IT UNIVERSITY OF COPENHAGEN

Domain-aware embeddings for anomaly detection

Christoffer Ebbe Sommerlund (csom@itu.dk)

Master thesis
Data Science
KISPECI1SE

May 31, 2024
Supervisor: Stella Grasshof (stgr@itu.dk)

Abstract

Industrial anomaly detection can greatly benefit companies when it comes to quality assurance, preventing component failure and ensuring safety and reliability. In this paper, we collaborate with the company MAN Energy Solutions (MAN-ES) to tackle the task of industrial anomaly detection in the maritime maintenance domain. This combination of domain and task poses a challenging combination, as the evaluated images have uncontrolled environments which can trigger false alarms. Our approach uses in-domain data to fine-tune embeddings of a pre-trained backbone model with the goal to specialise it to the domain. If the backbone is trained on a useful proxy task, this would improve the discriminating capabilities of an anomaly classification model. We explore three different methods of fine-tuning backbones along with two anomaly classification algorithms, and perform a qualitative evaluation of both backbones and classifiers. We find marginal improvements of 1%, and discuss potential ways of improving upon this. The code is available at `https://github.com/circle-queue/anomaly-detection-public`. The data is property of MAN-ES, and cannot be shared.

Contents

1 Introduction

This project aims to detect unexpected user input using AI. It is produced in collaboration with MAN Energy Solutions, a maritime company wanting to improve the monitoring of ship engine components. Currently, experts at MAN are tasked with examining images produced by the ship owners at sea, and provide valuable feedback which could prevent damage to the engine. However, the volume of data is rapidly expanding, and they see a potential for providing immediate feedback on typical cases using a smartphone application.

Since this application aims to directly communicate with the customer, it should not provide feedback on components it was not trained on as it will be incorrect and degrade trust. Therefore, it should be capable of rejecting user input. Furthermore, the company also has an internal need for filtering out unrelated material, since data from the ship owners is often provided as an unordered camera-roll containing any number of things. By filtering and organizing the data, it can be used as training or testing data for other systems, or to simplify the labeling process. Finally, if the model becomes familiar with expected conditions of a particular component, it may also detect unexpected conditions of a component and raise an alarm.

This project aims to tackle the above problem using machine learning, specifically deep anomaly detection. The problems can be split into two categories:

- Detecting images belonging to unexpected categories in a dataset (e.g. a cat among engine components)
- Detecting damaged images within a category (e.g. a crack among otherwise smooth surfaces)

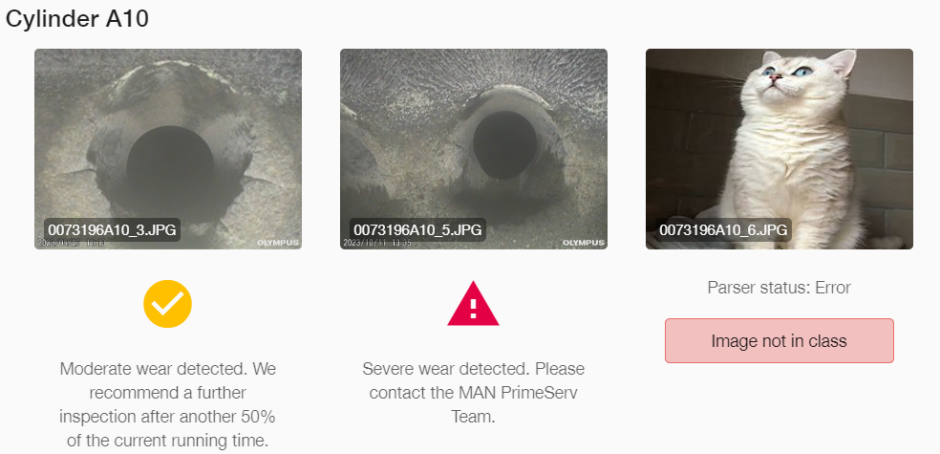


Figure 1.1: Examples of the two types of tasks. The two left-most images are looking anomalous compared to expected wear on that type of component, whereas the right-most image is completely unrelated and should not be evaluated.

A previous project[1] attempted to tackle the issue of filtering unrelated material using suggestions from [19], which proposed using pre-trained models to quickly produce results and found moderate success[2]. To get the best results, however, the paper [19] suggest specializing the model to the domain. While this has been effective in the manufacturing domain, the maintenance domain poses unique challenges that make data collection tricky, primarily due to data being collected from uncontrollable and diverse environments [28, 2].

This project aims to improve upon the previous project by specializing the embeddings to the domain. First, as suspected by the previous project and demonstrated by [2, 22] lower level features will be used

[1] `https://github.com/circle-queue/component-classifier`

[2] The best model utilized a one-class support vector machine model on ResNet50 ImageNet embeddings resulting in around a 5% false negative rate and 50% false positive/anomaly rate

instead of the logits layer used by the previous project. Secondarily, since ImageNet features are not very specialized to out domain, we specialize the backbone to create hopefully more informative embeddings for specifically the engine domain. Multiple approaches are explored in order to utilize different levels of data quality to alleviate the issues of anomaly data collection in the maintenance domain. This includes varying degrees of dataset pollution, volume, labeling and how the learning task is formulated, all with the intent of providing useful embeddings for anomaly detection.

The research question can be summarized as follows: When targeting the two categories of tasks, either classifying unexpected new image categories in a dataset or detecting component damage, how effective is fine-tuning a backbone on domain-specific data compared to applying an off-the-shelf pre-trained model?

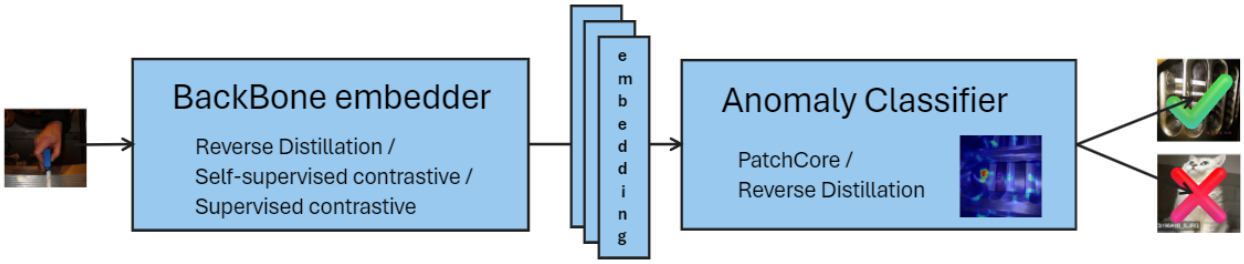


Figure 1.2: The architecture used in this paper. A backbone is optimised using one of the three listed methods, after which it can return embeddings at multiple scales, which are used by a classifier to create a 2D anomaly map. High anomaly scores result in the image being flagged.

2 Related work

Below are definitions for common terms in the following section which may be ambiguous in a broader context, but has a specific meaning in the context of this paper.

- Anomaly: An image with different, undesirable characteristics compared to a majority of other samples. Synonyms include outlier, positive case (y=True). Antonyms include "good", healthy, negative case (y=False)
- Embedding/feature/activation/representation: A model's representation of an image in the form of a vector.
- Domain: Images with a set of features, along with different likelihoods of observing those features[18]. E.g. the domains "cat" and "dog" images are similar, but fur-less features are more common in cats compared to dogs (domain-shift)
- Task: The objective of some model, e.g. distinguish between cats and dogs.

2.1 Anomaly detection in manufacturing

Autonomous anomaly detection can greatly benefit industry, since it can substitute human supervision in the form of e.g. batch inspections, and instead perform complete- and real-time monitoring of the production. Recent efforts have demonstrated great results in this area. Of note is the MVTec dataset[3], which contains close-up images of various small products on a plain background. Examples of objects include hazelnuts, pills, fabric etc.. Recent approaches like PatchCore[22] and Reverse Distillation[7] achieves near-perfect anomaly detection on this dataset. These are described in more detail in section 3.

PatchCore and Reverse Distillation apply a one-class approach, which is common in both general anomaly detection[19], but especially in the domain of industrial[5] anomaly detection. The one-class approach trains exclusively on the healthy/good samples, and new samples are evaluated on how well they fit into the previously seen samples. This makes data collection simpler due to the fact that anomalies can be tricky to accurately represent[19]. The reasons can be summarized in their rarity of occurrence[2, 28] and the diverse ways a sample can be anomalous, e.g. ways that never have occurred before.

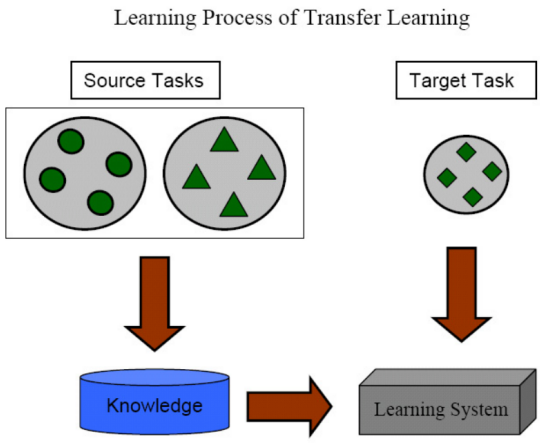


Figure 2.1: Source tasks can improve a target task. Source: [18]

Figure 2.2: Cherry-picked example images in the "piston-ring-overview" category showing the diversity of perspectives and illuminations within a single category

The mentioned approaches also require less data by utilizing pre-training and transfer learning, e.g. first learning the task of classifying images into one of 1k classes using the ImageNet1k[8] dataset. The idea is that learning to perform one task can add to the foundational understanding[1] of the model, which may benefit other tasks[18] as depicted in figure 2.1. For this to be effective, the original (source) domain/task must share characteristics of the final (target) domain/task, otherwise *negative transfer*[18] may actually degrade performance.

The assumption of PatchCore and ReverseDistillation is that anomalous samples are represented differently by models compared to ordinary samples. This works great in a manufacturing setting where one can control the data collection process and produce consistent images with variations mostly arise in anomalous samples. However, maintenance is unlike manufacturing as noted by the authors of datasets on maintenance anomaly detection[2, 28]. Given that the product is no longer indoor on an assembly line, these perfectly isolated and aligned closeups are not attainable, e.g. because the camera operator expects a human audience, or because the operator may not be human at all (drones, CCTV), or because a straight-on angle is not attainable after the product is assembled. The papers[2, 28] specifically lists the following primary difficulties:

1. Uncontrolled Surface, e.g. expected wear over time or dirt
2. Uncontrolled Background, e.g. objects or lighting
3. Uncontrolled Viewpoints

Figure 2.2 demonstrates this through three very different images belonging to the same class.

2.2 Targeting the maintenance domain

These authors and their datasets demonstrate the inadequacies of existing models when it comes to the above diversity of the maintenance domain, as the pre-trained approaches are unable to directly model these diverse settings. [28] proposes an architecture to deal with these unique challenges called Masked Multi-scales Reconstruction (MMR). MRR adds a model with the task of reconstructing the features of cropped out (masked) patches of an image from the remaining image, whereas previous approaches instead aim to model the image itself. The authors hypothesise that it is able to infer causal relationships from the remaining image to the masked patches, which makes it more robust when new surfaces, backgrounds or viewpoints are presented[2]. The reconstruction objective of MMR is another example of transfer learning, more specifically feature representation transfer[18] (FRT). The goal of FRT is to make the pre-trained representations more effective in the target task. Returning to MMR again as an example, we're not actually interested in the result of the intermediate task, that being the masked out patches, since we already have the original unmasked

[1] Models do not "understand", but this phrasing conveys the message of weights which can meaningfully separate samples in the embedding space in order to perform well in downstream task(s), e.g. classification

[2] The authors liken their method to the mathematical technique of conditional probabilities Pr(x|y) in contrast to other unconditional probability modeling Pr(x), where x is the reconstructed/masked patches and y is the original unmasked patches. As a concrete counterexample, PatchCore evaluates a new image by comparing patches of the image to previously known good patches in an embedding space. However, the embedding tries to exactly model that patch, encouraging a "memorisation" or inverse mapping function, whereas the conditional approach of MMR learns to reconstruct the patch features from the surrounding patches much like the in-painting capabilities of generative image models.

image. However, by encouraging the model to infer features from neighbouring patches, it means it generalizes better if the viewpoint changes, as the learned relationship is constant. E.g. if we see a palm we are likely to see fingers. The authors also note that the model performs worse than PatchCore and ReverseDistillation when these biases are not necessary[28], as is the case for the MVTec dataset with a fixed viewpoint and illumination.

However, learning these biases requires a separate module in the model and increases complexity. One could instead train the model from scratch, but that may fail to generalize without vast data volumes[28]. Another approach is to fine-tune the existing weights of the pre-trained network, incurring no additional space requirements. This is often done without labels (due to previously mentioned data collection issues for maintenance data), as that allows using large unlabeled collection of domain-specific data. Common approaches include unsupervised approaches, along with the aforementioned one-class approach of only training on healthy samples[5, 19]. Specifically, self-taught and self-supervised learning can improve a task by using knowledge from data which may not even fit into any of the classes within the task[20, 14]. This is particularly relevant when data may be polluted with unexpected data, as data is not constrained to fit into one of a number of classes. However, in case we have access to labels, we may be able to utilize this to improve model, even if the information does not directly relate to the target task (as demonstrated by MRR). One example of this is Supervised Contrastive Learning[14], which tries to create embeddings maximally distant embedding of other classes, while close to embeddings of the same class. This is similar to the triplet loss function.

The mentioned approaches (Reverse Distillation, MMR, PatchCore, (supervised) Contrastive Learning) fall into one of two categories: Reconstruction based or representation based[5]. The first approach can be thought of as a lossy-compression task, formally called auto-encoders, where a model should try to store an image using very little space, and then reconstruct the original image as accurately as possible. The assumption of MRR and Reverse Distillation[7] is that anomalous samples are harder to represent if one does not train on anomalous samples. Representation based methods instead skip the last step, and just try to create an *effective* compressed representation of an image. This representation can then be compared to other images, and if they're very different, one can classify them as anomalous. This is the approach of PatchCore[22]. Contrastive learning is also representation based, but it differs in its aim to minimize the distance between different augmentations of the same image (or images of the same class for the supervised setting), and maximise the distance to all other images.

Many approaches within anomaly detection[5] (including MMR, PatchCore and Reverse Distillation) use earlier layers of pre-trained model for its features, as the final layers are very close to the classification layer, and therefore biased towards that specific task. Since our binary anomaly classification task is significantly different from classifying a thousand different objects, this bias is likely unhelpful. Furthermore, the layer we select impacts the scale of the features we extract. Therefore, by combining multiple features, we can cover multiple scales of anomalies thereby improving the overall performance[28, 22].

This paper aims to overcome the issue of poor generalisation by using data at different levels of quality, e.g. by pre-training a backbone using a unsupervised and supervised proxy task, before training the actual anomaly classification task.

3 Methods

In this section we discuss the methods more in-depth, along with how the methods are combined to target different tasks and scenarios.

3.1 ResNet

ResNet[12] is the primary backbone architecture used in this paper. The ResNet model architecture aims to solve common issues that arise in very deep neural networks. Their core contribution is a stable and flexible design that supports many scales of models, e.g. ResNet18, ResNet34, ResNet50 etc., where the digits represent the model depth.

The stability arises from "residual learning", where data uses shortcut connections to skip computational blocks in the model using an identity function. The idea is that these shortcuts make learning easier, as the model does not have to learn this identity function of a complex image, but just has to learn the residual, i.e. the modification to perform.

The flexibility arises from stacking residual blocks, as layers can arbitrary increase in computational complexity (the paper demonstrates ResNet1202), while keeping the model conceptually simple: The model always has four layers of building blocks, and each layer always has the same output dimensions and activations are batch-normalized to a reasonable range. When we discuss using layers of a model, it is the output of these four conceptual layers we refer to. To merge multiple layers of differing dimensionality, we can upscale them to a common resolution using bilinear interpolation before aggregating them. This is discussed in more details in section 3.5.

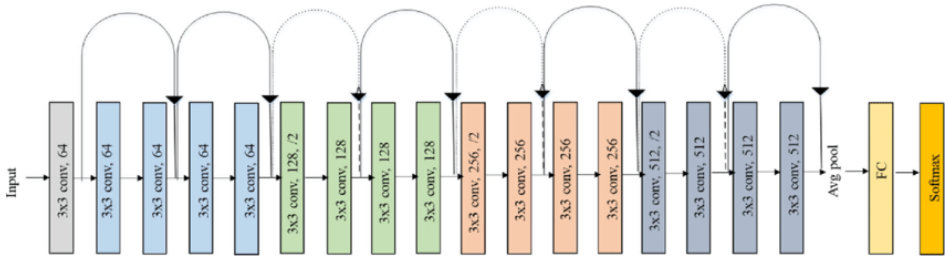


Figure 3.1: The ResNet18 architecture of our backbone. In this paper, layer1 refers to the first modular layer, that being the blue layer. Image source: [21]

It is also worth noting the resolution of each layer. The output of the four layers (layer1, layer2, ...) has a 2d image resolution (dimensionality) of 56x56, 28x28, 14x14 and 7x7 respectively. This is also worth remembering for 3.5, as a visualisation using activations from only the third layer has a maximum resolution of 14x14 pixels, which can be upscaled to e.g. 256x256. An example of this can be seen in appendix D.

3.2 PatchCore

PatchCore[22] does not perform training in the traditional sense, as it doesn't include any additional parameters other than a frozen backbone. Instead, it stores training samples in a "memory bank". Upon evaluation, PatchCore compares the new and potentially anomalous images to a memory bank of good observations seen during training. If the new image is very far from training samples, the image is marked as anomalous.

Storing images in their pixel representation would require vasts amount of space and would be inefficient for finding anomalies, as anomalies are different in some higher-level semantic sense which cannot be measured

in pixel level differences. This is especially true due to the high variability of the maintenance domain as discussed in 2. To better represent images, each image is embedded using a pre-trained model such as ResNet pre-trained on ImageNet and split into overlapping patches. For training, the patches are down-sampled to avoid storing near-duplicate patches in the memory bank. This reduces memory consumption and inference time, as we need to compare each memory bank patch to each patch of our test image. The anomaly score is the largest distance measured when comparing each pair of patches: the test- vs memory bank patches. For a more in-depth presentation, please refer to the PatchCore paper [22].

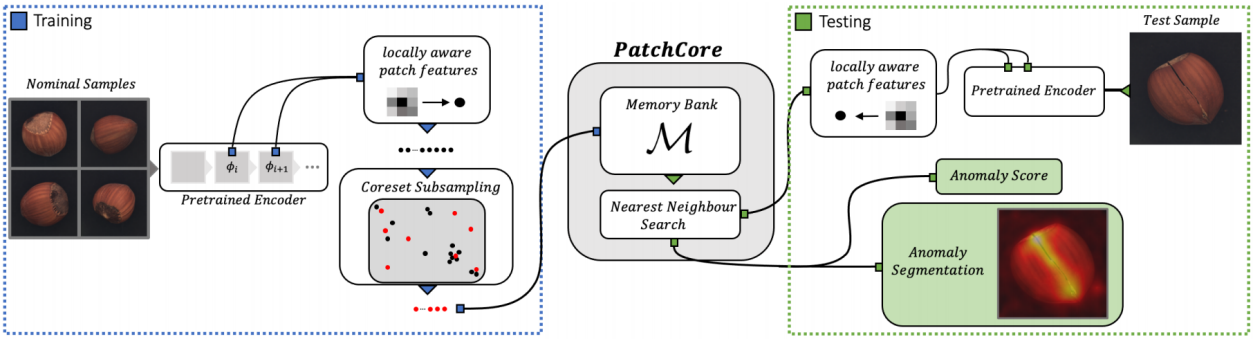


Figure 3.2: The architecture of PatchCore. We explore different pre-trained ResNet18 encoders for constructing the patch features. Source: [22]

To formalize this, we summarize the formula of the paper: Let $P(x)$ denote the function for creating patch-level embeddings m of a test image x^{test}, and let M be the memory bank created by applying P on the training set and down-sampling it. Then the unweighted anomaly score s^* is defined as follows:

$$m^{test,*}, m^* = \underset{m^{test} \in P(x^{test})}{\arg\max} \underset{m \in M}{\arg\min} ||m^{test} - m||_2 \tag{3.1}$$

$$s^* = ||m^{test,*} - m^*||_2. \tag{3.2}$$

The unweighted anomaly score s^* is then weighted and becomes s to make it more robust against outlier features in the training data. Consider again the memory bank M, specifically the neighbourhood N_b of m^* consisting of the b nearest embeddings to m^* in M. The anomaly score is amplified if this neighbourhood is distant to the test embedding $m^{test,*}$:

$$s = s^* \cdot \left(1 - \frac{\exp s^*}{\sum_{m \in N_b(m^*)} \exp ||m^{test,*} - m||_2}\right). \tag{3.3}$$

For improving upon PatchCore, we can fine-tune the pre-trained model used in P. This may result in embeddings which better model the characteristics of our data and hopefully the anomalous data in particular. If the pre-trained model creates creates embeddings where the anomalous data clusters differently compared to the "good" data, then we increase the anomaly score of anomalous samples when compared to the memory bank of "good" samples.

3.3 Reverse Distillation (Teacher-Student/T-S)

Typically, knowledge distillation is the term for transferring knowledge from a large model, the teacher, to a smaller one, the student. The goal is for the student to preserve some of the capabilities of the teacher while requiring less computational time and storage during inference. This is often done by first training the teacher on vast amounts of data on the task of interest (e.g. image classification), then freezing the teacher before finally training the student to mimic the response (activations) of the larger model on the original dataset. This accelerates learning, as the loss function doesn't just provide binary feedback (success, failure)

at the final classification layer, but across all activations. Additionally, it guides the training towards the already "optimal" solution of the teacher.[10]

The authors of Reverse Distillation [7] use a similar technique, but for another purpose. They create an anomaly score based on how poorly the student mimics the teachers response. The hypothesis is that the student will only be able to mimic the response of "good" data, if it was only trained on "good" data. The teacher, which has been frozen, will perform equally well on any data it sees, as its training is not impacted by the filtered input data, so there will be a large difference for anomalous images.

They achieve this by having the same, but mirrored, architecture for the teacher and student similar to that of autoencoders; the teacher being the encoder and the student being the decoder. The student is fed the response of the teacher, and is taught to mimic the teachers activations across a number of layers on "good" samples. Since it is difficult to recreate low-level responses from the high-level output of the teacher, the authors make use of multi-scale feature fusion by passing not just the teacher's output, but multiple intermediate layers to the student. However, this also simplifies the task of recreating these features, so the authors first condense this representation to a lower dimensionality using a one-class embedding block, before passing it along to the student. This block is trained along with the student model, as the authors hope for this embedding to be even more biased towards accurately embedding the "good" samples.

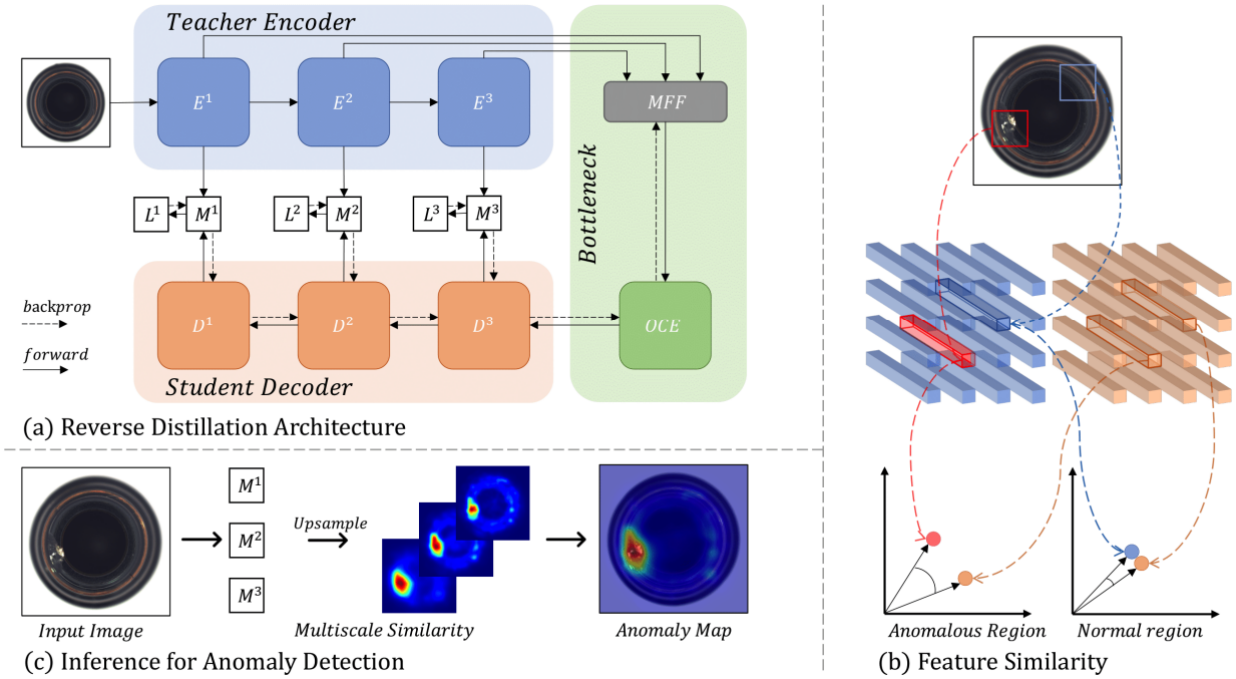


Figure 3.3: The architecture of Reverse Distillation (T/S) shown in (a), along with a visualisation of how feature similarity is computed in (b) and the resulting anomaly map in (c). (c) is discussed further in 3.5. Source: [7]

In practice, this architecture is simple if you know the ResNet architecture. Each image is passed through the first three layers of ResNet and the 3 layers are concatenated before passing them through the fourth layer[1], which functions as the one-class embedding block. The decoder then simply passes this embedding through the student ResNet's 4th, 3rd and 2nd layer to get three output activations, which can be compared with the 3 input layer activations. The magnitude of these differences are summed to produce the final loss function. This is also shown visually in (a) of figure 3.3.

Once more, we summarize the formula of the authors: The anomaly scores at each position (h, w) between the encoder layer f_E^k and decoder layer f_D^k is calculated using the "cosine similarity loss across the channel

[1]The three input layers are up-scaled to the same dimensionality using convolutions before concatenation, and the fourth layer is correspondingly modified to take 3x the input dimension

axis [which results in] a 2-D anomaly map M^k"[7]:

$$M^k(h,w) = 1 - \frac{(f_E^k(h,w))^T \cdot f_D^k(h,w)}{||f_E^k(h,w)||\,||f_D^k(h,w)||} \tag{3.4}$$

where $\cdot$ is the dot product. During training, the sum of each averaged anomaly map is minimized as the loss function. For the anomaly score, the authors suggest picking the largest value of this map such that even small anomalous regions are captured; a small anomaly is still an anomaly. However, we found better results using the mean. As a side note, PatchCore also seems to indirectly favor a mean-like function as show in appendix D.

For evaluation, each anomaly map is bi-linearly up-sampled so that they can be summed to a single aggregated 2-D anomaly map. This anomaly map is further smoothed by a Gaussian filter to remove noise.

3.3.1 As a backbone

The previous section discussed Reverse Distillation in its original formulation, however, in this paper we also utilize it as a method of fine-tuning a backbone. For this, all we need to do is decide on a dataset and unfreeze the teacher/encoder. The model will then aim to minimize the difference between the activations at each layer. This method may be effective for polluted datasets which contains anomalous data, since the teacher will be very good at representing all data. When the student is then re-initialized and re-trained on only "clean" data it will only easily approximate the "good" data it was trained on, and the anomalous data will have a comparatively larger difference between the teacher and student.

3.4 (supervised) Contrastive learning

Unlike the previous approach, Supervised Contrastive Learning[14] was designed to create an embedding model. It formulates the learning task as both a minimization and a maximisation at the level of each batch. It does this both in an self-supervised setting along with a supervised setting:

- Supervised: maximizes the distance between each pair of sample that belong to different classes and minimize the distance between any pair belonging to the same class.
- Self-supervised: maximizes each sample's distance to all other samples and minimize the distance to itself.

The last part of the self-supervised method is illogical in its current formulation, as the distance between oneself should be 0. To non-trivialise this, each sample is actually duplicated and passed through an augmentation before being embedded and compared using one of the above two approaches. Each sample in the self-supervised case therefore has exactly one self-comparison to minimize, and $2 * (batchSize - 1)$[2] comparisons to maximize.

More formally, consider a batch of samples with indices I. Let $z_i = Proj(Enc(\tilde{x}_l))$ be the output embedding from an augmented sample x_i passed through an encoder in the form of a ResNet model Enc and a projection layer in the form of an arbitrary neural network $Proj$. Then, for the self-supervised case, $j(i)$ is the matching augmentation of i and let $A(i)$ be the index of all other sample. The loss is then defined as follows:

$$L^{self} = -\sum_{i \in I} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_{j(i)}/\tau)}{\sum_{a \in A(i)} \exp(\mathbf{z}_i \cdot \mathbf{z}_a/\tau)} \tag{3.5}$$

where $\cdot$ is the dot product and τ is a temperature hyper-parameter.

For the supervised case, the function $j(i)$ is swapped with a comparison to each sample in $P(i)$, the set of remaining samples with the same class as i. $A(i)$ is still defined to be all other samples. The loss is then defined as:

[2]Each augmentation is compared to each of the two augmentations in the remaining batch

$$L^{sup} = -\sum_{i \in I} \frac{1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_p / \tau)}{\sum_{a \in A(i)} \exp(\mathbf{z}_i \cdot \mathbf{z}_a / \tau)} \tag{3.6}$$

Since contrastive learning is performed at a batch-level, the batch-size has an impact on the loss function and therefore model performance, especially for the supervised contrastive learning. This is because it encourages samples of the same class within the batch to produce similar embeddings. The original authors used a batch size above 1k when training on the 1k classes of ImageNet [8]. This is computationally infeasible for this paper. We use a batch size around an order of magnitude smaller (128), but our dataset also contains orders of magnitude fewer classes (10-ish classes, albeit imbalanced).

3.5 Activation maps

Since we have so little validation data, we want to get as much information from it as possible. Therefore, we not only use aggregated quantitative methods, but we will also use qualitative methods. We can examine the models response to individual images using an approach similar to the popular grad-CAM [23] technique. We do this by overlaying the image and the activations of the 2-D anomaly scores outputted by PatchCore or Reverse Distillation using a color-gradient. The pixels of the image contributing to a high anomaly score will thereby be colored in one way (often red), and ordinary regions will be colored another (often blue). As a human, we can then evaluate whether we believe the highlighted regions to be sensible. If our data had segmentation annotations of the anomalous region, we could do this quantitatively by comparing the models regions with the annotated regions, but instead we do this qualitatively by hand.

The activation heatmaps can be misleading if viewed without context of their generation process. This is because they are lower-resolution than the original image, meaning the upscaling process may give the perception of higher fidelity than actually exists. Below we visualize an example of this. The left-most image contains the activations of the third ResNet layer after passing through an image. The spacial resolution of this layer is 14x14 pixels. The middle picture shows the bilinear interpolation performed to upscale the image to the original size (e.g. 512x512). The third image demonstrates the effects of a Gaussian filter which smooths the image. More examples are shown in the appendix D.

Figure 3.4: The two post-processing steps of visualising activations: A bilinear upscaling followed by a Gaussian filter.

3.6 t-SNE

As with visualising the activations, we can also infer something about the quality of our backbones by examining the embeddings they produce. These embeddings contains hundreds of dimensions, so they cannot be directly visualised. Instead, we can apply a dimensionality reduction technique to summarize these hundreds of dimensions in just two dimensions such that they can be plotted on the x- and y-axis in a scatter plot. For this, we apply the method t-distributed stochastic neighbor embedding (t-SNE)[25]. This method models the distance between points as probabilities in two spaces: The original high-dimensional space and a randomly initialized low-dimensional space. The low-dimensional space is then adjusted to produce similar probabilities between datapoints as the original space.

To allow for more separable clusters, the low-dimensional space is cleverly biased towards accurately representing local regions, while compromising on the accuracy of medium- to distant regions. This means

that two nearby points in the high-dimensional space are likely to be close in the low-dimensional space, but two distant points in the high-dimensional space have little effect on each other in the lower space. One can therefore not meaningfully interpret distances between different clusters.

Mathematically, this happens by minimizing the Kullback–Leibler divergence between the two probability distributions using gradient descent. The low-dimensional space utilizes a heavy-tailed distribution, which enables the locality bias. The standard deviation of the probability distributions are controlled by a hyperparameter "perplexity". Larger perplexity values result in capturing larger scale patterns in the dataset.

As with any summarizing technique, it is important to not interpret the resulting visualisation too literally and keep the above discussion in mind with respect to cluster distances and perplexity values.

4 Experiments

4.1 Environment

Experiments were run primarily on an AWS SageMaker instance of type 'ml.g4dn.2xlarge' with 8 CPUs, 32 GB RAM and a NVIDIA T4 GPU with 16 GB vRAM . All experiments can execute in this environment, however, some experiments also ran on a local Windows desktop with 128 GB ram, two Intel Xeon Gold 624 CPUs and a NVIDIA Quadro P4000 GPU with 8GiB VRAM. Note that Linux is required for PatchCore, as it depends on the faiss[13] library.

To reduce IO, all images are resized to be at most 512x512 while maintaining the aspect ratio and stored on disk. As a preprocessing step, we apply the TorchVision[17] ResNet transform function[1], which consists of a resize to 256x256 followed by cropping to 224x224 and finally standard-scaling.

4.2 Hyperparameter optimisation (Optuna)

For hyperparameter optimisation, this paper utilizes the Optuna[1] framework. Optuna claims to improve upon naive hyperparameter optimisation techniques by using better algorithms. It frames the problem as a set of hyperparameters to adjust with a single output to optimize (e.g. loss). It then sequentially performs a number of trials (we use 100 trials), each proposing hyperparameters and observing the outcome, which informs the next values to select. This paper uses the default algorithm TPESampler[2].

Each trial is evaluated on a development (dev) dataset unseen during training. For the final results presented in the tables, we use the test set which is not used for anything else. For the experiments that fine-tune the backbone, we minimize the loss function, whereas for the anomaly classifiers we instead maximize the potential accuracy using the area under the curve (AUC) of the receiver operating characteristic (ROC) curve. If we run out of memory (VRAM or RAM), we return a score of 0% AUC to avoid an infinite loop of Optuna attempting an impossible configuration. This would frequently happen for PatchCore which would trigger the Linux out-of-memory killer, preventing Optuna from logging the failure. Therefore, we implement a fail-safe when building the memory bank, which would raise an out-of-memory exception if the RAM usage exceeded 50%.

4.3 Training

To provide the best overview, we first describe the training phases, after which we describe in detail the datasets used in each phase of the training. The reader may benefit from returning to this overview after reading about the datasets to get a thorough understanding of the experiments. The architecture and experiments are summarized visually in figue 4.1.

The training of the models are split into two disjoint phases. Firstly, a backbone model is trained to minimize some loss function on a training dataset. The second phase freezes the backbone and adds the anomaly classification model on top using another dataset. All ResNet models, backbone and classifier, are initialized with ImageNet weights, and we use the smallest ResNet architecture (ResNet18). The appendix B contains more details.

First Phase: The backbones are trained on the "train" split of either the vessel archive dataset or the scavenge port dataset (described in 4.4). The loss function to minimize can either be from Reverse

[1] `https://github.com/pytorch/vision/blob/v0.17.0/torchvision/models/resnet.py#L315`

[2] Optuna summarizes this algorithm as follows: "On each trial, for each parameter, TPE fits one Gaussian Mixture Model (GMM) l(x) to the set of parameter values associated with the best objective values, and another GMM g(x) to the remaining parameter values. It chooses the parameter value x that maximizes the ratio l(x)/g(x)." `https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/003_efficient_optimization_algorithms.html` (26-03-2024)

Distillation (Teacher Student/T-S) or the (un)supervised contrastive loss function. Since the scavenge port dataset contains labeled information, that dataset can utilize the "supervised contrastive loss function" for the 8-classes. The loss function is used to guide Optuna's hyperparameter optimisation when evaluated on the "dev" split of each dataset (see 4.2).

Second Phase: The classifiers are trained on one of the 5 "good" datasets. For anomaly classification experiments, the training set is synonymous with "good" samples as they do not contain any anomalies. The 5 datasets consist of the scavenge port dataset and each of the 4 sub-datasets of the condition dataset. After training, each classifier is then evaluated on their corresponding test set. As a concrete example, you can see in table 4.4 that the ring-condition dataset is trained the one "good" training sample and tested on 50 "good" or anomalous test samples[3]. We use Optuna in the same way as in the first phase.

The "dev" set is used during development such that the "test" set is only used once for evaluation. This also means the test results suffer high variance due to small sample sizes of both train, dev and test data, and individual results may therefore be unrepresentative, which is why we will focus on the larger trends of the results.

For more details such as hyperparameter ranges, see appendix B. For a complete list of experiments and their results, see C. Furthermore, an example image within each image class can be found in the appendix A.

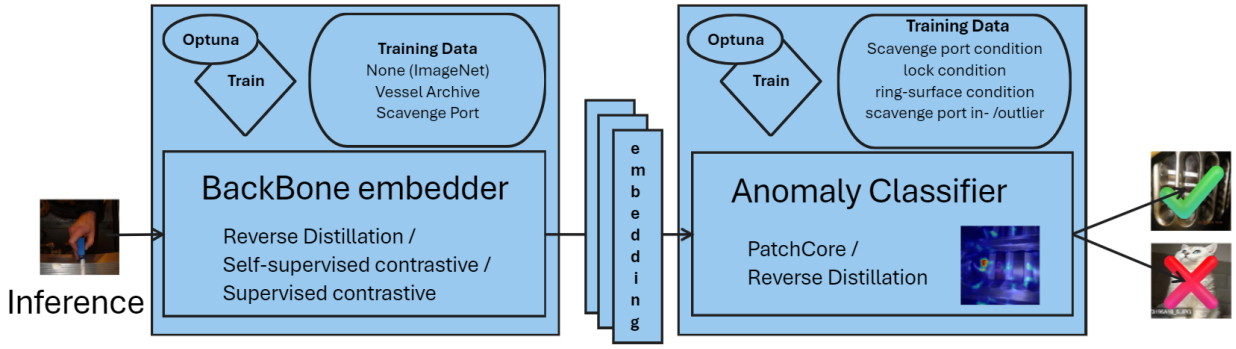


Figure 4.1: The architecture overview figure 1.2 from the introduction, but with added training information. Note the separate training phases being 1) the backbone embedder and 2) the anomaly classifier, each having multiple possible training datasets.

4.4 Backbone training datasets

This section presents the datasets which will be used to fine-tune a backbone model, which will produce embeddings for the anomaly classification model. For fine-tuning, we can annotate and produce high quality data, which often ends up in a higher quality model. However, the more time we spend on quality, the smaller quantity of data we end up with, which has a negative effect on model performance. As noted in related works, the maintenance domain requires a larger quantity of data compared to the manufacturing domain to be effective. For this reason, we examine two different scenarios for tackling the task of anomaly detection.

- Vessel archive (polluted, unannotated, 100k train, 14k dev)
- Scavenge port (filtered, unannotated or annotated, 5k train, 1k dev)

The datasets are produced by previous or ongoing projects at the company MAN-ES and therefore cannot be shared. Similar public datasets could not be found. The closest we found was the DIMO dataset [6] containing real and simulated metallic surfaces in different configurations and illuminations.

[3]The "50" is computed from the table 4.4 in the "test" column for ring-condition classes containing 1 good sample, along with 3, 17 and 29 samples in the categories Broken, Collapsed and Missing

4.4.1 Vessel archive dataset

The vessel archive data represents the lowest quality and lowest effort in-domain data. This data is collected by scraping a network drive for images. This results in a large quantity of data, but also some data which is unrelated to engines, such as loading cranes or people sitting at their desks. This may be an advantage in training the backbone, as anomaly classifiers benefit from the backbone more accurately representing anomalous regions from ordinary regions.

Table 4.1: The distribution of samples across the different splits of the vessel archive dataset

class name	train	dev	test
vessel-archive	128623	14325	0

One of the downstream application of the final model is to filter out these unrelated images, such that the remaining images could be used for training a model or for more granular annotation by humans. The scavenge port in-/outlier evaluation dataset in 4.5.1 represents this task of removing non-scavenge port images from a polluted dataset.

4.4.2 Scavenge port dataset

This dataset only contains samples belonging to one of 8 categories. Each of the categories represent some component which is examined during a scavenge port inspection. The 8 categories are also present in the vessel archive dataset, so the vessel archive dataset can be considered a "polluted" version of the scavenge port dataset. If we managed to collect enough samples within a single of the 8 categories, we may be able to apply anomaly detection on only that category to detect component damage. As an example, the single-piston-ring category may experience damage such as cracks or wear, which we would like to automatically detect. A complete listing of types of component damages is shown in table 4.4, which lists a few small datasets for evaluating component damage. This is discussed more in section 4.5.2.

Table 4.2: The different categories under the scavenge port dataset

class name	train	dev	test
liner	1147	246	246
piston-ring-overview	1466	314	314
piston-rod	37	8	8
piston-top	198	42	42
scavange-box	138	30	29
single-piston-ring	916	196	196
skirt	211	45	45
topland	408	87	88
total	4521	968	968

4.5 Anomaly classification datasets

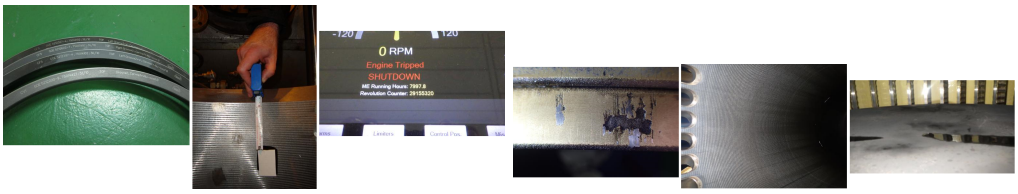
For validation data, we want to test whether the model is able to detect data which differs from the distribution it was trained on. We have two applications for this.

4.5.1 Scavenge port in-/outlier

The first application is for filtering inputs to other models. If a user dumps a camera roll containing partially irrelevant images, we do not want to provide feedback on the irrelevant images for two reasons: Firstly, it may lower the trust in our system if we always provide an evaluation, e.g. if a user accidentally uploads a selfie and is told their component looks worn out and should be changed. Second, we do not wish to waste

compute on expensive downstream models if the input is irrelevant. For this purpose, we use a dataset constructed from the same images as the scavenge port dataset, meaning the inlier class should follow the same distribution as that dataset, whereas the outlier class should accurately reflect the types of noise we receive along with the inlier data.

The outliers in this dataset is characterized by images of unrelated components, disassembled parts, microscopic images, screenshots and miscellaneous images the crew may take during their day.



(a) scavenge port anomaly (b) scavenge port inlier

Figure 4.2: 4.2a shows example anomalies for the scavenge port dataset. Examples include (from left-to-right) 1) a disassembled set of piston rings 2) a disassembled "liner" wall 3) a screenshot. 4.2a shows inliers such as (from left-to-right) 1) a "single-piston-ring", 2) a "liner", 3) a "piston-top". The left- and right-most images of 4.2a coincidentally are also anomalies in the context of 4.5.2, as metal is either chipped off the "single-piston-ring" or oil/water has leaked onto the "piston-top".

Table 4.3: The first evaluation dataset containing inlier and outlier images with respect to the Scavenge port dataset. The outlier class is defined as not belonging to any of the 8 scavenge port classes. *The inlier training dataset is the same dataset as the training dataset of the scavenge port dataset, as they should be from the same distribution

class name	train	dev	test
scavenge-port-inlier	4521*	373	373
scavenge-port-outlier	0	150	150
total	0	523	523

4.5.2 Condition dataset

The second application of anomaly detection is for building a classification model when little supervised data is available, and when the characteristics of anomalies may still be unknown. In our case, we have anomalous classes containing only 2 total samples (ring-surface-condition-signs of abrasive wear). The data is divided into 4 sub-datasets, each with a "good" class, which may be used for training in a one-class manner. For 2 examples of anomalous images in this category, see figure 4.2.

Table 4.4: The condition dataset, split into two groups. The healthy/good components and those which are characterized by some sort of damage

class name	train	dev	test
images-scavengeport-overview[good]	67	68	67
lock-condition[good]	20	21	21
ring-condition[good]	1	2	1
ring-surface-condition[good]	5	5	5
images-scavengeport-overview[abnormal]	0	42	42
lock-condition[Broken]	0	3	3
lock-condition[Burn Mark]	0	5	4
lock-condition[Coating Missing or peeled off]	0	31	31
lock-condition[Visible Cracks]	0	11	12
ring-condition[Broken]	0	4	3
ring-condition[Collapsed]	0	17	17
ring-condition[Missing]	0	29	29
ring-surface-condition[Coating Cracks]	0	6	6
ring-surface-condition[Coating Peel-off]	0	14	14
ring-surface-condition[Embedded iron]	0	31	32
ring-surface-condition[scuffing]	0	13	12
ring-surface-condition[signs of abrasive wear]	0	1	1
ring-surface-condition[signs of adhesive wear]	0	8	9

5 Results

Each of the below sections use a different dataset to create the BackBone (BB). The backbone is created by fine-tuning ImageNet weights of a ResNet18 model. This backbone is then used to train the classifier (Clf) and this classifier is evaluated on a binary inlier/outlier test dataset using the area under the curve (AUC) of the receiver operating characteristic (ROC) curve. Our training and evaluation capture the two types of tasks mentioned in the introduction:

- Scavenge port: We train on the scavenge port data (ignoring class) and evaluate using AUC on the binary in-/outlier data.
- Condition: For each sub-dataset, we train on the good class and evaluate on the binary in-/outlier data (anything not "good" is an outlier). The AUC is averaged across the 4 sub-datasets.

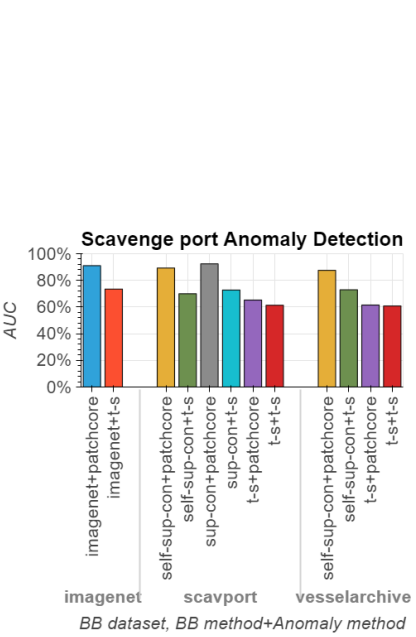
Keep in mind that the 4 condition sub-datasets have very little validation data, and even less training data, so results have high uncertainty. This especially true for *ring-condition* and *ring-surface-condition* with respectively 1 and 5 training samples.

The aggregated results are visualized in figure 5.1. The appendix C contains the same data represented as tables. We observe across both the Scavenge port dataset and the 4 condition datasets, that the PatchCore architecture generally outperforms the teacher student architecture when it comes to anomaly classification accuracy. For the scavenge port dataset, which is the easier of the 5 datasets, the best model achieves a 92% AUC score. This can be seen in the grey column representing a PatchCore classification model using an encoder trained using supervised contrastive learning on the scavenge port dataset. This aligns with the expectation that using supervised methods are better than unsupervised methods when trained on the same data. However, it is surprising that it is only a percentage point better than the baseline model using ImageNet weights. This means the proxy task of classifying the 1k ImageNet categories on out-of-domain data is almost as useful for training a backbone compared to training on in-domain data using any of our three backbone training approaches. This may be reasonable for the task of classifying anomalies in the scavenge port dataset, where images may contain categories of data vastly different from engine components.

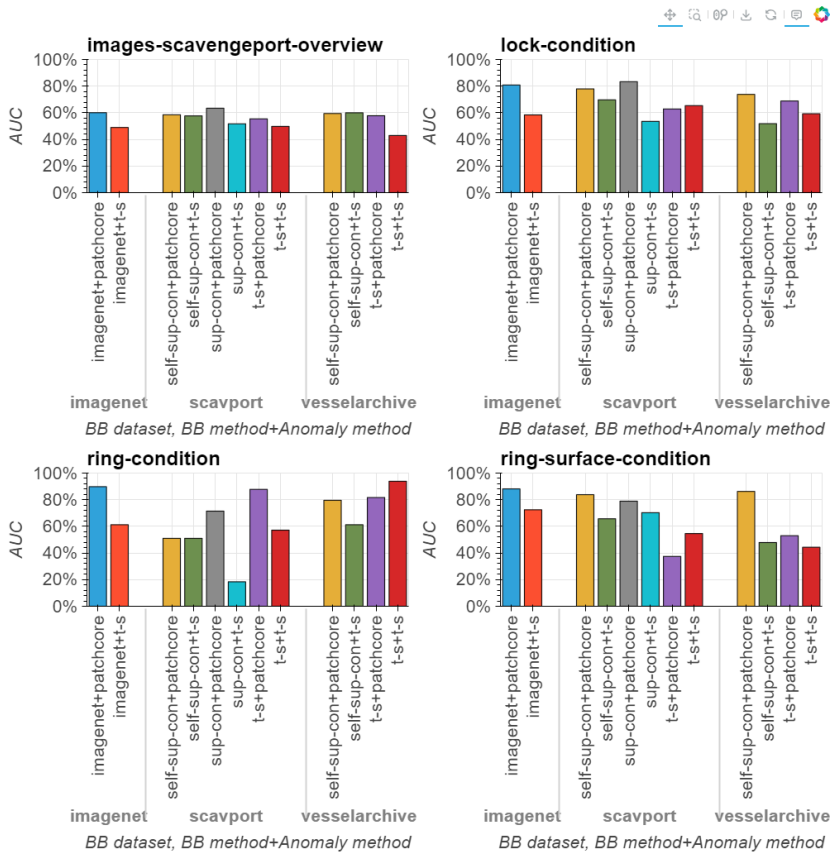
For the 4 sub-datasets, we see that the datasets vary significantly in performance. The lock-condition and ring-surface-condition datasets have a much higher AUC score of 80%+. For the ring-condition dataset we also see some results surpassing 80%, however, we also see results dipping significantly below random guessing of 50%, all the way down to 20%. For this reason, we should not trust the reported performances for this dataset, as the model training is produces wildly fluctuating results. This is explained by the training set containing a single sample, after which the models are evaluated on 50-ish samples. Surprisingly, we find the ring-surface-condition dataset to have much more stable results using only 5 training samples. This may be due to the anomaly classification task being easier compared to the other condition datasets, but more likely it is due to having a fortunate split for the development and training split. Despite the scavenge port overview condition dataset training on the most samples, it consistently performs poorly around 60% AUC. This is likely because an "overview" image has much higher variability in perspectives as compared to a zoomed in "ring" or "lock" image. This variability makes learning harder, as discussed in the related works 2. An alternate explanation is the complement of the previous explanation of seemingly good performance: Due to the many training samples, the anomaly threshold and resulting AUC can be accurately estimated. In section 5.6.1 we discuss methods of dealing with this variability.

5.1 Impact of vessel archive dataset

It is unlikely that the large volume of ImageNet is the cause of the good baseline performance, as the scavenge port dataset has orders of magnitude less data for training compared to the vessel archive dataset, but resulted in marginal differences in either direction. These results are also surprising when viewed in the context of



(a) Scavenge port in-/outlier



(b) Condition datasets

Figure 5.1: The classification AUC performance (100% is best) using different backbone dataset, training methodology and anomaly classification methodologies. Note: Matching colors correspond to identical training methodology on different backbone datasets. Also note that odd-numbered columns use the PatchCore method whereas even-numbered columns use the Reverse Distillation/Teacher-Student classification method (should not be confused with the embedding finetuning method under the same name). Graph is intentionally not zoomed in as to not magnify the small differences between model performance on the small dataset. For raw numbers, see appendix C.

the methods section 3: We hypothesised that training the model backbone on both the vessel archive dataset which contains both "good" and the "outlier" classes would result in embeddings which better model these classes, notably the anomalous class. This should amplify the errors between the backbone and a one-class trained model which has only observed "good" samples. However, as mentioned previously, we see marginal differences between the two datasets when measuring anomaly AUC. This may be explained by the model improving for anomalous samples but worsening for "good" samples when measuring the loss. The loss is shown in table 5.1. In contrast to the above, the Reverse Distillation method decreases in loss for both types of samples when fine-tuned on the vessel archive dataset. Despite improving the embedding loss, this doesn't lead to an improvement in our anomaly classification task as seen in 5.1. We believe this may be attributed to the model over-generalizing and learning the same features irrespective of the images. We explore this more in the next section, and discuss a method of combating this in section 5.6.4.

Table 5.1: The average loss of the scavenge port anomaly dataset. We present the results partitioned by whether the sample is anomalous, and by the backbones (bb) method and dataset. Specifically, we are interested in seeing how the characteristics of the backbone datasets effect the loss of anomalous or "good" samples. The vessel archive embedding fine-tuning dataset contains both anomalous (1) and "good" (0) samples, whereas the scavenge port embedding fine-tuning dataset only contains "good" samples.

bb-model	anomaly	bb-dataset	loss
T-S	0	scavport	9.1e-07
		vesselarchive	1.3e-05
	1	scavport	1.1e-06
		vesselarchive	3.1e-05
self-sup-con	0	scavport	2.0e-03
		vesselarchive	2.7e-03
	1	scavport	9.9e-03
		vesselarchive	5.4e-02

5.2 Investigation of backbones

The question arises whether the backbones have at all improved upon the baseline representations. To investigate this, we visualize the embeddings of samples using each backbone by applying t-SNE as seen in figure 5.2, and compare clustering patterns across backbones. In the figures, one should notice the grey "vessel archive" images spanning most of the image. The red "anomalous" samples are also very scattered, whereas the remaining coloured samples represent each of the components of the "scavenge port" dataset. The broad span of the grey dots align with the expectation that the vessel archive dataset is a superset of both datasets.

Ideally, we would observe little-to-no overlap between red anomalous and remaining coloured samples, as that means anomalous embeddings are far from "good" samples, which directly translates to a high anomaly score for the PatchCore model. For the Reverse Distillation/Teacher-student, the reasoning differs: If the teacher model is trained to embed all samples, but the student is only trained on "good" samples, this should lead to a large difference between the teacher and student when an anomalous sample is seen.

Returning to figure 5.2, we generally observe many anomalous samples close to or overlapping coloured regions, especially around the central green and yellow cluster, but otherwise being mostly in the region in between colored clusters. This is not true for the Reverse Distillation / t-s models, which seems to have overlapped all classes, anomalous or components. This may be explained by contrasting the learning algorithm with that of ImageNet classification or Contrastive learning: For Reverse Distillation, the objective is only to minimize the difference between the teacher and student response, albeit batch-normalized after each layer as per the ResNet architecture. One method for achieving this is by simply propagating activations similar to a positive feedback loop. If we contrast this with the classification task of ImageNet and contrastive learning, they instead aim to maximize the distance to unlike images/classes while minimising to like images. Therefore, the Reverse Distillation has no inherent mechanism of maintaining class separation, so if it can reduce its loss without modeling the differences in characteristics of the classes, it will. Unfortunately, this seems to also severely reduce the AUC performance, not just for the distance-based PatchCore method, but also for the Reverse Distillation anomaly detection method.

Returning to the non-Reverse Distillation embedding models, we see some red samples on the extremities close to the many clusters of grey samples. These clusters all represent different types of digital figures, e.g. bar-plots, flat lines, timeseries curves, heatmaps or photographs of digital curves. These images are vastly different from engine components, so it is expected for these to not cluster close to the coloured regions, and they likely incur a high distance anomaly score when applying PatchCore.

When it comes to differences between the backbones, we do not see significant differences between the baseline in the top-left and the remaining 3 non-Reverse Distillation backbones besides a rotation which

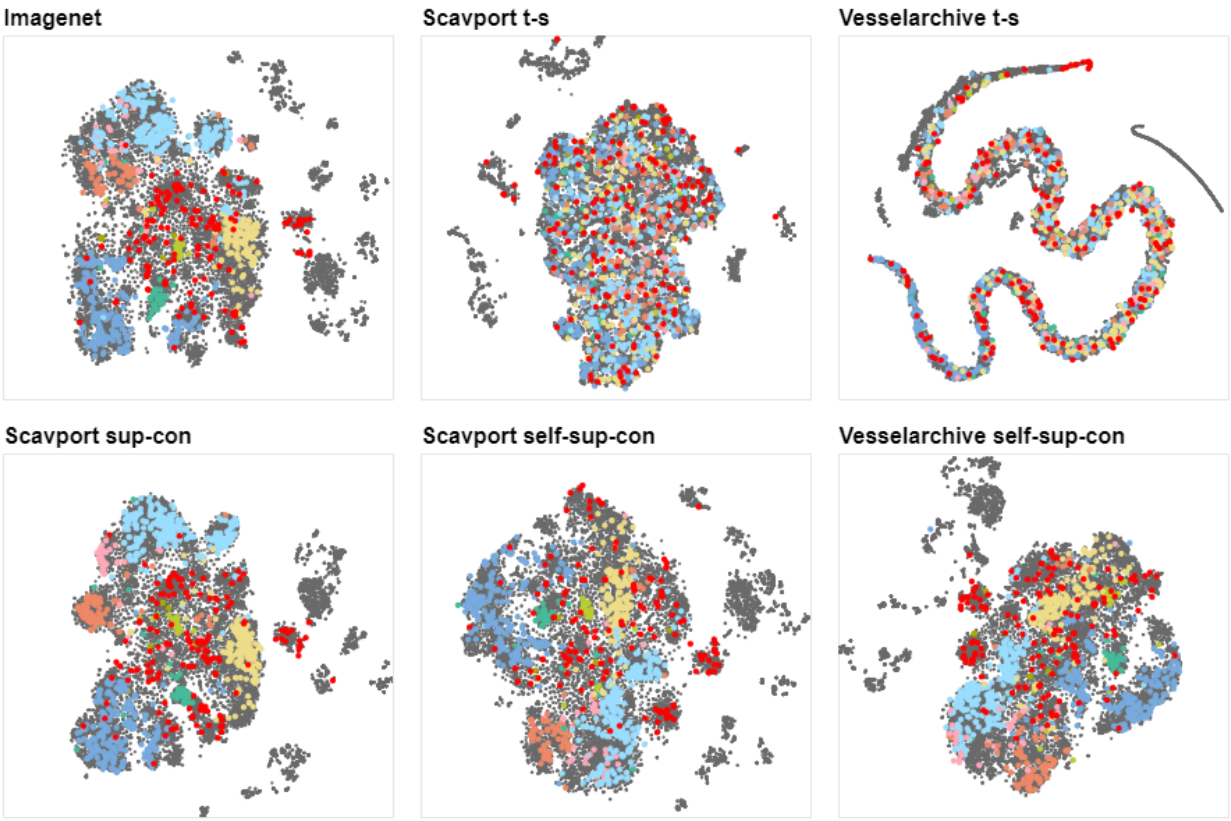


Figure 5.2: Datasets embedded using each of the backbones and visualized using t-SNE. Each point represents an image colored by type. The Vessel archive dataset samples are shown in the background in grey and using one tenth the size, whereas the scavenge port anomalies are shown in red in the foreground. The remaining 8 colors represent images of the components within the scavenge port dataset. The condition datasets are not shown for simplicity. The perplexity used is 100. Perplexities 50 and 20 were also tried, but did not lead to different interpretations.

can be attributed to the random initialisation of the t-SNE algorithm. The one exception is the supervised contrastive learning algorithm, which seems to cluster coloured regions tighter with less overlap with other classes. Specifically, the clay and pink cluster is not as scattered. Recall, that the training objective is specifically to maximize the distance between these categories. If we instead consider the self-supervised contrastive method, we expect to be able to produce form new clusters without annotations, as images are trained to resemble themselves after data augmentation is performed. As an example, the images may undergo perspective transforms, so this should indirectly make the images closer to other images that have a different perspective. Surprisingly we do not seem to form any new clusters using the self supervised contrastive learning method. One may see an additional cluster on the extremities, but this clustering already existed in the center of the ImageNet figure. This is shown in the appendix F, therefore it is considered mostly a misleading visualisation of t-SNE. At most, it may have made the existing cluster more tightly connected or more isolated, but not in any regard which seems to improve the AUC.

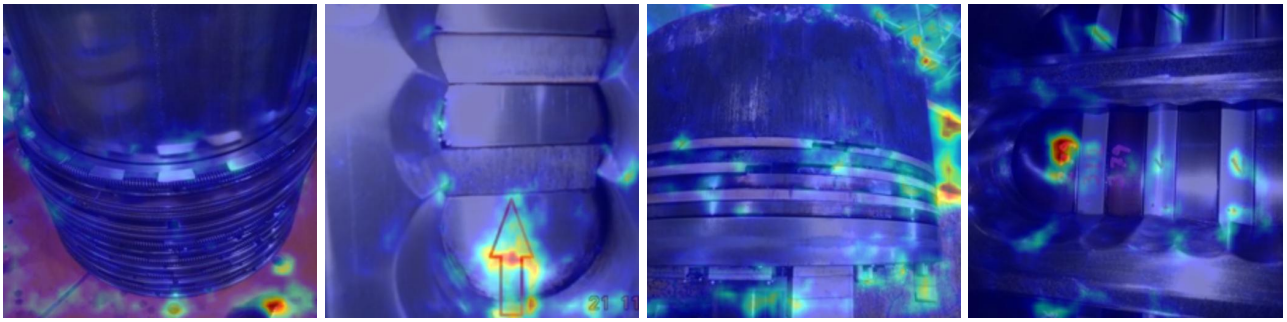
In summary, while the backbones do produce slightly different embeddings, these embeddings do not translate to significant increases in accuracy, as we see at most 1% point anomaly AUC increase for the supervised contrastive learning backbone. This leads us to believe the training of the backbones is inefficient for the downstream task. This may be due to either the backbone learning task or our method of optimising the backbone learning task, which is discussed in future works section 5.6.4.

5.3 Qualitative error analysis

It's hard to say whether one class of anomalies are easier to detect than another due to the low sample size. In appendix figure E.2 we see minor differences in how accurately the model predicts each anomaly, but nothing is worth highlighting due to the low sample sizes and therefore low statistical significance. The instability of the AUC metric is best exemplified by plotting the ROC curves which the AUC is computed from. In the appendix figure E.2b we see that the AUC is computed from just two samples for the "ring-condition" dataset. Since these metrics are unreliable to summarize the models performance, we would instead like to characterize the "good" samples which the model flags as anomalous, or fails to flag an anomalous image when it is supposed to. Before we can do this, we note that we will not be visualizing the best performing PatchCore model due to its large patchsize. The reason behind it is demonstrated in depth in appendix D.1. To summarize, for interpretation purposes we will be using the best hyperparameter configuration, but with a patchsize of 1, yielding a 10% points drop in AUC performance, but the resulting heatmap still represent the anomaly map of the best performing model, just prior to a patch aggregation.

Examining the most anomalous images of the scavenge port dataset, we see both anomaly classification architectures (PatchCore and T-S) have similar issues for misclassifications. In figure 5.3 we see the 4 most anomalous images for a T-S model. We see the model commonly reacts to surroundings of disassembled components or drawn symbols such as arrows or marker. The disassembled components in 5.3a and 5.3c are actually mislabelling and should be anomalous, as disassembled components are not a part of the scavenge port component dataset. This can be improved by better annotation guidelines with more examples and having multiple annotators and inter-annotator agreement to avoid miss-clicks. When it comes to scribbled digits, these are actually common in the dataset and an example of what we would like to avoid categorizing as anomalous by fine-tuning on these scribbled images. The practical reason for them is to later recall which cylinder you're looking at when reviewing the images. This is actually another business use-case, since if we can detect and parse the digits on the engine, we can better organize the images and provide more detailed feedback. Returning to the interpretations, we also sometimes see the model flag reflections as anomalous. Unfortunately, we also find unexplainable heatmaps of "good", e.g. the corner or edges are flagged as anomalous, or just a normal part of the component.

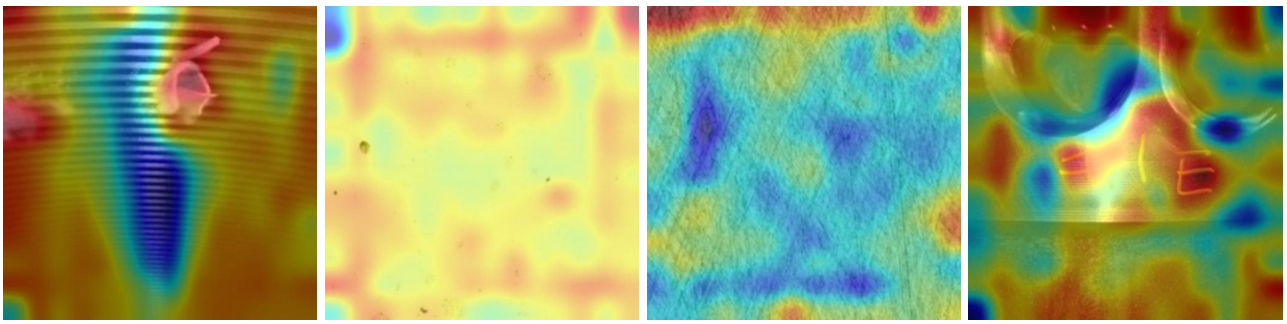
Before we examine the inverse cases where the model fails to catch anomalous samples, one should note that the samples are min-max scaled, such that there is always at least one red point on each image despite the image being classified as "not anomalous". Therefore, despite these images looking very red, this simply means that there is no area which is particularly anomalous. Looking at figure 5.4, we commonly see that plain, blurry or even surfaces fail to be classified as anomalous. Other times we find slightly more understandable examples, such as a liner image with a digit or a part of a component with some part missing. The latter are more acceptable, as these images are of actual engine components. To capture



(a) A disassembled piston (b) A piston ring with a digital arrow (c) A disassembled piston (d) A piston ring overview with hand-written digits

Figure 5.3: Anomaly heatmaps of the 4 most anomalous images from the scavenge port "inlier" class. (Model: ImageNet+T-S using the mean of layers {1,2,3}

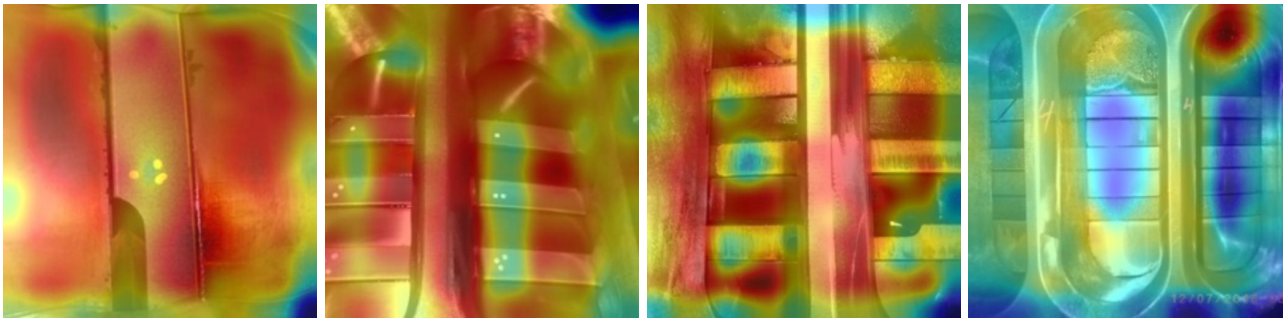
finer-grained anomalies, the type of component should be classified by one model and sent to specialized downstream anomaly classification models more rigorous in their filtration. It is unlikely that a single model will be able to have rigorous filtration, since the PatchCore memory bank is very sensitive to features in its training memory bank.



(a) A liner image with marker (b) A plain surface, maybe microscopic (c) A microscopic image (d) Almost "good" image

Figure 5.4: Anomaly heatmaps of the 4 least anomalous images from the scavenge port "outlier" class. (Model: ImageNet+PatchCore using layer=3

In figure 5.5 we examine the errors made in the condition dataset with the most training samples, that being the "scavengeport-overview" with 67 samples, we see that the model suffers from the characteristic difficulties of those mentioned in the related works 2. The model determines that the most anomalous regions are the surroundings and not areas around the 3 piston rings. Curiously, these rings actually seem to have some of the lowest anomaly scores. Specifically, the white dots are very distinct features which are present during training, and this lowers the anomaly score for the "good" samples of 5.5a and 5.5b. For 5.5a, the exact area containing the anomaly is marked as not anomalous, that being the middle of the 3 piston rings, which contains vertical surface scars. For 5.5d, one specific area is marked as more anomalous as the rest of the image, but this area doesn't look out of the ordinary except that it contains some reflections. To summarize, the model completely fails to capture the areas of interest, and is much more sensitive to the surroundings.



(a) "Good", but flagged (b) "Good", but flagged (c) Missed anomaly (d) Missed anomaly

Figure 5.5: Two false positives and two false negatives. Notice that the 3 rings are rarely marked as anomalous compared to the surroundings, despite these areas containing the anomalies. (Model: SupContrastive@ScavPort+PatchCore using layer=1,2

5.4 Importance of layers

Initially, only PatchCore models had a hyperparameter for selecting which layers participate in the anomaly score. However, we observed that Reverse Distillation would perform drastically worse performance on otherwise identical configurations, e.g. 91% vs 71% AUC after hyperparameter optimising on the dev set. We also observed that Optuna seemed to only select the third layer for the best performing PatchCore models. This lead us to the hypothesis that Reverse Distillation may benefit similarly from the possibility to optimise the layers contributing to the anomaly score. This indeed increased performance by 3% points on the development set. When examining the layers and their performance in figure 5.6, we see the trend that using deeper layers, specifically the third layer, generally perform better. This is especially true for PatchCore. This is unexpected when viewed in the context of previous works, which find optimality using intermediate layers, especially when combined. We do not observe this same trend, as individual deeper layers are often equally or more effective compared to combining them with earlier layers.

For PatchCore, we also note many zeros when multiple layers are used for the scavenge port dataset. This is discussed in the following section, section 5.5, but it is caused by out-of-memory errors.

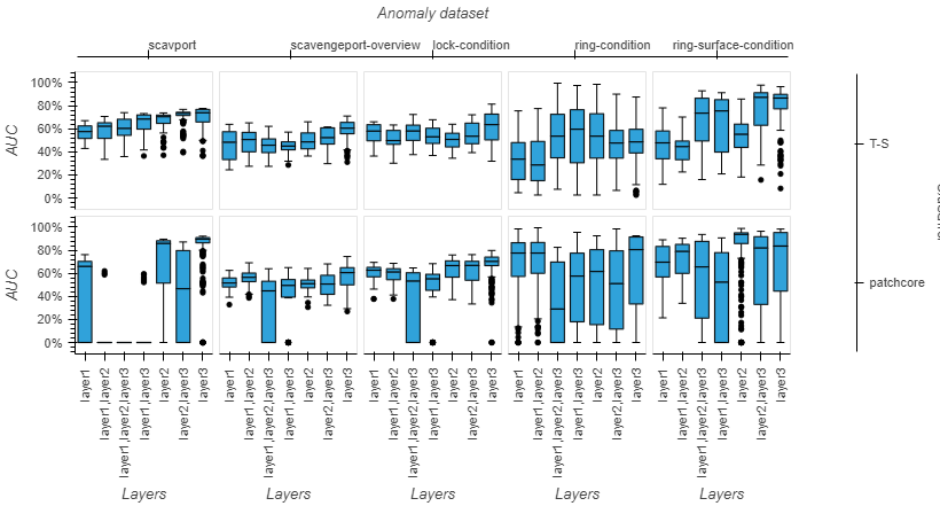


Figure 5.6: Boxplots showing the AUC across all optuna runs on the development set using different layers of ResNet for classification, partitioned by anomaly dataset and classification architecture. Note that the x-axis is ordered by increasing depth.

Since we observe the third layer to be so important, it would be interesting to see the effects of incorporating the fourth layer. This was originally dismissed since other papers dismissed the usage of this layer.

5.5 PatchCore instability

PatchCore was difficult to perform hyperparameter searching on, since parameters significantly effect memory consumption proportional to other parameters. As an example one can increase the number of samples, but only if you also decrease the embedding dimension or the feature retention percentage. This means you have to play around with configurations for each dataset or create formulas and conditional logic for suggesting hyperparameter configurations to Optuna. As discussed in section 4.2, we did not pursue this, and instead just logged a 0% AUC score if the process approached running out of memory. This has the side-effect of often resulting in out-of-memory errors when using multiple layers and high number of samples, which is the case for the scavenge port training set. Since we generally do not observe greater performance by combining layers, we present the results as-is and do not implement conditional hyperparameter searching.

5.6 Discussion and Future work

5.6.1 Data volume & augmentation

The largest issue for this project is data. With the few training samples in each "condition" dataset, the models are very prone to over-fitting and not being representative of the population of possible samples. So despite getting a seemingly great test score, this should not be trusted if our sample is small and possibly biased towards a few engines or environments. Another approach is to better utilize the data using cross validation, so we train multiple models on different splits of the data and evaluate the variance of these models' performance, e.g. by using statistical bootstrapping. However, this still assumes that we have sufficient training data to be representative: If our sample is biased towards "easier" cases, we cannot produce a fair evaluation.

Another approach which could be implemented with very little work is data augmentation. We already employ this in the self-supervised contrastive learning algorithm as a part of it's design, but we could use it for all methods. The reason it is useful is due to cameras commonly being flipped 90 degrees or having more or less zoom. Therefore, we can inject some useful biases in the model by applying random augmentations mimicking these actions prior to training on a batch. This will allow us to learn more from the same number of samples which should reduce overfitting. This would somewhat reduce the issues with getting a "lucky" training set as we train on more perspectives.

Another type of data augmentation we could apply is CutPaste[16], which aims to simulate anomalous regions by copying and pasting a region from one location to another on the same image. Specifically, the method of CutPaste (Scar) may be suitable for simulating some types of scar-like errors, such as surface abrasions or lock breakage. This could also be used for evaluation, however, the augmentation is not guaranteed to be representative of real anomalies, and definitely does not represent all types of anomalies, e.g. an entire ring missing. For other types of anomalies, we could implement specialized detection models. As an example, the company MAN already has a piston-ring detection algorithm. Therefore, if we detect only 2 rings, this tells us that a ring may be missing. Another specialisation could be to create a dataset of digits on surfaces and train a supervised model to classify this, after which an OCR model could extract the text. This would help two business cases, but at the cost of complexity.

Finally, the best solution is to have experts annotate more data. This is a slow process, but with the new models we can create a human-in-the-loop flow to improve the experience. This could include the binary anomaly classifier and an 8-class scavenge port component classifier. The models could suggest an annotation, and if the human agrees, they can click "next", and otherwise correct the annotations. This could be improved further by showing images in bulk within the same predicted category, e.g. anomalous piston-ring images in a 4x4 grid of images, and the human annotator just has to glance over the images and correct the false positives. This should still include improved annotation guidelines and multiple annotators per image to avoid the incorrect annotations found in the datasets as shown in figures 5.3a, 5.3c and debatably 5.4a. These could also examine samples in the embedding visualisation, and re-evaluate samples that cluster closely to

anomalous samples. These boundary points or incorrect labelling can cause problems for PatchCore, since the architecture is sensitive to pollution in the feature space due to comparisons with the closest sample in the "good" memory bank when evaluating whether a sample is anomalous or not.

5.6.2 Better backbone training

Previously we mention how data augmentation or larger datasets may benefit the low-data regime. However, data augmentation may also benefit us in the same way when we have thousands of samples for fine-tuning the backbone. As mentioned in the related works, the data volume may be too little to create robust features, but perhaps it is possible with data augmentation and unsupervised methods or other backbone training methods using transfer learning with proxy tasks. One example of training a backbone may be using a similar strategy leading to the success of [28] where they employ masked feature reconstruction, which is a type of masked auto encoder (MAE). By using the proxy task of reconstruction of masked out image patches, we could train from scratch or fine-tune a backbone in a self-supervised manner to create embeddings that are accurate at inferring the remaining of the masked out image. This masking approach has been effective in natural language processing (NLP) using e.g. BERT [9], and similar approaches have developed for images [27] where one particular implementation of MAE [11] seems promising using a transformer architecture. We can even extract a pre-trained MAE model trained to perform image segmentation on millions of images from the Segment Anything model [15], which uses MAE as the image encoder. Alternatively, we could train the model from scratch, possibly with the addition of data sources to increase data volume and make the representations more robust towards our domain. This may also reduce the redundant generic representational capabilities of the model, which can be an advantage during inference since less generic features can mean more discriminating features. A possible dataset for this could be DIMO [6], which contains metallic objects with reflective surfaces in diverse environmental illuminations. This may make the model better understand reflections of surfaces, which is the source of some of the false-positives.

5.6.3 Specialized segmentation models

We saw in the qualitative analysis of section 5.3 that much of the anomaly scores are attributed to regions surrounding the region of interest. Therefore, if we were able to eliminate irrelevant regions, our model would only be sensitive to relevant regions. This could be advantageous at the inference stage, but it would also significantly simplify the model training. If we consider PatchCore as an example: If we only trained on the regions of interest, our memory bank would not be filled with features related to the surrounding environment and more features could be used to discriminate for the relevant area. MAN has already developed one segmentation model for isolating the piston-rings, and more could be developed for e.g. detecting the "lock" of a ring or the surface of a piston. With a powerful model like the Segment Anything model [15] mentioned in the last section, this task is easier to build data for. The clear disadvantage is that two models now have to be trained instead of one, and if the first segmentation model fails, not only may be capture too much of an image, but we may omit the anomalous region completely. These disadvantages could be limited with an interactive UI on the end-user application, which could support adjustments of the segmentation when applying the model.

5.6.4 End-to-end training/optimisation

Currently, we're training the model in two distinct phases. First, we train the backbone to optimize some loss function, after which we freeze it and hyperparameter optimize the the anomaly classifier. However, optimising the first phase does not grantee performance for the second. Therefore, the training could be combined in one of two ways. The first is an end-to-end training, where the loss produced by the anomaly classifier propagates and updates the weights of the backbone. This is proposed as an effective method for some architectures as noted in [19], however, as we saw in the results, the Reverse Distillation architecture is not one of those cases. The second, more stable approach is to still train the two independently, but guide the hyperparameter optimisation of the embedding model by the performance of an anomaly classification model. This would switch the fitness evaluation of the embedding model from its performance on the loss function to the performance on our downstream task, and would avoid the hyperparameters overfitting to

the pre-training embedding task. This leads to a chicken-and-egg problem, as we need the anomaly classifier to evaluate the backbone model, but the backbone model is a part of the classifier. To solve this, and to avoid having to jointly train a backbone for each classifier, we can select a fixed classifier to be representative of the anomaly classification task, e.g. the baseline, and have this guide the hyperparameter search of the embedding model.

6 Conclusion

This paper applied in-domain data to improve the performance of an anomaly classification task and found little success: Only one training configuration was able to improve upon the baseline performance from 91% to 92% AUC when detecting novel classes with respect to a dataset. This was achieved by fine-tuning the ImageNet backbone weights using a Supervised Contrastive learning algorithm on an annotated, in-domain dataset of around 6000 images, and applying PatchCore as the anomaly classification model. Generally, it seems PatchCore performs much better, but it is highly sensitive to hyperparameter configuration, specifically the layers used of the backbone. Due to low sample sizes and dubious qualitative results, we can not claim to have a successful model when it comes to evaluating damage of components.

Unfortunately, we did not succeed in utilizing the large volumes of unlabeled data for improving a backbone, as the representations of the self-supervised algorithm did not deviate significantly from their ImageNet baselines. Even worse, the Reverse Distillation architecture completely overfit on its proxy task, which resulted in significantly degraded performance when evaluated on the target task. This was due to the two tasks being optimised in isolation. For the future, we believe the way of optimising the source task can still be improved, e.g. by changing the backbone's training task. However, the primary issue when it comes to classifying component damage is data volume. Our sample size is too little to account for the diverse nature of maintenance domain data. This diversity can be reduced by using segmentation models to only examine the regions of interest, which is likely to improve performance.

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] T. Bao, J. Chen, W. Li, X. Wang, J. Fei, L. Wu, R. Zhao, and Y. Zheng. Miad: A maintenance inspection dataset for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 993–1002, 2023.

[3] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9592–9600, 2019.

[4] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.

[5] Y. Cui, Z. Liu, and S. Lian. A survey on unsupervised anomaly detection algorithms for industrial images. *IEEE Access*, 2023.

[6] P. De Roovere, S. Moonen, N. Michiels, and F. Wyffels. Dataset of industrial metal objects. *arXiv preprint arXiv:2208.04052*, 2022.

[7] H. Deng and X. Li. Anomaly detection via reverse distillation from one-class embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9737–9746, 2022.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] J. Gou, B. Yu, S. J. Maybank, and D. Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[11] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[14] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

[15] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[16] C.-L. Li, K. Sohn, J. Yoon, and T. Pfister. Cutpaste: Self-supervised learning for anomaly detection and localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9664–9674, June 2021.

[17] T. maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

[18] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[19] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38, 2021.

[20] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766, 2007.

[21] F. Ramzan, M. U. Khan, A. Rehmat, S. Iqbal, T. Saba, A. Rehman, and Z. Mehmood. A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks. *Journal of Medical Systems*, 44, 12 2019.

[22] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler. Towards total recall in industrial anomaly detection. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14298–14308, 2022.

[23] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[24] L. N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pages 369–386. SPIE, 2019.

[25] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[26] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, et al. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[27] C. Zhang, C. Zhang, J. Song, J. S. K. Yi, K. Zhang, and I. S. Kweon. A survey on masked autoencoder for self-supervised learning in vision and beyond. *arXiv preprint arXiv:2208.00173*, 2022.

[28] Z. Zhang, Z. Zhao, X. Zhang, C. Sun, and X. Chen. Industrial anomaly detection with domain shift: A real-world dataset and masked multi-scale reconstruction. *arXiv preprint arXiv:2304.02216*, 2023.

A Example imgs

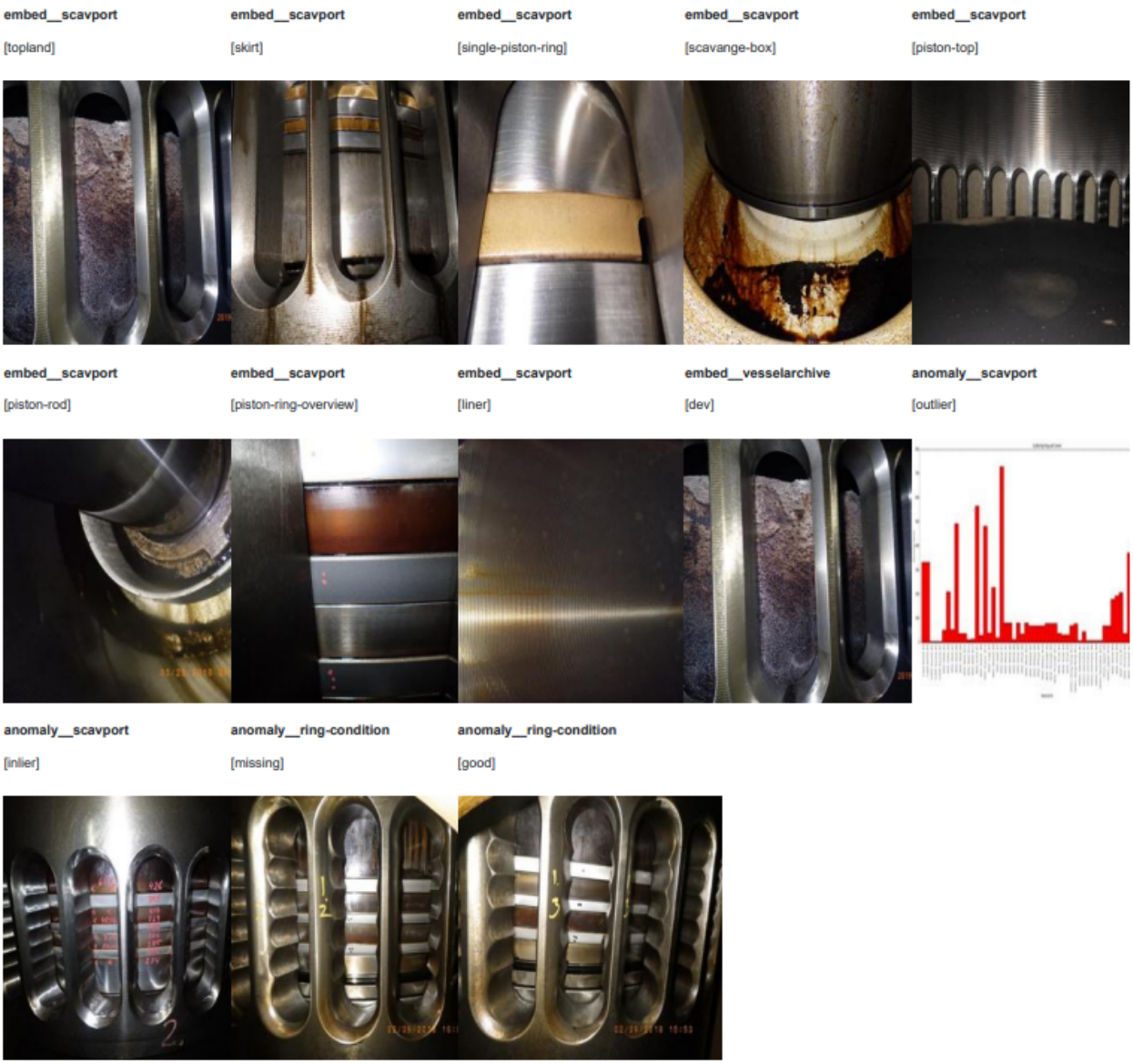


Figure A.1: One example image per class within each dataset used in this paper (1/2). Continued in A.2

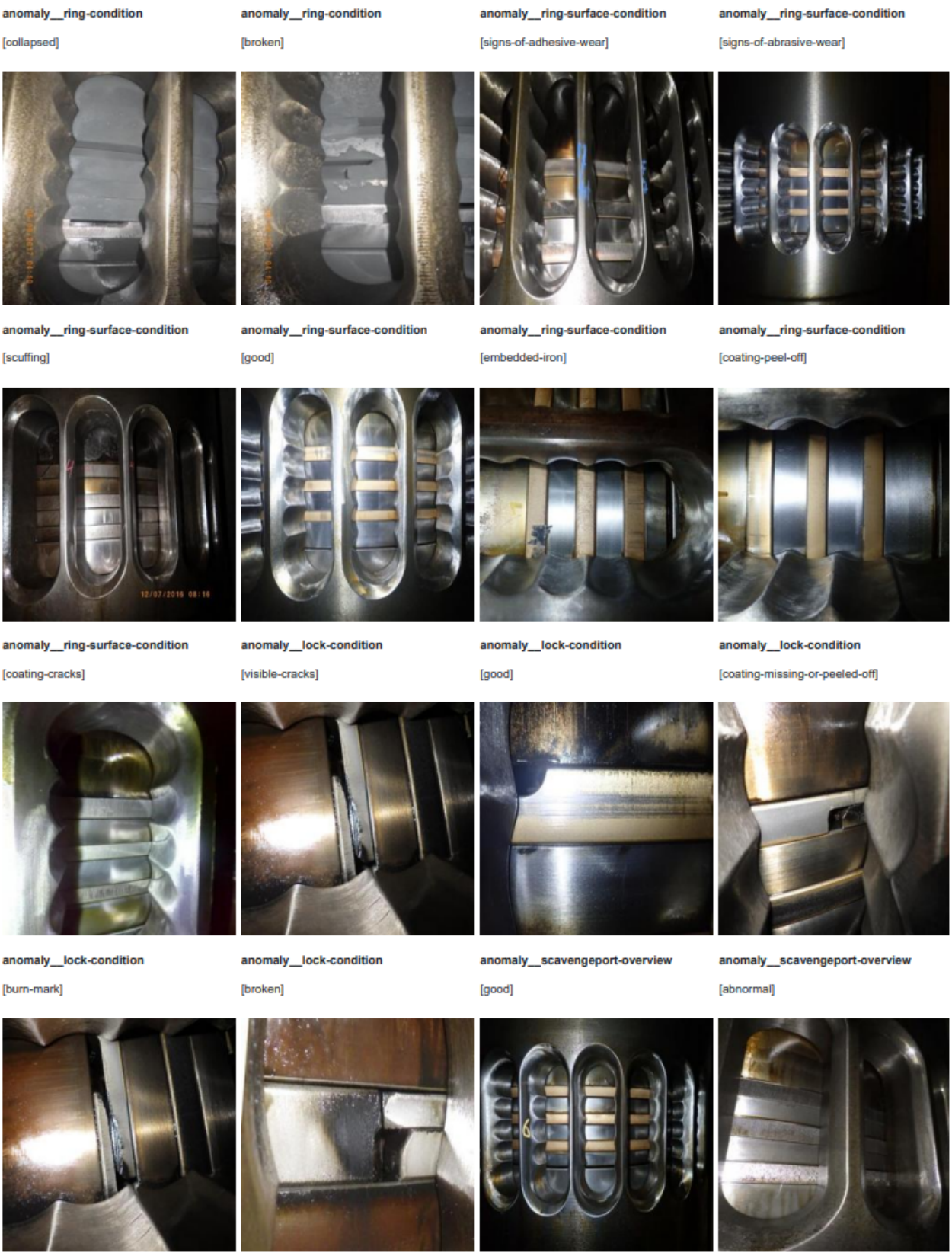


Figure A.2: One example image per class within each dataset used in this paper (2/2).

B Training details

This section describes implementation details that are insignificant for the main line argumentation of the paper, but helpful if you intend to read the source code.

We first describe how experiments are executed. Each experiment is invoked from the command line in three steps. Firstly, the model is run with optuna to find the best hyperparameters. The hyperparameters and their valid values are defined in a "XYZParams" python class. Each experiment may also include overrides. The below example shows this with the arguments "–trial-overrides" and "–params-cls".

```
python -m anomaly_detection
    dataloader-condition
        --sub-dataset=images-scavengeport-overview
        --transform=resnet18
        --train-classes=['good']
        --test-classes=['good','abnormal']
        --batch-size=1
    set-outliers
        --labels-inliers=['good']
        --labels-outliers=['abnormal']
    add-model
        --model-name=resnet18_T-S
        --weights=imagenet
    add-task--teacher-student-anomaly
        --cfg='"optuna"'
    run-optuna
        --optuna-direction=maximize
        --optuna-metric-name=anomaly_auc
        --trial-overrides="{"n_samples":lambda trial:trial.suggest_int("n_samples",1,1000,log=True)}"
        --params-cls=TSAnomalyParams
```

This logs the hyperparameters and metrics to MLflow[26]. The best hyperparameters are then retrieved and used to train and store a model in MLflow. This model can be loaded later for evaluation purposes or, in the case of backbones, for training models on top.

For models with parameters (all except for PatchCore), we utilize the Adam optimizer with a 1cycle learning schedule[24]. Since we assign learning rates per each layer, we cannot use a fixed max learning fate for the 1cycle scheduler. Instead, we use a magnification hyperparameter, such that each layer is assigned a max learning rate of some multiple of the base learning rate. We increment the scheduler at every batch, and randomly sample the dataset at every batch instead of doing a set number of epoch loops. This allows us to define a target number of samples to train on (again, excluding PatchCore).

PatchCore cannot repeatedly train on the same sample, so in these cases we select at most the number of samples in the dataset as the training data. PatchCore also has different parameters, since it's not training a ResNet18 architecture. This includes features like embedding dimensionality, number of neighbours used during inference, patchsize and feature retention percentage.

For all numerical values (number of samples, Adam betas, learning rate, etc.) we use a logarithmic sampling scale. For classification, we pick from all combinations of the first, second and third ResNet layer, e.g. layer1 & layer2.

For contrastive learning, as per the paper, we utilize a data augmentation strategy to allow for a self-comparison. We use the default parameters of RandAugment[4] from the torchvision library[17].

C Result table

Table C.1: The table of results for the scavenge port classification dataset. The metric is Area Under the ROC Curve (AUC). There's two metrics, one for the development set of the optimal OPTuna model, and one using that same model evaluated on the TEST set.

clf-ds	model	bb-model	bb-ds	auc@OPT	auc@TEST
scavport	patchcore	imagenet	imagenet	0.91	0.91
scavport	patchcore	T-S@scavport	scavport	0.73	0.65
scavport	patchcore	self-sup-con@scavport	scavport	0.90	0.89
scavport	patchcore	self-sup-con@vesselarchive	vesselarchive	0.90	0.87
scavport	patchcore	sup-con@scavport	scavport	0.92	0.92
scavport	T-S	imagenet	imagenet	0.78	0.73
scavport	T-S	T-S@scavport	scavport	0.68	0.61
scavport	T-S	T-S@vesselarchive	vesselarchive	0.67	0.61
scavport	T-S	self-sup-con@scavport	scavport	0.73	0.70
scavport	T-S	self-sup-con@vesselarchive	vesselarchive	0.76	0.73
scavport	T-S	sup-con@scavport	scavport	0.75	0.73

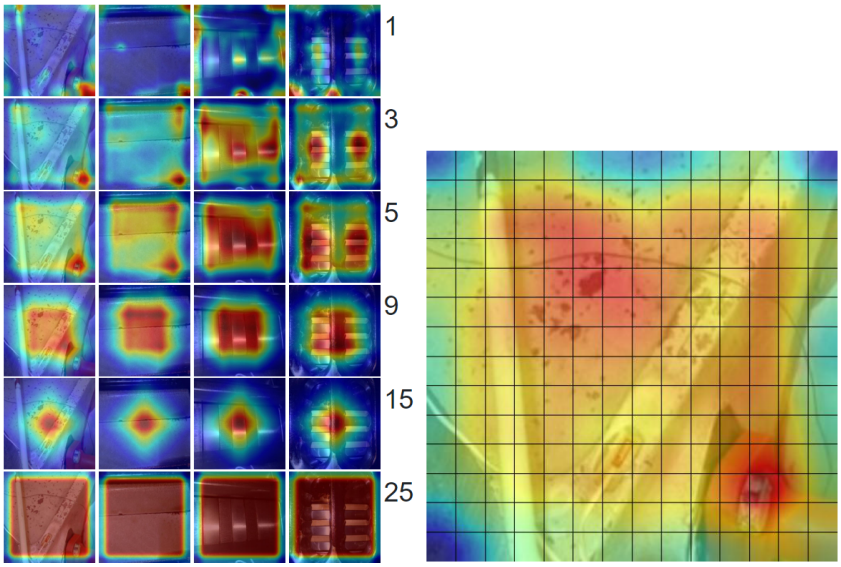
Table C.2: The table of results for the 4 condition sub-datasets. The metric is Area Under the Curve (AUC) of the operator receiver characteristics curve. There's two metrics, one for the development set of the optimal OPTuna model, and one using that same model evaluated on the TEST set.

clf-ds	model	bb-model	bb-ds	auc@OPT	auc@TEST
scavengeport-overview	patchcore	imagenet	imagenet	0.69	0.60
scavengeport-overview	patchcore	T-S@scavport	scavport	0.68	0.55
scavengeport-overview	patchcore	T-S@vesselarchive	vesselarchive	0.68	0.58
scavengeport-overview	patchcore	self-sup-con@scavport	scavport	0.66	0.58
scavengeport-overview	patchcore	self-sup-con@vesselarchive	vesselarchive	0.67	0.59
scavengeport-overview	patchcore	sup-con@scavport	scavport	0.75	0.63
scavengeport-overview	T-S	imagenet	imagenet	0.64	0.49
scavengeport-overview	T-S	T-S@scavport	scavport	0.65	0.50
scavengeport-overview	T-S	T-S@vesselarchive	vesselarchive	0.64	0.43
scavengeport-overview	T-S	self-sup-con@scavport	scavport	0.71	0.58
scavengeport-overview	T-S	self-sup-con@vesselarchive	vesselarchive	0.71	0.60
scavengeport-overview	T-S	sup-con@scavport	scavport	0.66	0.52
lock-condition	patchcore	imagenet	imagenet	0.81	0.81
lock-condition	patchcore	T-S@scavport	scavport	0.65	0.63
lock-condition	patchcore	T-S@vesselarchive	vesselarchive	0.69	0.69
lock-condition	patchcore	self-sup-con@scavport	scavport	0.80	0.78
lock-condition	patchcore	self-sup-con@vesselarchive	vesselarchive	0.74	0.74
lock-condition	patchcore	sup-con@scavport	scavport	0.83	0.83
lock-condition	T-S	imagenet	imagenet	0.65	0.58
lock-condition	T-S	T-S@scavport	scavport	0.76	0.65
lock-condition	T-S	T-S@vesselarchive	vesselarchive	0.67	0.59
lock-condition	T-S	self-sup-con@scavport	scavport	0.81	0.70
lock-condition	T-S	self-sup-con@vesselarchive	vesselarchive	0.70	0.52
lock-condition	T-S	sup-con@scavport	scavport	0.66	0.54
ring-condition	patchcore	imagenet	imagenet	0.93	0.90
ring-condition	patchcore	T-S@scavport	scavport	0.98	0.88
ring-condition	patchcore	T-S@vesselarchive	vesselarchive	0.99	0.82
ring-condition	patchcore	self-sup-con@scavport	scavport	0.94	0.51
ring-condition	patchcore	self-sup-con@vesselarchive	vesselarchive	0.95	0.80
ring-condition	patchcore	sup-con@scavport	scavport	0.93	0.71
ring-condition	T-S	imagenet	imagenet	0.68	0.61
ring-condition	T-S	T-S@scavport	scavport	0.98	0.57
ring-condition	T-S	T-S@vesselarchive	vesselarchive	0.99	0.94
ring-condition	T-S	self-sup-con@scavport	scavport	0.77	0.51
ring-condition	T-S	self-sup-con@vesselarchive	vesselarchive	0.79	0.61
ring-condition	T-S	sup-con@scavport	scavport	0.90	0.18
ring-surface-condition	patchcore	imagenet	imagenet	0.98	0.88
ring-surface-condition	patchcore	T-S@scavport	scavport	0.88	0.37
ring-surface-condition	patchcore	T-S@vesselarchive	vesselarchive	0.92	0.53
ring-surface-condition	patchcore	self-sup-con@scavport	scavport	0.95	0.84
ring-surface-condition	patchcore	self-sup-con@vesselarchive	vesselarchive	0.99	0.86
ring-surface-condition	patchcore	sup-con@scavport	scavport	0.98	0.79
ring-surface-condition	T-S	imagenet	imagenet	0.96	0.72
ring-surface-condition	T-S	T-S@scavport	scavport	0.86	0.55
ring-surface-condition	T-S	T-S@vesselarchive	vesselarchive	0.91	0.44
ring-surface-condition	T-S	self-sup-con@scavport	scavport	0.94	0.66
ring-surface-condition	T-S	self-sup-con@vesselarchive	vesselarchive	0.98	0.48
ring-surface-condition	T-S	sup-con@scavport	scavport	0.96	0.70

D Resolution and patchsize

Below we visualize the activations of a PatchCore model on only the third layer (14x14 resolution) to classify 4 images from the Scavenge Port dataset. Each row has the same 4 images, but the PatchCore model has a different patch size. Since the third layer has a spacial activation dimensionality of 14x14, a patch size of 30 means any of the 14x14 pixels will span the entire image. We can observe this behaviour in the 25 row below, where only the border is non-red, as the pixels on the left border does not contribute to the average of the pixels on the right border. This reduces the anomaly score, as the average is lowered due to zero-padding. The patch size 15 image has exactly one square which is red for the 4 images, which can be explained in a similar fashion as above. This square is the only one which covers the 14x14 resolution when the patch size is 15, and therefore has the least padding, resulting in the highest average.

Below we also add gridlines to an image which visually demonstrate these 14x14 cells after they have been upscaled and smoothed. The perceived higher fidelity of pixels not aligning with the grid is an artifact of the bilinear interpolation, along with the Gaussian filter applied.



(a) Example effects of increasing patch-sizes

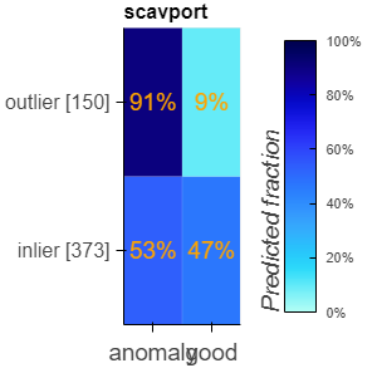
(b) 14x14 grid overlayed a heatmap

Figure D.1: Heatmap visualisations of the PatchCore activations in the third layer. Red indicates the largest activations, and blue indicates the lowest.

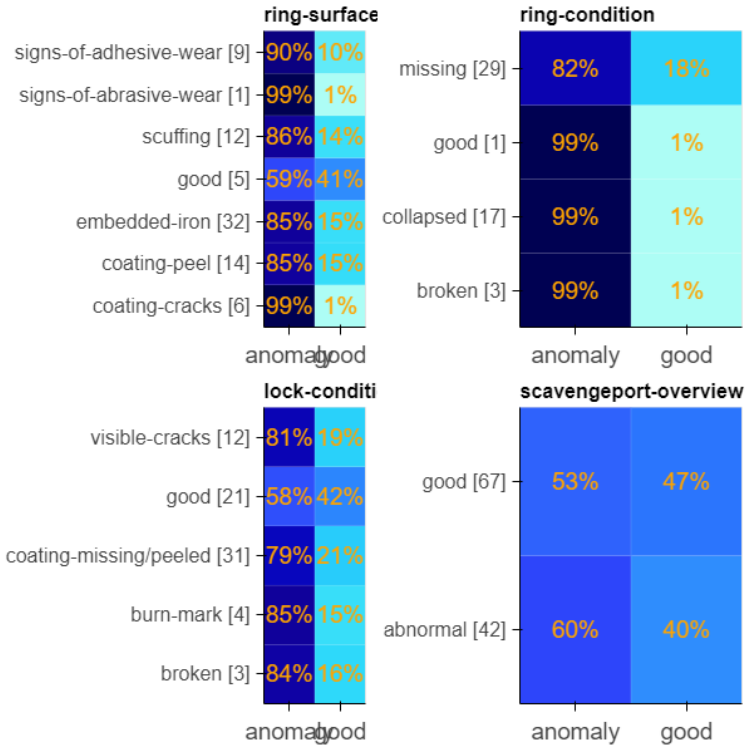
PatchSize	Anomaly map agg. func	AUC
29	max	0.91
29	mean	0.91
1	max	0.64
1	mean	0.83

Table D.1: Trainimg different PatchCore models using the optimal hyperparameters but adjusting the anomaly map aggregation function and the patch size. The "mean" & patchsize=1 is almost equivalent to having a patchsize of 29 when the resolution is 14x14. The difference can be attributed to zero-padding. The backbone used is the baseline ImageNet weights.

E Confusion matrices

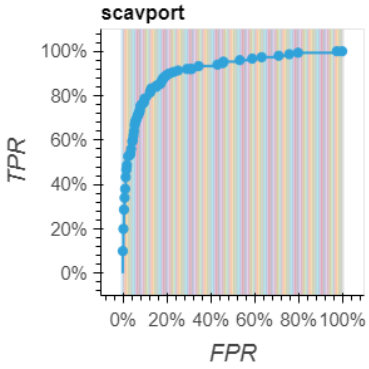


(a) Confusion matrix - Scavenge port

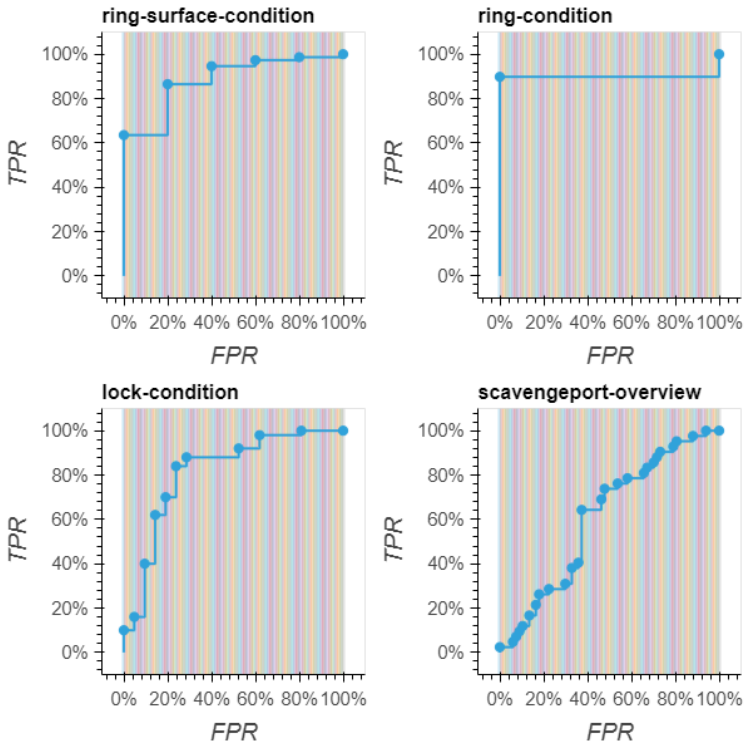


(b) Confusion matrices - Outlier datasets

Figure E.1: Figures E.1a and E.1b show the *[total]* and percentage of predictions being anomalous/good per class when using the PatchCore baseline model (ImageNet). The percentages are de-noised by averaging 100 anomaly thresholds along the False Positive Rates (FPR) x-axis of the ROC curve as shown in figures E.2a and E.2b. By definition, this should theoretically result in a FPR of 50%, i.e. the "good" class is expected to have a 50/50 split, however, due to the discrete sampling and low sample size, this is not the case.



(a) ROC curve - Scavenge port



(b) ROC curves - Outlier datasets

Figure E.2: E.2a and E.2b show ROC curves corresponding to the figures of E.1a and E.1b. Each vertical line segment corresponds to a sampling at some threshold. Specifically, the threshold is found at the left point of the line segment intersecting the vertical line and the the ROC curve frontier. This threshold is used to classify the anomaly scores as either anomalous or not, which are appended to the results of the confusion matrices. After having performed 100 samplings, the confusion matrices will therefore contains 100 times the samples as if we had only used a single threshold value, e.g. the average, while being less prone to variability compared to selecting an arbitrary threshold value.

F Self-supervised contrastive learning cluster

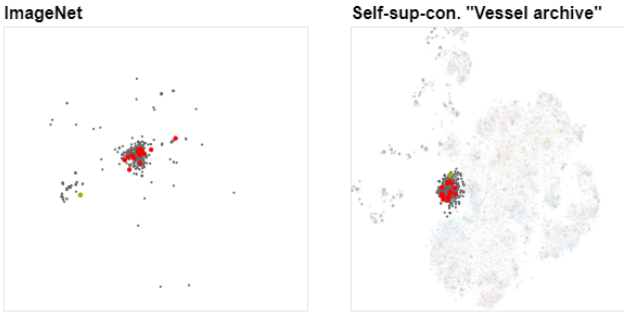


Figure F.1: A selection of samples from the t-SNE figure 5.2 using two of the embedding backbones. The right image is used to make the selection, where the transparent samples are excluded. The left image shows only the samples selected in the right image. The clustering of the left image means that no novel cluster was formed by employing the self supervised algorithm.