Universiteit
Leiden
The Netherlands

# Unveiling the Roof:
# Anomaly Detection in Train Pantographs

Levi Ari Pronk

First supervisor and second supervisor:
Lu Cao & Arno Knobbe

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

**Abstract**

Currently, anomaly detection models are tested mostly on benchmark datasets with constant backgrounds and minimal changes in lighting. This thesis presents a study comparing two anomaly detection models, *PatchCore* and *GLASS*, on real-life datasets with images of train roofs. Using computer vision models, the aim is to reduce manual labor in train maintenance by detecting defects via unsupervised methods. By using both statistical and qualitative results, the models were evaluated on their performance. *PatchCore*, known for its patch-level anomaly detection, showed promise in handling small datasets, but was generally challenged by background variation. Meanwhile, *GLASS* combined feature-level and image-level synthesis strategies, offering less sensitivity to varying backgrounds, but also to anomalies in general. This thesis primarily focuses on logical anomaly detection (detecting large objects that are out-of-place) and found that *PatchCore* detects more anomalies at a cost of more false positives, whereas *GLASS* shows narrow anomaly localization with less false positives but equally less detection of weak (less visible) anomalies.

**Keywords:** Anomaly Detection, Train Pantographs, Machine Learning, Computer Vision, PatchCore, GLASS, Unsupervised Learning.

# Contents

# 1 Introduction

The Train Maintenance department of NS (*Dutch Railways*) offers repair services to trains [nsD]. Currently, conducting controls of the state of the train is manual and happens mostly at night. To reduce manual labor, NS also uses computer vision models to make accurate predictions about possible defects on trains. This is important because manually checking every component of a train is labor-intensive and the operation is relatively expensive because technical personnel working at night receive a shift differential or premium pay. After successful implementation of computer vision models, the work would shift to only double-checking the images that were marked defect (digitization) or directly acting on computer predictions (automization). **The specific problem that this thesis will address is the detection of anomalies on train roofs, especially around *pantographs*, the linkages between the train and the overhead lines that power the train** [Wu18].

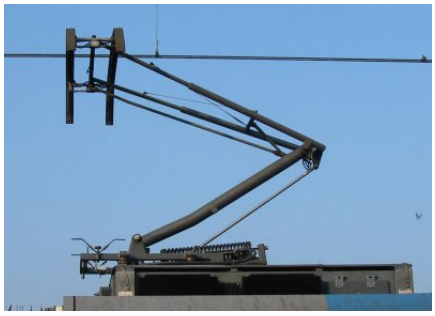


Figure 1: Example of a train pantograph

In order to detect objects on the train roof, there is a range of different computer vision models that can be used. For this research, an unsupervised *anomaly detection model* is devised. In anomaly detection, learning systems depart from known information (normal data and patterns) to infer abnormal or different patterns with respect to the normal situation, or *anomalies* [CBK09]. The normal situation is defined as a situation, in which there are no anomalies.

The reason for the choice of an anomaly detection model is the distribution of the given dataset. Of the approximately 5500 input images, 31 images are known to have an anomaly. It will therefore be easier to train a model to recognize the normal situation than to train a model on a smaller subset of irregularities. These data are not labeled, except for the 31 images known to feature an anomaly. In a supervised approach, the classes of anomalies themselves (for instance, a bag or a bird) would have to be labeled. A classifier could then accurately be trained to distinguish specific objects. The lack of labeled data and anomalous data in general has led to the choice of an unsupervised approach. Moreover, the variability in shape, color and size makes it difficult for a supervised learning model to encapsulate sufficient statistical information about abnormal image patterns [YXQS22].

Within the field of anomaly detection, there are three main steps: detection, localization and classification. Detection is about predicting whether a certain image is anomalous or not by attributing an anomaly score (2.3.2), localization finds the region that is predicted to be anomalous and classification involves classifying the anomaly found in the identified region. For the scope of this thesis, the focus will lie primarily on detecting anomalies. Localization serves as a sanity check

to see if the region identified as anomalous corresponds to the ground truth.

Current anomaly detection models are compared on benchmark datasets with a constant background and lighting scenarios. The contribution of this thesis is to replicate two state-of-the-art anomaly detection models, *PatchCore* and *GLASS*, on a real-world dataset. This dataset contains differing backgrounds, lighting and weather situations. In this thesis, the performance of both models will be compared on a new dataset. In the measurement of performance, the detection of 'strong' defects (logical anomalies) will be more important than those of 'weak' defects (structural anomalies). The reason for this choice is that structural anomaly detection usually requires domain knowledge, such as when a mechanical part should be considered 'bent' or 'broken' (see: 2.3.1).

Combining these criteria, the main research question of this thesis is:

- What is the difference in logical anomaly detection performance between *PatchCore* and *GLASS* in the context of train pantographs?

To compare the performance, several commonly used performance metrics will be devised, such as AUROC, error, accuracy, precision and recall.[1] There are other subquestions that will also play a role in this research:

- What is the difference in structural anomaly detection performance between *PatchCore* and *GLASS* in the context of train pantographs?

To further evaluate models, the detection of structural anomalies will substantiate the performance of a model.

- What potential workload reduction does the produced model entail?

Models like these could be used in future, so a lower number of misclassifications leads to a reduction in (manual) workload.

The contents of this thesis are as follows: section 2 introduces definitions specific to our use case as well as general computer vision and anomaly detection definitions; section 3 summarizes related work, such as components of an anomaly detection model, and specific information about the two chosen models, *PatchCore* and *GLASS*; section 4 describes the research methodology of this thesis; in section 5, the results and set-up of the experiments; finally, section 7 concludes this research by means of a conclusion and discussion.

[1] For more information, see: Evaluation metrics, (5.1)

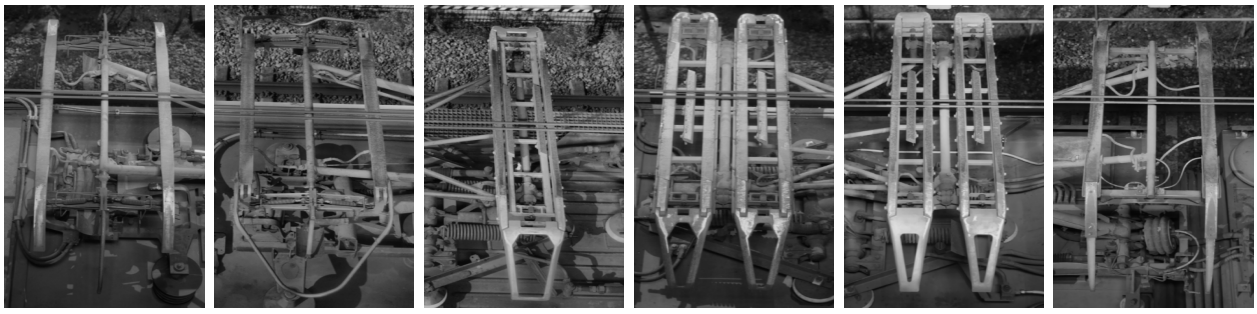
# 2 Definitions

## 2.1 Domain-specific definitions

### 2.1.1 Pantographs

A *pantograph* is a device mounted on a train to collect the current of overhead contact wires, providing the train with electricity. The pantograph consists of a pantograph head, frame, base, and drive system [Wu18]. NS trains have various types of pantograph heads with some trains sharing the same type and other train models having multiple types of pantographs.

### 2.1.2 Overview of pantograph heads



(a) (b) (c) (d) (e) (f)

Figure 2: This is an overview of the six different pantograph heads that are in the provided dataset, named 2a: `VIRM_4`, 2b: `SLT_4_6_VIRM_4`, 2c: `ICM_1_4`, 2d: `VIRM_6`, 2e: `DDZ_4_6`, and 2f: `SNG_3_4`

Data filtering (4.1.1) will happen based on these different types of pantograph heads. This thesis will approximately follow NS nomenclature for the pantograph heads, which is:

- 2a: VIRM IV (series 9547 - 9597) or `VIRM_4`
- 2b: SLT IV & VI and VIRM IV (series 9501 - 9546 & 9401 - 9481) or `SLT_4_6_VIRM_4`
- 2c: ICM I & IV or `ICM_1_4`
- 2d: VIRM VI or `VIRM_6`
- 2e: DDZ IV & VI or `DDZ_4_6`
- 2f: SNG III & IV or `SNG_3_4`

## 2.2 Computer vision definitions

### 2.2.1 Feature extraction

Similarly to human vision, computer vision models can distinguish *features* in an image, which are regions of interest bounded by edges or corners [LGRN11]. The goal of feature extraction is to extract a set of features. There are different levels of abstraction for feature extraction:

- *Low-level features*: edges, contours, angles
- *Mid-level features*: patterns, textures, shapes
- *High-level features*: local curvatures, objects

The distinction between high-level and low-level features is quite clear. Low-level features are found at a pixel-level, whereas high-level features offer us a semantic representation of a scene. Low-level features can be automatically extracted without any shape information [NAN12]. Mid-level features bridge the gap between high-level and low-level features, by building on the extracted angles [Pei15].

### 2.2.2 Feature adaptation

Feature adaptation or domain adaptation refers to adjusting or transforming extracted features to improve model performance when dealing with different domains or datasets [MKM25]. It is a special case of transfer learning 3.2.2 with the goal of exploiting information on the dissimilarity between the source and target domains. This information can be extracted from the available data to make the source distribution more similar to the target distribution [KvdMKL16].

Feature adaptation can be especially useful in images with varying angles and circumstances. By adapting the features, models can capture a wider variation of features. The reason for this is that is not known beforehand how well the source data (training data) will correspond to the target data (test data). Conditions in the target data might be different to the conditions in the source data, e.g. pictures taken on a snowy day when it has not snowed during the period in which the source data were captured. This subsequently shifts the feature distribution. Domain adaptation offers a solution by adapting a pre-trained model to dynamically changing environments with different lighting and weather conditions [WWG+19].

### 2.2.3 Image synthesis

*Image synthesis* is the process of converting the input text, sketch, or other sources, i.e., another image or mask, into an image [BLN23]. For the scope of this thesis, image synthesis will mostly involve *image-to-image translation*, which involves translating an input image into a corresponding output image [IZZE18]. Examples of image synthesis task involve translating daytime to nighttime images and transforming images with edges only to a photo. The process of image synthesis is especially useful in the use case of this thesis due to the natural variations in our dataset (lighting differences, weather conditions and differences in background).

## 2.3 Anomaly detection definitions

### 2.3.1 Classes of anomalies

Within anomaly detection, there is a distinction between different classes of anomalies. *Structural anomalies* primarily include in-object irregularities, such as scratches and cracks, whereas *logical anomalies* concern the placement, form or number of objects, such as the misplacement of labels or bent cables [mvt]. Yet another distinction between anomalies are *strong* and *weak* anomalies. Strong anomalies deviate from the norm by a higher magnitude than weak anomalies, which are

more difficult to detect. Weak defects are anomalies with small areas or low contrast [CLLZ24]. Another class of anomalies, are *synthetic anomalies*, which are generated anomalies based on normal images.

### 2.3.2 Anomaly score and threshold

The anomaly score of an image is the percentage (0-100%) that an image deviates from the normal situation [CBK09].[2] An anomaly score of 0% means that an image is normal. A higher anomaly score is an indication of an anomalous image. However, a normal image can get an anomaly score as high as 80-90%. There are several reasons for this, such as noise, intra-class variance or unusual lighting changes. This is why the introduction of an *anomaly threshold* is useful to determine at what anomaly score an image is considered to be anomalous. This score also determines the number of false positives, since it sets the threshold for what the model has correctly determined as being a normal or abnormal image.

# 3 Related Work

Anomaly detection models for computer vision have been used in medical imaging and industry contexts [ZXY+20, SDSS16]. In this section, an overview of (relevant parts of) anomaly detection models is provided. First, relevant benchmark datasets are explained (3.1). A brief overview of four different components of a typical anomaly detection model (3.2) will then be provided. This section concludes with a summary of how different anomaly detection models incorporate these components. The last two sections (3.3 and 3.4) will elaborate on the architectures of *PatchCore* and *GLASS*, respectively.

## 3.1 Benchmark datasets

In order to make comparisons across different anomaly detection models, *benchmark datasets* are used. These datasets are oftentimes acquired in a controlled environment with a constant background and no movement of the objects in the frame. The most widely-used dataset is the MVTec AD dataset (3.1.1).

### 3.1.1 MVTec AD

The MVTec dataset is a dataset for benchmarking anomaly detection methods with a focus on industrial inspection, containing over 5,000 images in different object and texture categories; [BFSS19]

### 3.1.2 VisA

The VisA (Visual Anomaly) dataset is another dataset with industry inspection objects, such as printed circuit boards, transistors and chips. The dataset contains 12 subsets with a total of more than 10,000 images of which around 10% are anomalous [ZJP+22].

[2] For more information on how PatchCore and GLASS calculate the anomaly score, see sections 3.3.5 and 3.4.6, respectively.

### 3.1.3 MPDD

First introduced in 2021, the Metal Parts Defect Detection (MPDD) dataset is a smaller dataset with 6 subsets and around 1,000 images. The dataset is particularly helpful because of its pixel-precise annotation masks, which can be helpful in the classification of anomalies. [JJB+21]

### 3.1.4 WFDD

This dataset, the Woven Fabric Defect Detection (WFDD) dataset, features 4,101 woven fabric images and was first introduced in the *GLASS* paper. The anomalies in the dataset are textural in nature and hence provide a benchmark in measuring the performance on weak defects [CLLZ24].

### 3.1.5 DTD

Lastly, the Describable Textures Dataset (DTD) contains 5,640 texture images with textures from a wide range of images collected 'in the wild' [CMK+13]. What sets this dataset apart is the fact that it was not acquired in a fully-controlled industrial environment.

## 3.2 Components of an anomaly detection model

### 3.2.1 Auto-encoders

An *auto-encoder* (also: *autoencoder*) is, in its most basic form, a neural network that is trained to reconstruct a compressed, encoded input into a decoded output that approximates the original input [CBK09, BKG23]. The decoded output is a reconstructed, decoded image in this case.

A train-related use case is a paper by Gasparini et al., which focused on obstacle detection on railways from the point of view from a moving camera, with a changing background [GPB+20]. In our use case, the camera is static, but the train is moving and the background may differ.

The paper by Gasparini et al. uses an auto-encoder structure to establish a normal situation. The model subsequently reconstructs an image from the input image, after having applied some smoothing to the input image to reduce the effect of different ballast between and on top of the tracks [GPB+20].

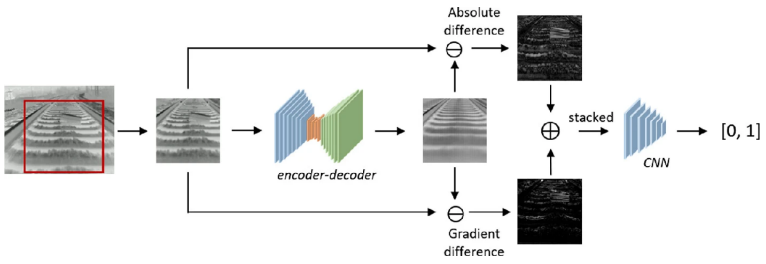



Figure 3: Auto-encoder architecture that the paper by Gasparini et al. uses. The frame is first cropped and subsequently fed to the encoder-decoder. Next, the model takes the absolute and gradient difference to compare the input image to the reconstructed frame from the auto-encoder. Finally, the stacked absolute and gradient difference image are fed into a classifier network that either outputs the presence (1) or absence (0) of anomalies.

In the most minimal case for an anomaly detection model, an auto-encoder model compares the reconstructed image to the input image on the basis of criteria, such as the absolute difference and gradient difference.

### 3.2.2 Transfer learning

*Transfer learning* involves first training a model on a problem similar to the actual problem (on a different sample) and then using that model for another task [TS09, OGS+09]. The addition of a student-teacher feature offers an unsupervised equivalent to transfer learning, which uses discriminative embeddings from pretrained networks and is popular in supervised learning.

The addition of a student-teacher feature in computer vision takes away the limitations of models that deal with large high-resolution datasets, such as necessary dimensionality reduction and heavy training data subsampling. In a typical student-teacher model, such as *Uninformed Students*, student networks are trained to regress the output of a descriptive teacher network. If the output from the student network differs from the teacher network, those differences are considered anomalous [BFSS20].

### 3.2.3 Global-local

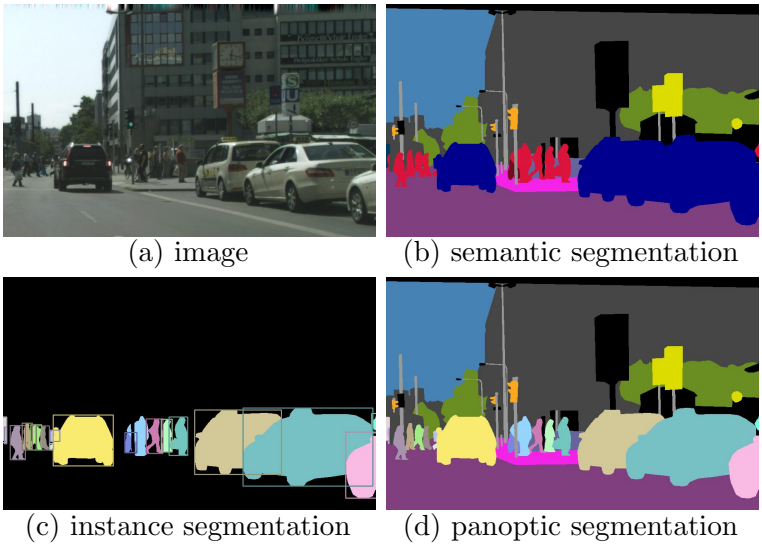
Another anomaly distinction that has not yet been addressed is that of *global* and *local* anomalies. Local anomalies are related to low-level features (2.2.1) and can be identified by their neighborhood, whereas global anomalies are mostly semantic and require a larger receptive field to be detected [CCX+24].

Global-local anomaly detections models, such as GLAD (*Global-to-Local Anomaly Detector*) and GLASS (*Global and Local Anomaly co-Synthesis Strategy*), aim to extract information form an image that is useful in detecting both types of anomalies [CLLZ24, YLW+24].

### 3.2.4 Segmentation

Another contribution to anomaly detection, and to computer vision in general, is *segmentation*. Segmentation is important because it can be used to leverage domain-specific knowledge about the image in order to detect anomalies. After performing segmentation on objects, it is easier to detect both structural and logical anomalies. Segmentation works by either clustering pixels with similar texture properties (region-based) or by attempting to find 'texture edges' between pixels that come from different texture distributions (boundary-based) [SS01].

There are different types of segmentation: *semantic segmentation*, *instance segmentation*, and *panoptic segmentation*. Semantic segmentation offers a pixel-level classification, including amorphous or uncountable regions [CVC23]. Instance segmentation, on the other hand, focuses on distinguishing countable 'things' in these semantic segments, and to distinguish individual instances of the same class [FWR+17]. Within instance segmentation, *object detection* is defined as bounding areas of same class instances with so-called *bounding boxes*. Finally, *panoptic segmentation* seeks to combine both semantic and instance segmentation in order to assign labels to each pixel and identify objects. Figure 4, from the paper by Kirillov et al., shows a visual representation of the different types of segmentation [KHG+18].



(a) image (b) semantic segmentation

(c) instance segmentation (d) panoptic segmentation

Figure 4: This figure is from the paper by Kirillov et al. [KHG+18]. For a given image (a), this figure shows the *ground truth* for: *semantic segmentation* (b), *instance segmentation*, being the colored areas, and *object detection*, being the bounding boxes (c) and *panoptic segmentation* (d).

Segmentation-based anomaly detection adds a branch to the standard auto-encoder architecture by segmenting the input image after encoding. The model then proceeds to detect anomalies in these objects. Examples of this type of anomaly detection can be found in Part Segmentation-based Anomaly Detection (PSAD) and Component Segmentation-based Anomaly Detection (CSAD), which each adds more stages to the basic auto-encoder structure [KAC+24, HL24].

### 3.2.5 Combining different components

Models such as *PaDiM* and later *PatchCore* leverage transfer learning (3.2.2) in the context of computer vision by using a so-called 'pre-trained backbone' [RPZ+22, DSLA20]. These encoders (3.2.1) are pre-trained on a large dataset of images, such as *ImageNet*, which is a visual database with over 14 million images [ima]. Pre-trained backbones extract features from input images [ZAA+21]. Feature extraction (2.2.1) is facilitated by these pre-trained networks because they have generally been trained on a significantly larger dataset than the target dataset.

Other models, such as *CSAD* and *GLASS* use a combination of a global-local-architecture (3.2.3) and a student-teacher-architecture (3.2.2) [HL24, CLLZ24]. Outputs from both local and global student networks are compared to an established 'normal' output from a (pre-trained) teacher model. The introduction of global-local-architecture is chosen to detect both local and global anomalies (2.3.1).

The next two sections will elaborate on the architectures of two specific models, being *PatchCore* (3.3) and *GLASS* (3.4), and draw a comparison between them.

## 3.3 PatchCore

*PatchCore* is an anomaly detection method that focuses on detecting *patch-level* anomalies. Input images are broken down into patch-level features. The main idea is that the entire image can be

considered anomalous if there is a patch that is anomalous. The reason why *PatchCore* was chosen as a first model in this research is its approach to address the *cold-start* problem. The cold-start problem arises in scenarios where acquisition of normal images is easy, but acquiring anomalous images is expensive and elaborate. Other motivations behind the choice for *PatchCore* are its relatively high performance on the MVTec AD benchmark dataset (3.1.1) of up to 99.6%, as well as short training times related to the use of only a single epoch during training [RPZ+22].

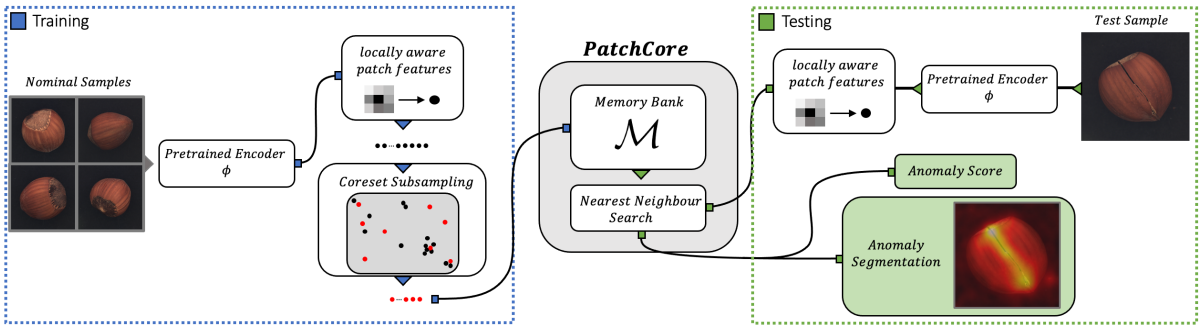



Figure 5: General overview of the *PatchCore* model showing a shared retrieval of locally aware patch features (3.3.2) in both training and testing alongside a pretrained encoder (3.3.1). Moreover, it presents training phase-specific coreset subsampling (3.3.4), which determines the size of the memory bank (3.3.3), against which the testing phase compares its locally aware patch features, derived from test samples by means of nearest neighbor search leading to its final anomaly score (3.3.5).

### 3.3.1 Pretrained encoder

*PatchCore* leverages *transfer learning* (3.2.2) by using a pre-trained network $\phi$. Due to a lack of data, it is often not feasible to train a computer vision model on all features present in the set of input images. Therefore, *PatchCore* uses architectures that were pre-trained on *ImageNet*. The architecture used in this thesis is WideResNet-50-2 [ZK16].

*PatchCore* builds a memory bank $\mathcal{M}$ consisting of mid-level feature representations from training data, serving to extract features from the test images (2.2.1). The reason for choosing this feature representation is to avoid features that are too generic or too heavily biased towards ImageNet classification. [RPZ+22].

### 3.3.2 Locally-aware patch features

This formally results in an image $x_i \in \mathcal{X}$, given dataset $\mathcal{X}$, with features of that image denoted as:

$$\phi_{i,j} = \phi_j(x_i), \tag{1}$$

where $j$ represents the index of feature maps from ResNet-like architectures $\phi$. In the case of WideResNet-50-2, we have that $j \in \{1, 2, 3, 4\}$. The formal definition of selecting mid-level features translates into taking $j \in [2, 3]$.

The collection of the features of an image $x_i$ is then defined as the *feature map*:

$$\phi_{i,j} \in \mathbb{R}^{c^* \times h^* \times w^*}, \tag{2}$$

where $\phi_{i,j}$ is a three-dimensional tensor of depth $c^*$, height $h^*$ and width $w^*$. Combining (1) and (2), we arrive at:

$$\phi_{i,j}(h,w) = \phi_j(x_i,h,w) \in \mathbb{R}^{c^*}, \tag{3}$$

where $\phi_{i,j}(h,w)$ denotes the $c^*$-dimensional feature slices at positions $h \in \{1,\dots,h^*\}$ and $w \in \{1,\dots,w^*\}$. This notation relates to image-patch feature representation. *PatchCore* aims to increase receptive field size without losing spatial resolution when composing each patch-level feature representation.

The neighborhood of a patch of patch size $p$ is represented as $\mathcal{N}_p^{(h,w)}$. *PatchCore* uses adaptive average pooling to aggregate these feature vectors.[3] The collection of all patch-features for a feature map tensor $\phi_{i,j}$ is $\mathcal{P}_{s,p}(\phi_{i,j})$.

### 3.3.3 Memory bank

Finally, we arrive at the definition of the *PatchCore* memory bank:

$$\mathcal{M} = \bigcup_{x_i \in \mathcal{X}_N} \mathcal{P}_{s,p}(\phi_j(x_i)), \tag{4}$$

which is the union of locally aware patch-feature collections of all training samples $x_i \in \mathcal{X}_N$, with each element denoted $m \in \mathcal{M}$.

### 3.3.4 Coreset subsampling

As the set of image samples $\mathcal{X}_N$ grows, the memory bank $\mathcal{M}$ grows exceedingly larger, which subsequently increases inference times. To reduce $\mathcal{M}$, *PatchCore* uses *coreset subsampling*, resulting in an $\mathcal{M}$-coreset $\mathcal{M}_C$ covering only parts of the original memory bank. *PatchCore* uses greedy coreset sampling instead of random sampling. The exact calculations behind the coreset subsampling can be found in the original paper [RPZ+22]. For the scope of this thesis, we will focus on the percentage $n$ to which the original memory bank was subsampled or the *coreset (sub)sampling ratio*, denoted $r$.

### 3.3.5 Anomaly score and nearest neighbor search

The image-level anomaly score $s \in \mathbb{R}$ for a test image $x^{\text{test}}$ is estimated by taking the maximum distance score $s^*$. This maximum distance score $s^*$ is calculated between test patch-features in the patch collection $\mathcal{P}(x^{\text{test}})$ and each respective nearest neighbor $m^*$.

Suppose we have memory bank features $m^*$ closest to anomaly candidate $m^{\text{test},*}$. The general idea is that anomaly score is increased if these memory bank features $m^*$ are far away from neighboring samples. This notion is formalized by the following two formulae:

$$s^* = ||m^{\text{test},*} - m^*||_2, \tag{5}$$

where $s^*$ is again the maximum distance score between patch features, $m^{\text{test},*}$ is the anomaly candidate and $m^*$ is the nearest neighbor in $\mathcal{M}$, calculated by taking the 2-norm or Euclidean

[3] For more information on the exact calculation, please refer to the paper by Roth et al. [RPZ+22]

distance between $m^{\text{test},*}$ and $m^*$, which are themselves defined as:

$$m^{\text{test},*}, m^* = \underset{m^{\text{test}} \in \mathcal{P}(x^{\text{test}})}{\text{argmax}} \underset{m \in \mathcal{M}}{\text{argmin}} ||m^{\text{test}} - m||_2, \tag{6}$$

where the Euclidean distance is calculated between each test patch feature $m^{\text{test}}$ and its nearest neighbor $m$ in the sampled memory bank $\mathcal{M}$ (or argmin). From all patch features, the one with the largest distance is identified as the anomaly candidate $m^{\text{test},*}$ by means of argmax.

The anomaly score $s$ is then calculated by performing some re-weighting on the Euclidean distance between $m^{\text{test},*}$ and $m^*$ before multiplication with the maximum distance score $s^*$:

$$s = \left(1 - \frac{\exp||m^{\text{test},*} - m^*||_2}{\sum_{m \in \mathcal{N}_b(m^*)} \exp||m^{\text{test},*} - m^*||_2}\right) \cdot s^*, \tag{7}$$

where $\mathcal{N}_b(m^*)$ represents the $b$ nearest patch-features in $\mathcal{M}$ for test patch-feature $m^*$ and $s^*$ is multiplied by 1 minus the re-weighting of (5).

### 3.3.6 Known deficiencies

Despite the state-of-the-art performance of *PatchCore* on benchmark datasets, there are some known deficiencies both on a theoretical and a practical level. First of all, there is a trade-off between short training times and GPU memory usage. *PatchCore* has an innate feature of requiring only a single epoch for training, which keeps training times relatively limited compared to other models requiring multiple epochs. On the other hand, *PatchCore* stores all extracted features from training images in memory to create its memory bank. This means that the memory usage of a *PatchCore*-model scales with the size of the dataset. *PatchCore* thus proves less suitable for larger datasets.

A more practical downside to the use of *PatchCore* in our use case is its core assumption that an anomalous patch directly translates into an anomalous image. The reason for this disadvantage is clear from the setup of our dataset: there is no constant background nor lighting. Since there is no segmentation pre-processing of the input data, *PatchCore* might classify normal images with shadows or different backgrounds as anomalous.

## 3.4 GLASS

*GLASS* stands for *Global and Local Anomaly co-Synthesis Strategy*, combining a global-local architecture (3.2.3) and a synthesis-based method (2.2.3). What sets *GLASS* apart is that it combines image-level and feature-level synthesis of a broader coverage of anomalies. The choice for *GLASS* as a second model comes with its near-perfect AUROC of 99.9% on the MVTec AD benchmark dataset (3.1.1), in addition to a 98.8% AUROC on the VisA dataset (3.1.2), 99.6% on the MPDD dataset (3.1.3) and 100% on the WFDD dataset (3.1.4). However, this model was not solely chosen on the basis of its good performance. The main idea behind *GLASS* is that it minimizes the overlap between anomalous and normal samples, as opposed to traditional anomaly synthesis strategies [CLLZ24].

The minimization of overlap between anomalous and normal samples is especially important in our use case. The reason for this is that there are only a limited number of anomalies, so a clear separation between normal samples and anomalies is an indication that the model performs well.

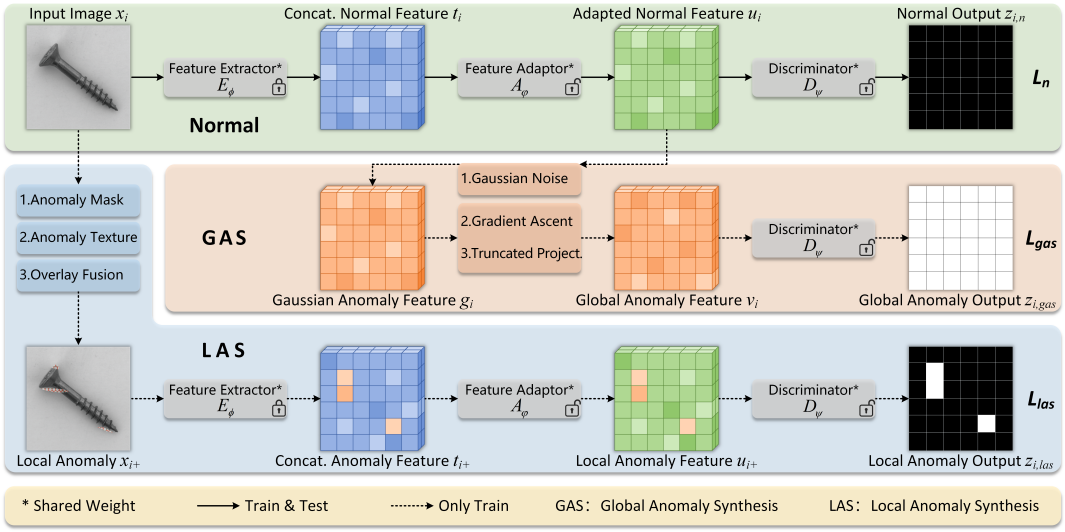



Figure 6: *GLASS* consists of three branches, shown in figure 6, being a Normal branch, GAS branch (3.4.4) and LAS branch (3.4.5). The Normal and LAS branch feature a feature extractor (3.4.1), a feature adaptor (3.4.2) and a discriminator (3.4.3). The GAS branch only shares the discriminator with the other branches.

### 3.4.1 Feature extraction

Similarly to *PatchCore*, *GLASS* uses a pre-trained backbone $\phi$, as defined in (1) and (2), combined with feature aggregation through adaptive average pooling[4]. This aggregation derives the locally aware vector $s_{i,j}^{h,w}$ from the neighborhood features of $\phi_{i,j}^{h,w}$, constituting the feature map $s_{i,j}$.[5] After upsampling and merging $s_{i,j}$, we obtain the concatenated feature map:

$$t_i = E_\phi(x_i), \text{with } t_i \in \mathbb{R}^{H_m \times W_m \times C}, \tag{8}$$

where $t_i$ is the concatenated feature map and $C$ is the channel size, defined as $C = \sum_{j \in J} C_j$ ($j$ being the index of feature maps, as defined in (1)).

### 3.4.2 Feature adaptation

In addition to a feature extractor $E_\phi$ (2.2.1), *GLASS* proposes a feature adaptor $A_\varphi$ (2.2.2). This feature adaptor is implemented to mitigate domain bias on the features of feature extractor $E_\varphi$, similarly to how *PatchCore* only extracts mid-level features from its backbone $\phi$ and uses a memory bank to mitigate ImageNet-specific bias.

Feature adaptation in *GLASS* is based on feature adaptation techniques from [LLS22]. Within the scope of this thesis, $A_\varphi$ employs a single-layer perceptron. We define the result after feature adaptation as:

$$u_i^{h,w} = A_\varphi(t_i^{h,w}), \tag{9}$$

where $u_i^{h,w}$ is the *adapted normal vector*, with $h$ and $w$ as defined in (3) and $A_\varphi$ the feature adaptor.

[4] *GLASS* uses WideResNet-50 and *PatchCore* uses WideResNet-50-2.

[5] Note that the notation of $s_{i,j}^{h,w}$ differs slightly from the notation in the *PatchCore* paper, $s_{i,j}(h,w)$. The notations will be used interchangeably in this thesis, depending on the model that is referred to.

### 3.4.3 Discriminator

All three branches of *GLASS* also share a discriminator $D_\psi$. The exact functioning of the discriminator is beyond the scope of this thesis and can be found in the original paper [CLLZ24]. After feature extraction and adaptation, $D_\psi$ gives a segmentation result, defined as:

$$z_i = D_\psi(u_i), \tag{10}$$

where $z_i$ is the segmentation result following from discriminator $D_\psi$ and where $u_i$ is itself the result from feature extraction and adaptation $u_i = A_\varphi(E_\phi(x_i))$ for input image $x_i \in X_{\text{test}}$.

The discriminator also employs a single, hidden layer perceptron with a sigmoid function, that outputs the *anomaly confidence* $z_i^{h,w} \in \mathbb{R}$ for each feature point which is obtained in the same way as (10).

### 3.4.4 Feature-level Global Anomaly Synthesis Strategy (GAS)

The GAS-branch is distinct from the other branches (Normal and LAS) because of the different input and operations that the branch features. This branch starts out by adding Gaussian noise to the retrieved normal features to synthesize anomalies (2.2.3). The Gaussian noise is adopted to simulate diverse anomalies. What makes *GLASS* more effective than other models that adopt Gaussian noise is the fact that its anomaly synthesis is guided by gradient ascent on a branch-loss function $L_{gas}$, which is calculated by taking the Binary Cross-Entropy (BCE) between the normal feature discrimination $z_{i,n} = D_\phi(u_i)$ and the ground truth of the feature map normal.

The GAS-branch implements a truncated projection, either Manifold or Hypersphere, based on the a *distribution hypothesis*. This distribution hypothesis determines the bounds of the anomaly features and their distribution in the feature space. Both the Manifold and Hypersphere hypotheses have a distinct *truncated projection* to constrain the range of gradient ascent-controlled anomaly synthesis.[6] In implementations of the model, the distribution hypothesis is determined by analysis of the image-level spectrogram.[7]

After adding Gaussian noise, based in the direction of gradient ascent on the loss function, and constrained by the relevant truncated projection, the GAS-branch arrives at a global anomaly feature $v_i$. This feature is passed through the discriminator to obtain a global anomaly output $z_{i,\text{gas}}$, with $z_i$ defined as in (10).

### 3.4.5 Image-level Local Anomaly Synthesis Strategy (LAS)

In addition to the synthesis of global, feature-level anomalies, *GLASS* also employs a LAS-branch, which is focused on the synthesis of image-level, local anomalies. As opposed to the GAS-branch, the LAS-branch departs from an input image instead of its adapted normal feature.

Before deriving potential local anomalies from the input image, the LAS-branch generates two binary masks by Perlin noise. These masks are used to obtain the foreground mask of a normal sample and shape of an anomalous region. After establishing the shape, an anomaly texture image is obtained by randomly selecting an image from the DTD dataset (3.1.5) and performing randomly

[6] For more information on the exact derivation of both the Manifold and Hypersphere truncated projections, please refer to pages 6 and 7 of the original paper [CLLZ24].

[7] Examples of this spectrogram

choosing three augmentation methods from a total set of 9 augmentation methods. Finally, the LAS-branch uses an overlay fusion to detect *weak defects* (2.3.1). The output local anomaly image is then processed by the extractor, adaptor and discriminator, similarly to the normal branch.

### 3.4.6 Anomaly score

The anomaly score is the result of taking the maximum value of all points in segmentation result $z_i$ (10):

$$s = \max_{h,w} z_i^{h,w}, \tag{11}$$

where $s$ is the anomaly score, $z_i^{h,w}$ is the anomaly confidence at location $(h, w)$ and $D_\varphi$ is the discriminator.[8] The anomaly thus returns the highest anomaly confidence found amongst different pixel-level anomalies.

### 3.4.7 Known deficiencies

As a model with multiple training epochs, *GLASS* is prone to overfitting in line with other models of a similar nature. In the original implementation of the paper, there is no early stopping to prevent the model from continuing to train in a local optimum. The best model is determined instead by performing inference at each iteration and saving the best model checkpoint. Following the original configuration used for all benchmark results, the model would only stop training after 640 epochs. In settings with limited time and computational resources, this configuration would pose an obvious challenge.[9]

Another point of attention is the way, in which the GLASS results were obtained. Rather than providing the model with a training, validation and test set, the GLASS paper uses only the train-test split that is already present in benchmark datasets, such as the MVTec dataset (3.1.1). The model will learn from the test set in next epochs, which means that the test set is not unseen. This would normally be considered bad practice in standard machine learning configurations. In anomaly detection papers, this approach is common practice because of the scarcity of the anomalies and the desire to compare models at their best performance. Since this thesis will compare *GLASS* to *PatchCore*, which only has one epoch and does not learn from the validation or test data, a different set-up for *GLASS* will be chosen to ensure fair comparison 4.3.

# 4 Methodology

The general study design is both comparative and iterative. The main challenge of this research is comparing *PatchCore* and *GLASS* and adapting these models for the introduced use case. However, a secondary goal of this research is to increase the chosen model's performance in both anomaly detection and localization and expand the scope to allow for images of different pantograph types at every angle and in every weather circumstance.

[8] In the thesis $S_{AD}$ is used to denote the image-level anomaly score, as opposed to the pixel-level anomaly score $S_{AL}$. In this thesis, $s$ will simply be used for the image-level anomaly score.

[9] As a reference, the author mentions training times of 4 hours to complete 640 epochs on a training set of 280 images, using an NVIDIA Tesla A800 as a GPU [CZ25a].

## 4.1 Pre-processing

### 4.1.1 Data filtering

The images of train roofs and pantographs are currently taken daily at two railway sections in the Netherlands, being Hekendorp and Hoofddorp [tre]. There is a degree of variation in the input images in terms of weather and lighting conditions, angle, and pantograph types. Images are taken in both rainy and sunny situations, at daytime and nighttime. Furthermore, there are three angles at which the images are taken, being a top view and a view from a 45 degree angle from either the left or the right side of the train.

The dataset consists of JPEGs and PNGs and comes with a CSV-file `metadata`, which encapsulates some of the variables described above. The CSV-file features several relevant column names, such as:

- `device`: the location of the camera
- `fleet`: the train type
- `side`: the side from which the image was taken
- `has_defect`: whether the image is normal (healthy) or abnormal (defect)
- `file_id`: the name of the image file

In order to filter the images based on the metadata, a `DataPrepper` class is devised to create a `pandas` DataFrame based on criteria in a `config.yml` file.

At the start of this research, the model will be restricted to one angle and one pantograph type. The angle will be chosen based on the most number of anomalies in that category, since the extra normal images can be scraped, if needed. The normal situation will be established using images taken in all weather and light circumstances to avoid overfitting on a certain scenario.

### 4.1.2 Cropping

For all models, the input image will be cropped around the pantograph. The pantograph image is cropped using a YOLO (*You Only Look Once*) object detection model (3.2.4). What is unique about this type of model is that it examines the entire picture only once, before identifying objects and their positions [RDGF16].

For this research, a pre-trained YOLOv5-model was devised [KH24a]. This model was trained by the computer vision team of the NS Maintenance Department based on a standard, non-finetuned YOLOv5x-model, and can classify active pantographs (in use), inactive pantographs (out-of-use) and pantograph heads. For this research, only active pantographs are of interest.

### 4.1.3 Flipping

Pantographs can have two orientations, which are visually similar to a >-sign and a <-sign. The orientation chosen for the first proof of concept was < since there were more pantograph images pointing in that direction. However, for later versions, images pointing in the opposite direction were flipped.

To facilitate the flipping process, a binary classifier (see: appendix B) was trained on manually sorted data using YOLOv11 [KH24b]. YOLOv11 is a more recent version than YOLOv5, which was used for cropping because it had already been pre-trained. The reason for the choice of a YOLOv11-model is its higher accuracy and faster training times compared to YOLOv5 [Ult].

## 4.2 Dataset setup

After pre-processing, the images are subdivided into a train, validate and test set. An 80-10-10 split was chosen with 80% of the images going towards training, 10% towards validation and another 10% towards testing. The splits are made on exclusive unit presence: trains with the same unit number cannot appear in more than one split. This split is made to prevent bias towards specific unit ranges.

## 4.3 General design

**PatchCore.** There are five iterations per model type, with every iteration being a new instance of the model on the same dataset split, but with a different seed. The reason for the introduction of five different seeds (0, 42, 200, 138, 159) is that model performance can fluctuate. Instances of a *PatchCore*-model produce different results on the same dataset, based on their seed. The reason for this is that *PatchCore* uses a k-center-Greedy method to perform coreset subsampling. Even though subsequent steps in its coreset subsampling algorithm 1 are deterministic, the initial cluster center for the k-center selection process is chosen randomly, by means of a seed. This approach is based on the Core-Set Approach introduced by Sener and Savarese [SS18]. In addition to this k-center-Greedy method, the *PatchCore*-paper introduces random linear projections to reduce the dimensionality of elements of the memory bank $m \in \mathcal{M}$, based on [SZG+19].

Since *PatchCore* uses only one epoch, multiple iterations with different seeds will be used to judge validation set performance. These iterations are then compared on their prediction scores, i.e. what anomaly score is given to each image. Based on the distribution of prediction scores, the anomaly threshold (2.3.2) is set, which determines whether a model has classified an image as anomalous or not. Selection from these different seeds happens on the same criteria as the combined scores in *GLASS*, (12) and (13).

**GLASS.** Since *GLASS* uses its test set as a validation set, the original implementation has been slightly altered to only use the validation set during training. In the testing phase, the model weights are loaded and inference is performed on the unseen test set. *GLASS* normally selects the model with the highest AUROC from the training-validation phase. Due to the small number of anomalies, not all validation sets contain anomalies. One of the challenges of the set-up of *GLASS* in this context is to find a suitable heuristic in selecting the 'best' model. Images should generally receive a low anomaly score and have a right-skewed distribution, meaning that higher anomaly scores are less frequent than lower anomaly scores. In cases where there are no anomalies in the test set, the criterion for choosing the 'best' model is based on a combined score:

$$s_{c,n} = \max(S) + \alpha \cdot \text{med}(S), \tag{12}$$

where $s_{c,n}$ is the combined score for a set of only normal images and $S$ the set of all scores, over which the maximum and the median is calculated. We take some constant $\alpha = \frac{1}{2}$ to weight these

scores against each other. The goal is to minimize this combined score.

In the original setup, the sum of the image-level AUROC and pixel-level AUROC was taken to select the best model. In cases where anomalies are present in the validation set, this thesis chooses to diverge from the AUROC as a heuristic for choosing the 'best' model. The reason for this is that the pixel-level AUROC of only a very small number of anomalies or even a single one is not representative of the fit on unseen anomalous images. Furthermore, the image-level AUROC can be high, even if there is a poor absolute separation between normal and defect images. This thesis proposes another combined score, which is calculated as follows:

$$s_{c,a} = \text{AP}_{\text{image}} + \text{med}(S_a) - \text{med}(S_n), \tag{13}$$

where $s_{c,a}$ is the combined score for a set of normal and anomalous images, $S_a$ is the set of scores for anomalous images, $S_n$ is the set of scores for normal images. The goal is to both maximize Average Precision (AP) and the difference between anomaly scores for normal and anomalous images. The Average Precision for an image is already calculated by the model by taking taking the area under the precision-recall curve (weighted mean of precisions at each threshold where recall increases) [ZZ09]. The *score gap* or difference between the median of the anomaly scores for normal images and the median of the anomaly scores for anomalous images is calculated at every epoch, during inference. This score gap will also be referred to as the *separation margin*.

In both cases, the 'best' model is defined as the model with the highest combined score. Due to computational and temporal constraints, the full 640 epochs will not be used for all experiments. The results in the proof of concept were run for only 50 epochs and the results after the proof of concept were run for 500 epochs.

### 4.3.1 Smallest possible subset

As a proof of concept, the input image is cropped around one pantograph type.[10] Moreover, the focus of this first version is selecting anomalies that are easiest to find, i.e. big and high contrast, and choosing the smallest subset (of the bigger dataset) with the largest number of anomalies. This version is taken at only a single location and with a single pantograph arm orientation.

### 4.3.2 One pantograph type

The subset of the data is then expanded to (1) include two pantograph arm orientations, by means of flipping (4.1.3), and (2) the two locations, Hekendorp and Hoofddorp. The goal of this expansion is to expose the model to more background variations during training, such that it can possibly infer the foreground-background separation. Note that (1) and (2) involve two different steps in the proof of concept.

[10] This might mean there are several train types sharing the same pantograph type.

# 5 Experiments

## 5.1 Experimental Details

**Datasets.** Based on the pantograph heads in 2.1.2, there are six main subsets of the dataset, corresponding to the different types of pantographs, being `VIRM_4`, `SLT_4_6_VIRM_4`, `ICM_1_4`, `VIRM_6`, `DDZ_4_6`, and `SNG_3_4`. The exact details of these six datasets can be found under A.2. In addition to these six datasets, the `ICM_1_4`-dataset, being the biggest, has substantiated this research by means of smaller *proof-of-concept*-subsets of the dataset:

- `ICM_1_4_Hekendorp_left`: The first version of the subset has images only taken from the left `side`, at `device` Hekendorp and with train `fleet` either ICM 1 or ICM 4, since these two fleets share the same pantograph type. Only cropping (4.1.2) was applied to these images. The resulting subset contains a total of 169 train images, 21 validation images and 23 test images (21 healthy images and 2 defects), roughly adhering to the 80-10-10 split.

- `ICM_1_4_Hekendorp_lt`: The second version of the subset features images taken in Hekendorp with pantograph arm orientations facing left (<, less than, abbreviated: lt). This orientation was either already present or flipped later (4.1.3). The resulting subset contains 299 train images, 31 validation images and 50 test images (48 healthy images, 2 defects).

- `ICM_1_4`: The final step in the proof of concept is to optimize the models on the ICM 1 & 4 with pictures from both locations and both sides (flipped to all be in the <-direction). This dataset consists of 671 train images, 98 validation images (97 healthy images, 1 defect) and 87 test images (85 healthy images, 2 defects).

The remaining datasets have the following sizes:

- `SLT_4_6_VIRM_4`: 446 train images, 51 validation images (50 healthy images, 1 defect) and 68 test images (66 healthy images, 2 defects);

- `SNG_3_4`: 278 train images, 45 validation images (44 healthy images, 1 defect), 39 test images (38 healthy images, 1 test image);

- `DDZ_4_6`: 155 train images, 19 validation images, 14 test images (13 healthy images, 1 defect);

- `VIRM_4`: 148 train images, 25 validation images, 18 test images (17 healthy images, 1 defect)

**Evaluation metrics.** The first phase consists only of fitting the model to the training and validation set. The reason for a 10% reservation for validation is to allow for intermediate evaluation of model performance on a separate set than the final test set and, eventually, hyperparameter optimization 5.3. Not all validation sets will have anomalies in them. In such cases, the distribution of the normal images will serve as a guide, as formalized in (12).

In case the validation set does contain anomalies, the evaluation metrics used are recall (or *true positive rate*) and AUROC. The AUROC is defined as the area under the ROC, which plots the true positive rate against the false positive rate and has a value between 0 and 1, with

a higher value indicating a better-performing model, whilst recall is equal to the number of true positives divided by the total number of positives [Mel13, Tin17].

The goal of this optimization is to establish what model has the least false positives on the validation set. Less number of false positives equals a higher *workload reduction*, which is the reduction in number of images that have to be manually checked. The false positive rate target that NS sets for the first version of its models is at less than 10% given the large number of normal images.

After training and validation, recall and AUROC are again considered to evaluate the performance of the model on the test set. Since there are only few anomalous images, missing one images can already have a significant influence on the recall. Therefore, the target is to have a first model with a recall of around 60%. Another metric of interest is the *error* (or the number of misclassified samples divided by the total number of samples). Models with a lower error are favorable over models with a higher error rate.

In addition to these performance metrics, the mean $\mu$, standard deviation $\sigma$ and deviation from the mean of the anomalous samples will serve as a guide to model significance. Judging if an anomaly falls in the appropriate percentile will require calculating the corresponding empirical percentile:

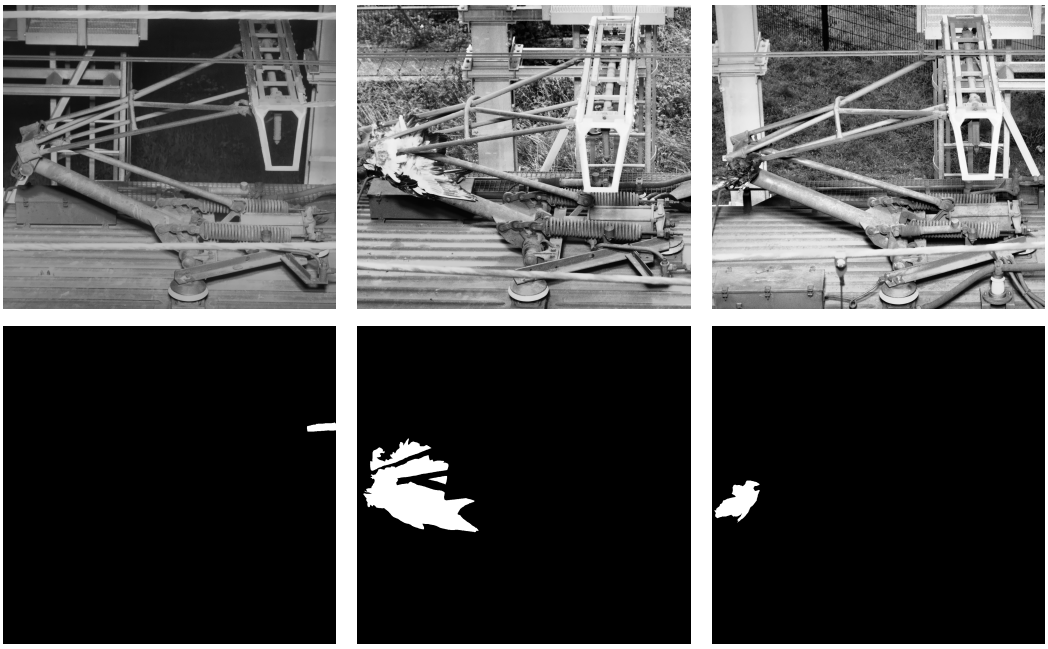
$$p_{emp}(s^*) = \frac{1}{n} \sum_{i=1}^{n} 1\{s_i \geq s^*\}, \tag{14}$$

where $p_{emp}$ is the empirical percentile of the anomaly score of the anomalous sample $s^*$. The empirical percentile is calculated by taking the sum of all normal samples that have an anomaly score $s_i \in \mathcal{S}$ that is higher than that of the anomalous sample, divided by the total number of samples $n$. This metric was chosen because no prior (normal) distribution of normal samples is assumed.

**Experimental set-up.** All experiments were run on an NCasT4_v3-series virtual machine, equipped with an NVIDIA Tesla T4 with 16 GB of GPU memory and AMD EPYC 7V12(Rome) CPUs with 8 cores and 56 GB RAM [mat]. Unless stated otherwise in the caption of the results, the experiments were run in the original configuration, found in Appendix C.

## 5.2 Proof of Concept

The proof of concept has an exploratory nature, to finetune settings and compare results in a computationally less expensive way, as well as a goal to substantiate the choices that were made in pre-processing and general design. Since the proof of concept will cover subsets of the same dataset that increase in size until they cover the original dataset `ICM_1_4`, this section lists the three anomalies in figure 7 to make it clear which anomaly is covered by which subset.



(a) (b) (c)

Figure 7: From left to right, we find anomalies 7a: a bent cable (structural anomaly) present in all subsets of the proof of concept, 7b: a white bird (logical anomaly) present in all subsets of the proof of concept, 7c: a small bird (logical anomaly) present only in the original dataset `ICM_1_4`.

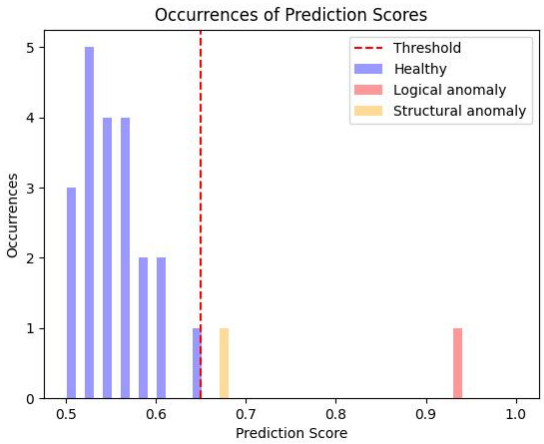
### 5.2.1 Experiments on ICM_1_4_Hekendorp_left

The first dataset used in the proof of concept is `ICM_1_4_Hekendorp_left`.

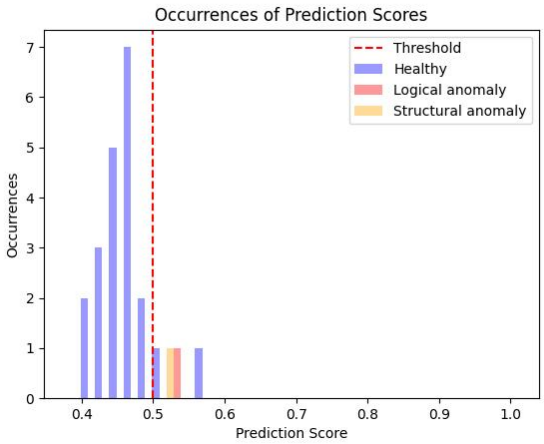
**Prediction score results.** After training the two models, a plot of the occurrences 8 of prediction scores will be devised to gain visual insight into the distribution of prediction scores for normal and anomalous images. A higher bar in the plot indicates that more samples received a certain score.

With a deviation from the normal samples of $4.02\sigma$, *PatchCore* has a stronger separation margin between the logical anomaly and normal samples than *GLASS*, which shows no significant separation for the logical anomaly at a deviation of $1.75\sigma$. For a final proof of concept, a better separation margin between the normal samples and both the logical and structural anomaly is desirable.

In addition to a smaller separation margin, *GLASS* misclassified one normal image as an anomalous image, which results in both anomalies 7a and 7b having in an empirical percentile $p_{emp}(s^*) = 0.04$ with a test size of 23 images.

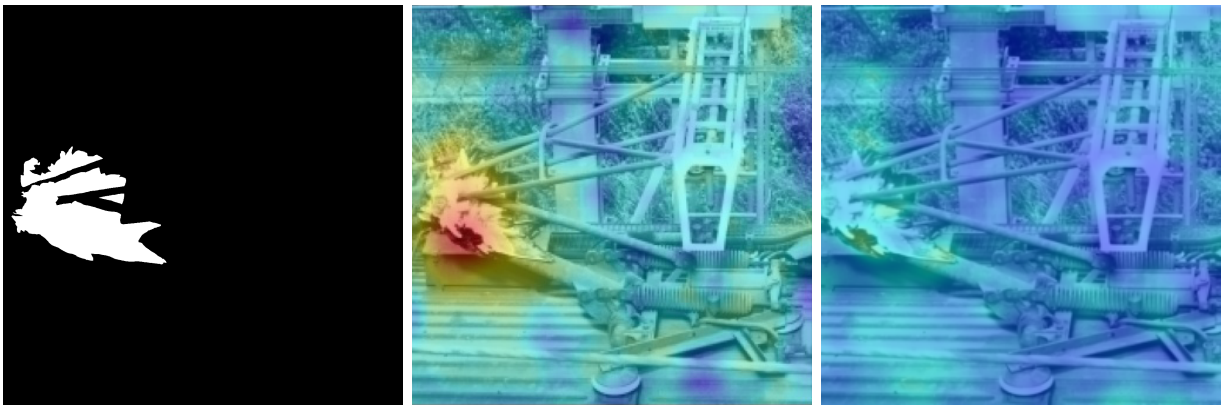



(a) *PatchCore* on ICM_1_4_Hekendorp_left




(b) *GLASS* on ICM_1_4_Hekendorp_left

Figure 8: On the left side 8a, we find the results for *PatchCore* with a coreset sampling ratio $r = 0.01$ and number of neighbors $k = 5$. Out of five different seeds 4.3, the first run (seed 0) was chosen. At an anomaly threshold of $t = 0.65$, *PatchCore* can distinguish both the structural 7a and the logical anomaly 7b. On the right side 10b, we find the results for GLASS. The best combined score (13) was achieved after 15 epochs out of a total of 50 epochs. *GLASS* has lower anomaly scores, in line with the combined score that is used to pick the best model based on (12) and can find both anomalies. *GLASS* has a smaller separation margin between the anomalies and normal samples, as well as one normal sample being classified as anomalous at anomaly threshold $t = 0.5$. Both figures display a skew to the right for the normal samples, which is expected because normal images should generally have a low anomaly score, with a mean score of $\mu = 0.58$ and $\mu = 0.46$ for *PatchCore* and *GLASS*, respectively.



(a) Ground truth (b) *PatchCore* (c) *GLASS*

Figure 9: In this figure, from left to right: 9a the ground truth containing a binary mask with the anomalous region being colored white, 9b the output anomaly map from *PatchCore* and 9c the anomaly map from *GLASS*. *PatchCore* displays a clear localization of the logical anomaly 7b in question, whereas *GLASS* fails to demarcate the anomalous area.

**Qualitative results.** To discover which parts of the image trigger the model, we will take a closer look at individual anomaly score maps in figure 9. The first two anomaly maps that will be compared are those of the logical anomaly as detected by *GLASS* and *PatchCore*.

One of the advantages of *PatchCore* becomes apparent in this qualitative example. *PatchCore* is known to perform effectively even when small data sets are provided [KC23]. The *GLASS* anomaly score map for 7b does not display a distinct anomalous region except for some brighter areas around the edges of the bird.

Since *GLASS* runs for several epochs and 'learns' from the training data, training for more epochs and adding more training data can both improve the performance of the model. The latter will be the focus of this proof of concept (5.2), whilst the former will be explored in the full experiments after the proof of concept (5.3).

**Performance metrics.** To quantify the qualitative and statistical differences between the two models that were addressed earlier, the first part of this proof of concept will be concluded by means of a table summarizing the performance metrics.

| **Method** | PatchCore | GLASS |
|---|---|---|
| **AUROC** ↑ | 0.99 | 0.95 |
| **Recall** ↑ | 1.0 | 1.0 |
| **Error** ↓ | 0 | 0.04 |
| **Misclassifications** ↓ | 0 | 1 |

Table 1: This table shows the different performance metrics for the *PatchCore* and *GLASS* models. This instance of the *PatchCore*-model achieves a near-perfect AUROC with zero misclassifications. The *GLASS*-model performs slightly less on the provided data set.

Judging solely by this table of performance metrics, the differences between the two models seem minute. However, for our specific use case, the *PatchCore*-model outperforms the *GLASS*-model on this dataset for several other reasons that are not captured by the table:

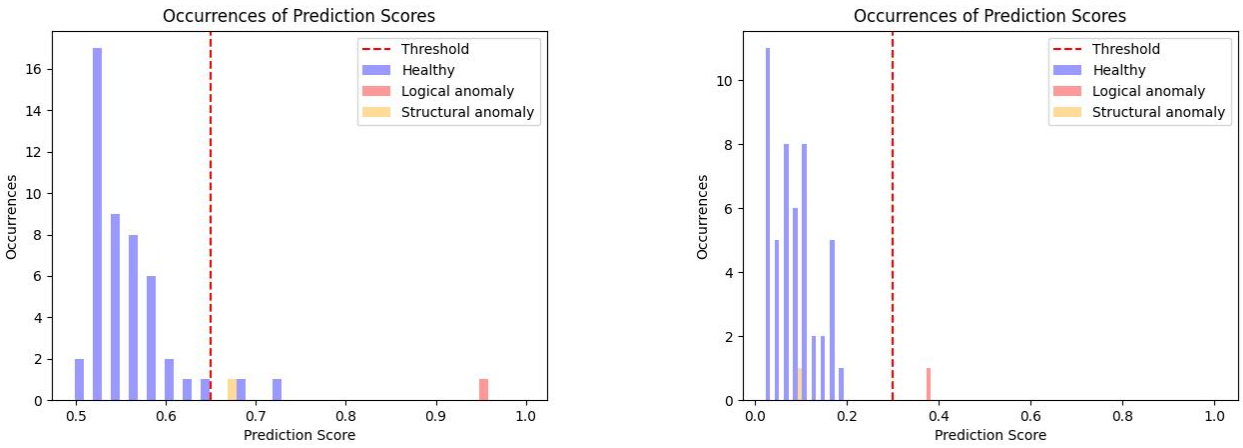
- First of all, the *PatchCore*-model shows a more significant separation margin between the anomalous and normal samples.
- Secondly, *PatchCore* is able to localize the anomalous region, which is useful in a shift to fully-automated anomaly detection.

The question is how well *PatchCore* will generalize on bigger datasets with more variation. The hypothesis for *GLASS* is that its performance will increase after expanding the datasets and increasing training epochs.

### 5.2.2 Experiments on ICM_1_4_Hekendorp_lt

In order to expand the data sets, flipping (4.1.3) was applied to images in the bigger dataset that had the opposite pantograph arm orientation. Whilst a flipped background does add extra background variation to the images, the expectation is that the extra data will lead to better foreground recognition and general model learning.

**Prediction score results.** The following results were obtained by running *PatchCore* and *GLASS* on the `ICM_1_4_Hekendorp_lt`-dataset.




(a) *PatchCore* on ICM_1_4_Hekendorp_lt

(b) *GLASS* on ICM_1_4_Hekendorp_lt

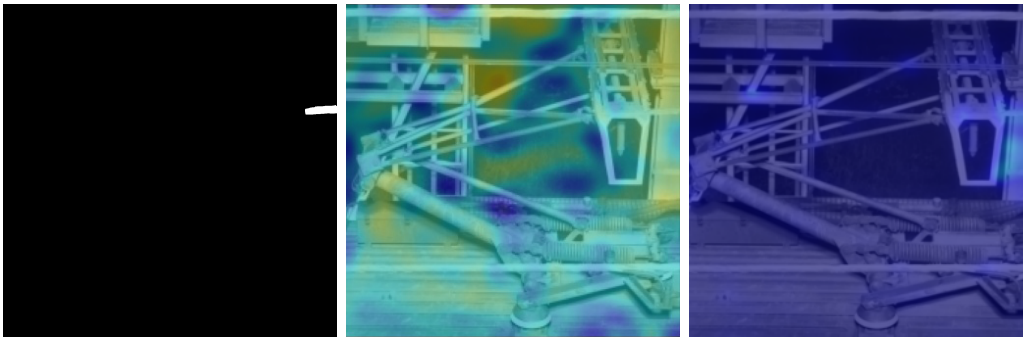
Figure 10: On the left side 10a, we find the results for *PatchCore*. Out of five different seeds 4.3, the first run (seed 0) was chosen. At an anomaly threshold of $t = 0.65$, *PatchCore* can distinguish both the structural 7a and the logical anomaly 7b. On the right side 10b, we find the results for GLASS after training for 50 epochs with the same parameters as the original paper. *GLASS* is not able to find the structural anomaly 7a. Both figures display a skew to the right for the normal samples, which is expected because normal images should generally have a low anomaly score.

#### 5.2.2.1 Original set-up.

Since *PatchCore* only runs for one epoch, adding anomalies only serves to evaluate the performance of the model after training and not during. In order to have the setups of *PatchCore* and *GLASS* match, the models were not presented with any anomalies during the time of training and validation for this experiment. Both models show a clear separation between the normal samples and the logical anomalies, though *GLASS* does not recognize the structural anomaly under the current threshold. The structural anomaly in question is small and hard to distinguish from its background. A possible reason why *PatchCore* does recognize the structural anomaly is because of its main paradigm that the entire image can be classified as anomalous if there is an anomalous patch [RPZ+22].

**Qualitative results.** We will hence zoom in on the images themselves and their respective anomaly maps in figure 11. As is visible from both figure 10 and 11, *GLASS* outputs lower anomaly scores, on average, than *PatchCore*. This behavior is directly correlated to the combined score function that were defined in (12), which penalizes high anomaly scores for normal images. One option is to include one anomaly in the validation set.

The structural anomaly 7a included in this dataset concerns a bent part of the pantograph, which is visually similar to the background. Due to its similarity to the pillar in the background of the image, this anomaly is particularly difficult to distinguish, even for the human eye. *PatchCore* seems to be more sensitive to the anomalous area. However, it should be noted that the model also picks



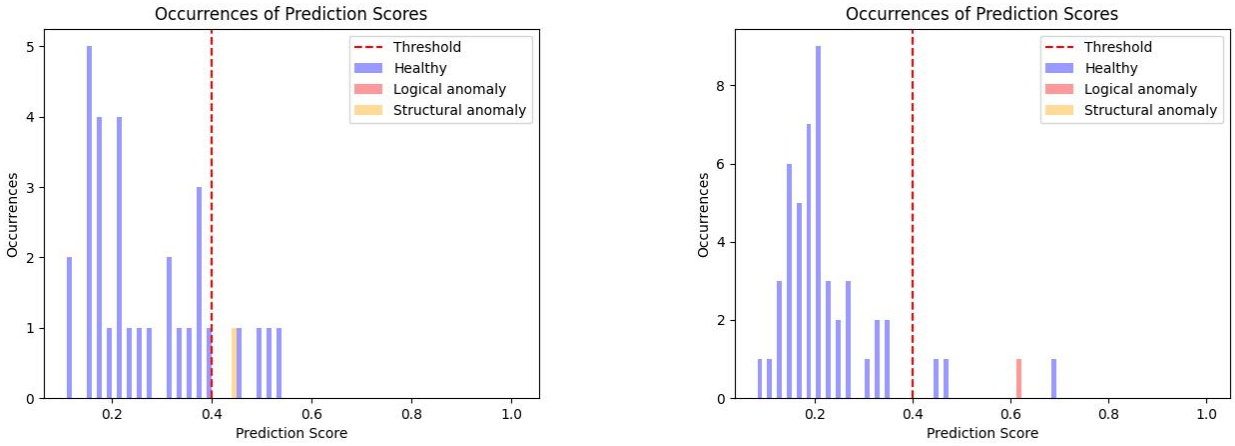
(a) Ground truth (b) *PatchCore* (c) *GLASS*

Figure 11: In this figure, from left to right: 11a the ground truth mask for anomaly 7a, 11b the output anomaly map from *PatchCore* and 11c the anomaly map from *GLASS*. The *PatchCore*-model is more sensitive to patches in the image that do not seem anomalous to the human eye. The *GLASS*-model, on the other hand, shows little sensitivity over the entire image, except for some area on and around the bent part of the pantograph.

up some non-anomalous regions, which could indicate that the model does not 'recognize' the anomaly as such.

#### 5.2.2.2 Anomaly in validation set

If one anomaly is included in the validation set, we get a better overview of *GLASS*-performance during training. This leads to the following results:

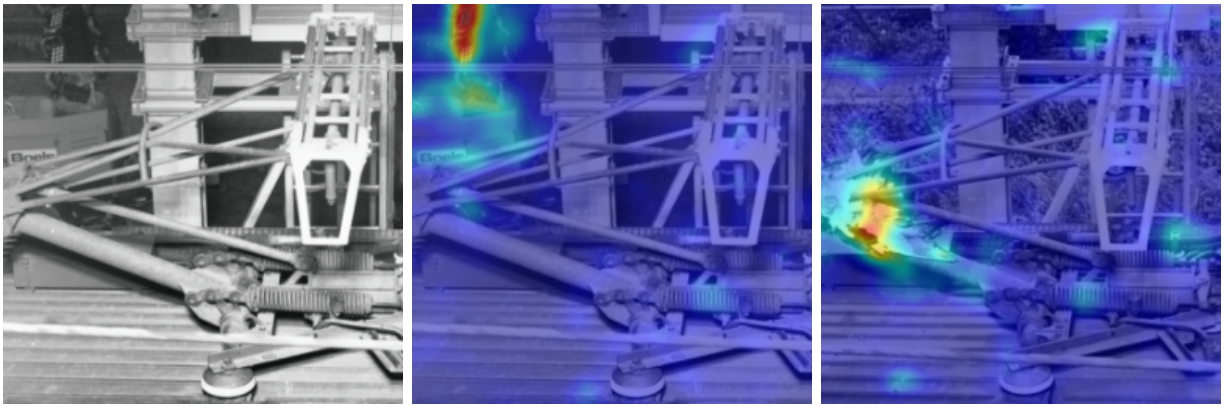



(a) *GLASS* on validation set (b) *GLASS* on test set

Figure 12: This figure shows results of the same *GLASS*-model on the validation set 12a and the test set 12b. The best combined score (13) was achieved after 47 epochs out of a total of 50 epochs. In both cases, there is some degree of separation between the anomaly and the normal samples. On the validation set, the distribution of prediction score occurrences does not show a right-skew and the separation margin is minimal. In the test set, the logical anomaly deviates $3.19\sigma$ from the mean of the distribution $\mu = 0.24$.

Compared to the configuration without an anomaly in the test set, there is a translation to the right for the distribution of prediction score occurrences ($\mu = 0.10$ to $\mu = 0.24$) and increase in misclassifications. The shift of the distribution and extra misclassifications can be explained by the difference in selection criterion for a 'best' model. For the configuration without anomalies in the validation set, model instances producing low maximum anomaly scores and median anomaly scores were favored (12). Meanwhile, the configuration with anomalies in the validation set favors model instances with a combination of higher image-level average precision for anomalous images and a higher score gap between median anomaly scores of anomalous and normal samples (13). The focus on image-level average precision and difference between medians does not necessarily lead to a lower number of misclassifications. For instance, a model that recognizes an anomaly well on a pixel-level may also be more sensitive to normal images.

Anomaly 7b with anomaly score $s = 0.62$ is in empirical percentile $p_{emp} = 0.02$ with test samples $n = 50$ because there is one normal sample that has a higher anomaly score. We will zoom in on this sample and compare its anomaly score map to the one of the logical anomaly, by way of example.



(a) Original: false positive (b) Overlay: false positive (c) Overlay: true positive

Figure 13: This figure shows the original image 13a and overlay of the false positive 13b with an anomaly score of $s = 0.71$, compared with the overlay image of anomaly 7b with anomaly score $s = 0.62$. In the original image, there is a piece of construction equipment in the background triggering the *GLASS* anomaly detection model.

As follows from figure 13, this *GLASS*-model instance shows some degree of sensitivity to the background. A possible reason for this is the fact that the construction equipment in the background was not possible in the training set. The aim to reduce background sensitivity is a core problem of this research. One possible solution is make manual splits and make sure that these types of variations appear in the training set. However, we do not want the model to consider the construction machinery as a natural or normal part of the background since it is not. In addition to this, manually selecting which images must appear in a certain split is labor-intensive.

Another solution would be to train a classification model recognizing objects in the image and then making splits based on objects falling within a segmented area. This would normally involve labeling all object instances manually. Recently advanced segmentation models have surfaced that could facilitate the task, such as Meta's *Segment Anything in Images and Videos* (SAM 2),

which can make segmentations of unseen images [RGH+24]. This option remains out-of-scope for this thesis due to the manual labeling.

We thus arrive at a trade-off between model performance and evaluation: adding an anomaly to the validation test provides a more robust way to choose a best *GLASS*-model, but will not always be possible, for example with subsets of the data that have only one anomaly. Furthermore, it decreases the number of anomalies in the test set, which is our only reference in evaluating how well the model performs on unseen data.

Given this trade-off, there are several options: gathering more anomalies, choosing different parameters or training for a longer period of time. Gathering more real anomalies is not possible, but generating synthetic anomalies and using them during validation is a possibility. This ensures that the number of anomalous samples in the test set is maximal. Choosing different parameters and longer training times are general solutions to boost performance but does not solve the problem of scarce evaluation metrics during training.

**Performance metrics.** Summarizing the results from the different configurations, we arrive at the following performance metrics:

| **Method** | PatchCore | GLASS[11] | GLASS-a[12] |
|---|---|---|---|
| **AUROC** ↑ | 0.98 | 0.78 | 0.98 |
| **Recall** ↑ | 1.0 | 0.5 | 1.0 |
| **Error** ↓ | 0.04 | 0.02 | 0.06 |
| **Misclassifications** ↓ | 2 | 1 | 3 |

Table 2: This table summarizes the performance metrics for the two different configurations of *GLASS*, as well as the results for *PatchCore*. Both models reach the same AUROC and recall due to a single anomaly being present. The error and number of misclassifications are lowest in the original configuration for *GLASS*, given an anomaly threshold of 0.65 for *PatchCore*, 0.35 for *GLASS* and 0.40 for *GLASS-a*. It should be noted, however, that *PatchCore* has one defect more in the presented test set.

In summary, the results of this second part of the proof of concept are:

- *PatchCore* outputs a higher anomaly score for the logical anomaly 7b, compared to the results of the smaller `ICM_1_4_Hekendorp_left` dataset. The anomaly score for the structural anomaly 7a is only slightly higher. These improvements come at the cost of 4 misclassifications, most likely due to the different backgrounds that are introduced when flipping some of the images.

- Adding an anomaly to the validation set, improves the AUROC from 0.78 to 0.98 for *GLASS* at a cost of 2 extra misclassifications. Additionally, there is a translation to the right for the distribution of prediction score occurrences (from $\mu = 0.10$ to $\mu = 0.24$).

- On average, *GLASS* outputs lower anomaly scores for normal samples with the bigger dataset. The mean prediction score shifts from $\mu = 0.46$ to $\mu = 0.10$ when an experiment is repeated

[11] No anomalies in the validation set
[12] One anomaly in the validation set

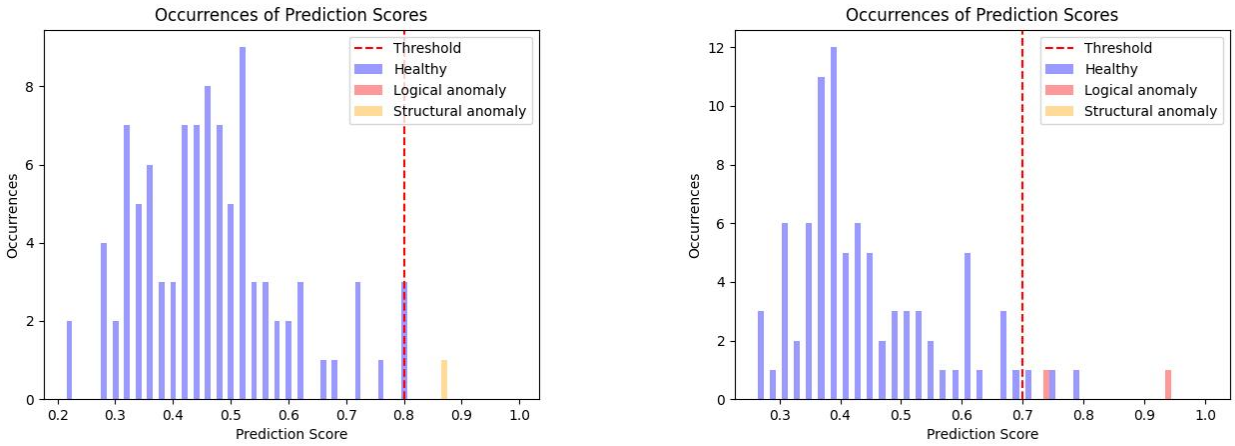
with the same configuration on a bigger dataset, whilst the anomaly score of the logical anomaly decreases by less. This is generally desirable and means that the separation between normal and anomalous images becomes more visible.

### 5.2.3 Experiments on ICM_1_4

After initial comparison and evaluation of results on a smaller subset, the scope data are now expanded to cover the entire `ICM_1_4`-dataset. This final part of the proof of concept serves to optimize the configurations of both models on a full subset of the data. For *GLASS*, this means running the model for 500 epochs, instead of 50, in the adapted train-validate-test-configuration that this thesis uses. The emphasis of this section is on *GLASS* more than on *PatchCore*. The section on hyper-parameter optimization (5.3) will elaborate more on *PatchCore*.

In order to ensure fair comparison to *PatchCore* later on, the adapted design described in 4.3 will be used for this experiment. The hypothesis is that a larger dataset and a more training rounds will improve *GLASS* performance compared to previous experiments.

**Prediction score results.** The following prediction score plots show the prediction scores of *GLASS* on the entire ICM-dataset:




(a) *GLASS* on validation set

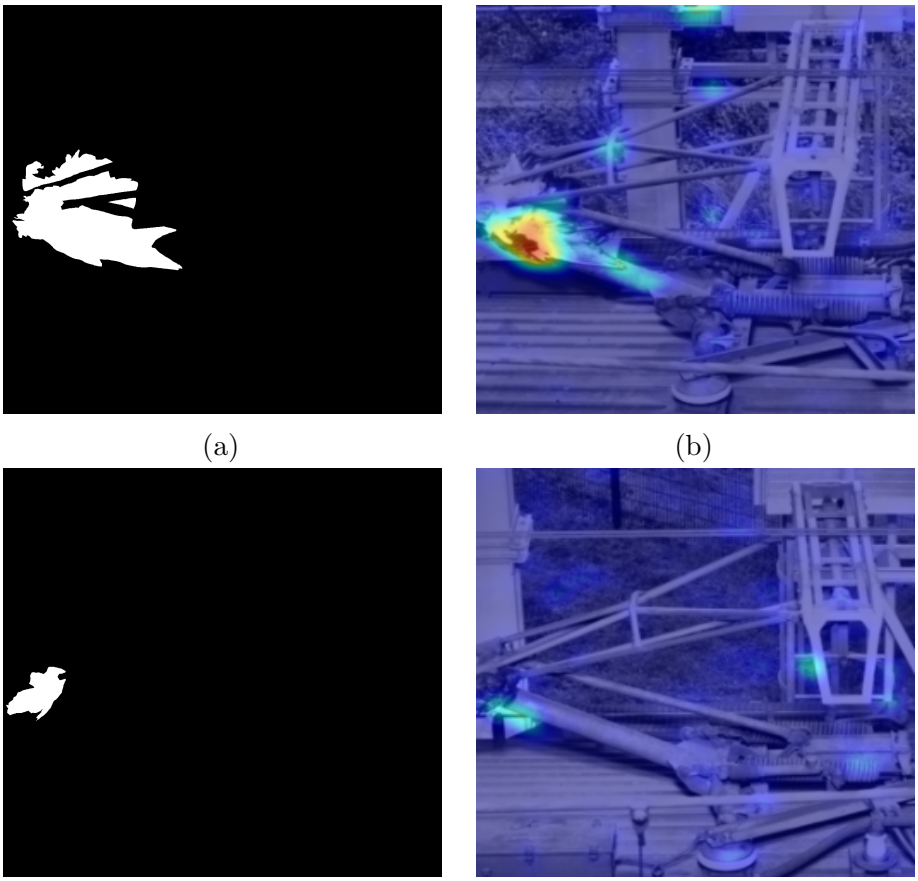
(b) *GLASS* on test set

Figure 14: Results for the entire ICM-dataset run over 500 epochs with structural anomaly 7a in the validation set and two logical anomalies 7b and 7c in the test set. After training for 328 epochs, the best combined score was found. The test set shows a right-skew with a mean of $\mu = 0.46$ and clear separation for logical anomaly 7b deviating $3.62\sigma$ from the normal sample mean, whereas the other logical anomaly 7c is closer to the anomaly threshold $t = 0.7$ with a deviation of $2.04\sigma$. At this anomaly threshold, there are three misclassifications.

As is visible from the prediction plots, *GLASS* attributes a higher anomaly score to 7b than previous versions of the model ($s = 0.93$ compared to $s = 0.37$ in the `ICM_1_4_Hekendorp_lt`-dataset). Another significant improvement from the first subset `ICM_1_4_Hekendorp_left` is the increase deviation from the mean of normal samples from $1.75\sigma$ to $3.62\sigma$.

Logical anomaly 7c has a $p_{emp} = 0.03$ with a sample size of $n = 87$. In the next section, the exact qualitative differences between the anomaly maps of both logical anomalies 7c and 7b will be entertained.

**Qualitative results.** To further evaluate the (localization) performance of this *GLASS*-model instance, we will zoom in on the anomaly score maps of the anomalies in the test set.



(a) (b)

(c) (d)

Figure 15: This figure shows the anomaly map and ground truth of anomalies 7b and 7c. Anomaly map 15b shows a clear localization for anomaly 7b, though not the entire ground mask 15a is covered. The localization for anomaly 7c in figure 15c does not show a strong overlap with its ground truth mask 15c, however.

This instance of the model detects all anomalies at anomaly threshold $t = 0.7$ but only partially localizes them. The partial localization could be related to low sensitivity.

**Performance metrics.** To draw a comparison between *GLASS* and *PatchCore* on ICM_1_4, a result for *PatchCore* from the HPO-section 5.3 is used:

| **Method** | PatchCore | GLASS |
|---|---|---|
| **AUROC** ↑ | 0.97 | 0.99 |
| **Recall** ↑ | 1.0 | 1.0 |
| **Error** ↓ | 0.20 | 0.03 |
| **Misclassifications** ↓ | 18 | 3 |

Table 3: This table shows the different performance metrics for the *PatchCore* and *GLASS* models. This instance of the *GLASS*-model achieves a near-perfect AUROC with three misclassifications at an anomaly threshold of $t = 0.7$. The *PatchCore*-model results were taken from table 4 with coreset subsampling ratio $r = 0.01$, number of neighbors $k = 5$ and anomaly threshold $t = 0.6$. *PatchCore* performs slightly less on the provided data set.

In this case, *GLASS* clearly outperforms *PatchCore*. A possible reason for this better performance is related to the presence of more training data. As stated before, *PatchCore* is known to perform well on smaller datasets because of its memory bank sampling design [KC23]. This advantage diminishes with larger datasets where 'traditional' epoch-based models have an edge. The increase in epochs allows the *GLASS*-model ample time to learn from the more elaborate data.

To wrap up this section, an overview will be provided of the specific design choices for *GLASS* and their effects, compared to *PatchCore*:

- Increasing the number of samples in the training set, increases the AUROC of the *GLASS*-model from 0.78 (ICM_1_4_Hekendorp_left) to 0.97 (ICM_1_4).
- *GLASS* keeps suffering from low sensitivity issues, partially due to the chosen heuristic (13). This leads to only partial localization.
- Flipping images is a 'cheap' method to gather more data. Instead of training two separate models, flipping images allows for a single model to be trained on both pantograph orientations. A side-effect is that the model will see more differences in background during training.
- Using images from both locations (Hoofddorp and Hekendorp) is advisable. The original idea to keep the first dataset small held variation at a minimum, but resulted in a small test set and, in the case of *GLASS*, a small separation margin. *PatchCore* fared relatively well on this single-location dataset with a perfect recall and 0% error rate.
- *GLASS* has longer training times at around two days for ICM_1_4 (a dataset of 800 images) if run for the full length of 640 epochs, whereas *PatchCore* takes around 20 minutes for the same dataset and 100 minutes for all five seeds. *GLASS* has slightly shorter inference times than *PatchCore* , which could play a role in live deployment.

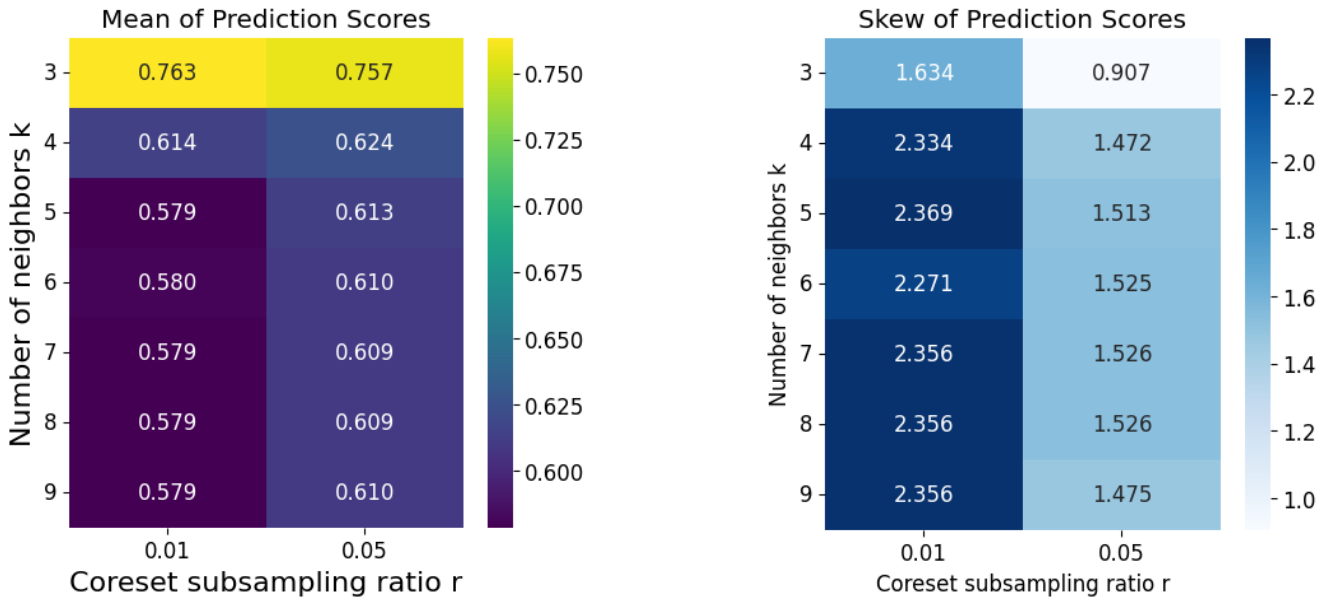
## 5.3 Hyper-parameter optimization

Training times for *PatchCore* are significantly shorter at approximately 30 minutes per training run for a dataset of around 800 images. Given its single epoch-structure, *PatchCore* does not 'learn' over different epochs. This leads us to this hyper-parameter tuning experiment, in which the coreset sampling ratio $r$ and number of neighbors $k$ is varied and averaged over five different seeds. The goal of this experiment is to find parameters $r$ and $k$, such that the distribution of anomaly scores shows a right-skew (positive skewness) and a low mean of the anomaly scores.

The chosen approach to this HPO-experiment is a simple *grid search* over $r \in [0.01, 0.05]$, $k \in [3, 4, 5, 6, 7, 8, 9]$ and seeds $\{0, 42, 200, 138, 159\}$. We will first evaluate the distribution properties of *PatchCore* on the ICM_1_4-validation set, based on a heatmap of mean scores, skewness and a line chart of prediction scores, similar to previously used prediction score results. The most promising parameters will also be used to run models on the test set.

### 5.3.1 Validation set evaluation

**Distribution properties.** First, we will look at the distribution of the prediction scores, averaged over the five different seeds. For the representation of the mean and skew of prediction scores over these different parameters, a heatmap was chosen.




(a) Mean prediction scores

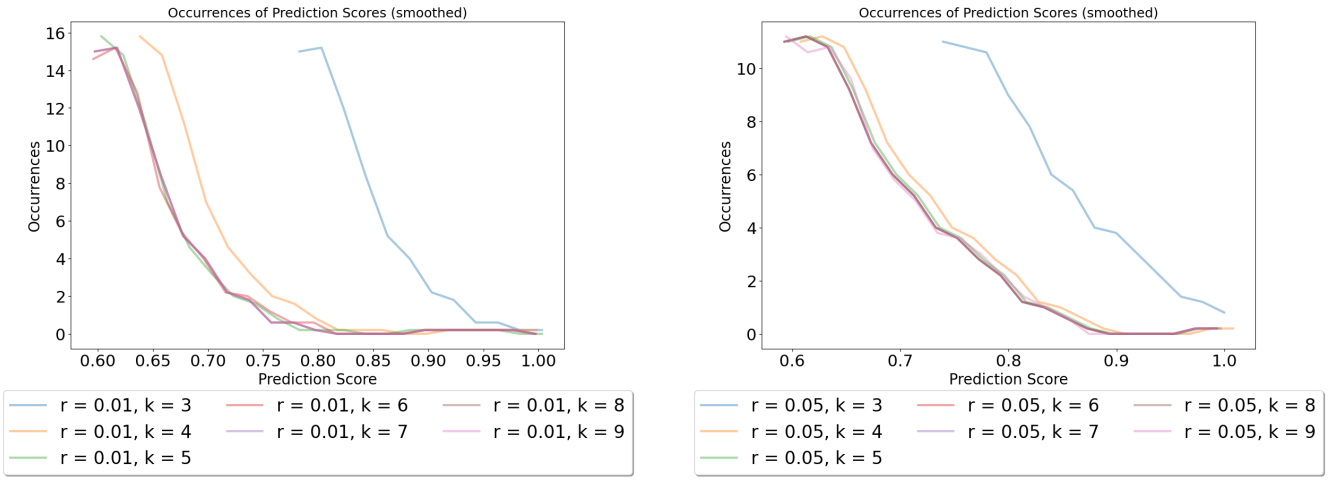
(b) Skewness

Figure 16: This figure shows the mean of prediction scores 16a and skewness values 16b for different configurations. A higher number of neighbors $k$ lowers the prediction scores mean. An increase in the coreset subsampling ratio $r$ does not show a similar trend. As is visible from the figure, there is no change in mean prediction scores or skewness from $k = 7$ to $k = 9$ number of neighbors. This could indicate that increasing the number of neighbors beyond $k = 7$ does not change model performance.

An optimal configuration does not only have a low mean of prediction scores, but should

also have a positive skewness. Judging from these two heatmaps, models with a coreset subsampling ratio of $r = 0.01$ and between $k = 5$ and $k = 7$ fulfill those criteria.

**Prediction score results.** In addition to the heatmaps themselves, we will take a look at the occurrences of prediction scores. The bar plot has been changed to a line chart to allow for multiple different parameters, but can be read in the same way, i.e. a higher number of occurrences means that an anomaly score appears more often.




(a) $r = 0.01$ (b) $r = 0.05$

Figure 17: This figure shows the prediction score distribution for parameters $k \in [3, 4, 5, 6, 7, 8, 9]$ combined with $r = 0.01$ 17a and $r = 0.05$ 17b with a smoothing window of 5. Both figures show a slightly more visible right-skew for a higher value for $k$, reflecting the findings from the presented heatmap 16b. $r = 0.01$ seems to lead to more promising distributions with lower anomaly scores, on average.

Generally, a lower number of neighbors $k$ is favored when increasing the number does not improve performance anymore. The reason for this is the increased memory usage when comparing a higher number of patches. We are thus looking for a saddle point or a point where the performance stops increasing.

Based on these criteria, parameters $k \in [4, 5, 6]$ and $r \in [0.01, 0.05]$ will be selected as candidate best parameters. The values for $k$ were chosen because of their resulting means and skews for both values of $r$ and because there is some degree of variation between the results of these these parameters. Both values of $r$ will be tested on because of the small difference in means and the unknown performance on the test set. Additionally, choosing only a single subsampling ratio based only on skewness in this stage would lead to a less representative hyper-parameter optimization.

### 5.3.2 Test set evaluation

To further critically evaluate our candidate best parameters, models with these parameters will be run on the ICM_1_4-test set containing all three anomalies 7a, 7b and 7c. Their prediction scores will be averaged over the results from five different seeds, from which the performance metrics will then be calculated.

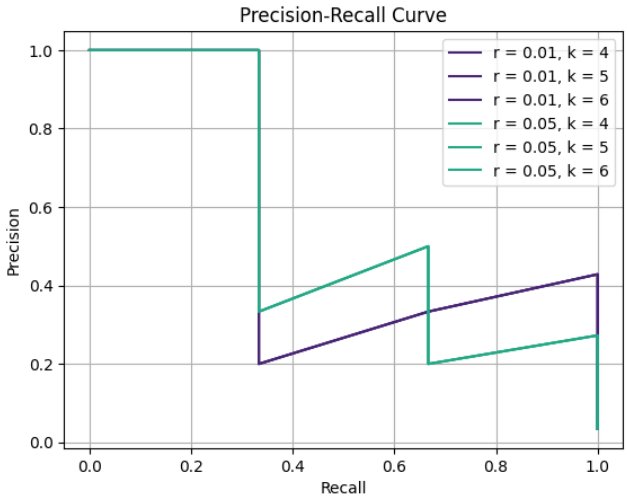
**Performance metrics.** To evaluate the performance of the *PatchCore*-model instances with different parameters, the same metrics as before will be used to compare the candidate best parameter combinations.

| **Parameters** | r = 0.01, k = 4 | r = 0.01, k = 5 | r = 0.01, k = 6 |
|---|---|---|---|
| AUROC ↑ | 0.97 | 0.97 | 0.97 |
| Recall ↑ | 0.33 | 0.33 | 0.33 |
| Error ↓ | 0.07 | 0.05 | 0.05 |
| Misclassifications ↓ | 6 | 4 | 4 |

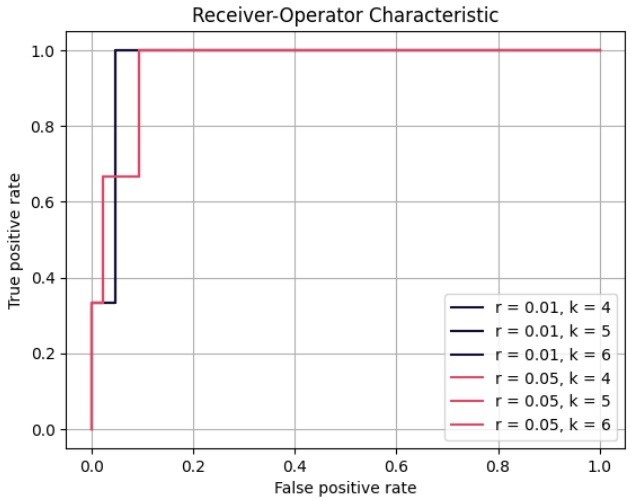
| **Parameters** | r = 0.05, k = 4 | r = 0.05, k = 5 | r = 0.05, k = 6 |
|---|---|---|---|
| AUROC ↑ | 0.96 | 0.96 | 0.96 |
| Recall ↑ | 1.0 | 0.67 | 0.67 |
| Error ↓ | 0.09 | 0.09 | 0.07 |
| Misclassifications ↓ | 8 | 8 | 6 |

Table 4: These two tables show the performance metrics results for the candidate best parameters $r \in [0.01, 0.05]$ and $k \in [4, 5, 6]$ at an anomaly threshold of $t = 0.7$. There is a lower recall for a coreset subsampling ratio of $r = 0.01$ than at $r = 0.05$. This difference could be related to a lower sensitivity, which is reflected in the mean prediction scores from figure 16a. As the number of neighbors $k$ increases, there is a slight decrease in the number of misclassifications and thus the error rate. For $r = 0.05$, an increase in the number of neighbors also corresponds to a decrease in recall.

Due to the small differences in AUROC and error, it is difficult to induce a 'best' set of parameters. The combination of $r = 0.05$ and $k = 4$ achieves the highest recall, whereas the combination of $r = 0.01$ and $k = 5$ has the lowest error rate and the highest AUROC at a threshold of $t = 0.7$. If this threshold is lowered to $t = 0.6$, all parameters get a perfect recall and the combinations $r = 0.01$ with $k = 5$ and $k = 6$ both get the lowest error rate at approximately 20%. For a more exact overview of the relation between the anomaly threshold and the performance metrics, figure 18 shows the precision-recall and receiver-operator (ROC) curves.




(a) Precision-Recall Curve




(b) Receiver-Operator Curve

Figure 18: This figure shows the precision-recall curve 18a and receiver-operator characteristics (ROC) curves 18b for *PatchCore* run on the ICM_1_4-dataset. Both parameter combinations with $r = 0.01$ and $r = 0.05$ have a perfect precision until a recall of 0.33 (one out of three correctly classified anomalies) and then drop to a precision of 0.33 and 0.2, respectively. All parameter combinations with $r = 0.05$ have an increasing precision between a recall of 0.33 and 0.67. However, at a perfect recall, the precision of the models run with $r = 0.05$ is lower than that of those run with $r = 0.01$. This is reflected in the ROC-curve, in which models with $r = 0.01$ have a slightly lower false positive rate to get a perfect true positive rate.

The addition of these two curves offers insight into the exact trade-off between between detecting anomalies and misclassifying normal samples as anomalous. This visualization offers the most general image of the impact of different parameters. In the rest of this paper, *PatchCore* will be run with $r = 0.01$ and $k = 0.05$. The reason for the choice of this parameter combination is the relatively higher precision at a perfect recall. A perfect recall is more important in this use case because of the limited number of anomalies.

We are aware that these results offer a limited generalization to other cases due to the single dataset that the different models were run on. As a consequence of limited computational resource and time constraints, it was chosen not to explore the influence of parameter-tuning on the other datasets.

## 5.4 Comparative experiments on the entire dataset

In this section, the results on the remaining datasets will be summarized in a more condensed fashion than in the proof of concept. This section will offer an overview of the AUROC scores for the two models on all subsets of the data, as well as a qualitative overview of the anomaly maps of *PatchCore* and *GLASS*.[13]

**Performance metrics.** Table 5 shows a full overview of the AUROC scores.

[13]For full reference of the prediction score results, please consult appendix D.1.

| Method | PatchCore | GLASS |
|---|---|---|
| ICM_1_4 | 0.97 | **0.99** |
| SLT_4_6_VIRM_4 | **0.53** | 0.43 |
| SNG_3_4 | 0.95 | **1.00** |
| DDZ_4_6 | **1.00** | 0.92 |
| VIRM_4 | 0.86 | **0.88** |
| Object average[14] | **0.86** | 0.84 |
| **Weighted average**[15] | **0.83** | 0.81 |

Table 5: This table shows the AUROC scores for *PatchCore* and *GLASS* models on all different subsets of the data. On average, *PatchCore* outperforms *GLASS* in terms of AUROC. AUROC scores for individual datasets differ only slightly between the two models.

From the AUROC table, there seems to be no clear difference between *PatchCore* and *GLASS*. Both models seem to score around the same, but there are some qualitative differences. This is why the qualitative overview is used: to offer insight in what types of anomalies are detected well by each model.

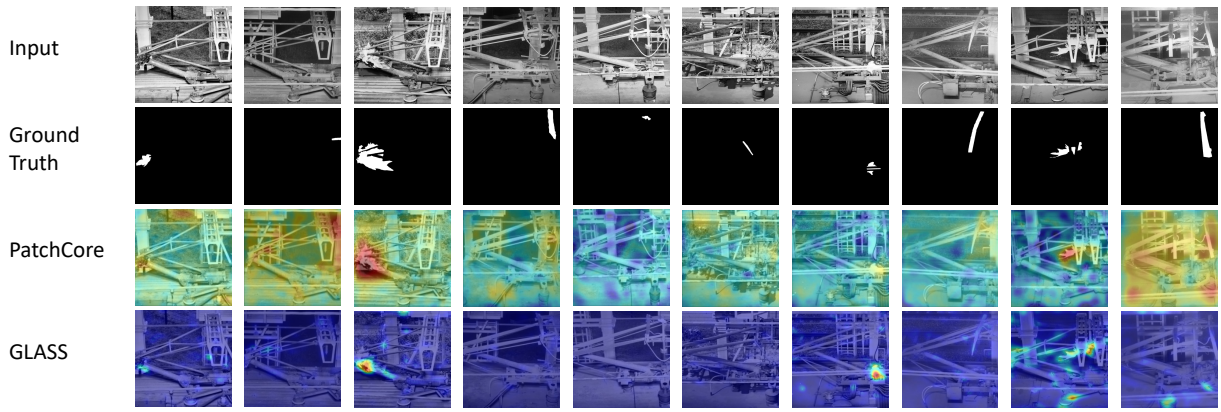



Figure 19: Qualitative overview of all different anomalies present in the dataset, alongside their ground truth masks and anomaly maps from *PatchCore* and *GLASS*. In general, *PatchCore* is sensitive to more areas in the input image, but is also capable of picking out smaller, logical anomalies. *GLASS*, on the other hand, shows overall low sensitivity due to the validation set selection criteria that have been chosen. When it does detect an anomaly, it localizes it in a precise fashion.

*PatchCore* scores well on both structural anomalies and is able to detect some of the structural anomalies, whereas *GLASS* only detects the logical anomalies. As stated before, *GLASS* anomaly maps contain less 'noise' (or areas that have been flagged anomalous, whilst being normal).

[14]The object average is the unweighted average of all different datasets.
[15]The weighted average is calculated by weighting based on the size of the test set.

# 6 Discussion

## 6.1 Summary of findings

The aim of this research was to establish the difference in anomaly detection performance between *PatchCore* and *GLASS* in the context of train pantographs. In the results section, a distinction has been made between logical and structural anomalies. Logical anomalies, such as objects or birds around the pantographs, were the main focus of our research, whilst structural anomalies, such as bent or damaged parts, were of secondary importance.

From the presented literature, *GLASS* was expected to outperform *PatchCore* because of its higher AUROC scores. In contrast to earlier findings on benchmark datasets, this thesis found varying results with regard to model performance. Looking purely at performance metrics, such as AUROC, recall and error rate, *GLASS* is outperformed by *PatchCore* in a configuration without anomalies in the validation set. It does, however, come to par with *PatchCore* when introducing an anomaly to the validation set even if this anomaly is a small, structural anomaly.

On the one hand, differences between the original paper and this thesis are related to more variations in background and lighting. On the other hand, the introduction of a separate validation and test set also played a role in performance results. The use of a validation set and an unseen test lead to a stricter evaluation criterion for a model, but does render results more representative in the case of real-life situation, in which the model will also encounter unseen situations.

*GLASS* generally proved less sensitive to variations in background with a decrease in exact anomaly localization as a trade-off, compared to *PatchCore*. *PatchCore* was generally triggered in the case of logical anomalies and even some structural anomalies. However, it was also prone to misclassifications with differences in background, lighting or weather conditions.

On a prediction score-level, both models showed a clear right-skew for normal samples, especially as the input data grew, in which case the separation margin between normal and anomalous samples also grew. This skewness diminished with smaller dataset sizes.

## 6.2 Limitations

### 6.2.1 Lack of anomalies & limited intermediate evaluation

A major source of uncertainty remains the problem of selecting a model based on a validation set without any anomalies. The design of the *GLASS*-model, required choosing a 'best' model out of several epochs. The chosen heuristics for *GLASS* (12) and (13) proved to be a limitation to its performance. In the case of a validation set without anomalies, sensitivity to anomalies was particularly low in favor of low average prediction scores. This low sensitivity influenced performance results for *GLASS*, even though it was not directly related to the architecture of the model itself. For the specific use case of detecting anomalies and a relative low cost of false positives, the choice of another model selection heuristic would be preferable. Comparing alternative heuristics could form as a basis of future research.

The addition of synthetic anomalies to the validation set was out of scope for this thesis, but seems to be promising in cases with few anomalies. This addition can facilitate intermediate model evaluation. When selecting synthetic anomalies, it is important to offer a supporting explanation on why a specific strategy was chosen. One of the reasons why the addition of synthetic anomalies

remained out of scope for this thesis was finding an adequate base of choosing some type of synthetic anomaly over another.

### 6.2.2 Model-specific adjustments

However, *GLASS* sometimes also underperformed compared to *PatchCore* in cases where the validation set did include an anomaly. Due to its longer training times of around 40 hours for a full training (500 epochs) on the ICM_1_4-dataset, hyper-parameter optimization was not considered for *GLASS* in this thesis. There are nonetheless many parameters that can be tweaked in the case of *GLASS* and more degrees of freedom to fine-tune the model than with *PatchCore*. Existing literature does not provide a full overview of these parameters most likely because of the relatively recent publication of the *GLASS*-paper (July 2024). In this thesis, it was thus decided to use the default settings, which were used in the paper and can be found under C. Exploration of hyper-parameter tuning, such as the introduction of random augmentations and different noise levels to the input image, could be explored in further research.

In the original configuration for *GLASS*, foreground masks of normal samples were used. According to the author of *GLASS*, leaving out these masks would not significantly reduce the accuracy of detection [CZ25b]. However, from 13, it became clear that there is some degree of background sensitivity. The most probable cause for this discrepancy is the nature of the used benchmark datasets, which have a uniform background per class. This uniform background acts similarly to a foreground mask. Creating foreground masks for our specific problem would create a new problem in and of itself, namely segmenting the pantograph, which leads us to our next point.

Segmentation was not used in either one of the compared models. The addition of segmentation adds a layer of complexity to image interpretation because the model needs some semantic knowledge about the objects in the image. One promising method, *CSAD*, uses Meta's *SAM* (Segment Anything) segmentation model and receives high performance in the case of logical and structural anomalies [HL24, RGH+24]. This method was not explored in this thesis due to limited code documentation compared to *PatchCore* and *GLASS* and possible patent issues due its indirect use of models recently patented by MVTec. In a business context, the use of these models without a license might cause problems. However, in a purely academical context it is worth exploring this method on real-life data. Furthermore, taking inspiration from a segmentation-based model, future research could focus on incorporating this type of segmentation in anomaly detection.

### 6.2.3 Pre-processing & data filtering

Additional improvements could be made in the area of pre-processing and data filtering. Data splits were currently only made based on fleet number. More elaborate splits could be made to ensure a representative distribution of background variations, weather and lighting conditions. In order to do this, one would either have to train a classifier or go over the data manually. However, in combination with the timestamps in the metadata, it would also be possible to filter times and even weather conditions if combined with public weather data.

Unfortunately, only half of the total training data could be explored within the time frame of this thesis. The addition of top-view data could serve as a an extra form of validation to model performance. Ideally, a model would combine the knowledge of top-view and side-view images to conclude whether a given image could be considered to be anomalous. We expect that this addition

could serve as a sanity check to potential false positives and a reinforcement to the localization of anomalies.

Another variable that has not been changed throughout this thesis is the image size. For both models, the input image size was kept at 256x256 to reduce computational strain on the models. Increasing the size of the input images, which were originally 10-20x bigger, could increase model performance, especially on structural anomalies.

# 7 Conclusion

This thesis has put into perspective the performance of *GLASS* and *PatchCore* by evaluating the models on basis of their prediction score, qualitative and performance metrics results. The inclusion of qualitative results showed not only *that* a model can correctly detect an anomaly, but also *why*, i.e. because of what parts of the image, it considered that image anomalous.

*PatchCore* is better used for small datasets (between 100 and 400 images) and in situations where short training times are preferable.[16] In cases with little to no background or lighting variation, *PatchCore* will achieve high performance for both structural and logical anomalies, leading to a workload reduction. Cases with more variation will incur more false positives. Increasingly bigger datasets will require more memory because of the design of *PatchCore*. The model will not see much improvement when more data are provided.

*GLASS*, in contrast, does see improvement with the introduction of more data. *GLASS* has longer training times at around 100-140x the time that *PatchCore* needs. *GLASS* has slightly shorter inference times than *PatchCore* , which could play a role in live deployment. In general, it is advisable to tweak the parameters for a *GLASS* model to achieve the desired sensitivity. In the case of this thesis, not optimizing these parameters led to low sensitivity.

In conclusion, both models are a decent, though not perfect fit for the problem. As stated earlier, the addition of segmentation and foreground masks could play a role in increasing performance, as well as more elaborate hyperparameter optimization. For big logical anomalies, both models can be used, whereas for structural anomalies, another method, such as *CSAD* could possibly give better results. For fast testing and deployment, *PatchCore* is the model of choice. Ultimately, the choice between *GLASS* and *PatchCore* should be informed by the specific demands and constraints of the dataset and deployment scenario. Further exploration and optimization could enhance the applicability of both models and effectiveness in anomaly detection tasks.

[16] Note that for datasets that are too small ($< 150$ images, aprroximately) results the model will also suffer from performance issues.

# References


[BFSS19] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad — a comprehensive real-world dataset for unsupervised anomaly detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9584–9592, 2019.

[BFSS20] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020.

[BKG23] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*, pages 353–374. Springer International Publishing, Cham, 2023.

[BLN23] Samah Saeed Baraheem, Trung-Nghia Le, and Tam V. Nguyen. Image synthesis: a review of methods, datasets, evaluation metrics, and future outlook. *Artificial Intelligence Review*, 56(10):10813–10865, Oct 2023.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), July 2009.

[CCX+24] Yichi Chen, Bin Chen, Weizhi Xian, Junjie Wang, Yao Huang, and Min Chen. Lgfdr: local and global feature denoising reconstruction for unsupervised anomaly detection. *Vis. Comput.*, 40(12):8881–8894, May 2024.

[CLLZ24] Qiyu Chen, Huiyuan Luo, Chengkan Lv, and Zhengtao Zhang. A unified anomaly synthesis strategy with gradient ascent for industrial anomaly detection and localization, 2024.

[CMK+13] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild, 2013.

[CVC23] Gabriela Csurka, Riccardo Volpi, and Boris Chidlovskii. Semantic image segmentation: Two decades of research, 2023.

[CZ25a] Qiyu Chen and Durong Zheng. cqylunlun/glass. https://github.com/cqylunlun/GLASS/issues/6, 3 2025. [Accessed 28-05-2025].

[CZ25b] Qiyu Chen and Durong Zheng. cqylunlun/glass. https://github.com/cqylunlun/GLASS/issues/18, 3 2025. [Accessed 01-07-2025].

[DSLA20] Thomas Defard, Aleksandr Setkov, Angelique Loesch, and Romaric Audigier. Padim: a patch distribution modeling framework for anomaly detection and localization, 2020.

[FWR+17] Alireza Fathi, Zbigniew Wojna, Vivek Rathod, Peng Wang, Hyun Oh Song, Sergio Guadarrama, and Kevin P. Murphy. Semantic instance segmentation via deep metric learning, 2017.

[GPB+20] Riccardo Gasparini, Stefano Pini, Guido Borghi, Giuseppe Scaglione, Simone Calderara, Eugenio Fedeli, and Rita Cucchiara. Anomaly detection for vision-based railway inspection. In *EDCC Workshops*, 2020.

[HL24] Yu-Hsuan Hsieh and Shang-Hong Lai. Csad: Unsupervised component segmentation for logical anomaly detection, 2024.

[ima] ImageNet — image-net.org. https://www.image-net.org/about.php. [Accessed 14-05-2025].

[IZZE18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[JJB+21] Stepan Jezek, Martin Jonak, Radim Burget, Pavel Dvorak, and Milos Skotak. Deep learning-based defect detection of metal parts: evaluating current methods in complex conditions. In *2021 13th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pages 66–71, 2021.

[KAC+24] Soopil Kim, Sion An, Philip Chikontwe, Myeongkyun Kang, Ehsan Adeli, Kilian M. Pohl, and Sang Hyun Park. Few shot part segmentation reveals compositional logic for industrial anomaly detection, 2024.

[KC23] Gyu-Il Kim and Kyungyong Chung. Patchcore-based anomaly detection using major object segmentation. *International Journal on Advanced Science, Engineering and Information Technology*, 13(4):1480–1485, Aug 2023.

[KH24a] Rahima Khanam and Muhammad Hussain. What is yolov5: A deep look into the internal features of the popular object detector, 2024.

[KH24b] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements, 2024.

[KHG+18] Alexander Kirillov, Kaiming He, Ross B. Girshick, Carsten Rother, and Piotr Dollár. Panoptic segmentation. *CoRR*, abs/1801.00868, 2018.

[KvdMKL16] Wouter M. Kouw, Laurens J.P. van der Maaten, Jesse H. Krijthe, and Marco Loog. Feature-level domain adaptation. *Journal of Machine Learning Research*, 17(171):1–32, 2016.

[LGRN11] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Commun. ACM*, 54(10):95–103, October 2011.

[LLS22] Sungwook Lee, Seunghyun Lee, and Byung Cheol Song. Cfa: Coupled-hypersphere-based feature adaptation for target-oriented anomaly localization, 2022.

[mat] mattmcinnes. NCasT4_v3 size series - Azure Virtual Machines — learn.microsoft.com. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/gpu-accelerated/ncast4v3-series?tabs=sizebasic. [Accessed 02-06-2025].

[Mel13] Francisco Melo. *Area under the ROC Curve*, pages 38–39. Springer New York, New York, NY, 2013.

[MKM25] Helia Mohamadi, Mohammad Ali Keyvanrad, and Mohammad Reza Mohammadi. Feature based methods in domain adaptation for object detection: A review paper, 2025.

[mvt] Anomaly Detection and Global Context Anomaly Detection [HALCON Operator Reference / Version 23.05.0.0] — mvtec.com. https://www.mvtec.com/doc/halcon/2305/en/toc_deeplearning_anomalydetection.html. [Accessed 20-02-2025].

[NAN12] Mark S. Nixon, Alberto S. Aguado, and Mark S. Nixon. *Feature extraction image processing for computer vision*. Academic, Oxford, 3rd ed. edition, 2012.

[nsD] Damage repair — Train maintenance — About NS — NS — ns.nl. https://www.ns.nl/en/about-ns/train-maintenance/damage-repair.html. [Accessed 13-02-2025].

[OGS+09] Emilio Soria Olivas, Jose David Martin Guerrero, Marcelino Martinez Sober, Jose Rafael Magdalena Benedito, and Antonio Jose Serrano Lopez. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2009.

[Pei15] Jonathan W. Peirce. Understanding mid-level representations in visual processing. *Journal of Vision*, 15(7):5–5, 06 2015.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[RGH+24] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024.

[RPZ+22] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection, 2022.

[SDSS16] Ljiljana Stojanovic, Marko Dinic, Nenad Stojanovic, and Aleksandar Stojadinovic. Big-data-driven anomaly detection in industry (4.0): An approach and a case study. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1647–1652, 2016.

[SS01] George Stockman and Linda G. Shapiro. *Computer Vision*. Prentice Hall PTR, USA, 1st edition, 2001.

[SS18] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach, 2018.

[SZG+19] Samarth Sinha, Han Zhang, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, and Augustus Odena. Small-gan: Speeding up gan training using core-sets, 2019.

[Tin17] Kai Ming Ting. *Precision and Recall*, pages 990–991. Springer US, Boston, MA, 2017.

[tre] NS zet camera's langs het spoor in voor efficiënter onderhoud - Treinenweb — treinenweb.nl. https://www.treinenweb.nl/nieuws/10959/ns-zet-camera-s-langs-het-spoor-in-voor-efficienter-onderhoud.html. [Accessed 13-02-2025].

[TS09] Lisa A. Torrey and Jude W. Shavlik. Chapter 11 transfer learning. 2009.

[Ult] Ultralytics. YOLO11 vs YOLOv5: A Detailed Comparison — docs.ultralytics.com. https://docs.ultralytics.com/compare/yolo11-vs-yolov5/#ultralytics-yolo11. [Accessed 18-06-2025].

[Wu18] Jiqin Wu. Chapter 2 - pantograph. In Jiqin Wu, editor, *Pantograph and Contact Line System*, High-Speed Railway, pages 27–71. Academic Press, 2018.

[WWG+19] Zuxuan Wu, Xin Wang, Joseph E. Gonzalez, Tom Goldstein, and Larry S. Davis. Ace: Adapting to changing environments for semantic segmentation, 2019.

[YLW+24] Hang Yao, Ming Liu, Haolin Wang, Zhicun Yin, Zifei Yan, Xiaopeng Hong, and Wangmeng Zuo. Glad: Towards better reconstruction with global and local adaptive diffusion models for unsupervised anomaly detection, 2024.

[YXQS22] Jie Yang, Ruijie Xu, Zhiquan Qi, and Yong Shi. Visual anomaly detection for images: A systematic survey. *Procedia Computer Science*, 199:471–478, 2022. The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 2021): Developing Global Digital Economy after COVID-19.

[ZAA+21] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Naveed Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *CoRR*, abs/2104.11892, 2021.

[ZJP+22] Yang Zou, Jongheon Jeong, Latha Pemula, Dongqing Zhang, and Onkar Dabeer. Spot-the-difference self-supervised pre-training for anomaly detection and segmentation, 2022.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

[ZXY+20] Kang Zhou, Yuting Xiao, Jianlong Yang, Jun Cheng, Wen Liu, Weixin Luo, Zaiwang Gu, Jiang Liu, and Shenghua Gao. Encoding structure-texture relation with p-net for anomaly detection in retinal images, 2020.

[ZZ09] Ethan Zhang and Yi Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009.

# A Dataset distribution

## A.1 Original dataset

| Pantograph head (type) | Number of occurrences | Frequency |
|---|---|---|
| ICM_1_4 | | |
| ICM I | 410 | 7.4% |
| ICM IV | 1916[17] | 34.5% |
| **Subtotal** | **2326** | **41.8%** |
| SLT_4_6_VIRM_4 | | |
| SLT IV | 410 | 7.4% |
| SLT VI | 411 | 7.4% |
| VIRM IV[18] | 119 | 2.1% |
| **Subtotal** | **940** | **16.9%** |
| VIRM_6 | | |
| VIRM VI | 399 | 7.2% |
| **Subtotal** | **399** | **7.2%** |
| DDZ_4_6 | | |
| DDZ IV | 405 | 7.3% |
| DDZ VI | 401 | 7.2% |
| **Subtotal** | **806** | **14.5%** |
| SNG_3_4 | | |
| SNG III | 412 | 7.4% |
| SLT IV | 406 | 7.3% |
| **Subtotal** | **818** | **14.7%** |
| VIRM_4 | | |
| VIRM IV[19] | 272 | 4.9% |
| **Subtotal** | **272** | **4.9%** |
| **Total** | **5561** | |

Table 6: Data distribution of entire dataset

[17]Additional images were added for the Proof of Concept (4.3.2), leading to a higher number of images than with other types of pantograph heads.

[18]Train series 9547 - 9597

[19]Train series 9501 - 9546 and 9401 - 9481

## A.2 Target dataset

Dataset excluding top-`side`.

| Pantograph head (type) | Number of occurrences | Frequency |
|---|---|---|
| `ICM_1_4` | | |
| ICM 1 | 175 | 6.8% |
| ICM 4 | 600 | 23.4% |
| **Subtotal** | **775** | **30.2%** |
| `SLT_4_6_VIRM_4` | | |
| SLT IV | 203 | 7.9% |
| SLT VI | 202 | 7.9% |
| VIRM IV | 94 | 3.7% |
| **Subtotal** | **499** | **19.4%** |
| `VIRM_6`[20] | | |
| VIRM VI | 187 | 7.3% |
| **Subtotal** | **399** | **15.5%** |
| `DDZ_4_6` | | |
| DDZ IV | 197 | 7.7% |
| DDZ VI | 199 | 7.8% |
| **Subtotal** | **396** | **15.4%** |
| `SNG_3_4` | | |
| SNG III | 202 | 7.9% |
| SNG IV | 200 | 7.8% |
| **Subtotal** | **402** | **15.7%** |
| `VIRM_4` | | |
| VIRM IV | 96 | 3.7% |
| **Subtotal** | **96** | **3.7%** |
| **Total** | **2567** | |

Table 7: Data distribution of target dataset

[20]This subset does not contain any anomalies in the target data.

# B Binary classifier

The binary classifier was trained on images with two classes `lt` (<-orientation) and `gt` (>-orientation). The distribution of the dataset is as follows:

- `lt`: 804 (562 train, 122 test, 120 validation)
- `gt`: 892 (624 train, 135 test, 133 validation)

A YOLOv11-model was trained on the images over 20 epochs, with 8 dataloaders.

# C Model information

## C.1 PatchCore

### C.1.1 Configuration

- backbone="wide_resnet50_2"
- layers=["layer2", "layer3"]
- coreset_sampling_ratio=0.01
- num_neighbors=5
- extensions=None
- train_batch_size=32
- eval_batch_size=32
- num_workers=8
- task=classification
- image_size=[256,256]
- transform=None
- train_transform=None
- eval_transform=None

### C.1.2 Memory bank reduction algorithm

**Algorithm 1:** *PatchCore* memory bank.

**Input:** Pretrained $\phi$, hierarchies $j$, nominal data $\mathcal{X}_N$, stride $s$, patchsize $p$, coreset target $l$, random linear projection $\psi$.

**Output:** Patch-level Memory bank $\mathcal{M}$.

**Algorithm:**

$\mathcal{M} \leftarrow \{\}$

**for** $x_i \in \mathcal{X}_N$ **do**

$\quad \mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{P}_{s,p}(\phi_j(x_i))$

**end**

/* Apply greedy coreset selection. */

$\mathcal{M}_C \leftarrow \{\}$

**for** $i \in [0, ..., l-1]$ **do**

$\quad m_i \leftarrow \underset{m \in \mathcal{M} - \mathcal{M}_C}{\operatorname{argmax}} \min_{n \in \mathcal{M}_C} \|\psi(m) - \psi(n)\|_2$

$\quad \mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{m_i\}$

**end**

$\mathcal{M} \leftarrow \mathcal{M}_C$

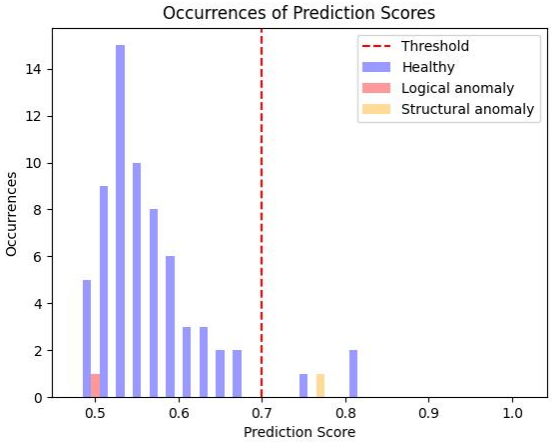
## C.2 GLASS

### C.2.1 Configuration

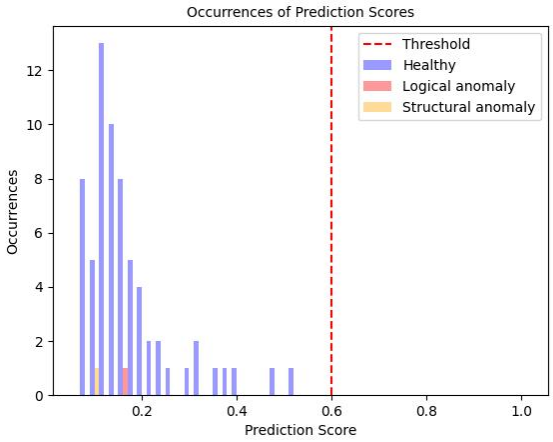
- backbone="wide_resnet50"
- layers=["layer2", "layer3"]
- patchsize=3
- image_size=[256,256]
- dsc_layers=2
- dsc_hidden=1024
- pre_proj=1
- mining=1
- noise=0.015
- radius=0.6
- p=0.5
- step=20
- limit=392
- distribution=2
- mean=0.5
- std=0.1
- fg=0
- rand_aug=1
- batch_size=8
- resize=288

# D Results on the entire dataset

## D.1 Prediction scores

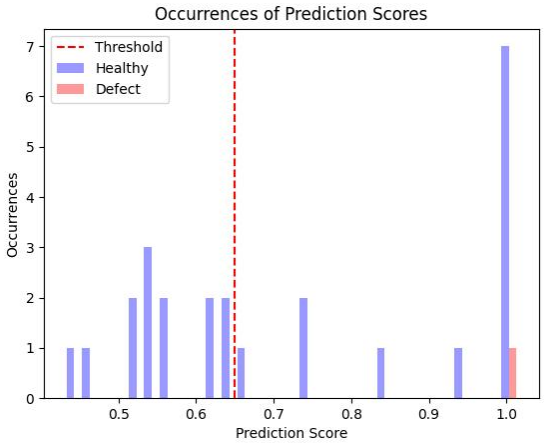



(a) *PatchCore* on SLT_4_6_VIRM_4

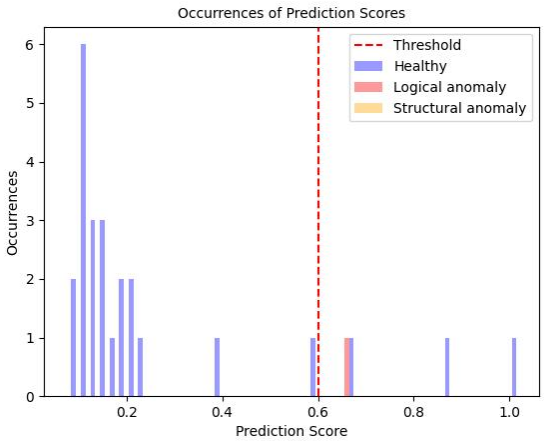



(b) *GLASS* on SLT_4_6_VIRM_4

Figure 20: On the left, 20a shows the prediction score results for best seed 200. On the right, 20b shows the results after the best combined score was obtained at epoch 454. *PatchCore* is unable to find the logical anomaly at anomaly threshold $t = 0.7$, but does find the structural anomaly at $2.91\sigma$ deviations from the mean of $\mu = 0.57$. There are, however, two normal samples with a higher anomaly score than the structural anomaly. *GLASS*, on the other hand, can find neither anomaly at an acceptable threshold.

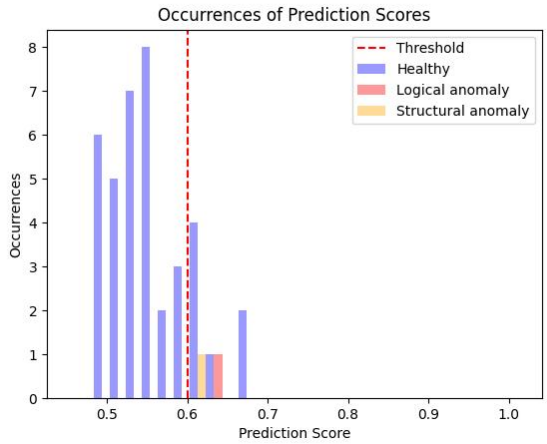



(a) *PatchCore* on VIRM_4

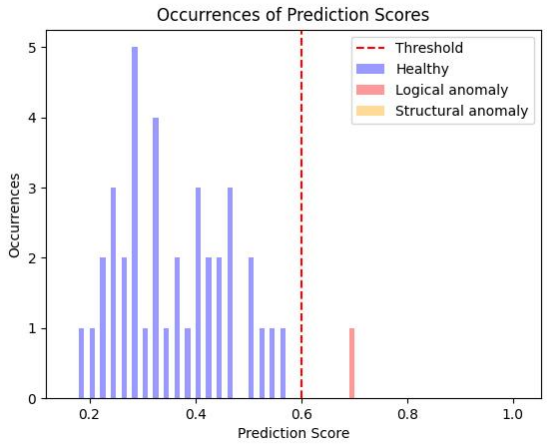



(b) *GLASS* on VIRM_4

Figure 21: This *PatchCore* model in 21a instance was run with seed 42, whereas the *GLASS* model instance in 21b is was picked as a best model after 116 epochs. *PatchCore* shows insignificant results, whereas *GLASS* still shows a right skew and some degree of separation.

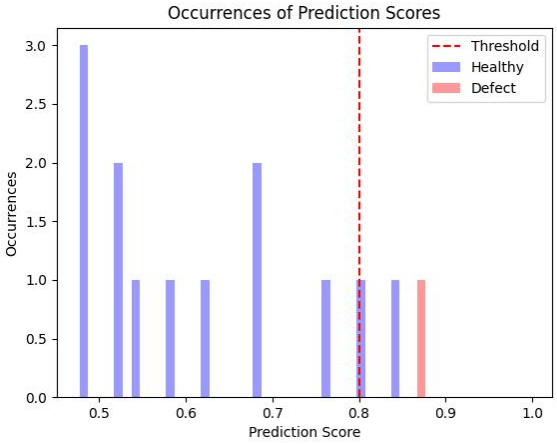



(a) *PatchCore* on SNG_3_4

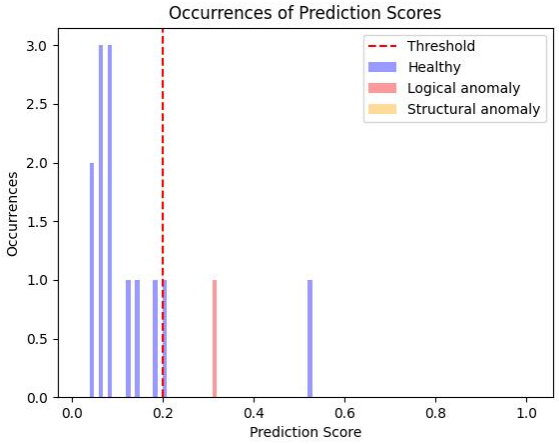



(b) *GLASS* on SNG_3_4

Figure 22: The left figure 22a shows the prediction score results for a *PatchCore*-model with seed 0. The right figure 22b displays the results at epoch 366, which resulted in the best combined score. *GLASS* was provided with a structurral anomaly during testing and shows a stronger separation margin than *PatchCore*. *PatchCore* does detect both anomalies, however, which is not possible in the case of *GLASS*.




(a) *PatchCore* on DDZ_4_6

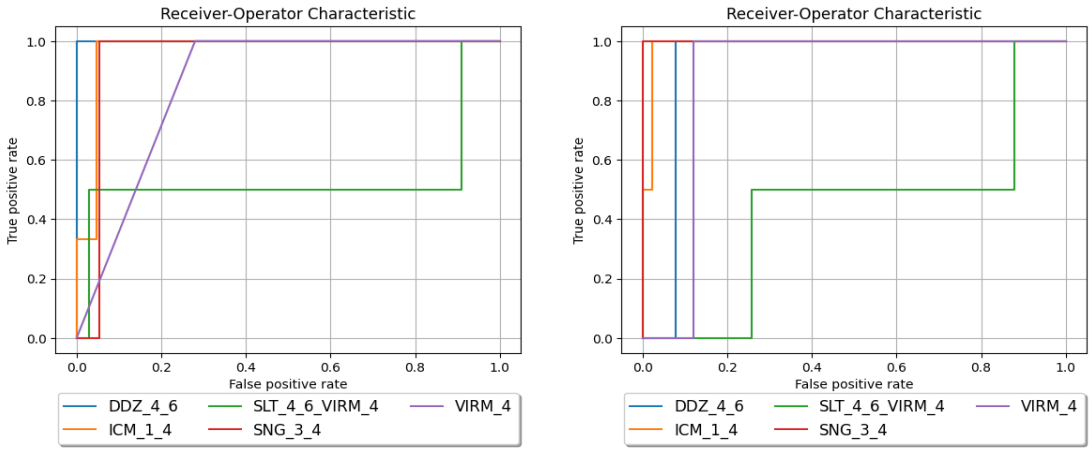



(b) *GLASS* on DDZ_4_6

Figure 23: In the left figure 23a, we find the resutls for a *PatchCore*-model run with seed 42 (best out of five). The right figure 23b displays the prediction score results for *GLASS* at epoch 150 (best out of 500 epochs). Both models detect the anomaly, but the separation margin is minimal in the case of *PatchCore* and the attributed score is relatively low for *GLASS*.

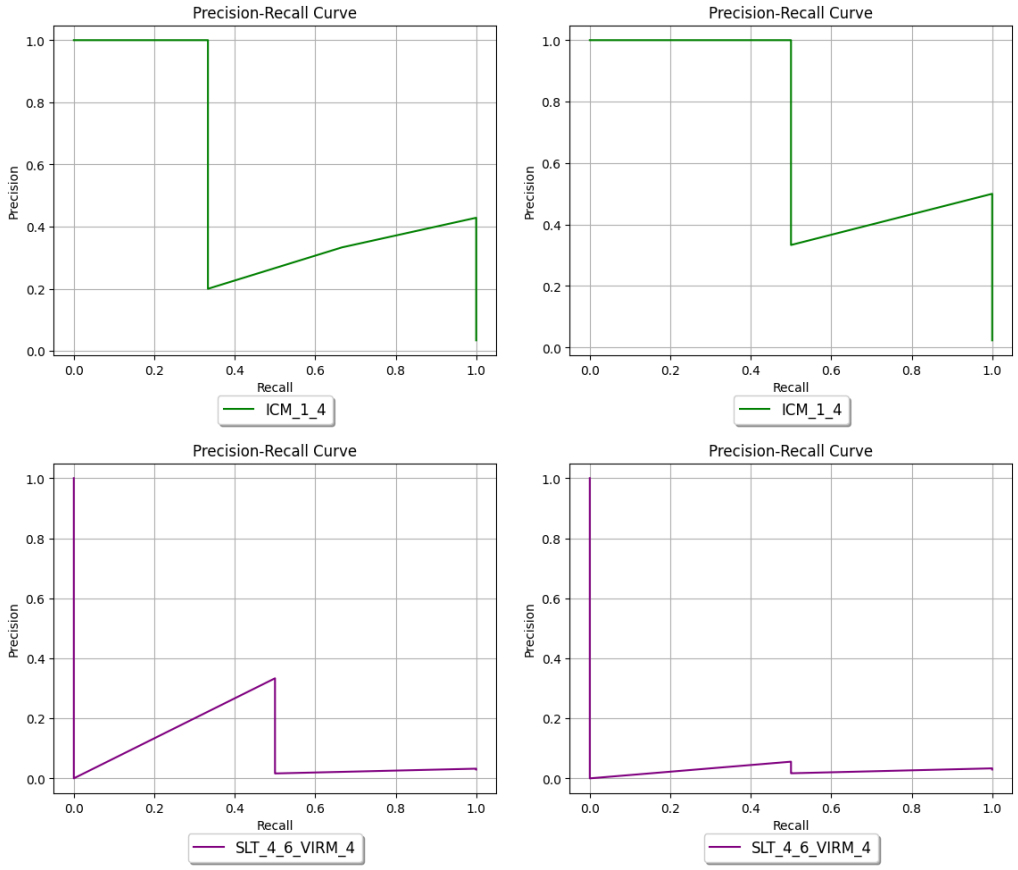
## D.2 Performance metrics

This is an overview of the ROC and Precision-Recall curves

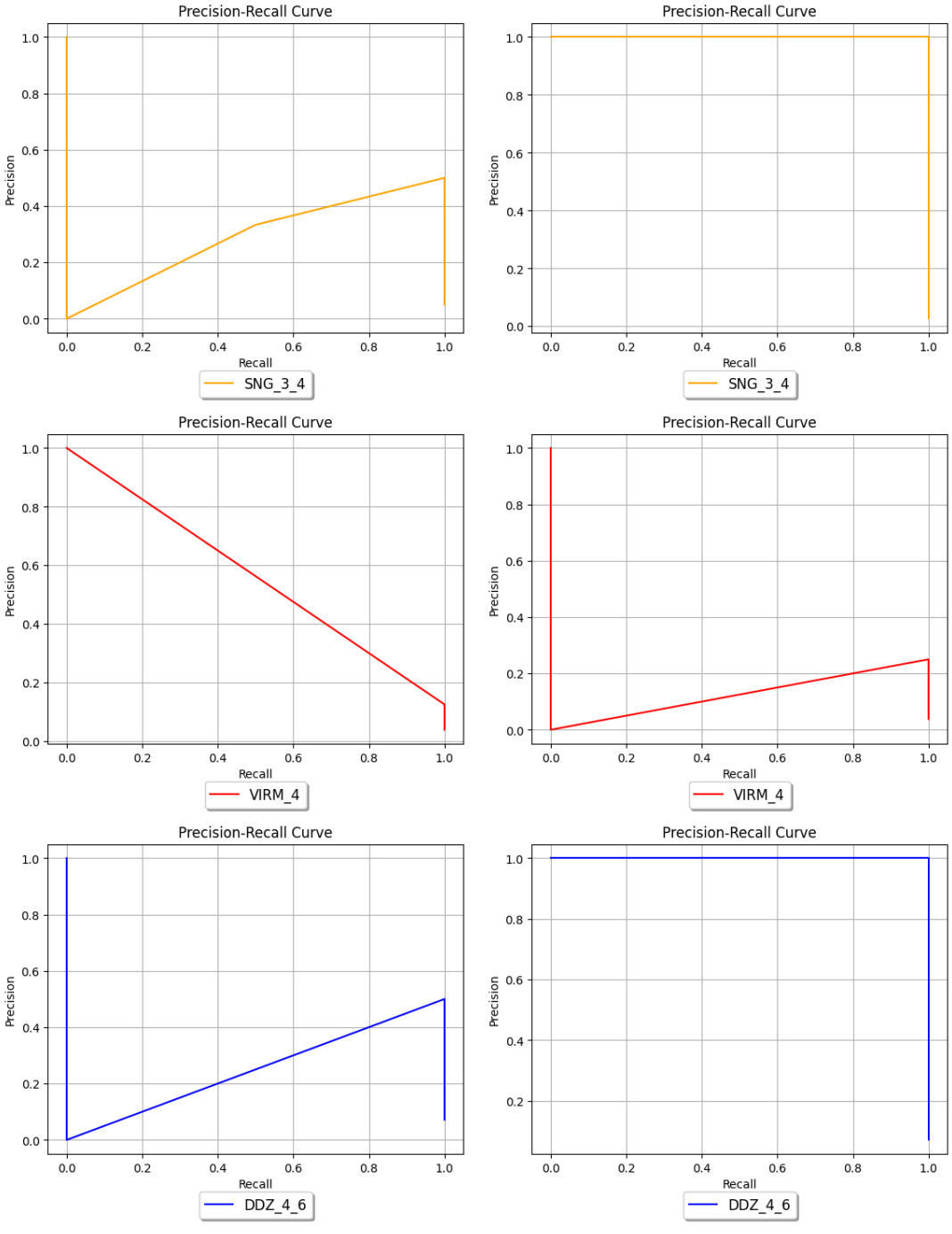



(a) *PatchCore* (b) *GLASS*

Figure 24: This figure shows the differences in ROC curves between *GLASS* and *PatchCore* run on all different subsets of the data. The Precision-Recall curves are shown separately underneath.




(a) *PatchCore* (b) *GLASS*




(a) *PatchCore* (b) *GLASS*