Active Learning for Visual Anomaly Detection Applied to Mobile Robots

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Artificial Intelligence

presented by
Alind Xhyra

under the supervision of
Prof. Alessandro Giusti Advisor
co-supervised by
Dario Mantegazza Co-Advisor, Prof. Domenico Giorgio Sorrenti Co-Advisor

October 2022

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Alind Xhyra
Lugano, 27 October 2022

"A computer would deserve to be called intelligent if it could deceive a human into believing that it was human."

Alan Turing

Abstract

State-of-the-art approaches for visual Anomaly Detection rely on Deep Learning techniques; these techniques require an enormous amount of labeled data for training. This data in some specific fields such as medical imagery, robotics, and safety-critical situations may require expert annotators. Because their time is limited, such labeling is expensive. Active Learning (AL) tries to overcome this issue by choosing from the available unlabeled data only the samples that improve most the knowledge of the model.

We adapt and compare different Active Learning techniques applied to Anomaly Detection tasks for ground mobile robots, using only the robot's camera stream as input. We test them on a proxy task based on the MNIST dataset. We acquire an additional dataset for experimentation by recording a robot's camera feed while it is moving around different corridors on a university campus. The dataset is collected from divers scenarios and stages more more than a dozen types of anomalies. The dataset is composed of more than 40GB of images counting up to 228,795 total frames. We replicate the experiments defined on the collected dataset to check the performance of the AL methods studied.

Contents

Figures

Tables

Listings

Introduction

Anomaly Detection (AD) problems try to find samples that show an unexpected behavior or pattern given a context of *normality*. These unexpected samples are referred to as *anomalies* or *outliers*. AD applied to mobile robots could be used to solve hazard detection problems without having prior knowledge nor definition of the hazards.

Recently, a solution [49, 28] to AD has been developed using autoencoders. Autoencoders have achieved state-of-the-art performance in the context of visual Anomaly Detection. However, such models require very large amounts of labeled data to be trained and tested on. This labeling can be very expensive in some fields where experts are required (i.e. Medical Images, Robotics, ...). The goal of Active Learning (AL) is to reduce this cost by carefully choosing the *best* samples to be labeled to increase the model performance on the given task.

This thesis reports the work and research done to implement an AL system applied to the task of *visual anomaly detection* for mobile robots using Undercomplete Convolutional Autoencoders. First, we define a *proxy* task based on the MNIST dataset, in which we consider a class as normal and all the others as anomalous. On this task, we compare different AL techniques adapted from literature. Later we compare the results of our model to a realistic dataset gathered using a ground robot.

Thesis structure

This thesis is structured in the following chapters:

- **chapter 1** discusses the background related work about Anomaly Detection, Active Learning, *uncertainty* and *representativeness* metrics;
- **chapter 2** defines our problem and the proposed solution;
- **chapter 3** explains the hardware and software setup, the models and the dataset used;
- **chapter 4** discusses the data collection and labeling process used to expand a real-life dataset;
- **chapter 5** explains the conducted experiments and the obtained results.

Chapter 1

Related work

In this chapter, we describe the background research conducted. We introduce both Anomaly Detection (AD) and Active Learning (AL) and show how they are treated in the literature. We then describe some AL techniques applied to different contexts.

1.1 Anomaly Detection

Anomaly Detection (AD) [34] is a research field that focuses on finding anomalous observations in mostly nominal data. Its application spans multiple disciplines such as engineering, machine learning, data mining, and statistics. AD has applications across different domains. Some of the domains are: cybersecurity [34, 25, 52], robotics [28, 49], biomedical imagery [38], automatic surveillance systems [6], insurance [47] and finance [1] fraud detection, and more [34].

Because AD is applied to such a broad range of contexts, a general definition of anomaly has to be as vague as possible. Ruff et al [34] define an anomaly as: "an observation that deviates considerably from some concept of normality", while Chandola et al [7] define it as: "patterns in data that do not conform to expected behavior." Both definitions make clear that a concept of *normality* has to be known and modeled to successfully perform AD.

In this work, we apply AD to corridor patrolling robots, described in chapter 2. We use AD techniques to images coming from the robot's camera. Currently, the state-of-the-art models for image AD are Deep Learning (DL) based models. Generally, these kinds of model training make use of supervised learning

approaches. However, anomalous events are *unexpected* and rare phenomena. For this reason, supervised learning approaches are not suitable because of the strong imbalance between the classes. Therefore in this setup unsupervised learning approaches are used. Models training uses normal data only. Their goal is to make the model learn a representation of *normality*.

Supervised methods are not suitable for AD because of this strong class unbalance and difficulty in modeling and formalizing anomalies. Since anomalous events happen rarely, *unsupervised* models for AD are used. These models are usually trained on normal data only.

1.1.1 Anomaly Detection in Images

Recently, Sabokrou et al [35] propose a novel model for image AD. They build a model composed of an Autoencoder (AE) and a Convolutional Neural Network (CNN), used as the network discriminator. This model is trained in an adversarial, unsupervised manner. The goal of the autoencoder is to learn how to reconstruct the input image after compressing it in such a way as to fool the discriminator. On the other hand, the discriminator has to tell whether or not the image it received is an original sample from the dataset. This model is trained on normal samples only. After training, when the model receives an anomalous image, the autoencoder will not be able to reconstruct it correctly. It will distort the anomalies making it simpler for the discriminator to reject the image.

Sarafijanovic et al [37] propose an inception [45] like Convolutional Autoencoder (CAE) and compared it against a CAE in the same AD task. The inception like AE combines convolutional filters with different kernel sizes at the same layer. The authors train the model on normal data only. Then instead of calculating the error map, they compute the distance between the pooled output of the bottleneck and its Nearest Neighbor (NN) in the same space. The models are tested over classical benchmark computer vision datasets: MNIST [24], Fashion MNIST [51], CIFAR10 [21], and CIFAR100 [22]. Their inception-like CAE outperformed state-of-the-art models in all tasks, except for Fashion MNIST.

1.1.2 Visual Anomaly Detection in Robotics

AD is a relevant problem in the context of robotics. It allows robots to find and avoid anomalies such as potential hazards never seen at design time. These

hazards could potentially be dangerous or affect the robot's operation, thus detecting an anomaly could be a *critical task* in some situations.

Early works, used an image processing pipeline with image and pattern matching algorithms for the task [6].

More recent works make use of Deep Learning (DL) approaches. Christiansen et al [8] propose DeepAnomaly, a modified AlexNet [23] implementation as solution to an AD problem for autonomous agricultural vehicles. Their goal is to avoid obstacles. AD is performed by applying *background subtraction* to some internal layer of the CNN, obtaining an *anomaly map*.

Wellhausen et al [49] compare three methods for AD applied to the context of traversability for legged robots. Their goal is to find a safe traversable path for the robot in an unknown environment. They compare an implementation of an AE with two different models which make use of the encoder part of the AE: *Deep SVDD* [39], and Real-NVP [33]. In the results, Real-NVP outperformed both the autoencoder and the Deep SVDD model, which scored the worst.

1.1.3 Autoencoder

Autoencoders [20] are artificial neural networks used in *unsupervised learning* and *representation learning* settings. In this kind of network, the output layer has the same size as the input one. The autoencoder is trained with the objective of learning to reproduce the input after encoding it in a lower-dimensional space (i.e., compressing the input data).

Figure 1.1 shows a schematic structure of an autoencoder. The depicted model has to learn how to encode all the information coming from the input X into the lower dimensional space z (Encoder part), named *bottleneck*. It has then to reconstruct the original input using only the encoded information obtaining the output X' (Decoder part).

A typical autoencoder learns the following mapping while trying to minimize the error (usually the Mean Squared Error (MSE)) between its input and output:

$$X' = D(E(X)) = D(z),\ z = E(X) \tag{1.1}$$

Where D is the decoder *forward* pass and E is the encoder one.

Autoencoders for anomaly detection in images

An autoencoder learns how to reproduce the data seen during training. This comes in handy when applied to image anomaly detection tasks.

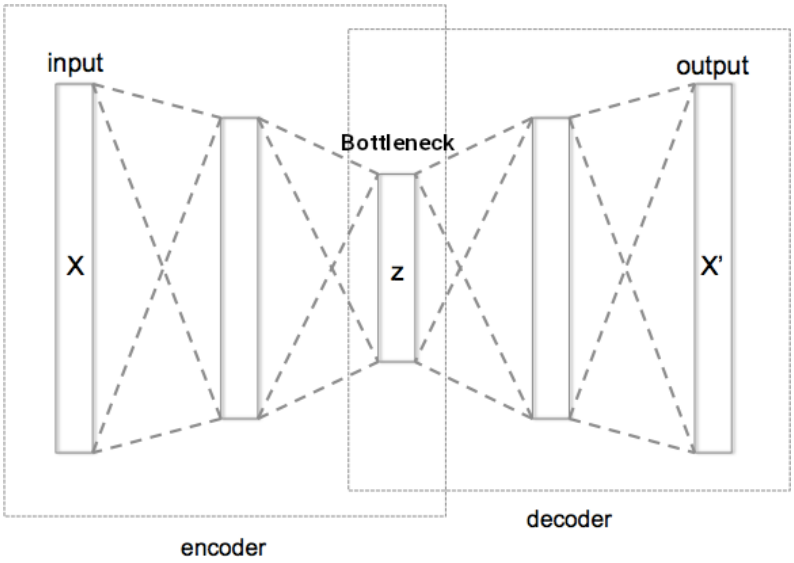

Figure 1.1. Schematic structure of an autoencoder [9], the code element z can be also referred to as latent space.

If an autoencoder is trained using only *non-anomalous* images, it learns to reconstruct them with no significant error. However, if an *anomalous* frame would be given as input to the model, it will not reconstruct the anomalies inside the image. Thus returning as output a new image with the anomalous part omitted. The autoencoders we use in this work are strongly limited by their bottleneck. This induces some reconstruction error even in normal samples.

By performing the difference between the input and output image, it is possible to compute an error map (Figure 1.2). In the Figure, we notice that the autoencoder can correctly reconstruct the scene. However, it was not able to reconstruct the human (anomaly) in the picture. By using the error map shown in the figure it is possible to compute an anomaly score which can be used to determine whether or not a given frame is *anomalous*.

1.2 Active Learning

Active Learning (AL) [41] is a branch of Machine Learning (ML) and Human-in-the-loop computing. Its goal is to find and query the best samples (i.e., the most *informative* to a Deep Learning (DL) model) to be labeled from an unlabeled pool of data.

Figure 1.2. Example of autoencoder for anomaly detection applied to our collected dataset. The model has been trained only in empty corridors. The human in the input image is thus an anomaly. From left to right: Input image of the model, Output of the model, Error map computed as $error = |input - output|$.

DL has become the state-of-the-art technique for many tasks. However, it relies on an enormous amount of data that needs to be labeled. In some contexts such as medical images, robotics, and safety-critical situations, the need for an expert makes the data labeling expensive, thus limiting the availability of large labeled datasets.

AL overcomes this problem by querying from the unlabeled pool of data the best samples to be labeled using some criteria. The goal is to obtain optimal model performance. This approach can drastically reduce labeling costs by reaching near state-of-the-art performance with a fraction of the data used.

Assuming a small labeled dataset L, a large pool of unlabeled data U, and a group of one or more experts (i.e., persons who label the data) are available. The goal of AL is to find a best $L^* \subseteq L$ given a current DL model $f(x|L^{'})$, $L^{'} \subseteq L$ where $L^{'}$ is an intermediate labeled dataset and U is the unlabeled pool from which query new samples for labeling to be added to L^*.

It is theorized [4] that it exists some $L^* \subseteq L$ which performance is close to the whole dataset L: $(f(x|L^*) \approx f(x|L))$, meaning that a DL model trained on L^* should achieve similar or equal performance to a model trained on the entire L dataset.

One crucial aspect of AL is the query approach. An AL framework queries the samples from the unlabeled set by using some criteria. These chosen samples are then labeled by one or more experts and used to extend the model's knowledge. This can be done in two ways: by fine-tuning the existing model or by retraining it using all the available data. Fine-tuning has been demon-

strated to outperform a network trained from scratch in the biomedical image field [46]. However, we will test only model retraining in our work.

This section will discuss and introduce some query approaches and several metrics for Active Learning.

1.2.1 Query approaches

Query approaches are used to sample the to-be-labeled data from the *unlabeled* data pool. In the following subsections, we describe some of the query approaches from the literature.

To develop an AL framework the first choice to consider is the type of query. Three choices are available:

- *Stream-based Selective Sampling*;
- *Membership Query Synthesis*;
- *Pool-based Sampling*.

Stream-based Selective Sampling

Stream-based Selective Sampling assumes data is coming in as a continuous stream. The model has to compute an *informativeness* score for each data point and decide whether or not to ask the oracle for annotations. Since the decisions are isolated this method does not provide significant benefits when compared to a random decision model.

Membership Query Synthesis

Membership Query Synthesis assumes that the to-be-labeled data is generated instead of being drawn from a real-world data pool. The data is generated to be the most *informative* to the model as possible. The generated data is going to be labeled by the oracle and integrated into the training set L. This approach can be very efficient in finite domains. However, it has some major drawbacks:

1. the model may not know certain unseen areas of the original distribution;
2. due to some small initial training set, it could also generate data that makes no sense to a human and require annotations for them.

The recent advance of *Generative Adversarial Networks* (GANs) [13] has shown great promise for these methods [4]. GANs can mimic real-world data with high accuracy, making them ideal for this setup.

Pool-based Sampling

Pool-based Sampling assumes a large unlabeled real-world dataset is available. It selects a batch of *N* samples to request labels for. This kind of method usually makes use of the currently trained model to make predictions on the unlabeled data to measure the informativeness metric and query the best *N* samples. Pool-based methods can be computationally expensive but are the most promising when combined with DL in a deep active learning framework [4].

We chose to work only on *Pool-based sampling* because we have access to the entire data available beforehand, making the stream-based methods not useful. On the other hand, we chose not to use membership query synthesis methods for time constraints since they require using and training Generative Adversarial Network (GAN) [13] models.

1.2.2 Informativeness metrics

After selecting the query method, an *informativeness* metric has to be defined to select the samples to label. Budd et al [4] explain the following types of metrics:

- Uncertainty
- Representativeness
- Generative adversarial networks for informativeness
- Learning active learning

For simplicity and time constraints we chose to work only with *uncertainty* and *representativess* metrics. Table 1.1 summarizes the metrics shown in the next subsections.

Uncertainty

Uncertainty can be a useful informativeness metric because it is argued that the more uncertain a model prediction is, the more information can be gained by including that sample in the training set.

Paper	Task	Type of data	Uncertainty estimation	Uncertainty metrics	Representativeness metrics
Wang et al [48]	Classification	Images		LC Rank, LC Rank, EN Rank	
Beluch et al [2]	Classification	Images	MC Dropout, Deep Ensembles.	Entropy, BALD, Variance	Core-set, REPR
Kirsch et al [18]	Classification	Images		BatchBALD	
Smailagic et al [43]	Classification	Images (Medical)		Distance between output of intermediate CNN layer	Distance between output of intermediate CNN layer
Gal et al [11]	Classification, Regression	Images, Time series	MC-Dropout	Bayesian approximation, Gaussian process	
Wen et al [50]	Classification (Binary)	Images (Medical)		Model posterior based	
Yang et al [53]	Segmentation	Images (Biomedical)	Bootstrapping	Bootstrapping	Cosine similarity
Ozdemir er al [30]	Segmentation	Images (2D, 3D, Medical)	MC-Dropout	Borda count	Cosine similarity, Entropy of intermediate CNN layer
Konyushkova et al [19]	Segmentation (Multi-class)	Images (2D, 3D, Medical)		Entropy: - Shannon, - Selection, - Conditional Geometric entropy	
Sourati et al [44]	Segmentation (Semantic)	Images (Medical)		Fisher information	Fisher information

Table 1.1. Different metrics found in the literature for active learning approaches

Uncertainty estimation Uncertainty can be estimated instead of calculated. In the literature, for the context of biomedical images, different methods for estimating uncertainty are proposed.

Monte Carlo Dropout (MC Dropout) [2, 11, 30] is one of these. Proposed by Gal et al [11], it makes use of regular dropout and interprets it as a Bayesian approximation of the Gaussian Model, a well-known probabilistic model. This approach applies dropout during model inference to generate T predictions for each input sample, which can then be averaged or their distribution can be analyzed, estimating the model uncertainty.

In addition to MC Dropout, Beluch et al [2] propose the use of *Deep Ensembles* to estimate model uncertainty. This approach trains N classifiers and then computes the average softmax output of the N predictions for each unlabeled sample.

Yang et al [53] make use *bootstrapping* [10] to estimate the model uncertainty. This method works by training a set of models on a different restricted random subset (random sampling with replacement) of the training data. The method then computes the variance between the model predictions.

Uncertainty metrics Uncertainty can be computed in several ways, depending on the task and field of the problem. It is usually computed using the lowest class probabilities, marginal sampling, or entropy [42].

Other uncertainty methods have been proposed, such as Cost-Effective Active Learning (CEAL) [48]. It uses traditional AL entropy-based methods to generate the D_L dataset with the most uncertain samples. It then introduces an additional step in which the most confident samples (whose *entropy* is below a threshold ω) are added to D_H. Both D_L and D_H are then used to fine-tune the model for a given number of iterations. Then the threshold ω is updated and the samples from D_H are added back to the unlabeled dataset. This method showed state-of-the-art performance using less than 60% of the available data [48].

Another approach uses Bayesian CNNs for AL in a method named Bayesian Active Learning by Disagreement (BALD) [12]. This approach is based on a Bayesian CNN. It produces a set of predictions using all the parameters and a set of stochastic predictions for each sample in the unlabeled dataset. Then the BALD acquisition function [15] is calculated as the difference in entropy between the average prediction and the average stochastic prediction. This method has been shown effective for AL [11]. However, it can result in redundant and very similar unlabeled samples being chosen. BatchBALD [18] solves this issue by calculating the mutual information in batches instead of on all the datasets.

In the field of medical image segmentation, Konyushkova et al [19] propose a novel uncertainty metric that considers the geometrical properties of the images. The proposed method builds a graph from the image. Each segmented region (i.e., *superpixel*) becomes a node, with edges linking neighbor regions. Edges in the graph are weighted with the probability of the transition to the same label as a neighbor. The intuition behind this metric is that the closer two regions are, the more likely they are to have the same label [19].

Representativeness

The representativeness metric is used to avoid problems that arise by focusing only on uncertainty to select the next samples to label (i.e., focusing on small regions of the distribution and redundancy/similarity in chosen data).

Representativeness metrics used are usually based on some distance metric between samples.

Beluch et al [2] uses *core-set* and *REPR* methods as *representativeness* metrics. The First method is a core-set-approach [40] that chooses p points that minimize the maximum distance between the point x_i and its closest neigh-

bor x_j in the chosen subset s. This approach combines both uncertainty and representativeness ideas in a single metric; the resulting metric u is computed greedily.

The REPR method on the other hand chooses points that best represent the rest of the data distribution greedily. Each sample of the unlabeled dataset U has a computed *representativeness* score, defined as the similarity between this point and its most similar one in the labeled set L. This approach encourages L to be as diverse as possible to represent in the best way the distribution of the unlabeled data. The similarity function Beluch et al [2] use is the Euclidean norm.

Samilagic et al [43] propose a novel model and a new metric. They try to combine uncertainty and representativeness by measuring the distance between the outputs of internal layers of a CNN. They test different distance metrics, with the Euclidean distance performing the best.

Ozdemir et al [30] take the work done by Smailagic et al [43] and try to improve it by letting the model learn representativeness by maximizing the activation entropy loss, a novel form of regularization.

Generative adversarial networks for informativeness

GANs can be used in an active learning framework to generate new data to be labeled and a metric of informativeness could be returned by the generator and/or the discriminator models [32, 26, 4].

A GAN is a neural network proposed by Goodfellow et al [13]. It is composed of two networks: a *generator* and a *discriminator*.

The two networks are set up in an adversarial manner: the objective of the generator is to generate samples that will fool the discriminator. The discriminator, on the other hand, has to predict whether the input sample was real or generated.

Both networks are trained at the same time, in a *minmax* setup: each network minimizes its loss and tries to maximize the loss of the other.

Learning active learning

The methods discussed so far rely on defined heuristics. The goal of *Learning active learning* techniques is to use a model to infer the best selection strategy based on the experience of previous AL outcomes. These techniques are an ideal use case for reinforcement learning (LR) methods [4]. A data selection policy learned this way can be agnostic to the data selection strategies, potentially

achieving better results and reaching state-of-the-art performance with even less sample than the heuristics shown above.

Chapter 2

Problem definition and solution design

In this chapter, we introduce the problem we are trying to solve, the metrics used, and the proposed solutions.

2.1 Problem introduction

AD applied to mobile robots allows them to find and avoid anomalies such as potential hazards never seen at design time. This is useful in many applications, such as hazard detection and avoidance during the exploration of tight tunnels to find sections that might need maintenance or replacement, or the autonomous exploration of a human-hostile environment. The robot's goal is to safely explore the environment and come back to the base to upload the collected data.

However, training DL models for AD with visual data requires a large amount of specific and labeled data. This labeling process requires experts in the field, whose time is limited, making the process expensive. The use of AL methods tries to reduce this cost by reducing the amount of data required. AL methods have never been applied to the context of AD for robots.

We study which approach from the literature can be adapted to work for our problem and we propose a new AL approach for AD in robotics.

Researchers of Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Intelligent Robotics lab in Lugano, Switzerland have developed a model [27] for visual anomaly detection in the context of mobile robotics. Their objective is to use only the visual sensing data stream coming from the robot's front-facing camera to detect hazards that could pose a risk to the robot.

The first goal of this thesis is to replicate the work done in IDSIA. Then, we extend the work by adapting AL techniques from literature to achieve similar performance while using a smaller training set.
For simplicity, the majority of work is conducted and focused on a proxy task based on the MNIST dataset: a class of the dataset is used as normal, and the others are considered anomalous. The reason for this choice is that working on MNIST is more time-efficient than working on a realistic dataset. It also is image-based, class balanced, and a well-known and widely used dataset in the literature.

The last experiments are conducted to compare the results of the found AL approaches on the realistic dataset gathered using a ground robot.

2.2 Active Learning metrics

2.2.1 Anomaly metrics

The anomaly score is a metric used to determine whether a frame is anomalous or not. For our work we used the MSE (Equation 2.1) between the input and the output of the AE [36].

Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a common metric used to evaluate the performance of regression models. It is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2 \tag{2.1}$$

where y_i is the ground-truth value and $\hat{y}_i$ is the predicted value for the i-th sample. The MSE is a good metric to evaluate the performance of regression models. However, it is not a good metric to evaluate the performance of classification models, since it does not take into account the class imbalance. We optimize this metric during the model training and we use it to compute an *anomaly score* during prediction.

Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.2}$$

where y_i is the true value and $\hat{y}_i$ is the predicted value for the i-th sample. The MAE is a good metric to evaluate the performance of regression models. As for the MSE, it is not a good metric to evaluate the performance of classification models. This metric is tracked during the model training.

2.2.2 Informativeness metrics

Average latent distance

As an informativeness metric, we use the metric proposed by Smailagic et al [43]. They try to combine uncertainty and representativeness by measuring the distance between the outputs of internal layers of a CNN and choosing the farthest sample:

$$s(x) = \frac{1}{N} \sum_{i=1}^{N} \text{dist}(e(x_i), e(x)), \quad x_i \in L_{train} \tag{2.3}$$

Where e is the feature extractor (in our case the encoder of the AE), x is the input frame from the unlabeled data, x_i is a frame from the training set, and N is the number of frames in the training set. $s(x)$ will be the informativeness of the sample x. This metric is computed for each sample in the unlabeled set. Then the samples with the highest score are selected for labeling.

Min-Max approach

The Min-Max approach is inspired by the metric proposed by Beluch et al [2] (Equation 2.4).

The proposed approach avoids selecting similar (i.e. close in latent space) samples. It works in the following way:

1. computes the distances between each sample in the labeled set and the unlabeled samples;
2. for each unlabeled sample selects the closest labeled one;
3. for each selected labeled sample selects the furthest unlabeled sample.

If the min-max approach is *naive* it queries all required samples in a single pass. Otherwise, every time a sample is queried, this is inserted into the labeled set. The distances are updated and a new sample is queried in the next pass.

$$u = \arg\max_{i \in [n] \setminus s} \min_{j \in s} dist(z_i, z_j) \tag{2.4}$$

2.3 Proposed solutions

We propose 8 AL setups, named ALi with i an integer from 1 to 8 (e., g., $AL1$, $AL2$, ...) briefly summarized in Table 2.1.

Method	Description	AL Metric	AL Space Dimensionality
AL1	Baseline no AL	None	None
AL2	Random AL	None	None
AL3.1	Most Anomalous	Anomaly score (MSE)	1
AL3.2	Most Anomalous Intermediate model	Anomaly score (MSE)	1
AL4	Mean distance on latent space	Distance on AE's bottleneck	bottleneck size
AL5	Hybrid: Most anomalous and mean distance	Anomaly score + Distance on AE's bottleneck	1 and bottleneck size
AL6	Naive MinMax	MinMax distance on AE's bottleneck	bottleneck size
AL7	Iterative MinMax	MinMax distance on AE's bottleneck	bottleneck size
AL8	Hybrid: MinMax and most anomalous	Anomaly score + AE's bottleneck	1 and bottleneck size

Table 2.1. Summary of the AL techniques used

2.3.1 Baselines

$AL1$ and $AL2$ are the baseline approaches. In the first one, a model is trained without the use of AL. The latter randomly queries samples from the unlabeled pool.

2.3.2 Anomaly-based approaches

Anomaly-based approaches use only the anomaly score as a metric to perform AL. These models are $AL3.1$ and $AL3.2$

For $AL3.1$ and $AL3.2$, we choose to query the most anomalous samples (i.e., the samples with the highest anomaly score) by using the MSE between the input and the output of the trained AE. Both approaches are similar in function: an AE is initially trained on a small initial training set. Then, this model is used to query new samples from the *unlabeled* pool using the model's anomaly

score. However, *AL*3.2 takes an intermediate step: it trains a second intermediate model with half of the required samples. It then uses this newly trained model to query the second half of the samples from the unlabeled set. The final model is trained on all of the newly labeled samples.

2.3.3 Informativeness-based approaches

Informativeness-based approaches make use of *informativeness* metrics to perform AL. These methods are *AL*4, *AL*6, and *AL*7.

As for the anomaly-based approaches, these methods first train an intermediate model on a small initial training set. This model is then used to query the samples from the unlabeled pool using *informativeness* metric.

*AL*4 queries the to-be-labeled samples from the unlabeled set using the *average latent distance* metric described in section 2.2.2.

*AL*6 and *AL*7 use the *MinMax* approach described in section 2.2.2. The difference between them is that while *AL*6 uses the *naive* variant of the approach, *AL*7 does use the iterative version. In this version, the distances are recomputed after each sample is queried and labeled.

2.3.4 Hybrid approaches

Hybrid approaches use both *anomaly* and *informativeness* metrics. These methods are *AL*5 and *AL*8.

*AL*5 Combines *AL*3 with *AL*4. It queries the first half of the required unlabeled samples using the criterion of *AL*3, the other half with *AL*4's one.

*AL*8 Uses a hybrid version iterative *min-max* approach, which includes some concept of *anomaly*. It begins by using the naive min-max approach to query a small subset of samples. Then, from these, it queries the one with the highest *anomaly score*. This sample is labeled and added to the labeled set. Distances are then recomputed and the method continues until all the required samples are selected.

Chapter 3

Implementation and experimental setup

In this chapter, we describe the experimental setup used to perform the experiments described in chapter 5. We begin by showing the software and hardware that made this work possible. Then we describe the dataset used to train and test the models. Finally, we show and explain the metrics used to evaluate the models.

3.1 Software

The project is developed in Python 3.9 and composed of multiple classes. A handful of libraries and frameworks are used to develop the code, namely:

- Pytorch [31]
- WandB [3]
- Scikit-learn
- Albumentations
- Matplotlib and Seaborn
- Pandas
- Numpy

The final code is available on GitHub and in Appendix A.

3.1.1 Pytorch

Pytorch [31] is an open-source Machine Learning (ML) Python framework used to build, train, and test the DL models. This framework was originally developed by Meta, but now it is part of the Linux Foundation. It is a Deep Learning (DL) framework that provides two high-level features:

- Numpy-like Tensor computation with strong GPU acceleration.
- Deep neural networks built on a tape-based autogradient system.

This framework is used as the base to develop, build, train and test the models used in this work.

3.1.2 WandB

Wandb [3] is a dashboard used to collect and display all the metrics of the models. It is a tool that allows to track and visualize the training of the models. It is also used to compare the results of different models.

We use this library to keep track of model metrics during training, validation, and testing.

3.1.3 Scikit-learn

Scikit-learn is a Python library used to compute model metrics. It is a Python Machine Learning (ML) library. It features various classification, regression and clustering algorithms. Some of which include support vector machines, random forests, and more. It is designed to interoperate with the Numpy and Scipy libraries.

This library is used to compute the model test metrics, in our case the Area Under the Curve (AUC) score.

3.1.4 Albumentations

Albumentations [5] is a computer vision library used to perform *data augmentation* in image datasets. This library is a part of the PyTorch ecosystem, it can be easilly integrated with the most used DL frameworks.

We use this library to perform data augmentation on the Hazards&Robots dataset, Corridors scenario.

3.1.5 Matplotlib

Matplotlib is a Python library used to generate charts. It is a comprehensive library for creating static, animated, and interactive visualizations in Python.

3.1.6 Numpy

Numpy is a widely used Python library used to manipulate arrays. It is a core library for Python scientific computing. It contains among other things: a powerful N-dimensional array object (ndarray), sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, useful linear algebra, Fourier transform, and random number capabilities.

Vectorization and broadcasting are two essential concepts in Numpy. Vectorization makes writing array operations easier by making loops and array indexing implicit. Broadcasting is a set of rules for applying binary functions (addition, subtraction, multiplication, ...) on arrays of different sizes.

The library provides a multidimensional array object named ndarray, various derived objects (such as masked arrays and matrices), and many routines for fast array operations, some of which include mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, statistical operations, random simulation and more.

The ndarray object is the core of the Numpy library. This object is a multidimensional array of homogeneous data types. It provides many operations written in C for improved performance. It is a fast and space-efficient multidimensional container for generic data.

We use this library in conjunction with Pytorch and Scikit-learn to compute model metrics.

3.2 Hardware

3.2.1 Workstation

The majority of experiments are conducted on a cluster available to IDSIA intelligent robotics lab. The cluster has the following hardware:

- 4 Nvidia RTX 2080TI GPUs
- 128GB of ram
- 2 Intel(R) Xeon(R) Gold 5217 8-core, 16-threads CPUs

Model training uses only one GPU at a time.

Some test experiments and initial model testing are conducted on a mobile workstation equipped with:

- Nvidia Quadro T2000 GPU
- 16GB of ram
- Intel(R) Core(TM) i7-9850H CPU

3.2.2 DJI Robomaster S1

The robot used for this project is the DJI Robomaster S1 (Figure 3.1, 3.2), the first consumer-level ground robot of the company. It is named after the company's annual robot combat competition.

The S1 is a remotely controlled, tank-like rover that can be operated in first-person view using Wi-Fi and an app on devices running Android, iOS, or Windows. The user has to assemble the robot out of the box from loose parts and learn to program it. It is intended to be an "advanced teaching robot" meant to teach kids how to code.

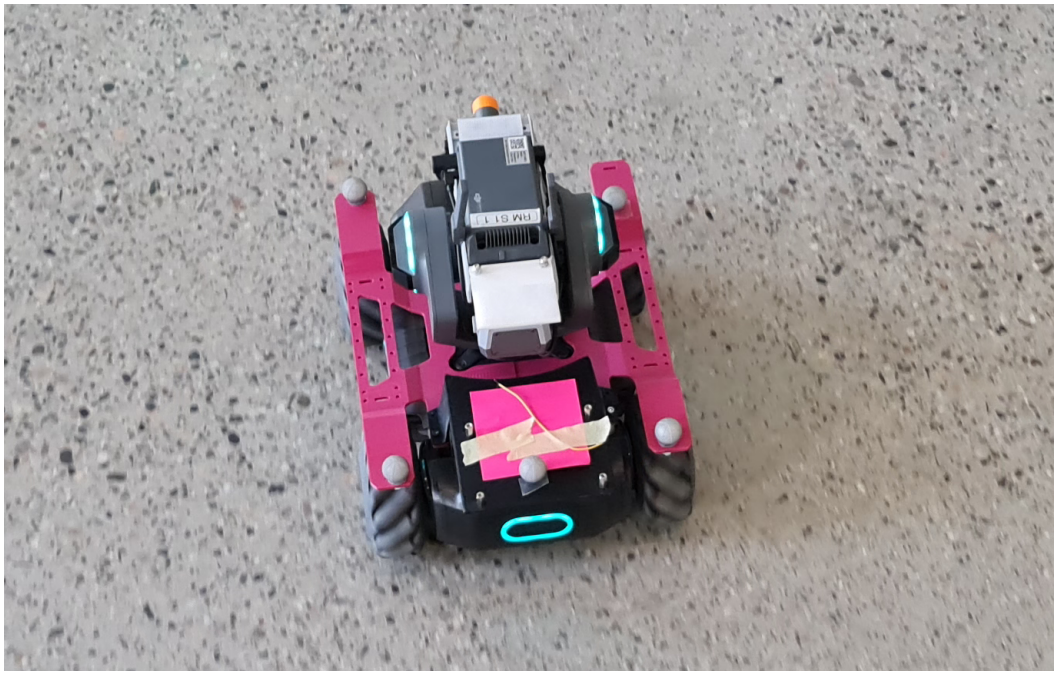

Figure 3.1. Robomaster S1, picture shot during data collection.

Hardware description

The Robomaster S1 is a ground robot capable of both indoor and outdoor drives. Thanks to its *Mecanum wheels* [16] it can move in any direction. It can achieve a maximum speed of 13 km/h moving forwards, 9 km/h moving backward, and 10 km/h moving laterally.

The robot comes with a 2400 mAh battery which lasts about 30 minutes when fully charged. A quick battery release system lets the user change an empty battery with a charged one in a matter of seconds.

The robot is usually remote-controlled using the Robomaster app on a smartphone, tablet, or computer. The robot can be configured to act as an Access Point (AP) or connect to a WLAN router. Depending on the connection type and the frequency used (the robot supports both 2.4 and 5 GHz networks) the operating range may vary from a minimum of 90m to a maximum of 300m.

Once connected to the robot the app provides the live camera feed from the robot's camera, letting the user control it in first-person view.

For the data collection phase, the robot is operated using the app and the videos are recorded to an onboard micro sd card.

Both python and scratch official SDK is available for the robot. The python SDK is a wrapper around the Robomaster's API and allows to control of the robot using python scripts. The robot can be also interfaced with Robot Operating System (ROS) via an unofficial API wrapper developed in IDSIA [14].

Camera

The Robomaster S1 comes with a 120° FOV 5 Megapixel camera capable of recording Full-HD video at 30 FPS with a maximum bitrate of 16Mbps. The camera is mounted on a 2-axis gimbal, which provides mechanical camera stabilization for Pitch and Yaw.

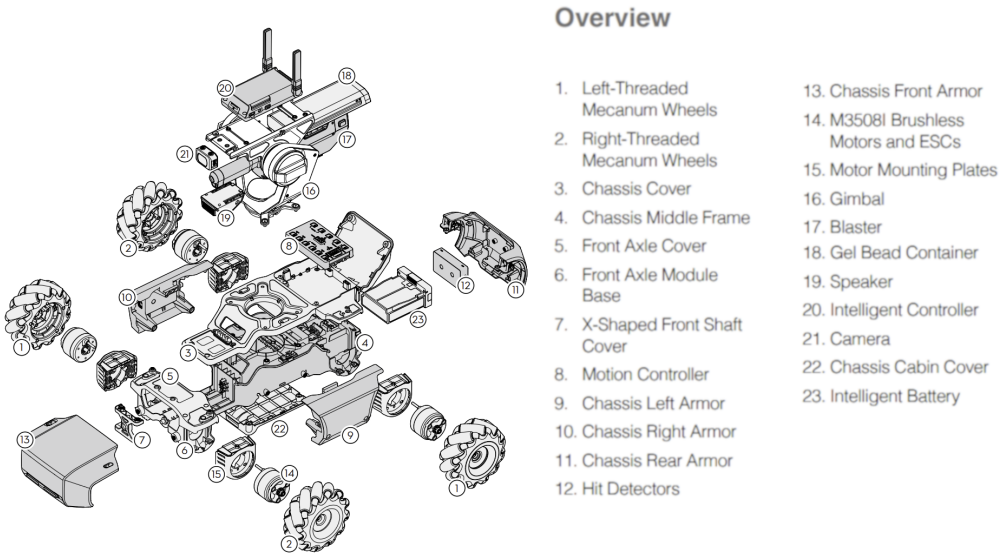

Figure 3.2. Robomaster S1, exploded-view from user manual with part names.

3.3 Anomaly Detection model

The DL model we use to solve the task of anomaly (shown in Table 3.1) is a Convolutional Autoencoder (CAE) implemented in pytorch [31], inspired from the work done in IDSIA [27].

The encoder is a Convolutional Neural Network (CNN) composed of 4 convolutional layers with dropout inserted after the second convolution and the third one.

The network bottleneck is a Fully Connected Network. Its input layer receives the flattened output of the encoder. The *hidden layer* has dimension *16*, and the output layer has the same size as the input one ($64 * 64 * 3$ for the RoboMaster dataset, $28 * 28$ for MNIST).

The decoder has the inverted structure of the encoder. It uses *Transposed Convolutional* layers and dropout.

The *Reconstruction error* is computed as the Mean Squared Error (MSE) (Equation 2.1) between the input frame and the output of the autoencoder. This metric is used to check whether a frame is anomalous: the higher the error (anomaly score), the higher the probability the frame is anomalous.

The models are trained using the Adam optimizer [17] with a learning rate

Layer			Output Shape	Number of Parameters
Autoencoder				
|-	ConvolutionalEncoder		[1, 2048]	
|	|-	Conv2d	[1, 16, 26, 26]	160
|	|-	Conv2d	[1, 32, 24, 24]	4640
|	|-	Dropout	[1, 32, 24, 24]	
|	|-	Conv2d	[1, 64, 11, 11]	18496
|	|-	Dropout	[1, 64, 11, 11]	
|	|-	Conv2d	[1, 128, 4, 4]	295040
|-	Bottleneck		[1, 128, 4, 4]	
|	|-	Linear	[1, 64]	131136
|	|-	Linear	[1, 2048]	133120
|-	ConvolutionalDecoder		[1, 1, 28, 28]	
|	|-	ConvTranspose2d	[1, 64, 12, 12]	294976
|	|-	ConvTranspose2d	[1, 32, 25, 25]	18464
|	|-	Dropout	[1, 32, 25, 25]	
|	|-	ConvTranspose2d	[1, 16, 27, 27]	4624
|	|-	Dropout	[1, 16, 27, 27]	
|	|-	ConvTranspose2d	[1, 1, 29, 29]	145

Total params: 900,801
Trainable params: 900,801
Non-trainable params: 0
Total mult-adds (M): 67.51

Input size (MB): 0.00
Forward/backward pass size (MB): 0.66
Params size (MB): 3.60
Estimated Total Size (MB): 4.27

Table 3.1. Model architecture

of 1e-4 and random noise has been added to the images during training.

3.4 Evaluation metrics

During the training phase, the model optimizes the Mean Squared Error (MSE) (Equation 2.1), with the Mean Absolute Error (MAE) (Equation 2.2) being tracked as a side metric to check the performance. During the test phase, the Area Under the Curve (AUC) (Equation 3.3) metric is used to evaluate the model performance on the test set.

Before introducing the metrics we first define the following terms:

- **TP** True Positive: the number of correctly classified positive samples;
- **TN** True Negative: the number of correctly classified negative samples;
- **FP** False Positive: the number of incorrectly classified positive samples;
- **FN** False Negative: the number of incorrectly classified negative samples.

3.4.1 True Positive Rate (TPR)

The True Positive Rate (TPR) is defined as:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.1}$$

where TP is the number of true positives and FN is the number of false negatives.

3.4.2 False Positive Rate (FPR)

The False Positive Rate (FPR) is defined as:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{3.2}$$

where FP is the number of false positives and TN is the number of true negatives.

3.4.3 Area Under the Curve (AUC)

The Area Under the Curve (AUC) is a common metric used to evaluate the performance of binary classification models. It is defined as:

$$\text{AUC} = \int_0^1 \text{TPR}(\tau) - \text{FPR}(\tau) \tag{3.3}$$

where τ is the threshold value. The AUC is a good metric to evaluate the performance of binary classification models.

An AUC of 0.5 means that the model is performing as well as a random model, while a AUC of 1.0 means that the model is performing perfectly. In probabilistic terms, the AUC is the probability of predicting a sample from the positive class as positive.

3.5 Datasets

The used datasets are MNIST [24], a simple dataset used as *proxy* that makes the model testing and debugging phase easier, and the Hazards&Robots dataset, Corridors scenario [28].

MNIST

MNIST was adapted to the task of AD by using a single class as normal, thus training the model on that class only.

The MNIST (Figure 3.3) dataset contains 60,000 training images and 10,000 test images of handwritten digits. Each image is a 28x28 grayscale image. The dataset is divided into 10 classes, one for each digit. The dataset is balanced, meaning that each class has the same number of samples.

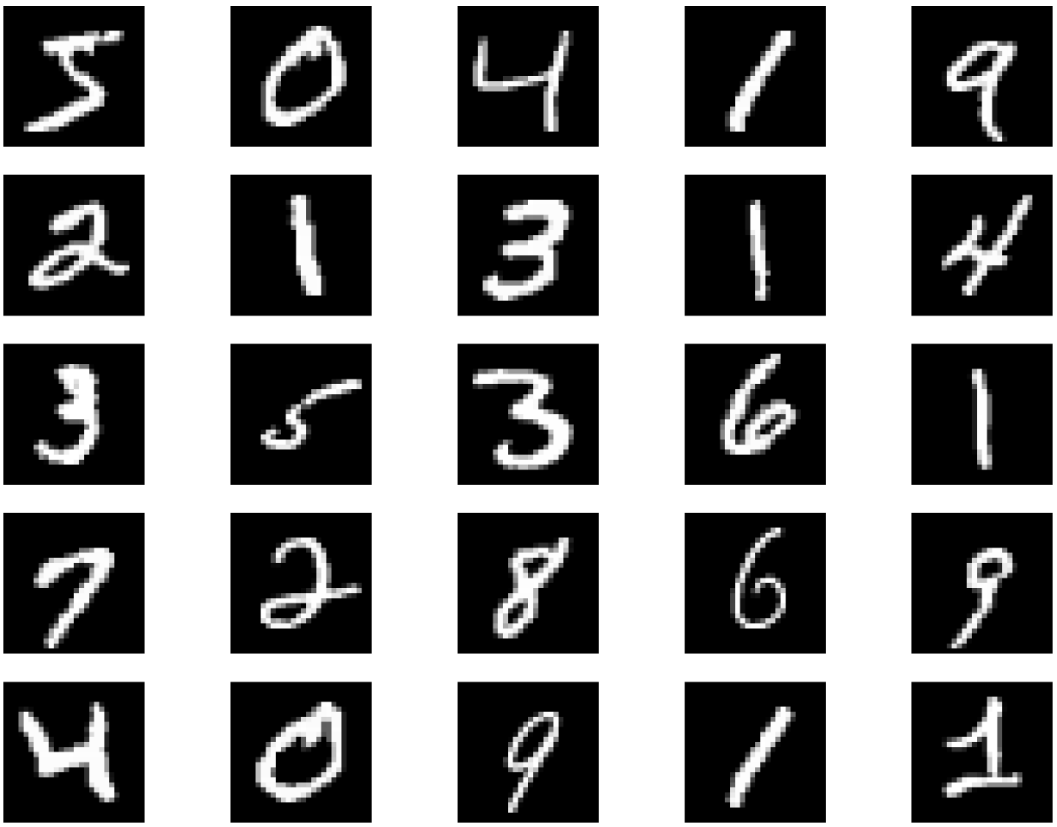

Figure 3.3. Images from the MNIST dataset.

Hazards&Robots dataset

We extend the Hazards&Robots dataset [28], corridors scenario (Figure 3.4) by collecting new frames using the ground robot described in subsection 3.2.2. The robot was driven around the campus of Università della Svizzera Italiana (USI), collecting images. The dataset contains 69,499 train images, 155,133 test images, and 4,163 images. Each sample is an RGB image with a resolution 64×64.

The dataset is structured into 21 classes, shown in Figure 3.4 and explained in chapter 4.

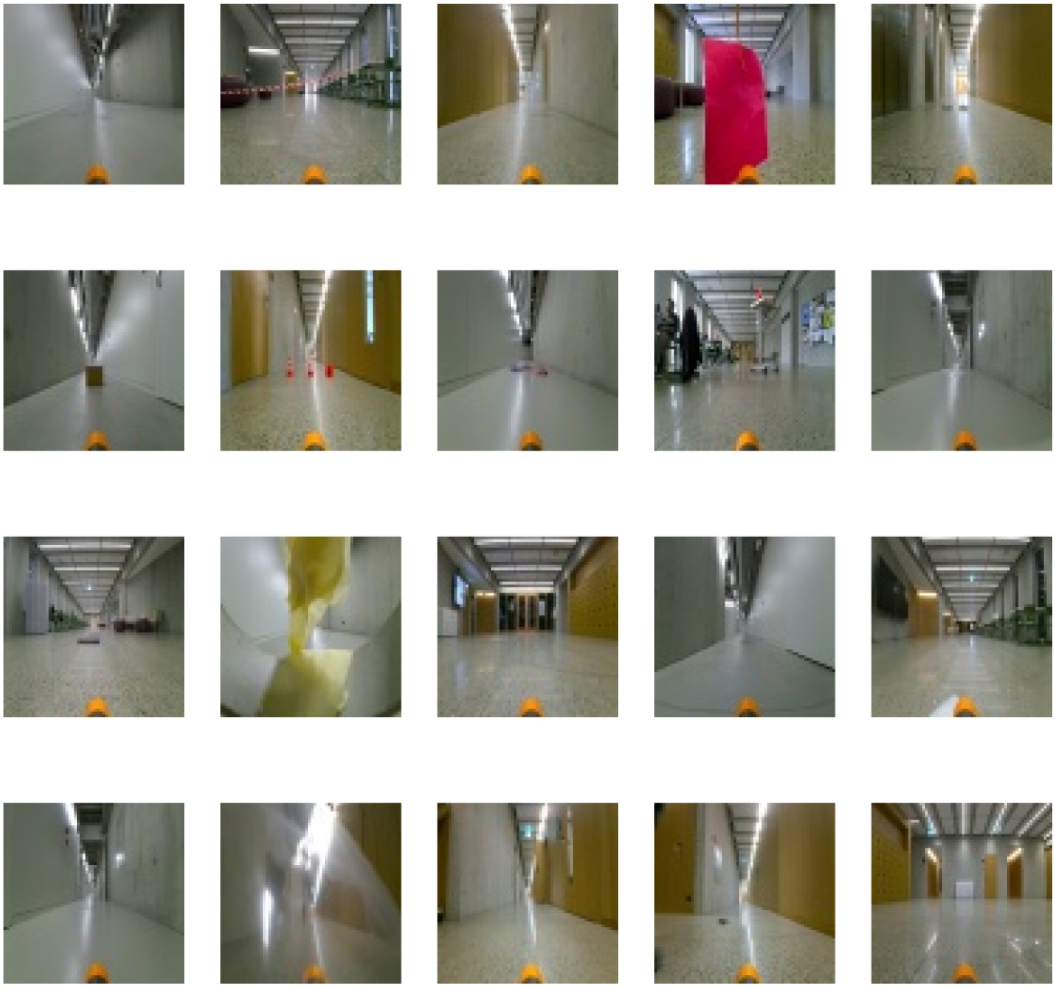

Figure 3.4. Images from the Hazards&Robots dataset.

Chapter 4

Data Collection

We extend the Robots&Hazards dataset [28] Corridors scenario to train and test the model we built. The dataset is collected by teleoperating the Robomoster S1 through the corridors of the Università della Svizzera Italiana (USI) east campus.

This extended version of the dataset is not yet published.

The Corridors scenario of the Robots&Hazards dataset consists of 6 corridors of the campus, divided into 3 scenarios:

1. *long corridor*: first floor corridors of sectors A and C;
2. *short corridor*: first floor corridors of sectors B and D;
3. *underground*: underground corridors of sectors B and C.

4.1 Data collection

Figure 4.1 shows the data collection process. The robot is teleoperated using a smartphone. For each corridor and both directions, the video feed of the robot traversing the empty corridor is saved.

For long videos, the robot has to move across the whole length of the corridor without stopping and while moving at a constant speed. The speed has to be fixed such that the robot would take approximately 40 seconds to traverse the corridor.

Short videos are recorded in small, random sections of the tunnel. The robot has to move at a constant speed and without stopping. The speed has to be fixed such that the robot would take approximately 10 seconds to traverse the section.

The anomaly in these videos is placed at 2m from the Robomaster. The robot has to move forward close to the anomaly while keeping it always in frame. Then it has to move back to the start position.

The hypothetical anomalies are staged by manually placing them into the environment and driving the robot near them while recording. Figure 3.4 contains 16 frames taken from the test set, and the majority of the frames shown is *anomalous*.

After all of the videos have been captured, they are processed with the *OpenCV* library. The videos are then divided into frames. Each frame is resized into $64x64$ RGB pictures.

Figure 4.1. Robomaster S1, picture shot during data collection.

4.2 Labeling

The videos are split into two groups: *long* and *short*, based on the video length.

4.2.1 Long videos

The long videos contain 5 classes. The labels are the following:

Normal

The normal class (Figure 4.2) contains frames where the camera is not obstructed by any object. The camera is clean and the robot's path is not obstructed.

Figure 4.2. Robots&Hazards dataset Corridors scenario, label normal

Human

The human class (Figure Figure 4.3) contains frames where a person is obstructing the robot's path. The person is usually standing in front of the robot and moves in the same direction as the robot.

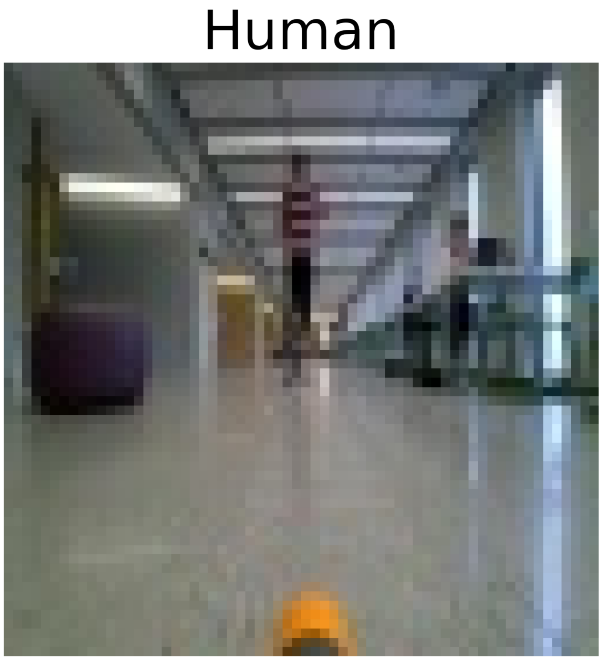

Figure 4.3. Robots&Hazards dataset Corridors scenario, label human

Defects

The defects class (Figure 4.4) contains frames where the camera has some kind of defect. The defect can be a scratch, a stain, or a fingerprint on the lens. This anomaly is simulated by placing an antistatic bag on the camera.

Figure 4.4. Robots&Hazards dataset Corridors scenario, label defects

Object on Robot

The object on robot class (Figure 4.5) contains frames where the robot FOV is obstructed by some kind of object fallen and stuck over the camera. Figure 4.6 shows the robot with the anomaly mounted on it.

Figure 4.5. Robots&Hazards dataset Corridors scenario, label Object on Robot

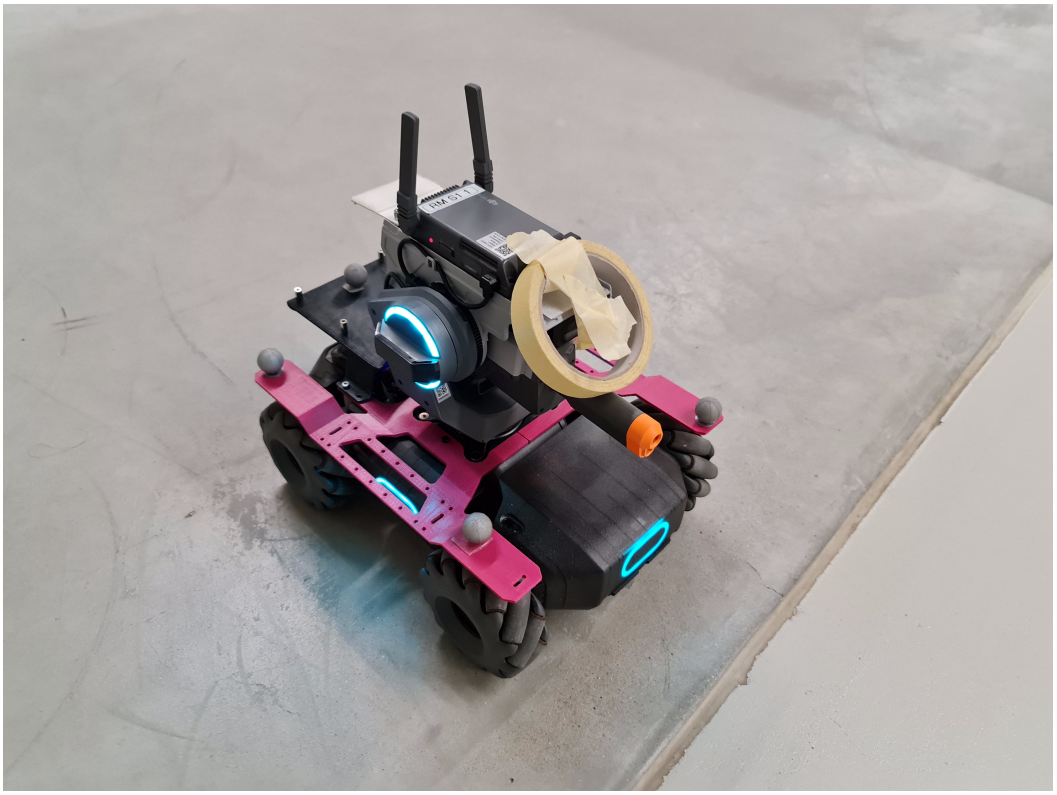

Figure 4.6. Robomaster S1 mounted with the Object on robot anomaly

Object on Robot 2

This class is similar to the previous one, but the object is placed on the robot differently. The object is placed further in the camera in a way to only obstruct the central part of the robot's FOV. In Figure 4.8 we can see the robot with the anomaly mounted on.

Figure 4.7. Robots&Hazards dataset Corridors scenario, label Object on Robot 2

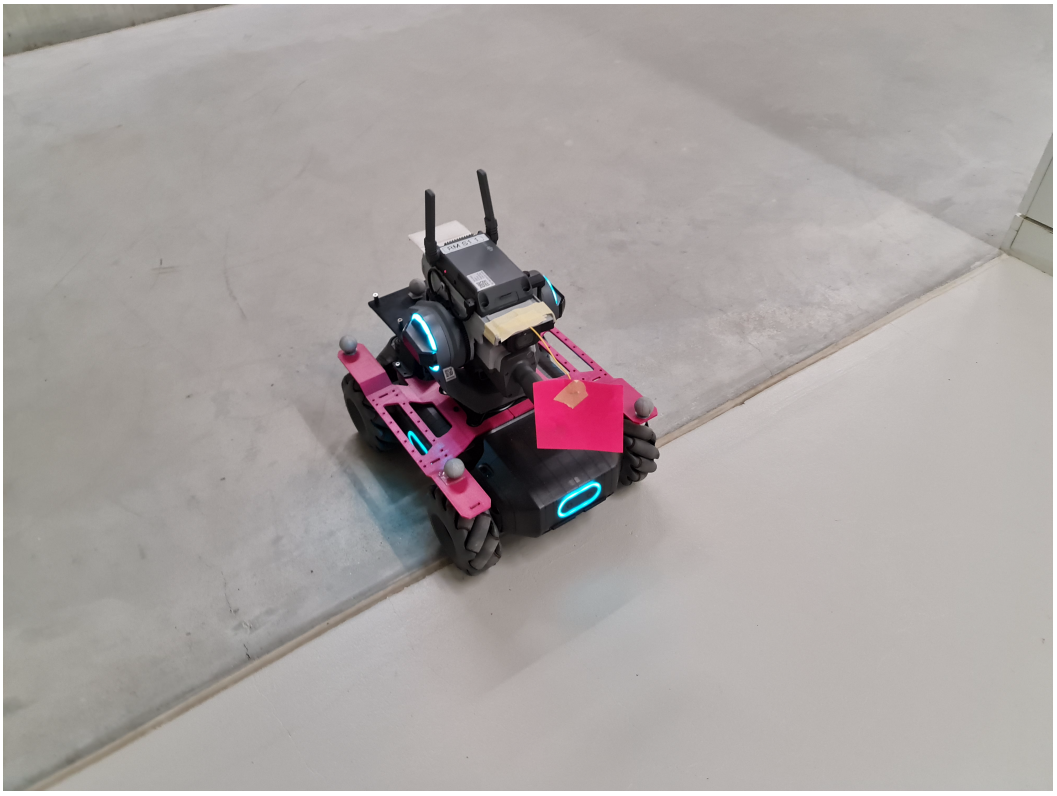

Figure 4.8. Robomaster S1 mounted with the Object on robot2 anomaly

4.2.2 Short videos

Short videos contain 7 classes. The labels are the following:

Cones

The cones class (Figure 4.9) contains frames where the path is obstructed by cones. The cones are usually placed in front of the camera, but they can also be placed on the sides.

Figure 4.9. Robots&Hazards dataset Corridors scenario, label cones

Debris

The debris class (Figure 4.10) contains frames where the camera is obstructed by some kind of debris. The debris can be a piece of concrete, a piece of wood, or a piece of plastic. This anomaly is simulated by placing in front of the robot a broken concrete slab.

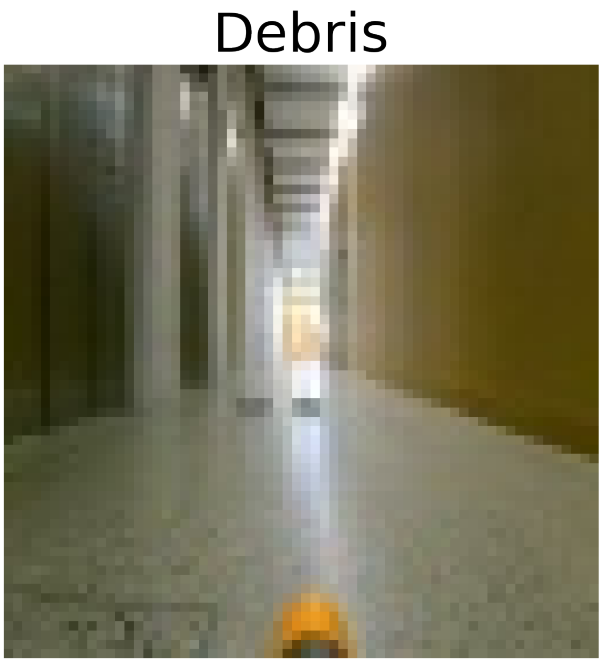

Figure 4.10. Robots&Hazards dataset Corridors scenario, label debris

Floor

The floor class (Figure 4.11) contains frames where the path is obstructed by some floor anomaly. The anomaly is simulated by placing a carpet on the floor.

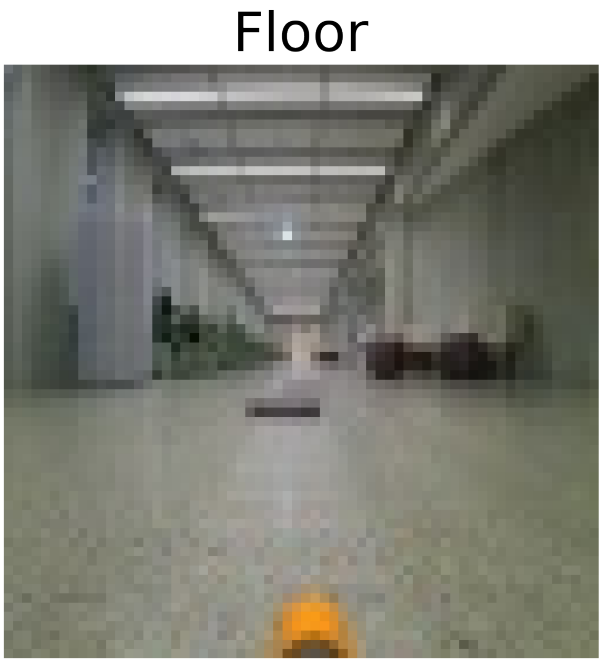

Figure 4.11. Robots&Hazards dataset Corridors scenario, label floor

Tape

The tape class (Figure 4.12) contains frames where the path is obstructed by some tape placed at random heights, even higher than the robot's height.

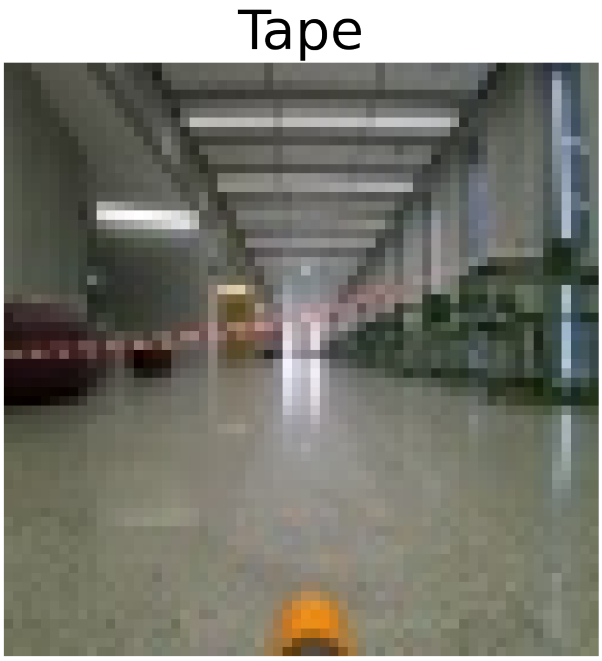

Figure 4.12. Robots&Hazards dataset Corridors scenario, label tape

Trolley

The trolley class (**??**) contains frames where the path is obstructed by a trolley. The trolley can be a small or large.

Figure 4.13. Robots&Hazards dataset Corridors scenario, label trolley

Cable

The cable class (**??**) contains frames where the path is obstructed by a cable. The cable can be a power cable, a network cable, or a USB cable.

Figure 4.14. Robots&Hazards dataset Corridors scenario, label cable

Box

The box class (**??**) contains frames where the camera is obstructed by a box. The box can be of any size.

Figure 4.15. Robots&Hazards dataset Corridors scenario, label box

Chapter 5

Experiments and Results

In this chapter, we introduce the experimental setup and explain the conducted experiments and the obtained results.

5.1 Experimental setup

For each experiment, the training set is divided into three partitions: *A*, *B*, and *C*, each one with a predetermined size. *A* and *B* partitions will be used as the initial labeled training set, while *C* will be used as the large *unlabeled* pool.

In case the number of samples in the obtained training set is less than the size of the entire training set (i.e., $|A|+|B|+|C|$), *resampling* is performed. This makes for a fair comparison between the different AL approaches.

More in detail, experiments are conducted by training and testing 10 runs off an autoencoder the AL techniques explained in section 2.3.

AL1

This baseline model does not use any AL method. The model is trained only on the *A* and *B* partitions of the training set.

AL2

This is our second baseline model, it randomly queries *n* samples from partition *C*. The final model is trained on partitions *A*, *B*, and the newly sampled data.

AL3.1

This approach trains an intermediate model ($AL1$) using only the A and B partitions. It then uses this model to query the most anomalous n samples from C to be included in the new training set for the model. The final model is trained on the new extended set.

AL3.2

This approach works similarly to $AL3.1$, although it uses another intermediate model:

1. the most anomalous $n/2$ samples (wrt AL1) from partition C are queried;
2. an intermediate (AL-tmp) model is trained on this new expanded dataset;
3. the final dataset is obtained by the AL-tmp training set plus the $n/2$ most anomalous samples (wrt AL-tmp).

AL4 to AL8

All these approaches use $AL1$ as intermediate model to query the n most *informative* samples from partition C following the criteria shown in section 2.3.

5.2 Model exploration

Preliminary tests are conducted to check whether the models were learning correctly.

These tests are conducted by both checking the AUC of the models in the test set and by visualizing the reconstructed images. Figures 5.1, 5.5 show model predictions for a *normal* and an *anomalous* sample. The model can reproduce any normal sample with little to no error. However, if the model receives an anomalous sample, it will not be capable of reconstructing it correctly. This results in the "brighter" anomaly map in both Figures.

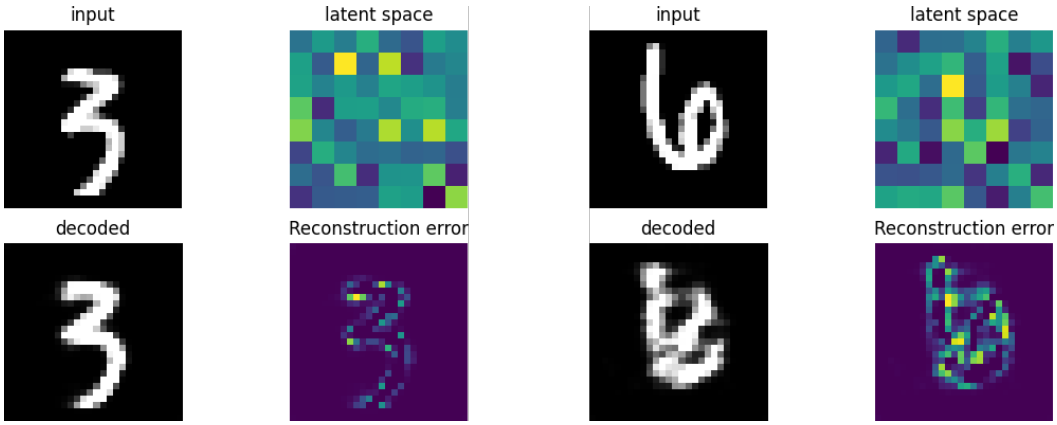

Figure 5.1. Example of predictions from a model trained on MNIST. The input image on the left is from the normal class, while the other on the right is anomalous

5.2.1 Early phase

In an early phase, we studied models with extreme architectural characteristics such as an autoencoder with no bottleneck. This was done to study the bottleneck size hyperparameter for the MNIST dataset shown in subsection 5.4.1.

5.2.2 Dummy models

We test some dummy models to check whether our model evaluation pipeline is working correctly. To test it we developed and deployed three dummy models:

- DummyFalse
- DummyTrue
- DummyRandom

The first model predicts any sample as *normal*, and the second predicts everything as *anomalous*. The third model randomly chooses between the two classes for any given input.

As expected the AUC of DummyFalse and DummyTrue is 0.5. For DummyRandom the boxplot in Figure 5.2 shows this model has an AUC score close to 0.5.

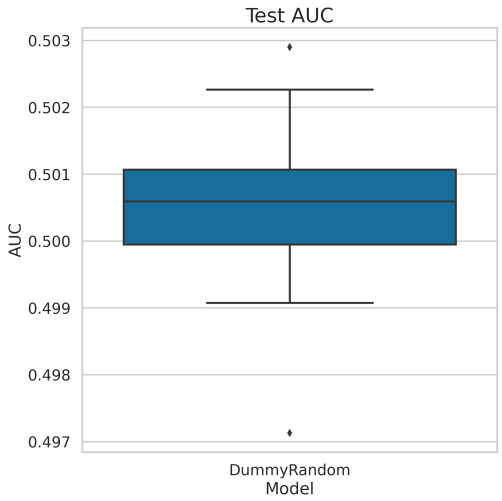

Figure 5.2. Test AUC for DummyRandom model, boxplot of 18 runs.

5.3 Proxy task

We use the MNIST dataset for a *proxy* task to perform AD. We achieve this by using one class (3 in our case) of the dataset as normal. All the other classes are considered *anomalous*.

With this proxy task, the autoencoder learns how to extract the most useful features and reproduce the normal input as best as possible. We then exploit the computed error map to solve our real task which is AD.

5.4 Experiments on MNIST

Most of the experiments are conducted on the *proxy task* defined on the MNIST dataset. In this section we show the experiments and results obtained.

5.4.1 Bottleneck size

One of the first experiments on MNIST is conducted to find the best bottleneck size for the task.

For simplicity and time constraints, this experiment is limited only to the approaches $AL1$, $AL2$, and $AL3.1$. It consists of training and testing 13 runs of models with different bottleneck sizes. The goal is finding the smallest bottleneck size which leads to the best results.

For these experiments we test the following bottleneck sizes:

- 0, 1, 2, 4, 6, 8, 16, 32, 64, 128, 256, 512

The results are shown in Table 5.1.

Based on the obtained results, we choose to use a bottleneck size of 64 for the next experiments with the MNIST dataset.

Model	Mean	Stdev	Model	Mean	Stdev	Model	Mean	Stdev
AL1-0	0.813	0.186	**AL2-0**	0.863	0.000	**AL3.1-0**	0.866	0.001
AL1-1	0.880	0.004	**AL2-1**	0.882	0.004	**AL3.1-1**	0.887	0.004
AL1-2	0.882	0.004	**AL2-2**	0.900	0.003	**AL3.1-2**	0.901	0.004
AL1-4	0.900	0.003	**AL2-4**	0.918	0.004	**AL3.1-4**	0.921	0.004
AL1-8	0.918	0.004	**AL2-8**	0.937	0.005	**AL3.1-8**	0.942	0.002
AL1-16	0.953	0.006	**AL2-16**	0.958	0.005	**AL3.1-16**	0.966	0.003
AL1-32	0.964	0.005	**AL2-32**	0.970	0.005	**AL3.1-32**	0.976	0.003
AL1-64	0.966	0.005	**AL2-64**	0.973	0.004	**AL3.1-64**	0.979	0.003
AL1-128	0.967	0.005	**AL2-128**	0.975	0.004	**AL3.1-128**	0.979	0.003
AL1-256	0.968	0.006	**AL2-256**	0.974	0.006	**AL3.1-256**	0.979	0.003
AL1-512	0.966	0.006	**AL2-512**	0.975	0.004	**AL3.1-512**	0.979	0.004

Table 5.1. Bottleneck research results. From left to right: Results for AL1 models, results for AL2 models, and results for AL3 models. The average and standard deviation are computed over 13 runs.

5.4.2 Bottleneck activation function

The second set of experiments is conducted to find the best activation function for the bottleneck layer.

The experiment is conducted by performing 10 runs of models with different bottleneck size activations. For simplicity, we limited the experiment only to the approaches $AL1$, $AL2$, and $AL3$.

We tested some of the most used activation functions:

- Sigmoid
- ReLu
- tanh
- no activation (linear)

The results of this experiment are shown in Figure 5.3.

The models with a linear bottleneck performed better than the others, with lower variance and a higher average AUC.

From now on we will only use models with a linear bottleneck layer.

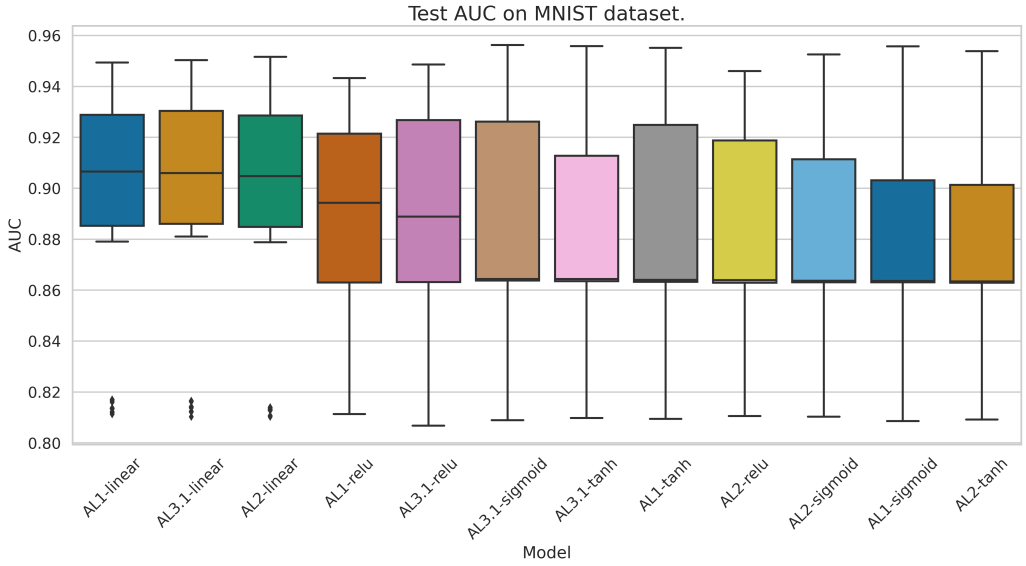

Figure 5.3. Test AUC for the bottleneck activation function experiment on the MNIST dataset.

5.4.3 Data split

Once the bottleneck size and the bottleneck activation are found, the next step is finding the best partition sizes for the task.

The experiment is conducted by performing 10 runs of models with different bottleneck data splits. For simplicity, we limited the experiment only to the approaches $AL1$, $AL2$, and $AL3$.

We try the following splits:

1. 100 ($|A|= 50$, $|B|= 50$, $|C|= 100$)
2. 200 ($|A|= 100$, $|B|= 100$, $|C|= 200$)
3. 400 ($|A|= 200$, $|B|= 200$, $|C|= 400$)
4. 1000 ($|A|= 1000$, $|B|= 1000$, $|C|= 2000$)

5. 2000 ($|A|$= 2000, $|B|$= 2000, $|C|$= 4000)

The results of this experiment are shown in Figure 5.4. Based on the obtained results, we choose to use a data splitting of 1000 images for the partitions A and B and 2000 images for C.

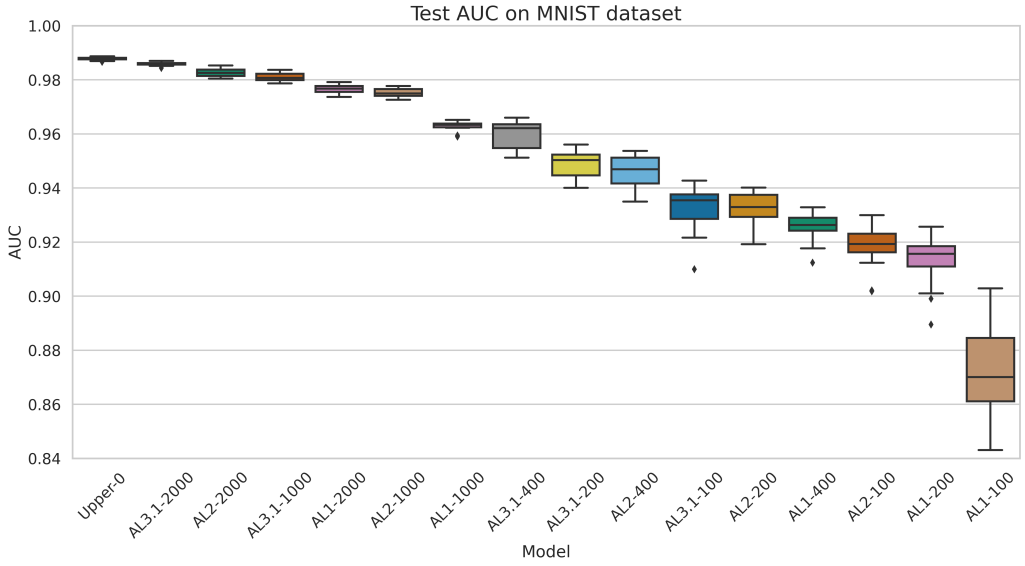

Figure 5.4. Test AUC for the partition size experiment on the MNIST dataset.

5.5 Experiments on the Hazards&Robots dataset, Corridors scenario

Similar experiments are conducted on the real-life Hazards&Robots dataset, Corridors scenario. This dataset is larger than MNIST, thus experiments require more time. Because of this, experiments in this section are not as broad as on the *proxy* task.

For the following experiments we perform *data augmentation* along with resampling.

We perform the following augmentation operations:

- Horizontal flip with a probability of 0.5;
- Random brightness contrast with a probability of 0.5, brightness and contrast limits of 0.1;

- Random crop with a probability of 0.5;
- Random rotation of maximum 10° with 0.5 probability.

5.5.1 Early phase

In an early phase, we adapted the autoencoder to work with the images from the Hazards&Robots, we then run some preliminary tests to check the correct function of the model (Figure 5.5).

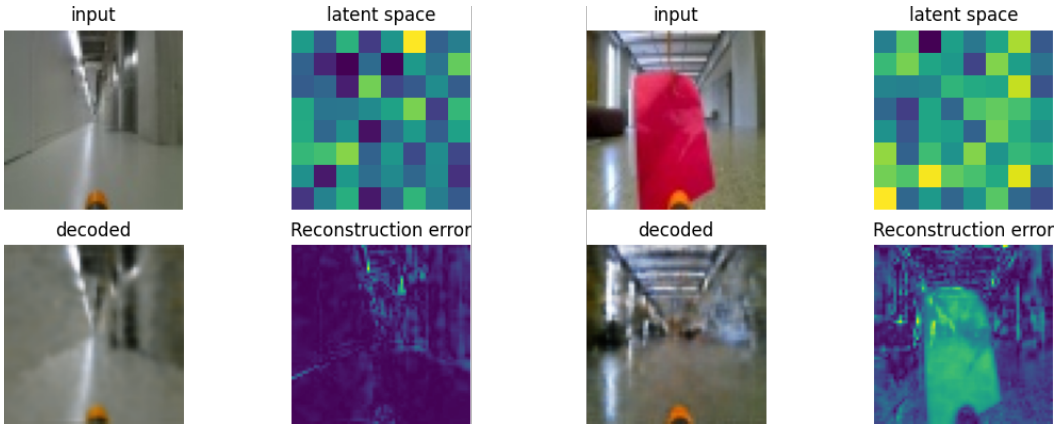

Figure 5.5. Example of predictions from a model trained on the Hazards&Robots dataset. The input image on the left is from the normal class, while the other on the right is anomalous

5.5.2 Bottleneck size

We choose to use a bottleneck size of 16 because Mantegazza et al [29] in their experiments on this dataset show it performs better.

5.5.3 Data split

The next step is finding the best partition size for this task.

Because of time constraints, since experiments on this dataset require more time than MNIST, we only try the following splits to $AL1$ only:

- 2 ($|A|= 1$, $|B|= 1$, $|C|= 2$)
- 20 ($|A|= 10$, $|B|= 10$, $|C|= 20$)

- 200 ($|A|$= 100, $|B|$= 100, $|C|$= 200)
- 2000 ($|A|$= 1000, $|B|$= 1000, $|C|$= 2000)

We choose to add to the comparison two more experiments: $AL1-2$, and $AL1-2000$ with no image augmentation.

The results in Figure 5.6 show how effective the data augmentation is when the training set is small, with $AL1-2$ outperforming its version with no augmentation. On the other hand, for the models with an initial training set of 2000 images, the augmentation led to a very small decrease in the model performance. We suppose this small decrease is due to noise introduced by the process of augmentation.

We choose to use a data splitting of 1000 images for the splits A and B partitions and 2000 images for partition C.

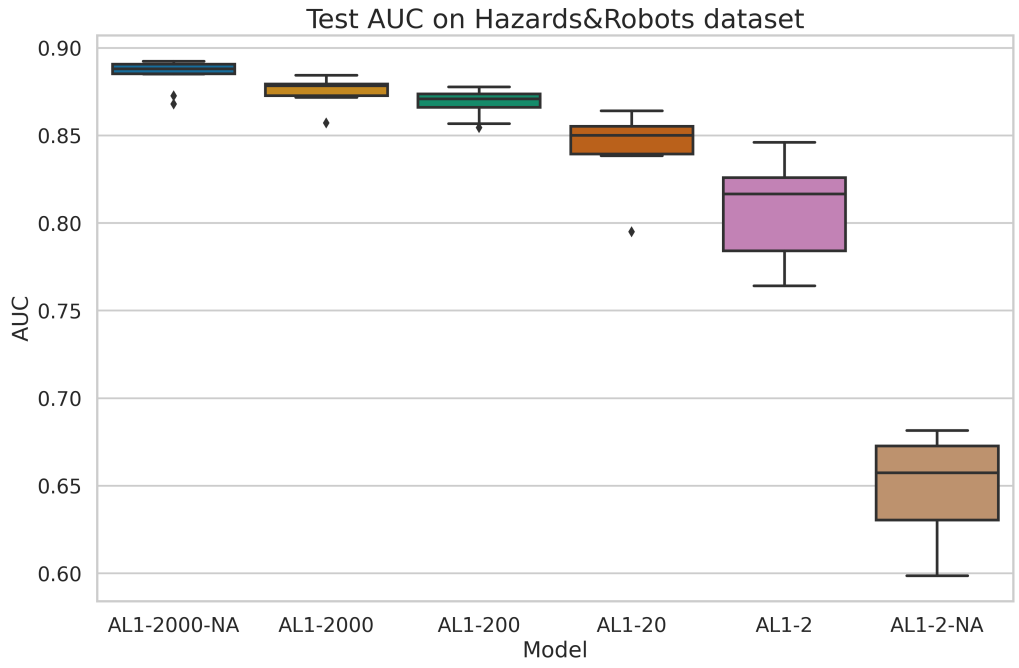

Figure 5.6. Test AUC on the Hazards&Robots dataset for different data split.

5.6 AL metrics comparison

Now we have the best bottleneck and data partition hyperparameters for both our tasks. The final experiments are conducted to test and compare the different

AL approaches proposed in section 2.3.

5.6.1 MNIST

The results of the final experiments on the proxy task are shown in Figure 5.7. From the chart, we notice that no *AL* approach has outperformed the others in a significant way. We notice that even with the random approach there is a significant improvement over the baseline model (*AL*1). However, for this task, the AL approaches proposed do not provide any significant improvement over the random query method (*AL*2).

We suppose this is because the MNIST dataset is too simple for this task, the model quickly learns which features to extract to correctly reconstruct the input. Thus finding clever solutions to query useful data for the model shows little to no improvements in this setup.

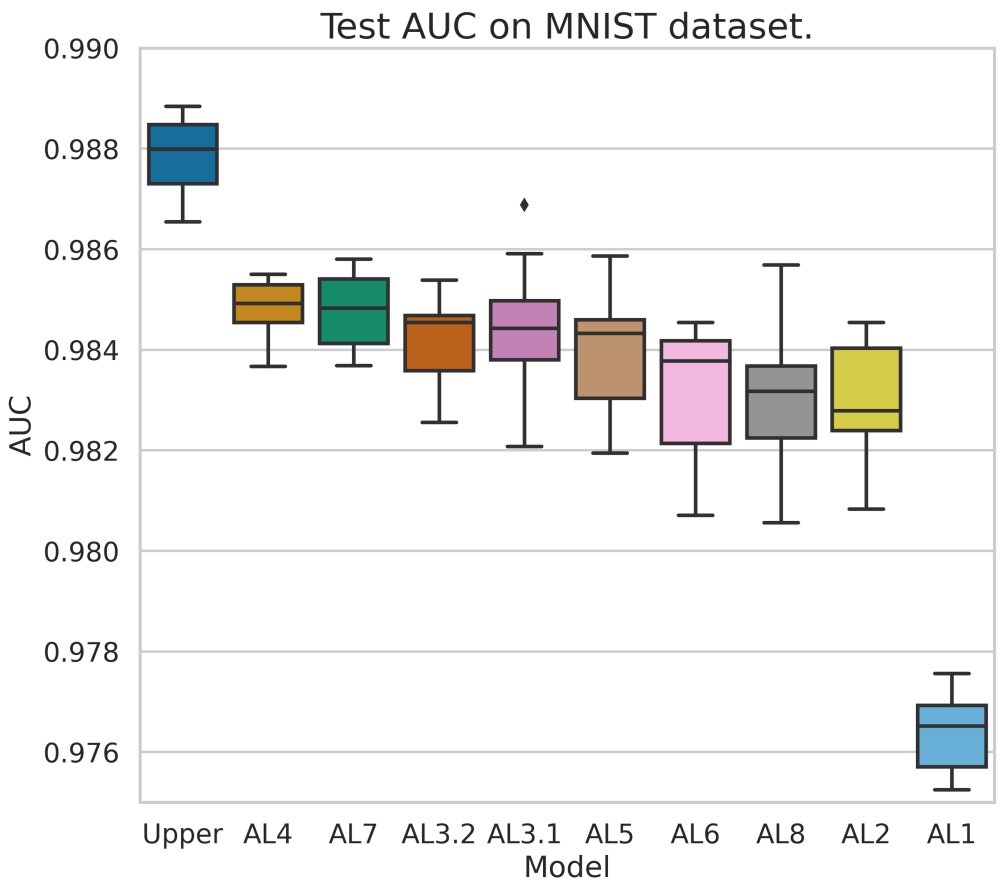

Figure 5.7. Test AUC on final experiments on the MNIST dataset

5.6.2 Hazards&Robots dataset, Corridors scenario

The results of the final experiments on the Hazards&Robots dataset, Corridors scenario are shown in Figure 5.7.

From this real-life dataset, the results show a different pattern. We still have significant improvements with any model compared to the baseline. However, in this case, we have some models outperforming the random. More in detail, the approach $AL8$ outperformed both the baselines ($AL1$, $AL2$), scoring an AUC close to the upper bound of the dataset while using a training set of only 4,000 images.

Another trend we notice is that methods that work on the AE's latent space show better performance than the ones which work in the image space.

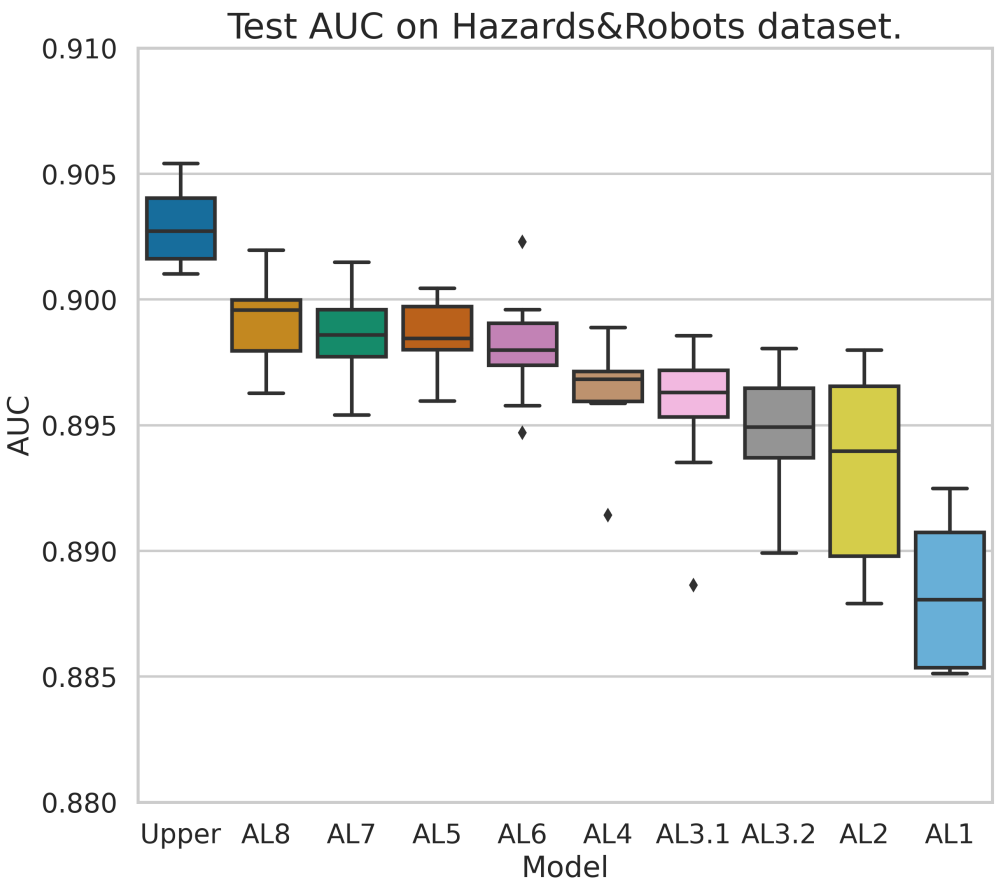

Figure 5.8. Test AUC on final experiments on the Hazards&Robots dataset, Corridors scenario.

5.6.3 Comparison

We now directly compare the final results for both tasks. Figure 5.9 shows an overview of the AUC for both the MNIST dataset and the Hazards&Robots dataset, Corridors scenario.

We immediately notice that the results for MNIST are more "compact" when compared to our real-life dataset. This encourages our hypothesis that the MNIST dataset is way too simple for the task, nullifying the effort of performing AL.

On the other hand, the chart of results on the Hazards&Robots dataset, Corridors scenario shows how AL can be effective when applied to more complex tasks, leading to an AUC close to the upper bound using only a small fraction

of the training data.

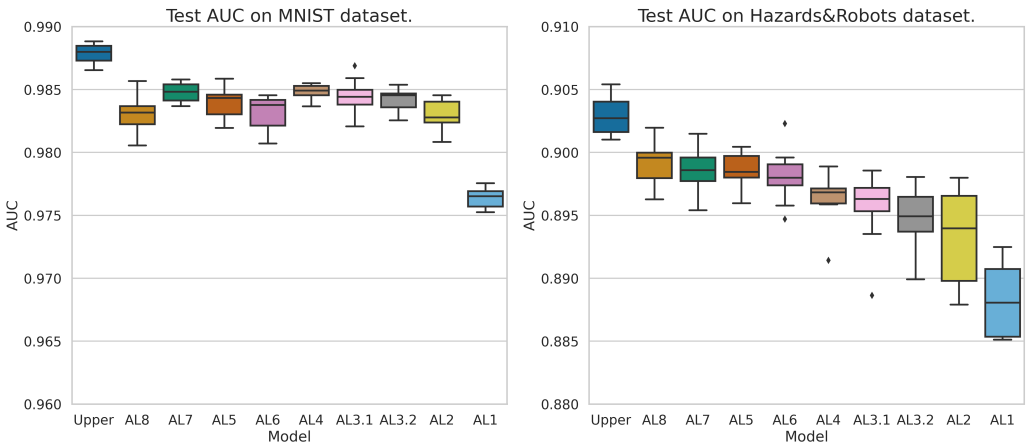

Figure 5.9. Test AUC for final experiments. Comparison between MNIST and the Hazards&Robots dataset, Corridors scenario

5.6.4 Discussion

In this chapter we found the best bottleneck size, split size, and data split for both our tasks. We then tested the proposed AL approaches to both our datasets. We found out that for MNIST the best bottleneck size is 64, while 16 resulted the best for the Hazards&Robots dataset.
A linear bottleneck for the autoencoder leads to better performance then using an activation function.
For both the datasets, the best split is 1000 images for partitions A and B, 2000 for partition C.
The approaches and the testing pipeline are validated using and testing some dummy models.
As for the AL approaches, MNIST turned out to be way too simple for the task, with AL not providing any significant performance boost. On the other hand, on the Hazards&Robots dataset the AL approaches which work in the latent space of the AE outperformed the baselines and performed better than the AL methods which work in the image space.
To summarize, AL applied the Hazards&Robots dataset let us reach performance near to the upper bound by using only 4,000 samples.

Chapter 6

Conclusions

The objective of this work is to study Active Learning (AL) techniques applied to Anomaly Detection (AD) to reduce the training data while retaining similar performance on the test set.

To achieve this we first build an AD model inspired by the work done at Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA). We then research the literature to find the metrics used for AL and explain them. After the literature research, we define our problem and design our solutions. We extend a real-life dataset for AD with new samples collected using a ground robot with a front-facing camera. Then we perform our experiments on both a *proxy* task defined on the MNIST dataset and the previously extended real-life dataset.

Our results show that on MNIST no AL approach has outperformed the others in a significant way. We assume this happened because MNIST is too simple for this task. On the other hand, for the dataset that we have collected, the most performing approach among all tested is the hybrid min-max with anomaly score. It reaches an Area Under the Curve (AUC) close to the upper bound model with only a small fraction of the training set. In the real-life dataset, we notice that the AL approaches that work on the latent space of the Autoencoder (AE) outperform the approaches that work in the image space.

6.1 Future Works

The results of this work show that the proposed AL approach is viable for AD applications in the context of mobile robots. To further improve the work we suggest the following:

- **More Complex Models**: In this work, we have only used a simple Autoencoder (AE). It is possible to use more complex models such as Variational Autoencoder (VAE) or Real-NVP based models to see if they can achieve better results.
- **More Challenging Dataset**: The Corridor scenario of the dataset we used turned out not to be very challenging. Testing the AL approaches in more challenging scenarios would be interesting to test the approaches on a more difficult real-life setting.
- **Transfer Learning techniques**: In this work, we have used the real-life dataset as a single Domain. However real-life scenarios are often diverse. It would be interesting to test our AL solution while keeping in consideration Domain Adaptation.

Appendix A

Code

A.1 Autoencoder classes

A.1.1 Autoencoder

```
import torch
import torch.nn as nn
import sklearn.metrics as metrics
from tqdm.auto import tqdm
from utils import compute_auc_score

import numpy as np
from torchinfo import summary

from .bottleneck import Bottleneck
from .convolutional_decoder import ConvolutionalDecoder
from .convolutional_encoder import ConvolutionalEncoder
from .decoder import Decoder
from .encoder import Encoder


class Autoencoder(nn.Module):

    def __init__(self, input_size, hidden_size, output_size, batch_size=128,
                 convolutional=True, dropout_rate=0.5, config=None, device=None,
                 bottleneck_activation=None):
        super(Autoencoder, self).__init__()

        self.device = torch.device("cuda" if torch.cuda.is_available()
                                   else "cpu") if device is None else device
        self.hidden_size = hidden_size
        self.in_channels = input_size[0]
        self.input_size = input_size[1:]
        self.batch_size = batch_size
        self.last_conv_size = (4, 4) if self.input_size == (28, 28) else (
            13, 13)

        if config is not None:
            input_size = config["input_size"]
            hidden_size = config["hidden_size"]
```

```
            output_size = config["output_size"]
            convolutional = config["convolutional"]
            dropout_rate = config["dropout_rate"]
        self.convolutional = convolutional
        if convolutional:
            self.encoder = ConvolutionalEncoder(dropout_rate=dropout_rate,
                                                in_channels=self.in_channels)
            if self.hidden_size > 0:
                self.bottleneck = Bottleneck((self.batch_size,
                                              self.last_conv_size[0],
                                              self.last_conv_size[1]),
                                             hidden_size,
                                             bottleneck_activation)
            self.decoder = ConvolutionalDecoder(
                self.input_size,
                dropout_rate=dropout_rate,
                out_channels=self.in_channels,
                last_conv_size=self.last_conv_size)
        else:
            self.encoder = Encoder(input_size, hidden_size, output_size,
                                   device=self.device)
            self.decoder = Decoder(input_size, hidden_size, output_size,
                                   device=self.device)

        self.threshold = 0
        self.config = {
            "input_size": input_size,
            "hidden_size": hidden_size,
            "output_size": output_size,
            "convolutional": convolutional,
            "dropout_rate": dropout_rate
        }

    def summary(self, input_size):
        return summary(self, input_size=input_size, device=self.device)

    def forward(self, x):
        encoder_out = self.encoder(x)
        if self.convolutional:
            if self.hidden_size > 0:
                bottleneck_out, encoded = self.bottleneck(encoder_out)
            else:
                bottleneck_out = torch.ones_like(
                    encoder_out.view(-1,
                                     int(np.prod((
                                         self.batch_size,
                                         4,
                                         4))))).to(self.device)
                encoded = torch.zeros(self.batch_size, 1).to(self.device)
            decoded = self.decoder(bottleneck_out)
        else:
            encoded = encoder_out
            decoded = self.decoder(encoder_out)
        return encoded, decoded

    def set_threshold(self, threshold_data_loader):
        self.eval()
        with torch.no_grad():
            for x, _ in tqdm(threshold_data_loader):
                x = x.to(self.device)
                encoded, y, _ = self(x)
```

```
            y_test = (x > 0.5).reshape(-1).cpu().detach().numpy().astype(
                np.uint8).tolist()
            y_score = y.reshape(-1).cpu().detach().numpy().astype(
                float).tolist()

            _, _, thresholds = metrics.roc_curve(y_test, y_score)
            self.threshold = thresholds[1]
```

A.1.2 Convolutional encoder

```
import torch
from torch import nn
import torch.nn.functional as F


class ConvolutionalEncoder(nn.Module):
    def __init__(self, dropout_rate=0.5, in_channels=1):
        super(ConvolutionalEncoder, self).__init__()
        self.dropout = nn.Dropout(dropout_rate)
        self.conv0 = nn.Conv2d(in_channels, 16, 3)
        self.conv1 = nn.Conv2d(16, 32, 3)
        self.conv2 = nn.Conv2d(32, 64, 3, stride=2)
        self.conv3 = nn.Conv2d(64, 128, 6, stride=2, padding=1)

    def forward(self, x):
        x = F.leaky_relu(self.conv0(x))
        x = self.dropout(F.leaky_relu(self.conv1(x)))
        x = self.dropout(F.leaky_relu(self.conv2(x)))
        x = F.leaky_relu(self.conv3(x))
        x = x.view(x.size(0), -1)
        return x
```

A.1.3 Bottleneck

```
import numpy as np
import torch
from torch import nn
import torch.nn.functional as F


class Bottleneck(nn.Module):
    def __init__(self, input_size, bottleneck_size, bottleneck_activation):
        super(Bottleneck, self).__init__()
        self.input = int(np.prod(input_size))
        self.output = input_size
        self.bottleneck_size = bottleneck_size
        self.bottleneck_activation = bottleneck_activation
        self.fc1 = nn.Linear(self.input, bottleneck_size)
        self.fc2 = nn.Linear(bottleneck_size, self.input)

    def forward(self, x):
        x = x.view(-1, self.input)
        if self.bottleneck_activation is not None:
            encoded = self.bottleneck_activation(self.fc1(x))
        else:
            encoded = self.fc1(x)
        x = self.fc2(encoded)
        x = x.view(-1, self.output[0], self.output[1], self.output[2])
        return x, encoded
```

A.2 Dataset classes

A.2.1 Dataset Handler

```
import copy
import json
from fractions import Fraction

import numpy as np
import torch
import albumentations as A
from torch.utils.data import Dataset
from torchvision import datasets, transforms

from dataset.dataset import AnomalyDetectionDataset, \
    CustomAnomalyDetectionDataset
from utils import get_most_anomalous, get_most_informative


class DatasetHandler:
    def __init__(self, dataset, transformations, anomalous_class=(3,),
                 splits=(500, 500, 1000, 1000), shuffle=True,
                 torch_builtin=True,
                 use_all_classes_except_anomalous_as_normal=False,
                 split_size=50000, augment=False, noise_flag=False,
                 noise_path=None):
        """
        > The function takes in a dataset, a list of transformations, a tuple
        of splits, and a boolean value for shuffle. It then creates a
        train_set and a test_set, and splits the train_set into a train_set
        and a valid_set. It then creates a split_a, split_b, and split_c,
        and creates a dictionary of the splits

        :param dataset: the name of the dataset to use. This is the name of
        the dataset class in torchvision.datasets
        :param transformations: a
        list of transformations to apply to the data
        :param anomalous_class:
        the class of the anomalous data. In this case, it's the digit 3
        :param splits: tuple of 4 elements, first three elements define the
        dimension of A, B, and C splits, while the
        :param shuffle: Whether to
        shuffle the dataset or not, defaults to True (optional)
        :param torch_builtin: If True, the dataset is assumed to be a torch
        builtin  dataset. If False, the dataset is assumed to be a custom
        dataset, defaults to True (optional)
        :param  use_all_classes_except_anomalous_as_normal: This is a boolean
        parameter. If set to True, it will use all the classes except the
        anomalous class as the normal class. If set to False, it will use
        only the anomalous class as the normal class, defaults to False (
        optional)
        :param split_size: The number of images to be used in the split,
        defaults to 50000 (optional)
        """
        self.idx = None
        self.split_size = split_size
        self.shuffle = shuffle
        for i, split in enumerate(splits):
            if isinstance(split, Fraction):
                splits[i] = float(splits[i])
```

```
if sum(splits) == 0:
    self.splits = None
else:
    self.splits = splits

if torch_builtin:
    train_set = datasets.__dict__[dataset](
        './data',
        train=True,
        download=True,
        transform=transforms.ToTensor())
    self.input_size = train_set.data[0].shape

    self.test_set = datasets.__dict__[dataset](
        './data',
        train=False,
        download=True,
        transform=transforms.ToTensor())

    train_set, valid_set = torch.utils.data.random_split(train_set, [
        int(len(train_set) * 0.95),
        int(len(train_set) * 0.05)])

    # Filtering the dataset to only contain the anomalous class.
    if use_all_classes_except_anomalous_as_normal:
        train_set = torch.utils.data.Subset(
            train_set,
            [i for i in range(len(train_set)) if
             train_set[i][1] not in anomalous_class])

        self.valid_set = AnomalyDetectionDataset(
            torch.utils.data.Subset(
                valid_set,
                [i for i in range(len(valid_set)) if
                 valid_set[i][1] not in anomalous_class]))
    else:
        train_set = torch.utils.data.Subset(
            train_set,
            [i for i in range(len(train_set)) if
             train_set[i][1] in anomalous_class])

        self.valid_set = AnomalyDetectionDataset(
            torch.utils.data.Subset(
                valid_set,
                [i for i in range(len(valid_set)) if
                 valid_set[i][1] in anomalous_class]))

else:
    assert len(
        dataset) == 3, 'dataset must be a tuple of' \
                       '(train_set, test_set, valid_set) paths'
    train_set_path, test_set_path, valid_set_path = dataset

    if augment:
        image_size = (64, 64)

        composed_transform = A.Compose(
            [
                A.transforms.HorizontalFlip(p=0.5),
```

```
                A.transforms.RandomBrightnessContrast(
                    brightness_limit=0.1, contrast_limit=0.1, p=0.5),
                A.RandomSizedCrop(min_max_height=[50, image_size[0]],
                                  height=image_size[0],
                                  width=image_size[1], p=0.5),
                A.Rotate(limit=10, p=0.5),
            ]
        )
    else:
        composed_transform = transformations

    train_set = CustomAnomalyDetectionDataset(
        train_set_path, (64, 64),
        aug_flag=augment,
        noise_flag=noise_flag,
        transform=composed_transform,
        noise_path=noise_path)
    self.split_size = len(train_set)
    self.valid_set = CustomAnomalyDetectionDataset(
        valid_set_path,
        (64, 64),
        transform=transformations)
    self.test_set = CustomAnomalyDetectionDataset(
        test_set_path,
        (64, 64),
        transform=transformations)

# Splitting the dataset into 3 parts, A, B, and C.
self.data = train_set
if self.splits is not None:
    if self.splits[0] < 1:
        self.splits[0] = int(len(train_set) * self.splits[0])
    if self.splits[1] < 1:
        self.splits[1] = int(len(train_set) * self.splits[1])
    if self.splits[2] < 1:
        self.splits[2] = int(len(train_set) * self.splits[2])

    split_a = self.splits[0]
    split_b = self.splits[1]

    self.splits[2] = self.splits[2] if len(self.splits) > 0 else len(
        train_set) - (split_b + split_a)

    self.train_part_a, self.train_part_b, self.train_part_c = \
        torch.utils.data.random_split(
            train_set,
            [split_a,
             split_b,
             len(train_set) -
             (split_b +
              split_a)])

    random_sample = np.random.choice(np.arange(len(self.train_part_c)),
                                     self.splits[2])
    random_sample = torch.utils.data.Subset(self.train_part_c,
                                            random_sample)

    # save on json file the number of images per splits
    with open('./data/split_size.json', 'w') as fp:
        json.dump({'split_a': len(self.train_part_a),
```

```
                    'split_b': len(self.train_part_b),
                    'total_split_c': len(self.train_part_c)}, fp)

        augment_a = self.augment(torch.utils.data.ConcatDataset(
            [self.train_part_a, self.train_part_b]))
        augment_b = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            random_sample]))

        self.split_a = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            augment_a]),
        )
        self.split_b = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            random_sample,
                                            augment_b]))
        self.split_c = None
        self.split_d = None
        self.split_e = None
        self.split_f = None
        self.split_g = None
        self.split_h = None
        self.split_i = None

        self.split_dict = {
            'AL1': self.split_a,
            'AL2': self.split_b,
            'AL3': self.split_c,
            'AL3.1': None,
            'AL3.2': self.split_d,
            'AL4': self.split_e,
            'AL5': self.split_f,
            'AL6': self.split_g,
            'AL7': self.split_h,
            'AL8': self.split_i,
            'DummyTrue': self.split_a,
            'DummyFalse': self.split_a,
            'DummyRandom': self.split_a
        }
    else:
        self.split_a = self.augment(train_set)
        self.split_b = None
        self.split_c = None
        self.split_d = None
        self.split_d_1 = None
        self.split_e = None
        self.split_f = None
        self.split_g = None
        self.split_h = None
        self.split_i = None
        self.split_dict = {
            'Upper': self.split_a,
            'AL1': self.split_a,
            'AL2': self.split_b,
            'AL3': self.split_c,
            'AL3.1': None,
            'AL3.2': self.split_d,
```

```
            'AL4': self.split_e,
            'AL5': self.split_f,
            'AL6': self.split_g,
            'AL7': self.split_h,
            'AL8': self.split_i,
            'DummyTrue': self.split_a,
            'DummyFalse': self.split_a,
            'DummyRandom': self.split_a
        }

    def augment(self, data):
        """
        This function takes a dataset and returns a new dataset with the same
        number of samples as the original dataset,
        but with the samples randomly selected from the original dataset

        :param data: the dataset to be augmented
        :return: a subset of the data.
        """
        if self.split_size - len(data) <= 0:
            return data

        idx = np.random.choice(np.arange(len(data)),
                               self.split_size - len(data))
        return torch.utils.data.Subset(data, idx)

    def gen_splits(self):
        """
        **gen_splits** should return a list of tuples of the form (train_set,
        test_set) where each tuple is a partition of
        the data
        """
        raise NotImplementedError("gen_splits is not yet implemented")

    def gen_split_c(self, model):
        """
        > We take the most anomalous data points from the train set,
        and augment them. Then we add them to the training set

        :param model: the model to use for anomaly detection
        """
        most_anomalous, _ = get_most_anomalous(model, self.train_part_c,
                                               self.splits[2])

        augment_c = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_anomalous]))

        self.split_c = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_anomalous,
                                            augment_c]))
        self.split_dict['AL3'] = self.split_c

    def gen_split_d(self, model):
        if self.split_d is None:
            most_anomalous, self.idx = get_most_anomalous(model,
                                                          self.train_part_c,
                                                          self.splits[3] // 2)
```

```
            self.split_d_1 = copy.deepcopy(most_anomalous)

            augment_d = self.augment(
                torch.utils.data.ConcatDataset([self.train_part_a,
                                                self.train_part_b,
                                                most_anomalous]))

            self.split_d = AnomalyDetectionDataset(
                torch.utils.data.ConcatDataset([self.train_part_a,
                                                self.train_part_b,
                                                most_anomalous,
                                                augment_d]))
            self.split_dict['AL3.1'] = copy.deepcopy(self.split_d)
        else:
            new_idx = [i for i in range(len(self.train_part_c)) if
                       i not in self.idx]
            part_c = torch.utils.data.Subset(self.train_part_c, new_idx)
            most_anomalous, _ = get_most_anomalous(model, part_c,
                                                   self.splits[3] // 2)
            augment_d = self.augment(
                torch.utils.data.ConcatDataset([self.train_part_a,
                                                self.train_part_b,
                                                self.split_d_1,
                                                most_anomalous]))
            self.split_d = AnomalyDetectionDataset(
                torch.utils.data.ConcatDataset([self.train_part_a,
                                                self.train_part_b,
                                                self.split_d_1,
                                                most_anomalous,
                                                augment_d]))
            self.split_dict['AL3.2'] = self.split_d

    def gen_split_e(self, model):
        most_informative = get_most_informative(model, self.train_part_c,
                                                torch.utils.data.ConcatDataset(
                                                    [self.train_part_a,
                                                     self.train_part_b]),
                                                self.splits[3],
                                                'latent_distance')

        augment_e = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative]))

        self.split_e = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative,
                                            augment_e]))
        self.split_dict['AL4'] = self.split_e

    def gen_split_f(self, model):
        most_informative = get_most_informative(model, self.train_part_c,
                                                self.split_a,
                                                self.splits[3] // 2,
                                                'latent_distance')
        most_informative2 = get_most_informative(model, self.train_part_c,
                                                 self.split_a,
```

```
                                                self.splits[3] // 2 +
                                                self.splits[3] % 2,
                                                'error_map')

        most_informative = torch.utils.data.ConcatDataset(
            [most_informative, most_informative2])

        augment_f = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative]))

        self.split_f = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative,
                                            augment_f]))
        self.split_dict['AL5'] = self.split_f

    def gen_split_g(self, model):
        most_informative = get_most_informative(model, self.train_part_c,
                                                torch.utils.data.ConcatDataset(
                                                    [self.train_part_a,
                                                     self.train_part_b]),
                                                self.splits[3], 'minmax')

        augment_g = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative]))

        self.split_g = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative,
                                            augment_g]))
        self.split_dict['AL6'] = self.split_g

    def gen_split_h(self, model):
        most_informative = get_most_informative(model, self.train_part_c,
                                                torch.utils.data.ConcatDataset(
                                                    [self.train_part_a,
                                                     self.train_part_b]),
                                                self.splits[3],
                                                'minmax_iterative')

        augment_h = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative]))

        self.split_h = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative,
                                            augment_h]))
        self.split_dict['AL7'] = self.split_h

    def gen_split_h2(self, model):
```

```
        most_informative = get_most_informative(model, self.train_part_c,
                                                torch.utils.data.ConcatDataset(
                                                    [self.train_part_a,
                                                     self.train_part_b]),
                                                self.splits[3],
                                                'minmax_iterative2')

        augment_h = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative]))

        self.split_h = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative,
                                            augment_h]))
        self.split_dict['AL7'] = self.split_h

    def gen_split_i(self, model):
        most_informative = get_most_informative(model, self.train_part_c,
                                                torch.utils.data.ConcatDataset(
                                                    [self.train_part_a,
                                                     self.train_part_b]),
                                                self.splits[3],
                                                'minmax_anomaly')

        augment_i = self.augment(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative]))

        self.split_i = AnomalyDetectionDataset(
            torch.utils.data.ConcatDataset([self.train_part_a,
                                            self.train_part_b,
                                            most_informative,
                                            augment_i]))
        self.split_dict['AL8'] = self.split_i
```

A.2.2 Dataset

```
import glob
import pickle
from typing import Iterator, Tuple

import numpy as np
import pandas as pd
import torch
import torchvision
from torch.utils.data.dataset import T_co, Dataset

import albumentations as A


def torch_standardize_image(image: torch.Tensor, epsilon: float = 0.0000001):
    channels, _, _ = image.shape
    im_mean = image.view(channels, -1).mean(1).view(channels, 1, 1)
    im_std = image.view(channels, -1).std(1).view(channels, 1, 1)
    return (image - im_mean) / (im_std + epsilon)
```

```
class AnomalyDetectionDataset(Dataset):

    def __getitem__(self, index) -> T_co:
        return self.data[index]

    def __len__(self):
        return len(self.data)

    def __iter__(self) -> Iterator[T_co]:
        return iter(self.data)

    def __init__(self, data):
        self.data = data


class CustomAnomalyDetectionDataset(Dataset):
    def __init__(self,
                 root_dir: str,
                 image_shape: Tuple[int, int],
                 aug_flag=False,
                 noise_flag=False,
                 transform=None,
                 noise_path='./noise/',
                 noise_alpha: int = 60,
                 noise_p=0.5,
                 ):
        list_files = sorted(glob.glob(root_dir + "/*"))
        assert len(list_files) != 0, "Error in loading frames"
        self.frames = list_files
        self.aug_flag = aug_flag
        self.transform = transform
        self.noise_path = noise_path
        self.image_shape = image_shape
        self.noise_flag = noise_flag
        self.noise_alpha = noise_alpha
        self.noise_p = noise_p
        self.labels = pd.read_csv(f"{root_dir}/../metadata/frames_labels.csv")

        # get label from frame_id using labels dataframe
        self.labels = self.labels.set_index("frame_id")
        # print(self.labels)
        if noise_flag:
            self.available_noises = len(glob.glob(noise_path + "/*"))
        else:
            self.available_noises = 0

    def __len__(self):
        return len(self.frames)

    def __getitem__(self, idx):
        pt_image = torchvision.io.read_image(self.frames[idx]).numpy()
        image = np.transpose(pt_image, (1, 2, 0))
        # we apply augmentations and apply standardization.
        if self.aug_flag and self.transform is not None:
            aug_image = self.transform(image=image)["image"]
            if self.noise_flag and np.random.rand() <= self.noise_p:
                int_image = self.add_noise_to_image(aug_image)
            else:
                int_image = aug_image
```

```
        else:
            int_image = image
        torch_float = torch.from_numpy(np.transpose(int_image,
                                                    (2, 0, 1))).float()
        # final_image = torch_float.div(255.0)
        final_image = torch_standardize_image(torch_float).contiguous()
        label = float(bool(self.labels.loc[idx]['label']))
        return final_image, label

    def add_noise_to_image(self, aug_image):
        noise_id = np.random.randint(self.available_noises)
        with open(self.noise_path + f"/noise_{noise_id}.pk", "rb") as pk_file:
            noise = pickle.load(pk_file)
        x_crop = np.random.randint(1000 - aug_image.shape[1])
        y_crop = np.random.randint(1000 - aug_image.shape[0])
        crop_noise = A.Crop(
            x_min=x_crop,
            y_min=y_crop,
            x_max=x_crop + aug_image.shape[1],
            y_max=y_crop + aug_image.shape[0],
            always_apply=True, p=1.0,
        )(image=noise)["image"]
        noise_rand_alpha = np.random.randint(
            low=self.noise_alpha, high=100) / 100.0
        final_image = aug_image / 255 + (crop_noise * (1.0 - noise_rand_alpha))
        return final_image
```

A.3 Utility Scripts

A.3.1 Function get_most_anomalous

```
def get_most_anomalous(model, data, n=100, criterion=None):
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model.eval()
    loss_fn = torch.nn.L1Loss()
    losses = []
    dataloader = torch.utils.data.DataLoader(data, batch_size=1, shuffle=True,
                                             pin_memory=True, num_workers=4)
    with torch.no_grad():
        for frame, _ in tqdm(dataloader):
            frame = frame.to(device)
            enc, dec = model(frame)
            loss = loss_fn(frame, dec)
            losses.append(loss.item())

    idx = np.argsort(losses)[::-1][:n]
    return torch.utils.data.Subset(data, idx), idx
```

A.3.2 Function get_most_informative

```
def get_most_informative(model, data, train_data, n=100, criterion="error_map"):
    '''
    Return the n most informative frames,
    informativeness score based on model uncertainty
    '''
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
dataloader = torch.utils.data.DataLoader(data, batch_size=64, shuffle=True,
                                         pin_memory=True, num_workers=4)
train_loafer = torch.utils.data.DataLoader(train_data, batch_size=64,
                                           shuffle=True, pin_memory=True,
                                           num_workers=4)
model.eval()

latent_spaces = []
train_latent_spaces = []

with torch.no_grad():
    for frame, _ in tqdm(dataloader):
        frame = frame.to(device)
        enc, dec = model(frame)
        for l_space in enc:
            latent_spaces.append(l_space.detach().cpu().numpy())
    for frame, _ in tqdm(train_loafer):
        frame = frame.to(device)
        enc, dec = model(frame)
        for l_space in enc:
            train_latent_spaces.append(l_space.detach().cpu().numpy())

latent_spaces = np.array(latent_spaces)
train_latent_spaces = np.array(train_latent_spaces)

latent_spaces = (latent_spaces - train_latent_spaces.mean(
    axis=0)) / train_latent_spaces.std(axis=0)
train_latent_spaces = (train_latent_spaces - train_latent_spaces.mean(
    axis=0)) / train_latent_spaces.std(axis=0)

if criterion.lower() == "error_map":
    informativeness = []
    with torch.no_grad():
        for frame, _ in tqdm(dataloader):
            frame = frame.to(device)
            enc, dec = model(frame)
            for in_frame, decoded in zip(frame, dec):
                error_map = np.clip(
                    abs(decoded.cpu().detach().numpy() -
                        in_frame.cpu().detach().numpy()),
                    0, 1)
                anomaly_map = error_map.reshape(-1)
                informativeness.append(np.mean(anomaly_map))

    idx = np.argsort(informativeness)[::-1][:n]

elif criterion.lower() == "latent_distance":

    l_mean = np.mean(train_latent_spaces, axis=0)

    informativeness = [np.mean(np.abs(l_space - l_mean)) for l_space in
                       tqdm(latent_spaces)]
    idx = np.argsort(informativeness)[::-1][:2 * n]
    informativeness = []
    for i in tqdm(idx):
        info_score = 0
        l_space = latent_spaces[i]
        for j in idx:
            l_space_j = latent_spaces[j]
            info_score += np.linalg.norm(l_space - l_space_j).item()
```

```
            informativeness.append(info_score / len(idx))

        new_idx = np.argsort(informativeness)[::-1][:n]
        idx = idx[new_idx]

    elif criterion == "minmax":

        distances = distance_matrix(train_latent_spaces, latent_spaces)

        # most_distant_per_train = np.argmax(distances, axis=1)
        most_close_per_unlabel = np.argmin(distances, axis=0)
        closeness_per_unlabel = [np.argsort(distances[:, i]) for i in
                                 range(len(distances[0]))]
        print(np.max(most_close_per_unlabel))
        lst = [[] for _ in range(np.max(most_close_per_unlabel) + 1)]
        idx = [[] for _ in range(np.max(most_close_per_unlabel) + 1)]
        for i, p in enumerate(most_close_per_unlabel):
            lst[p].append(distances[p, i])
            idx[p].append(i)
        ids = []
        for i in range(len(lst)):
            if lst[i]:
                ids.append(np.argmax(np.array(lst[i])))
            else:
                ids.append(0)
                idx[i].append(np.argmax(distances[i, :]))
        new_idx = [idx[i][num] for i, num in zip(range(len(ids)), ids) if
                   num is not None]
        idx = new_idx[:n]

        informativeness = []

        sorted_dist = []
        sorted_idx = []

    elif criterion == "minmax_iterative":
        # distances = distance_matrix(train_latent_spaces, latent_spaces)
        dtree = cKDTree(train_latent_spaces)
        closest_point = dtree.query(latent_spaces, k=1)[1]
        distances = [minkowski_distance(p, nn) for p, nn in
                     zip(latent_spaces, closest_point)]
        original_ids = [i for i in range(len(latent_spaces))]
        idx = []

        with tqdm(total=n) as pbar:
            while len(idx) < n:
                chosen = np.argmax(distances)
                true_chosen = original_ids[chosen]
                original_ids = original_ids
                idx.append(true_chosen)
                pbar.update(1)
                l_space = latent_spaces[true_chosen]
                # latent_spaces = np.delete(latent_spaces, chosen, axis=0)
                original_ids.remove(true_chosen)
                distances.remove(distances[chosen])
                new_dists = [minkowski_distance(l_space, other) for other in
                             latent_spaces]
                distances = [min(d1, d2) for d1, d2 in
```

```
                        zip(distances, new_dists)]

    elif criterion == "minmax_iterative2":
        ds = cdist(train_latent_spaces, latent_spaces)
        ds = np.min(ds, axis=0)
        rs = []

        for _ in trange(n):
            i = np.argmax(ds)
            rs.append(i)
            nds = cdist(latent_spaces, latent_spaces[i:i + 1])[:, 0]
            ds = np.minimum(ds, nds)
        idx = rs

    elif criterion == "minmax_anomaly":
        ds = cdist(train_latent_spaces, latent_spaces)
        ds = np.min(ds, axis=0)
        idx = []
        samples = None
        rs = []

        print("Selecting samples from clusters")
        for _ in trange(3 * n):
            i = np.argmax(ds)
            rs.append(i)
            nds = cdist(latent_spaces, latent_spaces[i:i + 1])[:, 0]
            ds = np.minimum(ds, nds)
        idx = rs

        print("Selecting most anomalous samples")
        dataset = torch.utils.data.Subset(data, idx)
        data, idx = get_most_anomalous(model, dataset, n)
        return data

    else:
        raise ValueError("Criterion not supported")
    return torch.utils.data.Subset(data, idx)
```

Acronyms

AD Anomaly Detection. xvii, 1, 2, 3, 13, 27, 50, 61

AE Autoencoder. 2, 3, 14, 15, 16, 57, 59, 61, 62

AL Active Learning. v, xiii, xvii, 1, 4, 5, 9, 13, 14, 16, 17, 47, 56, 58, 59, 61, 62

AP Access Point. 23

API Application Programming Interface. 23

AUC Area Under the Curve. 20, 26, 27, 52, 57, 58, 61

BALD Bayesian Active Learning by Disagreement. 8, 9

CAE Convolutional Autoencoder. 2, 24

CEAL Cost-Effective Active Learning. 9

CNN Convolutional Neural Network. 2, 3, 8, 9, 10, 15, 24

DL Deep Learning. 1, 3, 4, 5, 13, 20, 24

EN Entropy. 8

FOV Field Of View. 23, 35, 37

FPR False Positive Rate. 26

FPS Frames Per Second. 23

GAN Generative Adversarial Network. 7, 10

IDSIA Istituto Dalle Molle di Studi sull'Intelligenza Artificiale. 13, 14, 21, 23, 24, 61

LC Least Confidence. 8

MAE Mean Absolute Error. 14, 15, 26

MC Monte Carlo. 8

ML Machine Learning. 4, 20

MSE Mean Squared Error. 3, 14, 15, 16, 24, 26

NN Nearest Neighbor. 2

ROS Robot Operating System. 23

TPR True Positive Rate. 26

USI Università della Svizzera Italiana. 28, 31

VAE Variational Autoencoder. 62

Bibliography

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

[2] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9368–9377, 2018.

[3] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

[4] Samuel Budd, Emma C Robinson, and Bernhard Kainz. A survey on active learning and human-in-the-loop deep learning for medical image analysis. *Medical Image Analysis*, 71:102062, 2021.

[5] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL `https://www.mdpi.com/2078-2489/11/2/125`.

[6] Punarjay Chakravarty, Alan Miao Zhang, Raymond Austin Jarvis, and Lindsay Kleeman. Anomaly detection and tracking for a patrolling robot. In *Proc. of the Australiasian Conference on Robotics and Automation 2007*, pages 1 – 9, 2007. ISBN 978-0-9587583-9-0.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[8] Peter Christiansen, Lars N Nielsen, Kim A Steen, Rasmus N Jørgensen, and Henrik Karstoft. Deepanomaly: Combining background subtraction and deep learning for detecting obstacles and anomalies in an agricultural field. *Sensors*, 16(11):1904, 2016.

[9] Wikimedia Commons. Autoencoder structure, 2015. URL `https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png`. File: Autoencoder_structure.png.

[10] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[11] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[12] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pages 1183–1192. PMLR, 2017.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[14] Jérôme Guzzi. Robomaster-ros. `https://github.com/jeguzzi/robomaster_ros`, 2022.

[15] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[16] Bengt Erland Ilon. Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base, April 8 1975. US Patent 3,876,255.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32, 2019.

[19] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Geometry in active learning for binary and multi-class image segmentation. *Computer vision and image understanding*, 182:1–16, 2019.

[20] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[21] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). . URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[22] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). . URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[24] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[25] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[26] Dwarikanath Mahapatra, Behzad Bozorgtabar, Jean-Philippe Thiran, and Mauricio Reyes. Efficient active learning for image classification and segmentation using a sample selection and conditional generative adversarial network. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 580–588. Springer, 2018.

[27] Dario Mantegazza, Carlos Redondo, Fran Espada, Luca Maria Gambardella, Alessandro Giusti, and Jérôme Guzzi. Sensing anomalies as potential hazards: Datasets and benchmarks. *CoRR*, abs/2110.14706, 2021. URL `https://arxiv.org/abs/2110.14706`.

[28] Dario Mantegazza, Alessandro Giusti, Luca Maria Gambardella, and Jérôme Guzzi. An outlier exposure approach to improve visual anomaly detection performance for mobile robots. *IEEE Robotics and Automation Letters*, 7(4):11354–11361, 2022.

[29] Dario Mantegazza, Carlos Redondo, Fran Espada, Luca M Gambardella, Alessandro Giusti, and Jérôme Guzzi. Sensing anomalies as potential hazards: Datasets and benchmarks. In *Annual Conference Towards Autonomous Robotic Systems*, pages 205–219. Springer, 2022.

[30] Firat Ozdemir, Zixuan Peng, Christine Tanner, Philipp Fuernstahl, and Orcun Goksel. Active learning for segmentation by optimizing content information for maximal entropy. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 183–191. Springer, 2018.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/`

9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[32] Mahdyar Ravanbakhsh, Vadim Tschernezki, Felix Last, Tassilo Klein, Kayhan Batmanghelich, Volker Tresp, and Moin Nabi. Human-machine collaboration for medical image segmentation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1040–1044. IEEE, 2020.

[33] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402. PMLR, 2018.

[34] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021.

[35] Mohammad Sabokrou, Mohammad Khalooei, Mahmood Fathy, and Ehsan Adeli. Adversarially learned one-class classifier for novelty detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[36] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11, 2014.

[37] Natasa Sarafijanovic-Djukic and Jesse Davis. Fast distance-based anomaly detection in images using an inception-like autoencoder. In Petra Kralj Novak, Tomislav Šmuc, and Sašo Džeroski, editors, *Discovery Science*, pages 493–508, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33778-0.

[38] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Georg Langs, and Ursula Schmidt-Erfurth. f-anogan: Fast unsupervised anomaly detection with generative adversarial networks. *Medical image analysis*, 54: 30–44, 2019.

[39] Luke Scime and Jack Beuth. A multi-scale convolutional neural network for autonomous anomaly detection and classification in a laser powder bed fusion additive manufacturing process. *Additive Manufacturing*, 24: 273–286, 2018.

[40] Ozan Sener and Silvio Savarese. A geometric approach to active learning for convolutional neural networks. *arXiv preprint arXiv:1708.00489*, 7, 2017.

[41] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[42] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[43] Asim Smailagic, Pedro Costa, Hae Young Noh, Devesh Walawalkar, Kartik Khandelwal, Adrian Galdran, Mostafa Mirshekari, Jonathon Fagert, Susu Xu, Pei Zhang, et al. Medal: Accurate and robust deep active learning for medical image analysis. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 481–488. IEEE, 2018.

[44] Jamshid Sourati, Ali Gholipour, Jennifer G Dy, Sila Kurugol, and Simon K Warfield. Active deep learning with fisher information for patch-wise semantic segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 83–91. Springer, 2018.

[45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[46] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312, 2016.

[47] Guido van Capelleveen, Mannes Poel, Roland M Mueller, Dallas Thornton, and Jos van Hillegersberg. Outlier detection in healthcare fraud: A case study in the medicaid dental domain. *International journal of accounting information systems*, 21:18–31, 2016.

[48] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, 2016.

[49] Lorenz Wellhausen, René Ranftl, and Marco Hutter. Safe robot navigation via multi-modal anomaly detection. *IEEE Robotics and Automation Letters*, 5(2):1326–1333, 2020.

[50] Si Wen, Tahsin M Kurc, Le Hou, Joel H Saltz, Rajarsi R Gupta, Rebecca Batiste, Tianhao Zhao, Vu Nguyen, Dimitris Samaras, and Wei Zhu. Comparison of different classifiers with active learning to support quality control in nucleus segmentation in pathology images. *AMIA Summits on Translational Science Proceedings*, 2018:227, 2018.

[51] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[52] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *Ieee access*, 6:35365–35381, 2018.

[53] Lin Yang, Yizhe Zhang, Jianxu Chen, Siyuan Zhang, and Danny Z Chen. Suggestive annotation: A deep active learning framework for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 399–407. Springer, 2017.