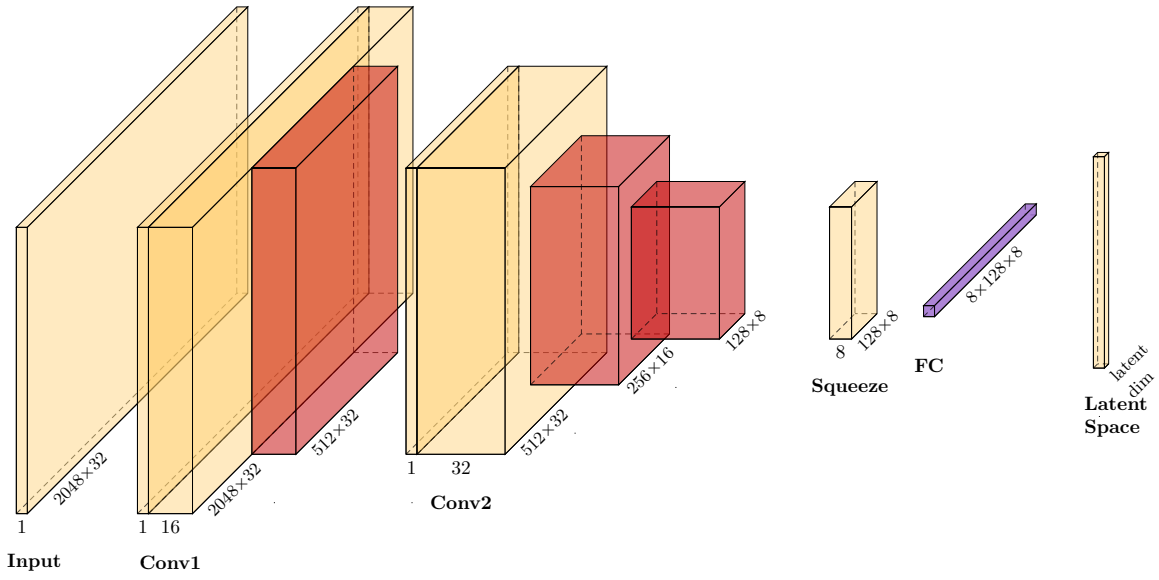


Figure 5.4: Architecture of the Efficient encoder. The input is a LiDAR range image of size $1 \times 2048 \times 32$ *(channels* $\times$ *width* $\times$ *height). The first block (**Conv1**) applies a depthwise–separable* 3×17 *convolution with circular padding along the azimuth, followed by batch normalization, LeakyReLU, and an aggressive horizontal pooling step, producing an intermediate representation of* $16 \times 512 \times 32$*. The second block (**Conv2**) applies another depthwise–separable convolution with channel shuffle, followed by two stages of* 2×2 *max pooling, yielding* $32 \times 128 \times 8$*. A* 1×1 *convolution (**Squeeze**) then reduces the channel dimension to 8, producing* $8 \times 128 \times 8$*. Finally, a fully connected layer (**FC**) flattens this feature map and projects it into the latent space of size* d*, where* d *is a tunable hyperparameter (*$32 \leq d \leq 1024$ *in our experiments).*

Decoder. The decoder (see figure 5.5) mirrors the encoder's structure but introduces changes to improve reconstruction stability:

- **Nearest-neighbor upsampling followed by convolution.** Instead of relying solely on transposed convolutions, each upsampling stage first enlarges the feature map using parameter-free nearest-neighbor interpolation, followed by a depthwise-separable convolution. This strategy reduces the risk of checkerboard artifacts while still allowing the network to learn fine detail.

- **Asymmetric upsampling schedule.** Horizontal resolution is restored more aggressively (e.g., scale factor 1×4) to reflect the anisotropic downsampling performed in the encoder.

- **Final convolution with circular padding.** The output is generated using a (3×17) convolution with circular padding along the azimuth similar to the new encoder, ensuring consistent treatment of the 360° LiDAR input.

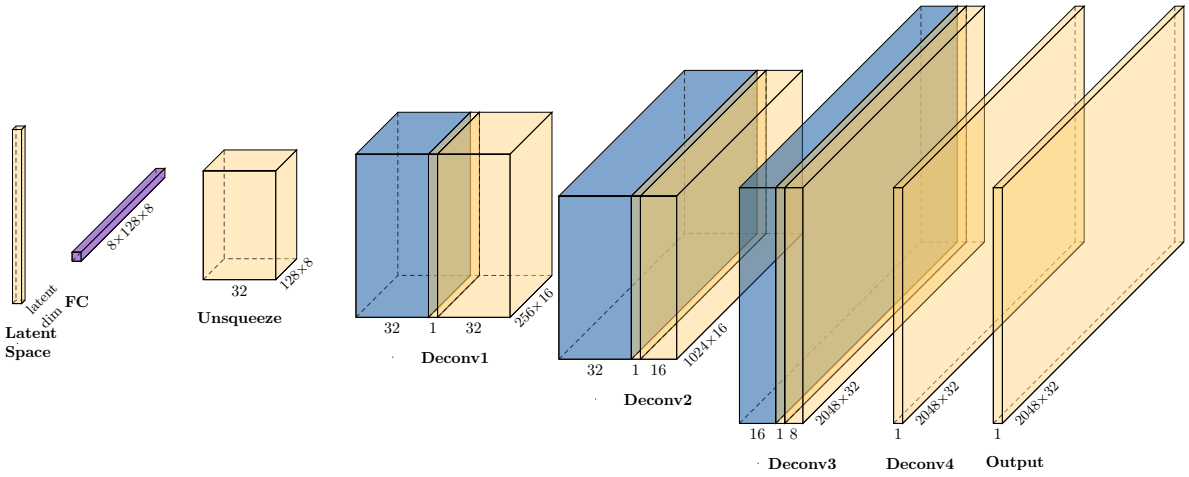


*Figure 5.5: Architecture of the Efficient decoder. The input is a latent vector of dimension d. A fully connected layer first expands this into a feature map of size $8 \times 128 \times 8$, followed by a 1×1 convolution (**Unsqueeze**) to increase the channel count to 32. The following three blocks (**Deconv1–3**) each consist of nearest-neighbor upsampling and a depthwise–separable convolution: **Deconv1** doubles both axes ($32 \times 256 \times 16$), **Deconv2** restores horizontal resolution more aggressively with a 1×4 upsampling ($16 \times 1024 \times 16$), and **Deconv3** doubles both axes again to $8 \times 2048 \times 32$. The final block (**Deconv4**) applies a (3×17) convolution with circular padding along the azimuth, reducing the channels to 1 and producing the reconstructed output of size $1 \times 2048 \times 32$, which matches the original input dimensionality.*

To compare the computational efficiency of the two architectures we show the number of trainable parameters and the number of multiply–accumulate operations (MACs) for different latent space sizes used in our experiments in table 5.1. Even though the efficient architecture employs more layers and channels which allows the network to learn to recognize more types of patterns when compared to the LeNet-inspired one, the encoders' MACs are quite similar. The more complex decoder design of the efficient network appears to contribute a lot more MACs, which leads to longer pretraining times which we report in section 5.3.

next paragrpah does not work anymroe?

As can be seen, the efficient encoder requires an order of magnitude fewer parameters and significantly fewer operations while maintaining a comparable representational capacity. The key reason is the use of depth–wise separable convolutions, aggressive pooling along the densely sampled horizontal axis, and a channel squeezing strategy before the fully connected layer. Interestingly, the Efficient network also processes more intermediate channels (up to 32 compared to only 8 in the LeNet variant), which increases its ability to capture a richer set of patterns despite the reduced computational cost. This combination of efficiency and representational power makes the Efficient encoder a more suitable backbone for our anomaly detection task.

mention that as we see in AE results the efficient arch is capable of reproducing inputs better and especially so in lower dimensional latent spaces

	Encoders				Autoencoders			
	LeNet		Efficient		LeNet		Efficient	
Latent z	Params	MACs	Params	MACs	Params	MACs	Params	MACs
32	0.53M	27.92M	0.26M	29.82M	1.05M	54.95M	0.53M	168.49M
64	1.05M	28.44M	0.53M	30.08M	2.10M	56.00M	1.06M	169.02M
128	2.10M	29.49M	1.05M	30.61M	4.20M	58.10M	2.11M	170.07M
256	4.20M	31.59M	2.10M	31.65M	8.39M	62.29M	4.20M	172.16M
512	8.39M	35.78M	4.20M	33.75M	16.78M	70.68M	8.40M	176.36M
768	12.58M	39.98M	6.29M	35.85M	25.17M	79.07M	12.59M	180.55M
1024	16.78M	44.17M	8.39M	37.95M	33.56M	87.46M	16.79M	184.75M

Table 5.1: Comparison of parameter count and MACs for SubTer_LeNet and SubTer_Efficient encoders across different latent space sizes.

Goal: how was training/testing adapted (networks overview), inference, ae tuning
Context: data has been loaded, how is it processed
Method: networks defined, training/testing k-fold, more metrics, inference + ae tuning implemented
Transition: training procesure known → what methods were evaluated

Goal: custom arch necessary, first lenet then second arch to evaluate importance of arch
Context: training process understood, but what networks were actually trained
Method: custom arch, lenet from paper and simple, receptive field problem, arch really important?
Transition: motivation behind archs given → what do they look like

Goal: show and explain both archs
Context: we know why we need them but what do they look like
Method: visualization of archs, explain LeNet and why other arch was chosen that way
Transition: both archs known → what about the other inputs/hyperparameters

Goal: give overview of hyperparameters
Context: deepsad arch known, other hyperparameters?
Method: LR, eta, epochs, latent space size (hyper param search), semi labels
Transition: everything that goes into training known → what experiments were actually done?

Baseline methods (Isolation Forest, one-class SVM) and feature extraction via the encoder

Goal: what methods were evaluated
Context: we know what testing/training was implemented for deepsad, but what is it compared to
Method: isoforest, ocsvm adapted, for ocsvm only dim reduced feasible (ae from deepsad)
Transition: compared methods known → what methods were used

To contextualize the performance of DeepSAD, we compare against two widely used baselines: Isolation Forest and One-Class SVM (OCSVM). Both are included in the original DeepSAD codebase and the associated paper, and they represent well-understood but conceptually different families of anomaly detection. In our setting, the raw input dimensionality (2048×32 per frame) is too high for a direct OCSVM fit, so we reuse the DeepSAD autoencoder's *encoder* as a learned dimensionality reduction (to the same latent size as DeepSAD). This choice is motivated by practicality (compute) and inductive bias: the encoder captures non-linear, domain-specific structure of the LiDAR range images, which linear methods like PCA may miss. Together, these two baselines cover complementary perspectives: tree-based partitioning (Isolation Forest) and kernel-based boundary learning (OCSVM), providing a broad and well-established basis for comparison.

Isolation Forest is an ensemble method for anomaly detection that builds on the principle that

anomalies are easier to separate from the rest of the data. It constructs many binary decision trees, each by recursively splitting the data at randomly chosen features and thresholds. In this process, the "training" step consists of building the forest of trees: each tree captures different random partitions of the input space, and together they form a diverse set of perspectives on how easily individual samples can be isolated.

Once trained, the method assigns an anomaly score to new samples by measuring their average path length through the trees. Normal samples, being surrounded by other similar samples, typically require many recursive splits and thus end up deep in the trees. Anomalies, by contrast, stand out in one or more features, which means they can be separated much earlier and end up closer to the root. The shorter the average path length, the more anomalous the sample is considered. This makes Isolation Forest highly scalable and robust: training is efficient since no explicit density estimation is required, and the resulting model is fast to apply to new data. In our setup, we apply Isolation Forest directly to the LiDAR input representation, providing a strong non-neural baseline for comparison against DeepSAD.

While Isolation Forest relies on random partitioning of the input space, OCSVM takes a very different approach by learning a flexible boundary around normal samples. OCSVM is trained only on data assumed to be normal, with the goal of enclosing the majority of these samples in such a way that new points lying outside this boundary can be identified as anomalies.

The boundary itself is learned using the support vector machine framework. In essence, OCSVM looks for a hyperplane in some feature space that maximizes the separation between the bulk of the data and the origin. To make this possible even when the normal data has a complex, curved shape, OCSVM uses a kernel function such as the radial basis function (RBF). The kernel implicitly maps the input data into a higher-dimensional space, where the cluster of normal samples becomes easier to separate with a simple hyperplane. When this separation is mapped back to the original input space, it corresponds to a flexible, nonlinear boundary that can adapt to the structure of the data.

During training, the algorithm balances two competing objectives: capturing as many of the normal samples as possible inside the boundary, while keeping the region compact enough to exclude potential outliers. Once this boundary is established, applying OCSVM is straightforward — any new data point is checked against the learned boundary, with points inside considered normal and those outside flagged as anomalous.

We adapted the baseline implementations to our data loader and input format

briefly describe file layout / preprocessing

, and added support for multiple evaluation targets per frame (two labels per data point), reporting both results per experiment. For OCSVM, the dimensionality reduction step is *always* performed with the corresponding DeepSAD encoder and its autoencoder pretraining weights that match the evaluated setting (i.e., same latent size and backbone). Both baselines, like DeepSAD, output continuous anomaly scores. This allows us to evaluate them directly without committing to a fixed threshold.

5.3 Experiment Overview & Computational Environment

Goal: *"What should the reader know after reading this section?"*
Context: *"Why is that of interest to the reader at this point?"*
Method: *"How am I achieving the stated goal?"*
Transition: *"How does this lead to the next question or section?"*

Goal: give overview of experiments and their motivations
Context: training setup clear, but not what was trained/tested
Method: explanation of what was searched for (ae latent space first), other hyperparams and why
Transition: all experiments known → how long do they take to train

Our experimental setup consisted of two stages. First, we conducted a hyperparameter search over the latent space dimensionality by pretraining the autoencoders alone. For both the LeNet-inspired and the Efficient network, we evaluated latent space sizes of $32, 64, 128, 256, 384, 512, 768$, and 1024. Each autoencoder was trained for 50 epochs with a learning rate of $1 \cdot 10^{-5}$, and results were averaged across 5-fold cross-validation. The goal of this stage was to identify the "elbow point" in reconstruction loss curves, which serves as a practical indicator of a sufficiently expressive, yet compact, representation.

Second, we trained the full DeepSAD models on the same latent space sizes in order to investigate how autoencoder performance transfers to anomaly detection performance. Specifically, we aimed to answer whether poor autoencoder reconstructions necessarily imply degraded DeepSAD results, or whether the two stages behave differently. To disentangle these effects, both network architectures (LeNet-inspired and Efficient) were trained with identical configurations, allowing for a direct architectural comparison.

Furthermore, we investigated the effect of semi-supervised labeling. DeepSAD can incorporate labeled data during training, and we wanted to investigate the impact of labeling on anomaly detection performance. To this end, each configuration was trained under three labeling regimes:

- **Unsupervised:** $(0, 0)$ labeled samples of (normal, anomalous) data.
- **Low supervision:** $(50, 10)$ labeled samples.
- **High supervision:** $(500, 100)$ labeled samples.

All models were pre-trained for 50 epochs and then trained for 150 epochs with the same learning rate of $1 \cdot 10^{-5}$ and evaluated with 5-fold cross-validation. Table 5.2 summarizes the full experiment matrix.

Table 5.2: Parameter space for the DeepSAD grid search. Each latent size is tested for both architectures and all labeling regimes.

	Latent sizes	**Architectures**	**Labeling regimes (normal, anomalous)**
Levels	32 64 128 256 512 768 1024	LeNet-inspired Efficient	(0,0) (50,10) (500,100)
Count	7	2	3
Total combinations		$7 \times 2 \times 3 = \mathbf{42}$	

Goal: give overview about hardware setup and how long things take to train
Context: we know what we trained but not how long that takes
Method: table of hardware and of how long different trainings took
Transition: experiment setup understood → what were the experiments' results

These experiments were run on a computational environment for which we summarize the hardware and software stack in table 5.3.

Pretraining runtimes for the autoencoders are reported in Table 5.4. These values are averaged across folds and labeling regimes, since the pretraining step itself does not make use of labels.

The full DeepSAD training times are shown in Table 5.5, alongside the two classical baselines Isolation Forest and One-Class SVM. Here the contrast between methods is clear: while DeepSAD requires on the order of 15–20 minutes of GPU training per configuration and fold, both baselines complete training in seconds on CPU. The OCSVM training can only be this fast due to the reduced input dimensionality from utilizing DeepSAD's pretraining encoder as a preprocessing step, although other dimensionality reduction methods may also be used which could require less computational resources for this step.

Inference latency per sample is presented in Table 5.6. These measurements highlight an important property: once trained, all methods are extremely fast at inference, with DeepSAD operating in the sub-millisecond range and the classical baselines being even faster. This confirms that, despite higher training costs, DeepSAD can be deployed in real-time systems without inference becoming a bottleneck.

Together, these results provide a comprehensive overview of the computational requirements of our experimental setup. They show that while our deep semi-supervised approach is significantly more demanding during training than classical baselines, it remains highly efficient at inference, which is the decisive factor for deployment in time-critical domains such as rescue robotics.

Table 5.3: Computational Environment (Hardware & Software)

System	
Operating System	Ubuntu 22.04.5 LTS
Kernel	6.5.0-44-generic
Architecture	x86_64
CPU Model	AMD Ryzen Threadripper 3970X 32-Core Processor
CPU Cores (physical)	32
CPU Threads (logical)	64
CPU Base Frequency	2200 MHz
CPU Max Frequency	3700 MHz
Total RAM	94.14 GiB
GPU	
GPU Name	NVIDIA GeForce RTX 4090
GPU Memory	23.99 GiB
GPU Compute Capability	8.9
NVIDIA Driver Version	535.161.07
CUDA (Driver) Version	12.2
Software Environment	
Python	3.12.11
PyTorch	2.7.1+cu126
PyTorch Built CUDA	12.6
cuDNN (PyTorch build)	90501
scikit-learn	1.7.0
NumPy	2.3.0
SciPy	1.15.3

Table 5.4: Autoencoder pretraining runtime (seconds): mean ± std across folds and across semi-supervised labeling regimes.

Latent Dim.	Autoencoder Efficient	Autoencoder LeNet
32	1175.03 ± 35.87 s	384.90 ± 34.59 s
64	1212.53 ± 35.76 s	398.22 ± 41.25 s
128	1240.86 ± 11.51 s	397.98 ± 33.43 s
256	1169.72 ± 33.26 s	399.40 ± 38.20 s
512	1173.34 ± 34.99 s	430.31 ± 38.02 s
768	1204.45 ± 37.52 s	436.49 ± 37.13 s
1024	1216.79 ± 34.82 s	411.69 ± 34.82 s

Table 5.5: Training runtime: total seconds (mean ± std).

Latent Dim.	DeepSAD LeNet	DeepSAD Efficient	IsoForest	OCSVM
32	765.37 ± 91.74 s	1026.18 ± 84.13 s	0.55 ± 0.02 s	1.07 ± 00.29 s
64	815.88 ± 93.07 s	1124.48 ± 60.84 s	0.55 ± 0.02 s	1.98 ± 01.57 s
128	828.53 ± 63.00 s	1164.94 ± 02.13 s	0.55 ± 0.02 s	3.17 ± 02.63 s
256	794.54 ± 97.04 s	986.88 ± 82.98 s	0.55 ± 0.02 s	12.81 ± 14.19 s
512	806.63 ± 99.83 s	998.23 ± 80.34 s	0.55 ± 0.02 s	22.76 ± 23.52 s
768	818.56 ± 86.38 s	1053.64 ± 78.72 s	0.55 ± 0.02 s	14.24 ± 01.21 s
1024	770.05 ± 86.22 s	1054.92 ± 87.49 s	0.55 ± 0.02 s	28.20 ± 24.04 s

Table 5.6: Inference latency (ms/sample): mean ± std across folds; baselines collapsed across networks and semi-labeling.

Latent Dim.	DeepSAD LeNet	DeepSAD Efficient	IsoForest	OCSVM
32	0.31 ± 0.04 ms	0.36 ± 0.05 ms	0.02 ± 0.00 ms	0.07 ± 0.02 ms
64	0.33 ± 0.06 ms	0.43 ± 0.04 ms	0.02 ± 0.00 ms	0.10 ± 0.06 ms
128	0.31 ± 0.04 ms	0.45 ± 0.02 ms	0.02 ± 0.00 ms	0.16 ± 0.09 ms
256	0.30 ± 0.04 ms	0.33 ± 0.02 ms	0.02 ± 0.00 ms	0.30 ± 0.21 ms
512	0.32 ± 0.04 ms	0.33 ± 0.02 ms	0.02 ± 0.00 ms	0.63 ± 0.65 ms
768	0.33 ± 0.03 ms	0.41 ± 0.06 ms	0.02 ± 0.00 ms	0.39 ± 0.07 ms
1024	0.27 ± 0.02 ms	0.39 ± 0.05 ms	0.02 ± 0.00 ms	0.94 ± 0.98 ms

6 Results and Discussion

Goal: Introduce the structure and scope of the results chapter
Context: The reader knows the experiments from the previous chapter, but not the outcomes
Method: State that we will first analyze autoencoder results, then anomaly detection performance, and finally inference experiments
Transition: Clear roadmap → prepares reader for detailed sections

The experiments described in Chapter 5 are presented in this chapter. We begin in Section 6.1 with the pretraining stage, where the two autoencoder architectures were trained across multiple latent space dimensionalities. These results provide insight into the representational capacity of each architecture. In Section 6.2, we turn to the main experiments: training DeepSAD models and benchmarking them against baseline algorithms (Isolation Forest and One-Class SVM). Finally, in Section 6.3, we present inference results on experiments that were held out during training. These plots illustrate how the algorithms behave when applied sequentially to unseen traversals, offering a more practical perspective on their potential for real-world rescue robotics applications.

6.1 Autoencoder Pretraining Results

The results of pretraining the two autoencoder architectures are summarized in Table **??**. Reconstruction performance is reported as mean squared error (MSE), with trends visualized in Figure 6.1. The results show that the modified Efficient architecture consistently outperforms the LeNet-inspired baseline across all latent space dimensionalities. The improvement is most pronounced at lower-dimensional bottlenecks (e.g., 32 or 64 dimensions) but remains observable up to 1024 dimensions, although the gap narrows.

Because overall reconstruction loss might obscure how well encoders represent anomalous samples, we additionally evaluate reconstruction errors only on degraded samples from hand-labeled smoke segments (Figure 6.2). As expected, reconstruction losses are about 0.05 higher on these challenging samples than in the overall evaluation. However, the relative advantage of the Efficient architecture remains, suggesting that its improvements extend to anomalous inputs as well.

It is important to note that absolute MSE values are difficult to interpret in isolation, as their magnitude depends on the data scaling and chosen reconstruction target. More detailed evaluations in terms of error in meters, relative error, or distance-binned metrics could provide richer insights into encoder quality. However, since the downstream anomaly detection results (Section 6.2) do not reveal significant differences between pretraining regimes, such detailed pretraining evaluation was not pursued here. Instead, we restrict ourselves to reporting the reconstruction trends and leave more in-depth pretraining analysis as future work.

	Overall loss		Anomaly loss	
Latent Dim.	LeNet	Efficient	LeNet	Efficient
32	0.0223	**0.0136**	0.0701	**0.0554**
64	0.0168	**0.0117**	0.0613	**0.0518**
128	0.0140	**0.0110**	0.0564	**0.0506**
256	0.0121	**0.0106**	0.0529	**0.0498**
512	0.0112	**0.0103**	0.0514	**0.0491**
768	0.0109	**0.0102**	0.0505	**0.0490**
1024	0.0106	**0.0101**	0.0500	**0.0489**

Table 6.1: Autoencoder pre-training MSE losses across latent dimensions. Left: overall loss; Right: anomaly-only loss. Cells show means across folds (no ±std). Maximum observed standard deviation across all cells (not shown): 0.0067.

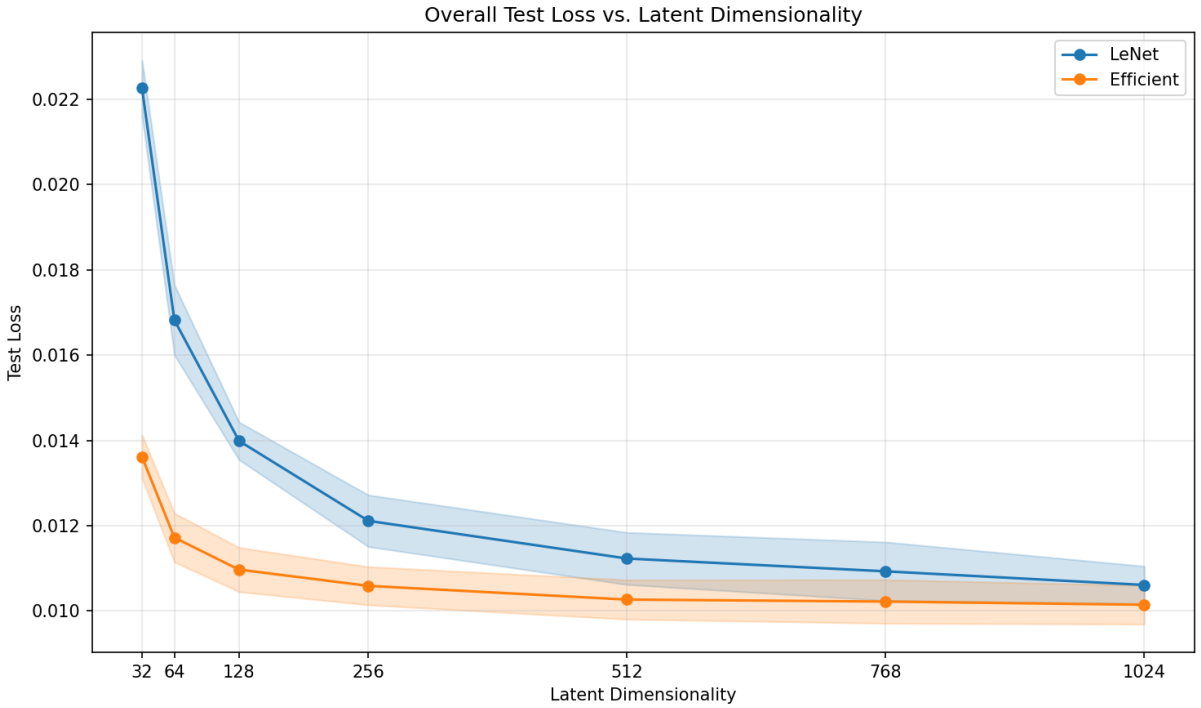


Figure 6.1: Reconstruction loss across latent dimensions for LeNet-inspired and Efficient architectures.

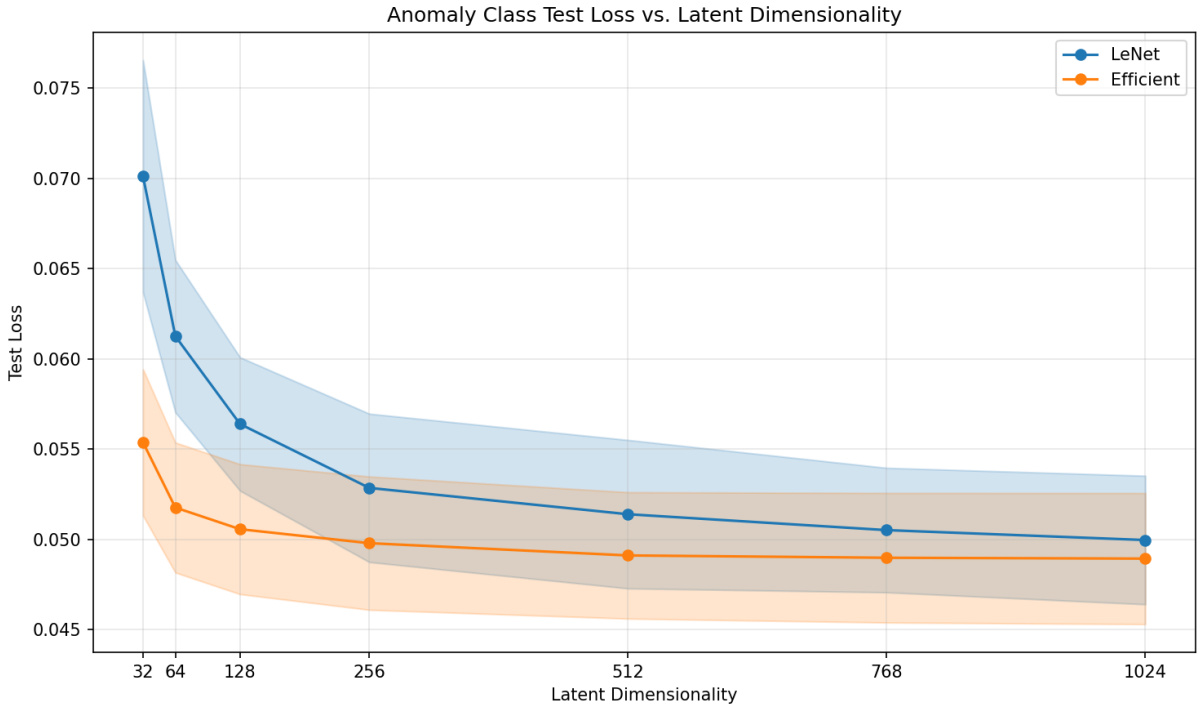


Figure 6.2: Reconstruction loss across latent dimensions for LeNet-inspired and Efficient architectures, evaluated only on degraded data from hand-labeled smoke experiments.

6.2 DeepSAD Detection Performance

Goal: Introduce DeepSAD anomaly detection results compared to baselines
Context: Core part of evaluation: shows if DeepSAD provides benefit beyond standard methods
Method: Explain ROC/PRC as evaluation metrics, show curves for all latent sizes, unsupervised case
Transition: Results here → baseline comparison and semi-supervised effects

Goal: Interpret unsupervised results across architectures and baselines
Context: Important to establish the baseline performance levels
Method: Compare AUCs: Isolation Forest weakest, OCSVM moderate (uses encoder), DeepSAD best
Transition: Sets expectation for whether supervision improves or harms performance

Goal: Present semi-supervised regimes and their effects
Context: Semi-supervision is central to DeepSAD; must show how labels change outcomes
Method: Show ROC/PRC plots for selected latent sizes under different labeling regimes
Transition: This leads → analysis of why few labels harmed but many labels improved

Goal: Discuss surprising supervision dynamics
Context: Reader expects supervision to always help; but results show nuance
Method: Interpret why few labels overfit, many labels help, unsupervised sometimes best
Transition: This discussion → motivates looking at model behavior over time via inference

	Experiment-based eval.				Handlabeled eval.			
Latent Dim.	DeepSAD (LeNet)	DeepSAD (Efficient)	IsoForest	OC-SVM	DeepSAD (LeNet)	DeepSAD (Efficient)	IsoForest	OC-SVM
Labeling regime: 0/0 *(normal/anomalous samples labeled)*								
32	**0.801**	0.791	0.717	0.752	**1.000**	**1.000**	0.921	0.917
64	0.776	**0.786**	0.718	0.742	**1.000**	**1.000**	0.917	0.931
128	**0.784**	**0.784**	0.719	0.775	**1.000**	**1.000**	0.921	0.967
256	0.762	0.772	0.712	**0.793**	**1.000**	**1.000**	0.918	0.966
512	0.759	0.784	0.712	**0.804**	**1.000**	**1.000**	0.920	0.949
768	0.749	0.754	0.713	**0.812**	**1.000**	**1.000**	0.923	0.960
1024	0.757	0.750	0.716	**0.821**	**1.000**	**1.000**	0.919	0.956
Labeling regime: 50/10 *(normal/anomalous samples labeled)*								
32	0.741	0.747	0.717	**0.752**	0.990	**0.998**	0.921	0.917
64	**0.757**	0.750	0.718	0.742	0.998	**0.999**	0.917	0.931
128	0.746	0.751	0.719	**0.775**	0.991	**0.999**	0.921	0.967
256	0.746	0.750	0.712	**0.793**	**0.999**	**0.999**	0.918	0.966
512	0.760	0.763	0.712	**0.804**	0.972	**0.999**	0.920	0.949
768	0.749	0.747	0.713	**0.812**	**1.000**	0.998	0.923	0.960
1024	0.748	0.732	0.716	**0.821**	**0.999**	0.998	0.919	0.956
Labeling regime: 500/100 *(normal/anomalous samples labeled)*								
32	0.765	**0.775**	0.717	0.752	**1.000**	**1.000**	0.921	0.917
64	0.754	**0.773**	0.718	0.742	**1.000**	**1.000**	0.917	0.931
128	0.758	0.769	0.719	**0.775**	**1.000**	**1.000**	0.921	0.967
256	0.749	0.768	0.712	**0.793**	0.999	**1.000**	0.918	0.966
512	0.766	0.770	0.712	**0.804**	0.989	**1.000**	0.920	0.949
768	0.746	0.750	0.713	**0.812**	**1.000**	**1.000**	0.923	0.960
1024	0.743	0.739	0.716	**0.821**	**1.000**	**1.000**	0.919	0.956

Table 6.2: ROC AUC means across 5 folds for both evaluations, grouped by labeling regime. Maximum observed standard deviation across all cells (not shown in table): 0.06.

6.3 Inference on Held-Out Experiments

Goal: Introduce inference evaluation on unseen experiments
Context: This tests real-world usefulness: continuous scan-level degradation quantification
Method: Explain setup: EMA-smoothed z-scores compared against heuristic degradation indicators
Transition: From static metrics → to temporal behavior analysis

Goal: Analyze correlation of anomaly scores with degradation indicators
Context: Important because it shows methods behave as intended even without perfect ground truth
Method: Discuss qualitative similarity, emphasize scores as degradation proxies
Transition: Sets stage → for clean vs degraded comparison

Goal: Compare anomaly score dynamics between clean and degraded experiments
Context: Tests whether scores separate normal vs degraded traversals reliably
Method: Show normalized z-score plots using clean-experiment parameters
Transition: Final confirmation → methods are meaningful for degradation quantification

7 Conclusion and Future Work

7.1 Conclusion

summarize what has been achieved

7.2 Future Work

confirm results with real smoke data

Bibliography

[1] F. E. and, "Xli. on discordant observations," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 23, no. 143, pp. 364–375, 1887. DOI: 10.1080/14786448708628471. eprint: https://doi.org/10.1080/14786448708628471. [Online]. Available: https://doi.org/10.1080/14786448708628471.

[2] Q. Wei, Y. Ren, R. Hou, B. Shi, J. Y. Lo, and L. Carin, "Anomaly detection for medical images based on a one-class classification," in *Medical Imaging 2018: Computer-Aided Diagnosis*, N. Petrick and K. Mori, Eds., ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 10575, Feb. 2018, 105751M, p. 105751M. DOI: 10.1117/12.2293408.

[3] M. Ul Hassan, M. H. Rehmani, and J. Chen, "Anomaly detection in blockchain networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 289–318, 2023. DOI: 10.1109/COMST.2022.3205643.

[4] D. Y. Oh and I. D. Yun, "Residual error based anomaly detection using auto-encoder in smd machine sound," *Sensors*, vol. 18, no. 5, 2018, ISSN: 1424-8220. DOI: 10.3390/s18051308. [Online]. Available: https://www.mdpi.com/1424-8220/18/5/1308.

[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009, ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. [Online]. Available: https://doi.org/10.1145/1541880.1541882.

[6] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K. Mü ller, and M. Kloft, "Deep semi-supervised anomaly detection," *CoRR*, vol. abs/1906.02694, 2019. arXiv: 1906.02694. [Online]. Available: http://arxiv.org/abs/1906.02694.

[7] P. Bergmann and D. Sattlegger, "Anomaly detection in 3d point clouds using deep geometric descriptors," in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, IEEE, Jan. 2023, pp. 2612–2622. DOI: 10.1109/wacv56688.2023.00264. [Online]. Available: http://dx.doi.org/10.1109/WACV56688.2023.00264.

[8] B. Rodríguez-Cuenca, S. García-Cortés, C. Ordóñez, and M. Alonso, "Automatic detection and classification of pole-like objects in urban point cloud data using an anomaly detection algorithm," *Remote Sensing*, vol. 7, no. 10, pp. 12 680–12 703, Sep. 2015, ISSN: 2072-4292. DOI: 10.3390/rs71012680. [Online]. Available: http://dx.doi.org/10.3390/rs71012680.

[9] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. [Online]. Available: http://dx.doi.org/10.1147/rd.33.0210.

[10] M. E. Villa-Pérez, M. Á. Álvarez-Carmona, O. Loyola-González, M. A. Medina-Pérez, J. C. Velazco-Rossell, and K.-K. R. Choo, "Semi-supervised anomaly detection algorithms: A comparative summary and future research directions," *Knowledge-Based Systems*, vol. 218, p. 106 878, Apr. 2021, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2021.106878. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2021.106878.

[11] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *Proceedings of the 2017 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Jun. 2017, pp. 90–98, ISBN: 9781611974973. DOI: 10.1137/1.9781611974973.11. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974973.11.

[12] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. Van Den Hengel, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2019, pp. 1705–1714. DOI: 10.1109/iccv.2019.00179. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2019.00179.

[13] F. H. Nahhas, H. Z. M. Shafri, M. I. Sameen, B. Pradhan, and S. Mansor, "Deep learning approach for building detection using lidar–orthophoto fusion," *Journal of Sensors*, vol. 2018, pp. 1–12, Aug. 2018, ISSN: 1687-7268. DOI: 10.1155/2018/7212307. [Online]. Available: http://dx.doi.org/10.1155/2018/7212307.

[14] J.-I. Park, S. Jo, H.-T. Seo, and J. Park, "Lidar denoising methods in adverse environments: A review," *IEEE Sensors Journal*, vol. 25, no. 5, pp. 7916–7932, Mar. 2025, ISSN: 2379-9153. DOI: 10.1109/jsen.2025.3526175. [Online]. Available: http://dx.doi.org/10.1109/JSEN.2025.3526175.

[15] T. Parsons, J. Seo, B. Kim, H. Lee, J.-C. Kim, and M. Cha, "Dust de-filtering in lidar applications with conventional and cnn filtering methods," *IEEE Access*, vol. 12, pp. 22 032–22 042, 2024, ISSN: 2169-3536. DOI: 10.1109/access.2024.3362804. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2024.3362804.

[16] A. Kyuroson, A. Koval, and G. Nikolakopoulos, "Efficient real-time smoke filtration with 3d lidar for search and rescue with autonomous heterogeneous robotic systems," in *IECON 2023- 49th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, Oct. 2023, pp. 1–7. DOI: 10.1109/iecon51785.2023.10312303. [Online]. Available: http://dx.doi.org/10.1109/IECON51785.2023.10312303.

[17] C. Zhang, Z. Huang, M. H. Ang, and D. Rus, "Lidar degradation quantification for autonomous driving in rain," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 3458–3464. DOI: 10.1109/IROS51168.2021.9636694.

[18] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Oct. 2018, pp. 4393–4402. [Online]. Available: https://proceedings.mlr.press/v80/ruff18a.html.

[19] H.-B. Mokrane, P. De Souza, P. Nordqvist, and N. Roy, "Modelling of lidar sensor disturbances by solid airborne particles," May 2021.

[20] F. Matos, J. Bernardino, J. Durães, and J. Cunha, "A survey on sensor failures in autonomous vehicles: Challenges and solutions," *Sensors*, vol. 24, no. 16, p. 5108, Aug. 2024, ISSN: 1424-8220. DOI: 10.3390/s24165108. [Online]. Available: http://dx.doi.org/10.3390/s24165108.

[21] K. Ebadi, L. Bernreiter, H. Biggie, G. Catt, Y. Chang, A. Chatterjee, C. E. Denniston, S.-P. Deschênes, K. Harlow, S. Khattak, L. Nogueira, M. Palieri, P. Petráček, M. Petrlík, A. Reinke, V. Krátký, S. Zhao, A.-a. Agha-mohammadi, K. Alexis, C. Heckman, K. Khosoussi, N. Kottege, B. Morrell, M. Hutter, F. Pauling, F. o. Pomerleau, M. Saska, S. Scherer, R. Siegwart, J. L. Williams, and L. Carlone, "Present and future of slam in extreme environments: The darpa subt challenge," *IEEE Transactions on Robotics*, vol. 40, pp. 936–959, 2024, ISSN: 1941-0468. DOI: 10.1109/tro.2023.3323938. [Online]. Available: http://dx.doi.org/10.1109/TRO.2023.3323938.

[22] A. Kyuroson, N. Dahlquist, N. Stathoulopoulos, V. K. Viswanathan, A. Koval, and G. Nikolakopoulos, "Multimodal dataset from harsh sub-terranean environment with aerosol particles for frontier exploration," in *2023 31st Mediterranean Conference on Control and Automation (MED)*, IEEE, Jun. 2023, pp. 716–721. DOI: 10.1109/med59994.2023.10185906. [Online]. Available: http://dx.doi.org/10.1109/MED59994.2023.10185906.

[23] T. G. Phillips, N. Guenther, and P. R. McAree, "When the dust settles: The four behaviors of lidar in the presence of fine airborne particulates," *Journal of Field Robotics*, vol. 34, no. 5, pp. 985–1009, Feb. 2017, ISSN: 1556-4967. DOI: 10.1002/rob.21701. [Online]. Available: http://dx.doi.org/10.1002/rob.21701.

[24] P. Li, Y. Pei, and J. Li, "A comprehensive survey on design and application of autoencoder in deep learning," *Applied Soft Computing*, vol. 138, p. 110 176, 2023, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.110176. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623001941.

[25] S. Rayana, *Odds library*, 2016. [Online]. Available: https://odds.cs.stonybrook.edu.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791. [Online]. Available: http://dx.doi.org/10.1109/5.726791.

[27] M. Ye, J. Nie, A. Liu, Z. Wang, L. Huang, H. Tian, D. Song, and Z. Wei, "Multi-year enso forecasts using parallel convolutional neural networks with heterogeneous architecture," *Frontiers in Marine Science*, vol. 8, Aug. 2021, ISSN: 2296-7745. DOI: 10.3389/fmars.2021.717184. [Online]. Available: http://dx.doi.org/10.3389/fmars.2021.717184.

List of Figures

List of Tables