

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

Anomaly Segmentation Using Neural Networks

Master's Thesis

BC. VÁCLAV HLOUŠEK

Brno, Fall 2023

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

Anomaly Segmentation Using Neural Networks

Master's Thesis

BC. VÁCLAV HLOUŠEK

Advisor: doc. RNDr. Pavel Matula, Ph.D.

Department of Visual Computing

Brno, Fall 2023



Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Václav Hloušek

Advisor: doc. RNDr. Pavel Matula, Ph.D.

Acknowledgements

I express my gratitude to Pavel Matula for his advice and help.

Abstract

The goal of this thesis is to examine recent successful anomaly detection methods which use neural networks in semi-supervised mode. These methods aim to detect anomalies on images from industrial settings. In this setting the anomalies are usually difficult to detect, hence the methods are required to have high accuracy. Furthermore, the methods locate the anomalies so their output is in the form of a heatmap. I provide an in-depth explanation of four neural network methods and the results of the experiments on them using public and externally supplied datasets.

Keywords

Anomaly Detection, Machine Learning, Neural Networks, Image Processing

Contents

1	Introduction	3
1.1	Goals	3
1.2	Challenges	4
I	Theoretical Part	6
2	Anomaly detection	8
2.1	Anomaly Detection in Broader Context	8
2.2	Anomaly Detection and Segmentation	9
2.3	Outputs of Anomaly Detection Methods	10
2.4	Anomaly Detection Surveys	10
3	General Methods of Anomaly Detection Using Neural Networks	12
3.1	Density Based Models	12
3.2	Classification Based	18
3.3	Reconstruction Based	20
3.4	Distance Based	21
II	Practical Part	24
4	Metrics	26
4.1	F1 score	26
4.2	Area Under the Receiver Operating Characteristics	27
5	Datasets	30
5.1	MVTecAD	30
5.2	FlangeAD	32
5.3	AutoAD	34
6	Experiments	35
6.1	Framework	35
6.2	FlangeAD	36

6.3	AutoAD	37
6.4	MVTecAD	39
6.5	Discussion	39
7	Conclusion	40
8	Supplementary	41
8.1	MVTecAD	41
8.2	FlangeAD	47
8.3	AutoAD	50

Chapter 1

Introduction

This thesis investigates image anomaly detection using neural networks in industrial settings. These anomalies are in a form of scratches, dents and other imperfections on wide range of devices in manufacturing. These types of anomalies are usually more challenging to detect than, for example, anomalies in the form of unsafe or harmful items inside of a baggage during an airport security check. This is mainly because of their generally small size and the fact that they often look quite similar to the non-anomalous areas of the image. Developing and researching new automatic methods for detection of these defects is highly demanded since it is more time efficient than manual inspection and in many cases also more accurate. The topic of this thesis focuses on neural network methods which use only anomaly-free images as inputs for training (this is called semi-supervised setting). After being trained on these images the network should distinguish between images with and without anomalies. It is also usually required of the network to locate the potential anomalies, hence the output is in a form of binary mask and/or a heatmap. This work was conducted with the cooperation of the industrial partner.

The main goal of the theoretical part is to examine a recent scientific papers and choose the most successful methods of image anomaly detection. In the practical part I perform experiments of these methods on the public dataset and the dataset provided by the industrial partner. For the provided data I create the scripts so the images are fit for the subsequent experiments. I also develop a script for the visualisation of the result of the experiments so they can be easily examined.

1.1 Goals

The objectives of the theoretical part are as follows:

- Research successful image anomaly detection methods that are using neural networks. These methods should use only the anomaly-free images for

training. They should also be targeted to work with anomalies occurring in industrial settings.

- Explore the topic of the visual anomaly detection and compile an informative text on it. The text should be easily accessible to a reader new to the topic. Choose the interesting and/or successful methods and explain how they work and the main principles they use for anomaly detection.
- Categorise these investigated methods based on their methodology. From each category pick a representative, best-performing method fit for industrial anomaly detection method and explain it in depth.

The practical part has following objectives:

- Implement a script for pre-processing the data provided by the industrial partner. Since the images in these datasets are quite large it is needed to implement a patching algorithm. The neural network methods will then be trained on these patches. Also, the datasets are not prepared to be used in a semi-supervised setting hence the script should divide the anomalous patches from the rest so the methods can train on them.
- For each representative method chosen in the theoretical part run the experiments on the public anomaly detection dataset and pre-preprocess dataset from the industrial partner.
- Compare the performance of the methods and discuss the results. Investigate the advantages and disadvantages of each method.

1.2 Challenges

Main challenge of the theoretical part was to orient myself in the recent scientific papers approaching the topic at hand. The amount of these papers can seem overwhelming at times. It also takes quite a lot of experience or time to ascertain if a particular paper is as good as its authors say it is. Reading the image anomaly detection surveys helped somewhat. However, the authors of these surveys are oftentimes contradicting other surveys or disagreeing on the taxonomy.

Another challenge is somewhat related to the one above. The authors of machine learning papers usually do not provide the implementation of their methods to back up the reported results. This is quite detrimental given the non-deterministic nature of their research. There are machine learning papers which do not provide enough information about the implementation of their methods but inform us of high accuracy scores. Such papers can be seen as questionable. Even more so when third-party implementation of the said methods do not amount to the same results. These outside implementation can be found on

GitHub and are developed by enthusiast. It is not rare to see that the performance of these models is few percent lower than the results in the original papers.

The challenges relating to the practical part of the thesis are no less significant. The sheer amount of data provided by the industrial partner and the difficulties with running the software on the external GPU servers were main impediments of the experiments' progress. Another practical challenge lies in time consuming nature of training of neural networks.

Part I

Theoretical Part

This part specifies the problem domain that this thesis is trying to tackle. It also contain a short presentation of the image anomaly detection. Then follows a categorisation of researched methods and a comprehensible introduction of each category. Within each category I include one method and I explain it more thoroughly.

Chapter 2

Anomaly detection

Anomaly detection is the task of deciding whether a particular sample is "not like the others", or more formally denoted as, *out of distribution*. In recent years the number of researchers engaging in developing methods to detect anomalies has been on the rise. This is mainly because of the unprecedented results of deep learning¹ which surpass traditional methods in recent years not only in anomaly detection [Chalapathy and Chawla, 2019] but in other domains, mainly in those where we work with large dimensionality of data. The goal of this chapter is to present, categorise and evaluate recent successful deep learning models tasked with sensory anomaly detection and segmentation in image data.

2.1 Anomaly Detection in Broader Context

With the surge of popularity also comes a need to tightly define a problem domain of anomaly detection. This section will shortly summarise tasks and contexts associated with detecting anomalies and will then fit it will define a topic of this thesis more precisely.

Firstly, anomaly detection is applied in myriad domains. These include, for example, fraud detection (in telecommunication, banking...), spam detection on social media, anomaly detection on video surveillance or on protein sequences. This thesis deals with image data, particularly in an industrial setting.

Anomaly detection (AD) can be divided into two categories based on the type of anomaly we are looking for:

Sensory Anomaly Detection: The model is tasked with detection of sensory anomalies. Sensory anomalies are tied to the lower levels of the feature hierarchy of an image, such as edges and textures while the overall image does not significantly deviate from other images. These anomalies can be, for instance, scratches on a ceramic surface or tears in fabric.

¹Deep learning refers to a family of methods utilising neural networks with multiple hidden layers.

Semantic Anomaly Detection: The model detects samples that are of a label other than normal. In this context, these samples are called semantic anomalies. Semantic anomalies refer, in contrast to sensory anomalies, to high-level features, such as scenes or objects in an image. One example of a utilisation of this approach is as follows: we have a set of images of dogs which, however, also contain images of other animals. The goal of the method would be to mark all the non-dog images as anomalies.²

From the nature of sensory AD, where the normal and anomalous images differ only in details, this task is more demanding for a particular model capability, memorising normal data distribution and faster inference times [Jiang et al., 2022]. Both sensory and semantic AD fall into a family of methods that tackle a problem of detecting out of distribution samples. This family also includes other groups of methods. For example open-set recognition and outlier detection:

Open-set recognition: One of the problems of machine learning models is that they tend to assign training labels to previously unseen samples with high confidence [Dhamija et al., 2018]. Open-set recognition tries to remedy this. The model works as a standard image classifier; it assigns training labels to the testing samples, but marks samples not seen during training as anomalies.

Outlier detection: Does not follow train-test procedure but directly processes all samples and detects those that differ from the rest.

Deep learning anomaly detection models can be trained in different settings depending on available labels for learning: *supervised*, where the model is presented with labels to each image during training. *unsupervised*, where the model does not have access to labels but needs to implicitly decide on normality of a training data. In image data, and mainly sensory AD, it is very often the case that we lack images of anomalies that we could use for training. For this reason these models are mainly developed for training in a *semi-supervised*³ setting. That is, during training we observe only normal images. All examined models in this thesis are trained in semi-supervised setting.

2.2 Anomaly Detection and Segmentation

The problem of finding anomalies in image data can be subdivided into two categories: *image-level AD* and *pixel-level AD* (also sometimes called anomaly segmentation). Broadly speaking, the task of image-level AD aims to decide whether an input image contains an anomaly or not, hence this decision is strictly binary for each image in a testing dataset. Pixel-level AD wants to decide, for each pixel, whether this pixel is anomalous or not giving us the localisation of anomaly regions in the image. It is not hard to see that if we

²Note that sometimes in literature semantic anomaly detection is called *novelty detection* [Jingkang et al., 2021]. And some authors distinguish between these two in terms of motivation, where novelty is not seen in a negative sense but are appreciated as learning resource and is incorporated into the learning process once discovered [Chalapathy and Chawla, 2019]. However, we can also see these two terms used interchangeably.

³In some literature this category is interchangeably used with the *unsupervised* category.

solve a problem of pixel-level AD we also get a solution for image-level AD. If the input image contains (enough or any) anomalous pixels then it can be marked as an anomaly.

2.3 Outputs of Anomaly Detection Methods

In practice the output of methods dealing with AD is an *anomaly score*; either for a whole image or for each pixel. This score is usually re-scaled to a range $(0, 1)$ giving us information as to how much a certain image or pixel is deemed anomalous by the network percent-wise. We can then decide on a suitable threshold for marking images or pixels anomalous. Or in case of pixel-level AD, where we have an anomaly score for each pixel, we don't threshold these scores and we get an anomaly heat-map which can be favourable over a binary mask.

2.4 Anomaly Detection Surveys

Many surveys have been published in the field of general AD across multiple domains. The work of [Chalapathy and Chawla, 2019] was the first one to survey deep learning methods for AD. The survey is done with respect to many applications such as fraud detection and video surveillance.

[Ruff et al., 2020] divide AD techniques into classification, probabilistic⁴ and reconstruction methods and in each of these discusses comprehensive connections between classic and deep learning approaches. For example, connections between PCA and autoencoders or between kernel density estimation and a method of normalising flows (these will be noted later in the text). The survey also makes note of the distance based approach as a secondary category to the proposed taxonomy which is, nonetheless, on the same level with other classes of methods in this thesis.

In the visual AD domain the work of [Jingkang et al., 2021] presents an extensive survey where the authors distinguish between sensory and semantic AD, depending on whether the task is aiming to detect a covariate or semantic shift in a dataset respectively. Together with *open set recognition* and *outlier detection* the authors envelop these categories under the name of *generalised out-of-distribution detection* and divide them further, similarly to the previous survey.

[Salehi et al., 2021] leaves behind an idea of categorisation made by previous surveys, finding connections between these methodologies as well as between categories in *generalised out-of-distribution detection* and introduces a closer study of each method through more mathematical approach.

[Jiang et al., 2022] is the first survey for sensory AD and categorises recent methods into semi-supervised⁵, weakly supervised (where labels are available,

⁴This terminology is identical with *density* category in this work.

⁵In a survey authors label this category as *unsupervised* and mentioned that only normal samples are used during training, hence it is written here as *semi-supervised*

but incomplete) and supervised learning. The authors also note that the semi-supervised category got the most attention in the research community in recent years mainly due to public dataset MVTecAD [[Bergmann et al., 2019](#)].

Chapter 3

General Methods of Anomaly Detection Using Neural Networks

Methods engaging in anomaly detection of image data using deep learning can be generally divided into four categories: Density Based, Reconstruction Based, Classification Based and Distance Based.

For each of these categories I will describe general strategies, introduce several interesting examples and characterise one recent and successful method whose results will be a topic in the practical part of this thesis.

3.1 Density Based Models

These methods aim to learn the probability distribution of the normal images denoted by $p(x)$. If we model this distribution, then test time samples with low log-likelihood will be marked as anomalous.

Images as samples of probability distribution

Image describes which pixel will have what values. The number and position pixels is pre-defined. As are the values which the pixels can have. For example we can work with an image of size 256×256 (65536 pixels) where each position can gain a integer value from interval of $I = [0, 255]$. We can create a model that would describe images based on their probability under this model since each image (a set of certain values for each pixel from the interval I) will be less or more likely to appear. One of the simplest models is a uniform distribution $U[0, 255]^{65536}$. Under this model the likelihood for each pixel to obtain a value from the interval I is the same ($1/256$) and random variables associated with each pixels are independent, meaning they do not influence each other. By

sampling from this model we would get a noise image and it is easy to see that it is very improbable that we would generate an image of, for example, a car. To reliably get an image of a car we need a different (and more complicated) distribution. These distributions can be described by deep learning models if they were successfully trained. The task of these models is twofold: first, given an image, we can ask the model if the image belongs to the learned distribution. In this case the model will output a likelihood (or mainly in practice the log-likelihood). Secondly, we can ask the model to generate a new image. More precisely, to sample from the learned distribution. As we will see, some neural network models can do only one of these tasks.

The core idea of density based methods is to use a training set of images and model their distribution.

There are few general sub-methods: Autoencoder and variational autoencoders, generative adversarial networks and normalising flows.

One can use classic density estimation methods (without the usage of neural network) to detect anomalies, for example, by fitting multivariate Gaussian over the training data¹ and use a Mahalanobis distance between a test sample and modelled distribution [Leys et al., 2018]. Mahalanobis distance can be considered a measure of how much a certain value fits into a learned distribution; if this distance is too large then the testing data point is considered anomalous. Other methods use mixed Gaussian distribution² [Eskin, 2000] to fit the training data or use kernel density estimation³ [Hu et al., 2020]. On their own, these methods are not suitable for high dimensional data samples such as images. However, we can still make use of these methods, if we use deep learning methods to lower the dimensionality of training data as we will see in the next section.

Autoencoders and Variational Autoencoders

Autoencoder (AE) is a type of neural network model comprising of two parts: encoder and decoder. Encoder maps high-dimensional (image domain) space into latent low-dimensional space. Decoder maps a latent space back into the high-dimensional one. We train this model to reconstruct images. This is done by implementing a loss function for reconstruction which is usually some sort of distance between an input and output images. This method is similar to principal component analysis and is also useful for data compression due to the lower dimensionality of the latent space.

¹Having the set of random vectors, this is simply done by computing mean value of each component of the vector and covariance matrix, telling us covariance between each pair of these components.

²The distribution comprised of several Gaussians, each with its own set of means and covariance matrices giving us more expressive models compared to fitting only one Gaussian.

³This is a non-parametric method closely related to histogram smoothing of training data. Intuitively and concisely explained, in a 1D case we would stack each data point as a building block on a x axis; this, after smoothing, gives us a particular curve which acts as a density function modelling a distribution the original data came from.

Variational autoencoder (VAE) [Kingma and Welling, 2013] has a similar topology but the encoder part maps an image into a normal distribution, represented by mean and standard deviation in the latent space, which is then sampled from and decoded back into the image domain. We can use this learned distribution and decoder part of the model to generate new images.

The latent space of AE/VAE networks is suitable for density estimation and anomaly detection by classical methods mentioned above [Zong et al., 2018] [Abati et al., 2019].

Generative Adversarial Networks

Generative adversarial networks [Goodfellow et al., 2014] (GANs) generally comprise of two sub-networks: generator G and discriminator D . The inputs of G are from low-dimensional noise distribution and the output is a target image domain. D takes images on the input and outputs a binary decision. Informally speaking G tries to create fake images from target distribution $p(x)$ and confuses D into thinking they are real. D on the other hand is trained to decide if the image on the input is real or not. D is trained alternately on real (training) set of images and those generated by G . After successful training, G will generate believable images similar to the original training set.

Unlike VAE, GANs lack an encoder part of the model, hence to use these models for AD we would need an inverse of G to map a test image to low-dimensional space where it is easier to compute its likelihood under latent distribution of the training set. ADGAN [Deecke et al., 2019] tries to remedy this problem by iteratively finding a point in latent space for a testing image.

Normalising Flows And FastFlow

This section will need a little bit of preface explanation. Our goal is to have a probability distribution function. Hence, for each input of that function (an image) we will get a likelihood value. To start simpler we can imagine a 1D probability distribution function (for an 1 pixel image). We do not have this function so we can not ascertain which values of pixels are anomalous and which are not. We add another probability distribution function. This one will be known and simple: a Gaussian distribution, a bell shaped curve. We can easily compute the values of this function for each value of the pixel. Now, let's have a mapping from the domain of the pixel values to the same domain. This mapping will transform the pixels which are under the Gaussian distribution to pixel which are under the goal distribution. For example, if our goal distribution would be, again, a Gaussian curve but shifted slightly to the right then the mapping would simply add a constant to each pixel value.

It is clear that for more complex goal distribution the mapping would be harder to find (and, unlike in the example, we do not know the goal distribution). For now, let's assume that we know such mapping.

Just by having a known Gaussian distribution and the mapping does not mean that we know the goal distribution. To solve this problem, the normalising flow

mechanism uses a certain mathematical theorem. According to this theorem, if the mapping is invertible, we can compute the goal distribution using the Gaussian distribution and the mapping:

$$p_X(x) = p_Z(f^{-1}(x)) \cdot \left| \frac{\partial f^{-1}(x)}{\partial x} \right|$$

Where x are the values of the pixel, $p_X(\cdot)$ is the goal probability distribution, $p_Z(\cdot)$ is the pre-defined simple distribution (Gaussian in our example) and f is the invertible mapping. Moreover, $f^{-1}(x) = z$ are samples from the Gaussian distribution.

Let's write this equation again by substituting $g = f^{-1}$:

$$p_X(x) = p_Z(g(x)) \cdot \left| \frac{\partial g(x)}{\partial x} \right|$$

To put this into the perspective of our example, we now have the invertible mapping which transform the domain from having a Gaussian distribution to have our goal distribution. The x_0 is a pixel value for which we need to know if it is anomalous or not, in other words: its likelihood. Just by plugging this x_0 to the right side of the equation above we get the desired result on the left.

The problem of this procedure is that we do not know the mapping $g = f^{-1}$. We will use a neural network to represent this mapping. We have only anomaly-free images during training, according to semi-supervised settings. Hence, this neural network will try to maximise the likelihood $p_X(x)$ of the inputs x . That is exactly the right side of the equality and will act as a learning objective of the training process. This way, the properly learned g will act exactly how we want it to.

In the context of FastFlow model [Yu et al., 2021] the x 's are features acquired through some pre-defined image feature extractor and z are samples of multi-variable Gaussian distribution of the same dimensionality as x . As said earlier, g will be described by the neural network. More precisely, g is implemented as a combination of simpler reversible functions (blocks) g_k with their own nets, since it is generally difficult to find one non-linear reversible function g [Kobyzev et al., 2021]. From the formula, which acts as a learning objective, it is clear that the Jacobian $\left| \frac{\partial g(x)}{\partial x} \right|$ should be easy to compute, since it will be needed with each pass through the network in the learning process. The decomposition of g to g_k does not make this process much more difficult since the final Jacobian of g will be only the multiplication of Jacobian of each g_k . The question remains how to implement these g_k function such as they are sufficiently expressive, invertible and their Jacobians are easy to compute

FastFlow utilises *affine coupling layers* [Dinh et al., 2014] where the channels of input x of a particular block g_p are firstly permuted and then the input is

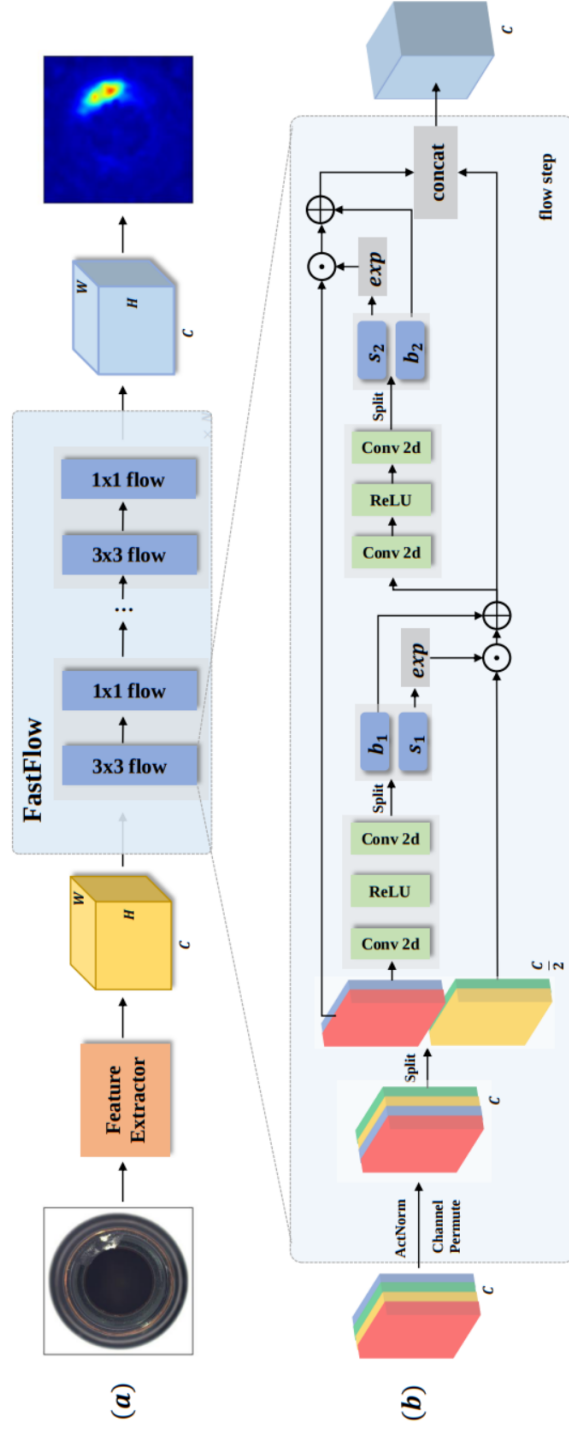
split into two x_A and x_B channel-wise. g_p is defined separately on these parts as follows:

$$x_A \xrightarrow{g_p} x_A$$

$$x_B \xrightarrow{g_p} s(x_A) \odot x_B + b(x_A)$$

Where s and b are convolutional neural networks. After this the two parts are again concatenated together and send to another block. This definition of g_p guaranties that the computation of the Jacobian is not demanding but the function is sufficiently expressive.

The following image explains the whole network:



3.2 Classification Based

Models of this category, in contrast with density based models, are not trying to learn a whole distribution of samples but are finding a decision boundary between anomalous and normal images.

Some methods in this category utilise feature adaptation, for example Deep SVDD [Ruff et al., 2018], which trains a network that maps the features of normal images, such that a radius of a hypersphere bounding these mappings is minimal. At test time, the features of the image go through the network and if they lie outside of the learned hypersphere then the model marks them as anomalous.

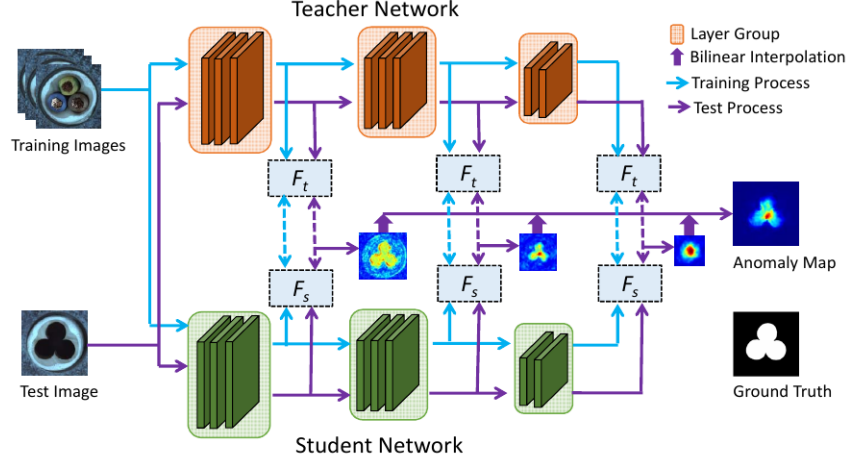
PANDA [Reiss et al., 2020] follows up on this method and attempts to negate a case where the networks learns a trivial solution that maps everything to one point resulting in a collapse of the hypersphere to a single point. To do this, PANDA introduces a continual learning method. The parameters of the network which are deemed important in the sense of a feature description capability of the network are forced to not change significantly during training.

CutPaste [Li et al., 2021] is using self-learning by data augmentation to find a decision boundary. Firstly it creates fake anomalies by cutting a random patch from a training image to a random location, then it implements a standard classification model which distinguishes between these fake anomalies and normal samples.

The work of [Wang et al., 2021b] uses discrepancy between features of a patch at some position of an image and features of the rest of the image. The patch goes through the net which has the patch on the input and extracts its features. Parameters of this network are fixed during the training of the model. The rest of the image goes through the global net which is trained by minimising the distance between the outputs of these two networks. At test time, if the distance between outputs of these networks is too large then the sample is deemed anomalous.

Student-Teacher Feature Pyramid Matching

Another way to find a decision boundary is by knowledge distillation. Student-Teacher Feature Pyramid Matching (STFPM) [Wang et al., 2021a] achieves state of the art results using this procedure. The method introduces two neural networks of the same topology: Teacher (T) and Student (S). T network is pre-trained on image classification task, e.g. ImageNet, and its parameters are fixed during the training of the model. By implementing a particular loss function the S network aims to imitate T on normal training images across its feature maps in multi-scale fashion. After training, S will have multi-scale knowledge from the feature pyramids which will help it to detect anomalies of different sizes. At inference the same loss function will be used to discern discrepancies between feature maps of the Student and the Teacher.



Now, formally the loss function is described by:

$$F_t^l(\mathbf{I})_{ij}, F_s^l(\mathbf{I})_{ij} \in \mathbb{R}^{d_l}$$

Where F_t^l are vectors from feature map l at position i, j of T and S respectively, where the features map is of depth d_l and $\mathbf{I}$ is an input image.

These feature vectors are then normalised:

$$\hat{F}_t^l(\mathbf{I})_{ij} = \frac{F_t^l(\mathbf{I})_{ij}}{\|F_t^l(\mathbf{I})_{ij}\|}, \quad \hat{F}_s^l(\mathbf{I})_{ij} = \frac{F_s^l(\mathbf{I})_{ij}}{\|F_s^l(\mathbf{I})_{ij}\|}$$

And loss for the position i, j and layer l is defined as a l_2 distance of T and S normalised feature vectors:

$$\ell^l(\mathbf{I})_{ij} = \frac{1}{2} \|\hat{F}_t^l(\mathbf{I})_{ij} - \hat{F}_s^l(\mathbf{I})_{ij}\|_{l_2}$$

The loss for an image at the layer l is defined as average through the dimensions of the feature map w_l, h_l :

$$\ell^l(\mathbf{I}) = \frac{1}{w_l h_l} \sum_{i=1}^{w_l} \sum_{j=1}^{h_l} \ell^l(\mathbf{I})_{ij}$$

The total loss is then a linear combination of losses through different scales weighted by hyperparameters α_l .

$$\ell(\mathbf{I}) = \sum_{l=1}^L \alpha_l \ell^l(\mathbf{I}), \quad \alpha_l \geq 0$$

During inference, to get the anomaly map of a test image $\mathbf{J}$, the 2D images $\ell^l(\mathbf{J})_{ij}$ (of different sizes depending on how deep in the neural network the layer is) are upsampled to have the same height and width. Then follows pixel-wise multiplication of these upsampled images to get the final anomaly heatmap.

3.3 Reconstruction Based

The base idea of these methods is that by training on anomaly-free images, the network is not able to reconstruct anomalies during inference. They usually implement an autoencoder (AE) scheme. But unlike VAE from density based category here the method does not use a latent space for distribution modelling but uses its reconstruction capabilities for AD. At test time, a location of anomalous areas is usually done by taking a distance from the input image and its reconstruction with respect to some measure and through all pixels of the image.

The work of [Sabokrou et al., 2018] is implementing a discriminator network attached to the denoising AE (noise is added to the inputs of the AE to improve reconstruction capabilities) and adding adversarial training between these two networks.

To add to this AE and discriminator model, Ganomaly [Akçay et al., 2018] is utilising another encoder after the AE and is training the reconstruction model not only with respect to image domain but also in latent space. The loss function is added that measures the distance between latent space of the AE and the output of the added encoder.

The SCADN [Yan et al., 2021] is enhancing the learning of reconstruction of normal images by partially masking these images before entering the network, then using GAN to recover these missing areas

OCR-GAN [Liang et al., 2022] firstly separates the input image I with respect to frequency information to get a series of images $\mathbf{I}_k$ $k = 1, \dots, n$, where $\mathbf{I}_1$ carries the lowest frequencies and $\mathbf{I}_n$ carries the highest frequencies, such that $\sum_{k=1}^n \mathbf{I}_k = \mathbf{I}$. Each $\mathbf{I}_k$ is then inserted to the AE networks M_k , of identical topology, accordingly and subsequent outputs $\mathbf{O}_k = M_k(\mathbf{I}_k)$ are summed to get the final reconstruction $\mathbf{O} = \sum_{k=1}^n \mathbf{O}_k$. The model also learns channel-wise attention which accentuate different channels of feature maps through all M_k according to their importance.

Discriminatively Trained Reconstruction Anomaly Embedding Model

From now on mentioned as DRAEM[Zavrtanik et al., 2021] is one of the successful models in this category. As noted earlier, reconstruction AD methods often rely on a distance between inputs and their reconstruction to detect anomalies. But anomalies that are not significantly different from normal areas in the image are often well reconstructed, hence the distance of these reconstructed areas from the input is small.

To alleviate this, DRAEM aims to train its model to learn the measure, which will calculate the distance of a input and its reconstruction, by itself. The model contains two sub-networks: a reconstruction sub-network, with the same goal as AEs in previous examples, to reconstruct its original input, and a discriminative sub-network, which from the input and its reconstruction learns to output anomaly segmentation maps.

DRAEM also deploys anomaly generation module which adds simulated anomalies to the normal, training images. Since added anomalies are known, the module also outputs binary masks which marks the areas where the normal image was tarnished by simulated anomaly.

During training the image with simulated anomalies $\mathbf{I}_a$ enters the reconstruction sub-network which tries to inpaint these anomalies by combination of patch based SSIM and l_2 loss functions.

$$L_{SSIM}(\mathbf{I}_a, \mathbf{I}_r) = \frac{1}{N_p} \sum_{i=1}^H \sum_{j=1}^W 1 - SSIM(\mathbf{I}_a, \mathbf{I}_r)_{i,j}$$

where $\mathbf{I}_r$ is the reconstruction of $\mathbf{I}_a$, H is a height and W is a width of $\mathbf{I}_a$. N_p is the number of pixels in $\mathbf{I}_a$ and $SSIM(\mathbf{I}_a, \mathbf{I}_r)_{i,j}$ is the $SSIM$ value for patch between a patches of $\mathbf{I}_a$ and $\mathbf{I}_r$ centred at position i, j .

This patch based $SSIM$ loss is then combined with l_2 loss:

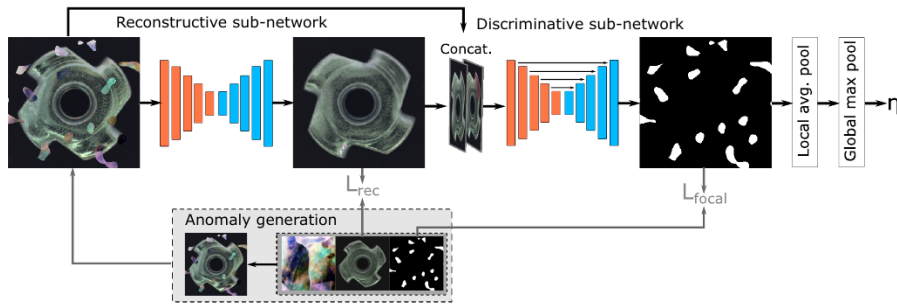
$$L_{rec}(\mathbf{I}_a, \mathbf{I}_r) = \lambda L_{SSIM}(\mathbf{I}_a, \mathbf{I}_r) + l_2(\mathbf{I}_a, \mathbf{I}_r)$$

where λ is a weighting hyperparameter.

$\mathbf{I}_r$ and $\mathbf{I}_a$ are channel-wise concatenated and inserted into the discriminative sub-network with a U-net architecture. This sub-network is trained by pixel-wise focal focus loss function [Lin et al., 2017] (denoted L_{seg}) between its output $\mathbf{O}$ and binary mask from the anomaly generation module $\mathbf{M}$. Final loss is then:

$$L(\mathbf{I}_a, \mathbf{I}_r, \mathbf{M}, \mathbf{O}) = L_{rec}(\mathbf{I}_a, \mathbf{I}_r) + L_{seg}(\mathbf{M}, \mathbf{O})$$

At inference the test image $\mathbf{J}$ goes through these two networks ($\mathbf{J}$ acts as $\mathbf{I}_a$ during training phase) resulting in an anomaly heatmap $\mathbf{O}$, which is an output of discriminative sub-network.



3.4 Distance Based

Generally employ some pre-trained network to features from training images to get a set of prototype vectors. At test time features are extracted the same way as in training and then the similarity of these features and training prototype

features is calculated. These methods differ from the previous ones in that there is usually no learning objective, we however still use feature extraction capability of neural network.

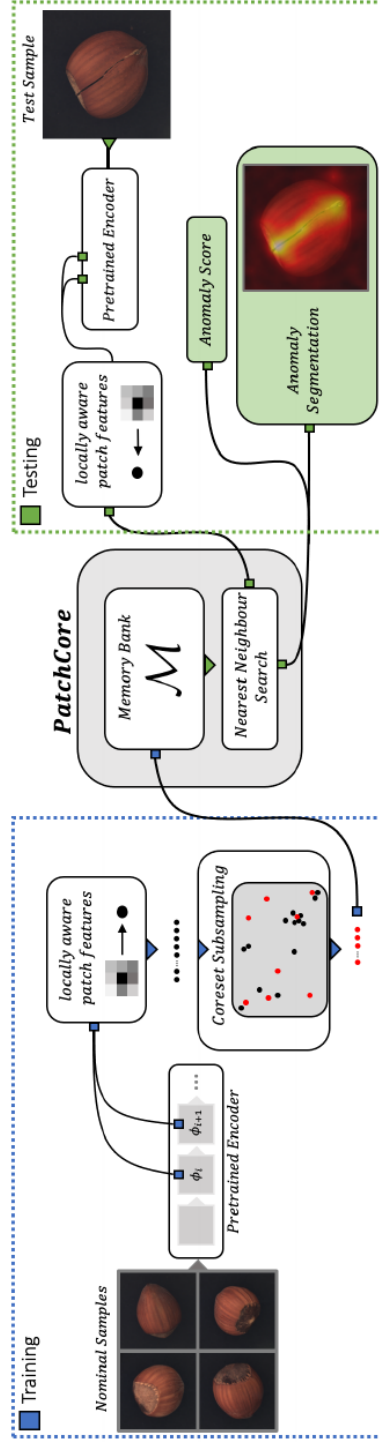
PatchCore

To get the feature information of the training images from the pre-trained network PatchCore uses feature maps from multiple layers to get the mix of low and high resolution features. The method employs context aware feature vectors. Denote a feature vector $v_{i,j}^l \in \mathbb{R}^{d_l}$, where d_l is a number of channels in the feature map of layer l and i, j is a position on the feature map. From this, one can define a context aware feature vector by aggregating feature vectors with respect to some neighbourhood (denoted $O_{i,j}$) as follows: $\hat{v}_{i,j}^l = f_{agg}(\{v_{s,t}^l; s, t \in O_{i,j}\})$, resulting in one-dimensional vector $\hat{v}_{i,j}^l$. Authors chose average pooling as the aggregation function f_{agg} .

To get a information of different resolution for a position i, j we concatenate $\hat{v}_{i,j}^{l_1}$ and $\hat{v}_{i,j}^{l_2}$ with linear re-scaling since there will be more context aware feature vectors of shallower layers then in deeper layers, which is given by sequential lowering of resolution of feature maps in standard CNN.

These vectors are then stored in the memory bank through all training samples. This bank is then reduced up to 1% of its original size thanks to greedy algorithm that assures that reduced bank will be satisfyingly representative.

During testing the method extracts context aware vectors of the test image in the same manner as in before and the anomaly score for each position is given by nearest distance of its feature vector from the memory bank.



Part II

Practical Part

The reader should now be informed of the problems and methods of visual anomaly detection. In the theoretical part I categorised methods tackling this task and choose one from each of them. The methods were mainly chosen because of their good performance on the public dataset. These four methods were examined and experimented using the public and externally provided datasets. The results and their discussions will now be provided to the reader as well as description of implemented scripts for data pre-processing.

Chapter 4

Metrics

In anomaly detection two metrics are usually used: AUROC and F1 score. Both of them are used in the experiments in versions for pixel-level and image-level AD. In pixel-level AD metrics are working with anomaly scores of pixel. From these score sets we get sub-sets sizes of interest such as *false positives*, *true negatives* and others. In image-level AD these sub-sets are acquired from anomaly scores of images. Both versions of metrics are calculated over the validation/testing batch and the final metric score is, in both cases, an average of the results from batches.

4.1 F1 score

An *accuracy*, which is a ratio of correctly assigned labels to the number of all samples, is not suitable for unbalanced datasets where we have only few samples of one class. Anomaly detection is usually one of these cases, hence we use usually use *F1 score*, which combine *precision* and *TPR*¹ (True positive rate)

$$precision = \frac{TP}{TP + FP} \quad TPR = \frac{TP}{TP + FN}$$

TPR naturally will be higher when we have more strict model that marks many samples as an anomaly. In an extreme case when everything will be deemed as an anomaly TPR will be 1. Precision on the other hand will usually increase with more lenient model, since it punishes false positives. F1 score balances these two measures using a harmonic mean.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{TPR}}$$

¹TPR is also called *recall* in some contexts.

Jaccard index is an intuitive way to judge the performance of a model, especially in images. If we would have a ground truth mask where A_{gt} would be a set of pixels that are truly anomalous and the result of the method, after thresholding, where A_{pred} would be a set of predicted anomalous pixels then a Jaccard index of this result would be computed as:

$$Jaccard = \frac{A_{gt} \cap A_{pred}}{A_{gt} \cup A_{pred}}$$

F1 score and Jaccard index are monotonic to one another meaning, by using them to evaluate machine learning models, they will sort them equally by their values [O'Connor, 2012]. Furthermore we can calculate Jaccard index from F1 score:

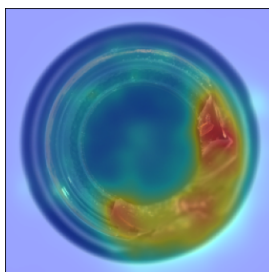
$$Jaccard = \frac{F1}{2 - F1}$$

4.2 Area Under the Receiver Operating Characteristics

F-1 Score presumes that we already have a threshold which divides anomalous samples from the rest. To decide where this threshold should be can be quite challenging and can be influenced by the desired performance of the particular model [Bergmann, 2022] (for example in some cases we want to permit less False positives.). This also makes it harder to compare two models with two different thresholds, hence it's reasonable to compare models with a metric that is removed from this decision. We use AUR (receiver operator characteristic) curve. This curve has FPR (False positive rate) on a X-axis and TPR on a Y-axis. FPR is ratio of false positives to all negatives and TPR is a ratio of true positives to all positives.

$$FPR = \frac{FP}{FP + TN}$$

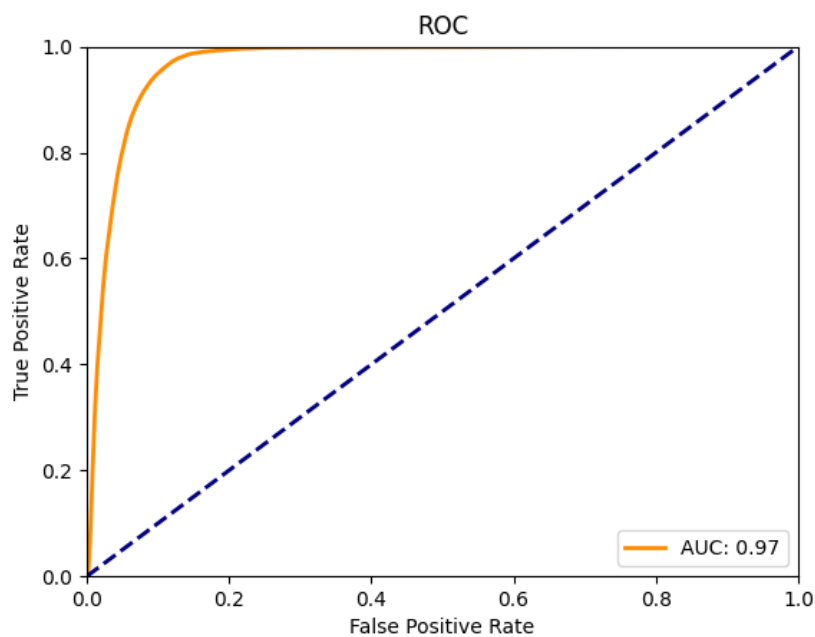
By thresholding the resulting score at different levels we get different points of this curve. In context of images we get heatmaps as the result of the inference. We also have a binary mask telling us which pixels are anomalous. We can imagine a threshold plane going up and down on the particular heatmap where everything above this plane is marked as an anomaly and everything below as normal pixels. By setting this plane at the bottom of the heatmap we get FPR = 1 and TPR = 1. As we are moving this threshold up more samples are marked as non-anomalous the FPR will be decreasing. A heatmap from well trained model will have high values where the true anomalous pixels are, hence in this example the TPR will not substantially decrease as we move the threshold plane up. In summary we hope that TPR will decrease much slower than FPR.



An example of the resulting heatmap overlaid with the original image.



Ground truth of the same image.



Example of fairly good ROC curve.

Examining the curve itself will give us an useful information about how good the model is. It can also be used to, for example, set the threshold line manually. The area under this curve (AUROC) is used as a comparative metric. AUROC

of 0.5 tells us that the model cannot differentiate between anomalies and normal pixels.

Chapter 5

Datasets

In this chapter I will describe each dataset, its size and procedure of preparation of it to act as an input for the subsequent experiments. The framework provided for the experiments require a certain structure for the training/testing data in semi-supervised mode. Specifically each dataset should be divided into *train* and *test* folders with the test folder further divided into *good* (anomaly free testing images) and *bad* (anomalous testing images); for the anomalous testing images we also need to provide annotation binary image of the same name contained in *ground truth* folder. The validation set is not created prior to running the experiment but is taken as a part of the testing set by the framework.

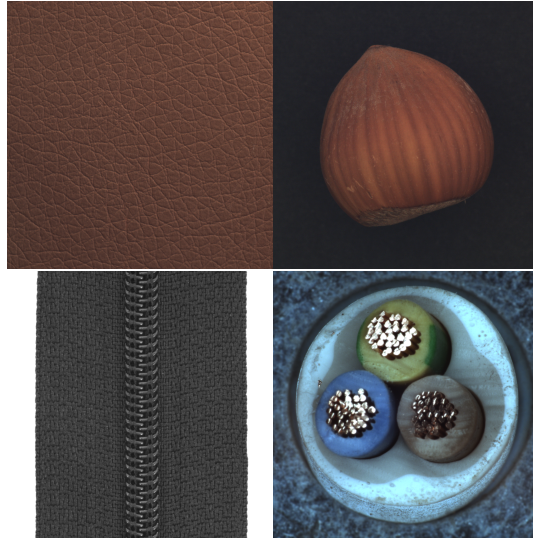
The experiments were done on three datasets: MVTecAD, FlangeAD and AutoAD. MVTecAD is a widely used public dataset. FlangeAD and AutoAD are datasets provided by the industrial partner; these datasets in accordance with non-disclosure agreement can not be presented here in their visual form.

5.1 MVTecAD

The standard and widespread dataset for evaluation and comparison of image sensory anomaly detection is *MVTecAD* [Bergmann et al., 2019] which is comprising of approximately 3600 training and 1700 testing images in 15 categories. The categories include relatively well-know objects such as *leather*, *hazelnut*, *zipper* or *cable*. With accordance of the semi-supervised nature of the problem, all training images are anomaly-free. Testing images come both with and without anomalies where those with anomalies are divided into sub-categories by the nature of the anomaly. For example a crack or discoloration in a category containing a medical *pill* or a spilled liquid or scratches in a *wood* category. Anomalous testing images have their own ground truth binary mask.

All images are of a squared shape and sizes of their sides are between 700 to 1024 pixels, however, the images are usually resized to 256x256 pixels size before entering the neural network. This is also the case in the experiments within this thesis. Images of three categories, including the *zipper* are grayscale. Otherwise

all images are RGB.



Examples of anomaly-free samples from *leather*, *hazelnut*, *zipper*, *cable* categories respectively.



A pill with a crack with its ground truth mask annotating the anomaly.



A discoloration of a pill.



A spilled liquid on a wood.



Scratches on a wood.

Category	Training size	Testing size
Grid	264	78
Zipper	240	151
Wood	247	79
Transistor	213	100
Toothbrush	60	42
Tile	230	117
Screw	320	160
Pill	267	167
Metal Nut	220	115
Leather	245	124
Hazelnut	391	110
Carpet	280	117
Capsule	219	132
Cable	224	150
Bottle	209	83

Table of numbers of all MVTecAD categories.

5.2 FlangeAD

FlangeAD dataset contains around 16000 images of pipe flanges. These metallic parts are used for connection between pipes or to connect pipes to other components in the system.

Each pixel contains normalised coordinates of the normal vector of the surface on the object. These coordinates aren't treated differently from RGB values during training/testing routines in subsequent experiments. Images are of various sizes, mainly 1456x1324, 1456x700, 2464x780. Because of the circular nature of pipe flanges not everything in the rectangular image is relevant for the training and is considered a background. For this reason each part of the flange is provided with a binary image acting as a mask and separating foreground and background. The dataset is not separated to normal and anomalous images, those images with anomalies are provided with a series of annotation binary images where each element of this series contains an annotation for a certain type of defect (pressure mark, scratch or damaged groove). Each image also contains anomalies which are deemed non-obstructive, these are not marked in the annotation images.

During the experiments it becomes apparent that this datasets is challenging. Mainly because of above mentioned irregularities of the material that do not count as anomalies in a testing process. Another reason is a variation in size of the true anomalies which can be as small as 15x15 pixels to encompass half of the 512x512 patch ground truth mask.

The developed script with the input directory containing the images does number of things to prepare the data for the models:

- The data does not have the needed structure described at the beginning of this chapter. It contains a large number of directories and sub-directories with images and associated annotations. These images are and their annotation are loaded by the script.
- Images are firstly masked with foreground/background masks and areas of background are set to zero values. There are 64 masks and be the name of a sub-folder of each image we can ascertain which mask to use.
- As opposed to MVTecAD input images are not resized but patched in order to preserve the fine texture of flanges' grooves. The size of the resulting patches¹ can be changed with the script argument; as can the stride with which the patching is done.
- Another argument provided for the patching algorithm is a minimum percentage foreground pixels within each patch. Patches which included large areas of background are discarded.
- Since we are only interested in binary decision for each pixel annotation series for a given image are merged and patched the same way a the associated image.
- All the patches are shuffled randomly.

¹This size has to be the power of 2 because of autoencoder nature of observed methods.

- Then we divide all patches into training set and testing set. In the training set all the patches are anomaly-free. These patches are removed from the testing set. Then the testing set is further divided into the images with and without anomalies. Because of the practical reasons the final number of patches is limited to 10000 training, 200 anomalous and 200 normal testing samples.

The resulting patches are of size 512x512 pixels and acquired with stride 64 and foreground ratio 95%.

5.3 AutoAD

AutoAD dataset contains images of three types of components, hence we train each model separately on each of the types. Images are of smaller and more convenient size compared to *FlangeAD*: 823x512 and 704x640 and their pixels are, as is the case with the *FlangeAD*, values of normalised normal vectors of the component surface. Images also contain foreground and background areas; these are already masked, hence background pixels are set to zero. This dataset appeared less challenging with more clearly defined anomalies compared to *FlangeAD*. It is also a lot smaller.

The included script processed this dataset as follows:

- Each component within one category has the same shape. The original structure of the data includes division into training and testing folders. This categorisation is of no use, however, since it does not follow semi-supervised mode. The script loads all the images homogeneously.
- The images are patched following manually pre-defined patch localisation and size specific to each category. This procedure is different from *FlangeAD* patching since *AutoAD* component parts are comparatively small and of more diverse shape with less foreground pixel ratio.
- For the last reason mentioned in previous point all the patches are included regardless of foreground ratio setting any restriction would result in discarding almost large number of patches of relatively small dataset.
- Lastly the script divides and saves the resulting patches according to semi-supervised mode.

These images were patched with resulting size 256x256. Resulting numbers of training, testing samples are noted in a following table.

Category	Training size	Testing Size
A	378	58
B	360	46
C	649	88

Training/testing samples sizes for all three categories in *AutoAD* dataset.

Chapter 6

Experiments

In this chapter I give a description of the framework which is providing the implementation of the chosen methods. Then I summarise and discuss their results on each dataset. Lastly I give a conclusion of the experiments.

6.1 Framework

The framework used for experiments is Anomalib [Akay et al., 2022] which uses PyTorch Lighting (PTL)¹.

Usually deep learning process comprises of three sub-routines: *training*, *validation* and *testing*. All of these are associated with their perspective sets in the framework where the validation one is obtained as a fraction of the testing set (and this fraction is removed from the testing set assuring that training is not influenced by the testing set.).

Metrics, AUROC and F1-score are firstly computed for each pixel in the validation set. We obtain the image-level anomaly score as a maximum of pixel-level anomaly score through all pixels in a given image.

Validation is usually performed after each epoch² and, as is its purpose, the validation set is not contributing to the gradient descent but is used to calculate above mentioned metrics and to stop the training process if its results of these metrics are no longer improving (early stopping) or, as used in the experiments, to save the weights in the point in time when the validation results are the best. PTL supports performing the validation process to run either less frequently skipping a given number of epochs or more often. We also need to provide which metric is to be watched for early stopping mechanism. We can also provide patience for early stopping which is given in a form of number of times that validation was performed and its result did not improve. For example, let us

¹Pytorch Lighting is framework above the Pytorch software that makes a better job of organising the code and separating the theoretical logic from machine learning routines.

²The epoch ends after all the training samples went through the network during training once. Then begins the next epoch. *Step*, on the other hand, is one update of the model's weights, which is usually performed after every batch of data.

have a scenario where we set a validation to be done after every epoch and set early stopping to watch the image F1-score with patience 10. During training on epoch 26 validation gives us the image F1-score of 60, which is the best score as of yet, the framework will save the weights at this moment. After this epoch the model will go through another 10 epochs and if within these epochs the validation image F1-score will not improve the training will stop.

For each method and each dataset a configuration file needs to be provided which contains above mentioned hyperparameters and paths to training/testing datasets. These configuration files, which acted as the main source of control for the experiments, needed to be manually edited separately and are part of the practical part of the thesis. The bulky source code of the framework also had to be thoroughly examined for clarification of its inner workflow.

The results below are acquired as an results of the framework with the configuration file and the patched datasets on the input.

6.2 FlangeAD

Here we can see the results of all methods on the FlangeAD dataset. All samples are of size 512x512 px. Since the dataset is so large, to have more control over the training process, the validation is performed one hundred times within the one epoch and the minimum number of epochs is set to 5.

Since DRAEM seems to be very memory demanding it's batch size was set to 1. The same problem was occurring during PatchCore feature extraction, hence the amount of samples on the input was reduces to 5% (500 samples). Third table shows the times of training processes in hours and minutes and the approximate time when the validation performed the best in epochs. For PatchCore we do not get this information since there is no standard training happening. It also shows the size of the saved weights (checkpoint size).

Model	Pixel F1-score	Pixel AUROC
STFPM	0.13	0.81
DRAEM	0.08	0.72
FastFlow	0.08	0.73
PatchCore	0.29	0.88

Model	Image F1-score	Image AUROC
STFPM	0.68	0.56
DRAEM	0.66	0.39
FastFlow	0.68	0.43
PatchCore	0.69	0.67

Model	Training Duration	Testing Duration	Best epoch	Checkpoint size
STFPM	1h 47m	1m 52s	3.1	140M
DRAEM	5h 20m	3m 12s	2.16	1.2G
FastFlow	1h 37m	1m 56s	3.2	135M
PatchCore	30m	2m 25s	-	1.3G

From the results we can see that PatchCore provides the best result for the complicated dataset such as FlangeAD. It also turns out to be the fastest method from all four. The disadvantage is its large checkpoint size. STFPM provides results with similar efficiency with respect to Pixel AUROC, lower memory demands and faster inference. DRAEM and FastFlow had similar results with the former being much memory and time demanding.

6.3 AutoAD

Patches in the AutoAD dataset were entering the models in sizes of 256x256 px. Batch size of DRAEM was returned to its original value of 32 since it seems to handle smaller sample resolutions unlike in the case of FlangeAD. Since the sizes of training sets are rather small it was decided to not sub-sample PatchCore data bank and hence use all the extracted feature vector for testing. Validation is run after every epoch and patience for early stopping mechanism is set to 20 which implicitly set a minimum number of epochs to be run to the same value.

Category A

Model	Pixel F1-score	Pixel AUROC
STFPM	0.31	0.97
DRAEM	0.25	0.82
FastFlow	0.23	0.92
PatchCore	0.34	0.97

Model	Image F1-score	Image AUROC
STFPM	0.80	0.95
DRAEM	0.88	0.97
FastFlow	0.80	0.93
PatchCore	0.87	0.86

Model	Training Duration	Testing Duration	Best epoch	Checkpoint size
STFPM	2m 34m	12s	90	35M
DRAEM	9m 31s	12s	7	1.1G
FastFlow	1m 26s	12s	11	85M
PatchCore	10m 24s	14s	-	2.4G

Category B

Model	Pixel F1-score	Pixel AUROC
STFPM	0.40	0.98
DRAEM	0.07	0.87
FastFlow	0.46	0.98
PatchCore	0.23	0.97

Model	Image F1-score	Image AUROC
STFPM	0.78	0.82
DRAEM	0.52	0.58
FastFlow	0.89	0.96
PatchCore	0.92	0.94

Model	Training duration	Testing duration	Best epoch	Checkpoint size
STFPM	1m 10s	8s	35	33M
DRAEM	5m 19s	10s	2	1.1G
FastFlow	45s	10s	2	83M
PatchCore	9m 24s	10s	-	2.3G

Category C

Model	Pixel F1-score	Pixel AUROC
STFPM	0.46	0.97
DRAEM	0.10	0.86
FastFlow	0.50	0.97
PatchCore	0.36	0.96

Model	Image F1-score	Image AUROC
STFPM	0.79	0.89
DRAEM	0.67	0.58
FastFlow	0.87	0.96
PatchCore	0.93	0.96

Model	Training duration	Testing duration	Best epoch	Checkpoint size
STFPM	3m 49s	15s	84	38M
DRAEM	9m 9s	18s	2	1.1G
FastFlow	1m 45s	17s	12	84M
PatchCore	30m 47s	21s	-	4.0G

The differences between the results of the methods are similar throughout all the categories with DRAEM lacking in performance, speed and the size of the checkpoint. The rest of the models perform similarly to each other. Things to note is the high image F1-score of PatchCore and fast learning duration of FastFlow.

6.4 MVTecAD

MVTecAD dataset became a staple for sensory anomaly detection. Below are the results of the models on this dataset. Each result of arithmetic average though all categories. Each model was trained and tested on all 15 categories separately. The results of all the categories can be found in the supplementary.

Model	Pixel F1-score	Pixel AUROC
STFPM	0.58	0.96
DRAEM	0.22	0.73
FastFlow	0.56	0.89
PatchCore	0.63	0.97

Model	Image F1-score	Image AUROC
STFPM	0.91	0.93
DRAEM	0.84	0.88
FastFlow	0.88	0.92
PatchCore	0.95	0.96

Model	Training duration
STFPM	12m 7s
DRAEM	28m 47s
FastFlow	7m 47s
PatchCore	14s

DRAEM again lacks behind the other methods in both performance and duration. Surprisingly FastFlow performs worse on this dataset compared to PatchCore and STFPM. The best with respect to all metrics is PatchCore which also can be trained in much shorter time than the other models.

6.5 Discussion

On the both provided datasets, in many cases, DRAEM seems to fail completely to locate the anomalies that the other methods found with decent accuracy. Generally speaking PatchCore is quite accurate but it is failing to locate anomalies with small areas. This can be because of the aggregation process of feature vector in the feature extraction procedure. On the other hand, FastFlow and STFPM locate these small anomalies more often but they add a lot of false positives in the process.

Chapter 7

Conclusion

Overall I deem the theoretical part of the thesis successful. It will hopefully act as a nice introduction to recent, cutting-edge research in sensory anomaly detection which the reader can use to familiarise themselves with various neural network categories and their prominent examples.

The practical part accomplished its stated goals despite facing the challenge of dealing with large data and other experimental impediments. I implemented the pre-processing scripts for both provided datasets and carried out the experiments. Looking at the results it becomes clear that *DRAEM* lacks in performance, speed of training and checkpoint size even on a relatively easy dataset as MVTecAD. Out of all methods *PatchCore* seems to have the best results with the best time. The only disadvantage is its large memory demand. Comparing the results throughout the datasets it seems that *FlangeAD* occurs to be quite challenging even for the *PatchCore* method. This is caused by the variation of its anomalies and material imperfections that are not deemed as anomalies by the neural network point of view. From the two real-life datasets the results on *AutoAD* are clearly better and the methods, with the exception of *DRAEM*, could be deployed in an industrial setting on this dataset.

Chapter 8

Supplementary

8.1 MVTecAD

Here are the results of the methods run on MVTecAD dataset.

STFPM

Category	Pixel F1-score	Pixel AUROC
Grid	0.53	0.99
Zipper	0.63	0.98
Wood	0.65	0.97
Transistor	0.33	0.78
Toothbrush	0.59	0.99
Tile	0.72	0.97
Screw	0.48	0.99
Pill	0.71	0.98
Metal Nut	0.57	0.93
Leather	0.59	0.99
Hazelnut	0.67	0.99
Carpet	0.60	0.98
Capsule	0.51	0.99
Cable	0.35	0.89
Bottle	0.75	0.98

Category	Image F1-score	Image AUROC
Grid	1.00	1.00
Zipper	0.90	0.71
Wood	0.93	0.98
Transistor	0.71	0.93
Toothbrush	0.84	0.86
Tile	0.91	0.93
Screw	0.89	0.93
Pill	0.92	0.90
Metal Nut	0.89	0.96
Leather	0.95	0.98
Hazelnut	1.00	1.00
Carpet	0.93	0.97
Capsule	0.95	0.96
Cable	0.80	0.84
Bottle	0.97	0.98

Category	Training duration	Best epoch
Grid	825	80
Zipper	323	11
Wood	971	49
Transistor	1012	55
Toothbrush	360	67
Tile	609	31
Screw	983	60
Pill	772	37
Metal Nut	407	26
Leather	894	45
Hazelnut	1332	46
Carpet	782	26
Capsule	700	25
Cable	461	8
Bottle	485	36

Draem

Category	Pixel F1-score	Pixel AUROC
Grid	0.23	0.52
Zipper	0.14	0.74
Wood	0.21	0.67
Transistor	0.13	0.65
Toothbrush	0.28	0.90
Tile	0.53	0.78
Screw	0.13	0.91
Pill	0.36	0.78
Metal Nut	0.23	0.71
Leather	0.11	0.57
Hazelnut	0.22	0.78
Carpet	0.09	0.57
Capsule	0.23	0.80
Cable	0.12	0.68
Bottle	0.44	0.82

Category	Image F1-score	Image AUROC
Grid	0.81	0.85
Zipper	0.88	0.83
Wood	0.95	0.96
Transistor	0.68	0.79
Toothbrush	0.79	0.74
Tile	0.82	0.83
Screw	0.85	0.58
Pill	0.92	0.75
Metal Nut	0.90	0.76
Leather	0.85	0.81
Hazelnut	0.78	0.75
Carpet	0.87	0.66
Capsule	0.87	0.71
Cable	0.78	0.62
Bottle	0.91	0.97

Category	Training duration	Best epoch
Grid	1623	15
Zipper	1664	14
Wood	1635	11
Transistor	2553	35
Toothbrush	402	8
Tile	2114	24
Screw	4045	45
Pill	2133	18
Metal Nut	1789	21
Leather	1589	10
Hazelnut	1880	3
Carpet	1237	1
Capsule	1741	15
Cable	1867	16
Bottle	1094	6

FastFlow

Category	Pixel F1-score	Pixel AUROC
Grid	0.61	0.99
Zipper	0.66	0.99
Wood	0.60	0.96
Transistor	0.54	0.91
Toothbrush	0.28	0.90
Tile	0.53	0.78
Screw	0.29	0.98
Pill	0.58	0.96
Metal Nut	0.75	0.97
Leather	0.65	1.00
Hazelnut	0.53	0.95
Carpet	0.58	0.96
Capsule	0.51	0.99
Cable	0.49	0.93
Bottle	0.71	0.98

Category	Image F1-score	Image AUROC
Grid	0.67	0.98
Zipper	0.95	0.95
Wood	1.00	1.00
Transistor	0.80	0.96
Toothbrush	0.79	0.74
Tile	0.82	0.83
Screw	0.87	0.74
Pill	0.94	0.96
Metal Nut	0.95	0.97
Leather	0.97	1.00
Hazelnut	0.85	0.83
Carpet	0.91	0.96
Capsule	0.95	0.96
Cable	0.78	0.82
Bottle	1.00	1.00

Category	Training duration	Best epoch
Grid	543	14
Zipper	845	62
Wood	672	25
Transistor	1218	69
Toothbrush	402	8
Tile	2114	24
Screw	1192	73
Pill	652	27
Metal Nut	588	45
Leather	524	16
Hazelnut	682	12
Carpet	671	19
Capsule	700	25
Cable	1150	51
Bottle	337	16

PatchCore

Category	Pixel F1-score	Pixel AUROC
Grid	0.82	0.99
Zipper	0.70	0.99
Wood	0.47	0.93
Transistor	0.40	0.81
Toothbrush	0.66	0.99
Tile	0.67	0.96
Screw	0.41	0.99
Pill	0.57	0.95
Metal Nut	0.83	0.98
Leather	0.48	0.99
Hazelnut	0.66	0.99
Carpet	0.94	0.99
Capsule	0.44	0.99
Cable	0.67	0.97
Bottle	0.73	0.98

Category	Image F1-score	Image AUROC
Grid	0.95	1.00
Zipper	0.98	1.00
Wood	0.98	0.99
Transistor	0.97	0.95
Toothbrush	0.93	0.94
Tile	1.00	1.00
Screw	0.87	0.95
Pill	0.92	0.93
Metal Nut	0.93	1.00
Leather	0.99	1.00
Hazelnut	0.99	0.83
Carpet	0.93	0.96
Capsule	0.96	0.92
Cable	0.85	0.93
Bottle	0.97	1.00

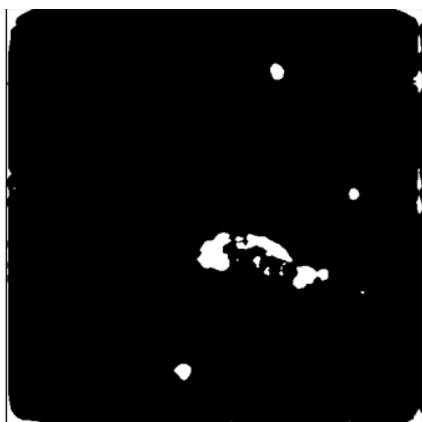
Category	Training duration
Grid	15
Zipper	15
Wood	15
Transistor	16
Toothbrush	12
Tile	16
Screw	16
Pill	18
Metal Nut	14
Leather	17
Hazelnut	16
Carpet	17
Capsule	17
Cable	18
Bottle	14

8.2 FlangeAD

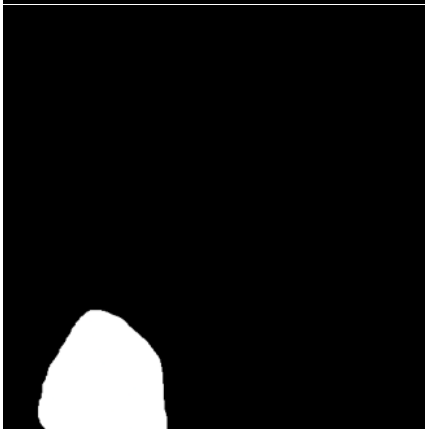
Here are the binary results of the methods on the *FlangeAD* dataset. On the left is the threshold of the output of the certain method. On the right is the ground truth.

FastFlow

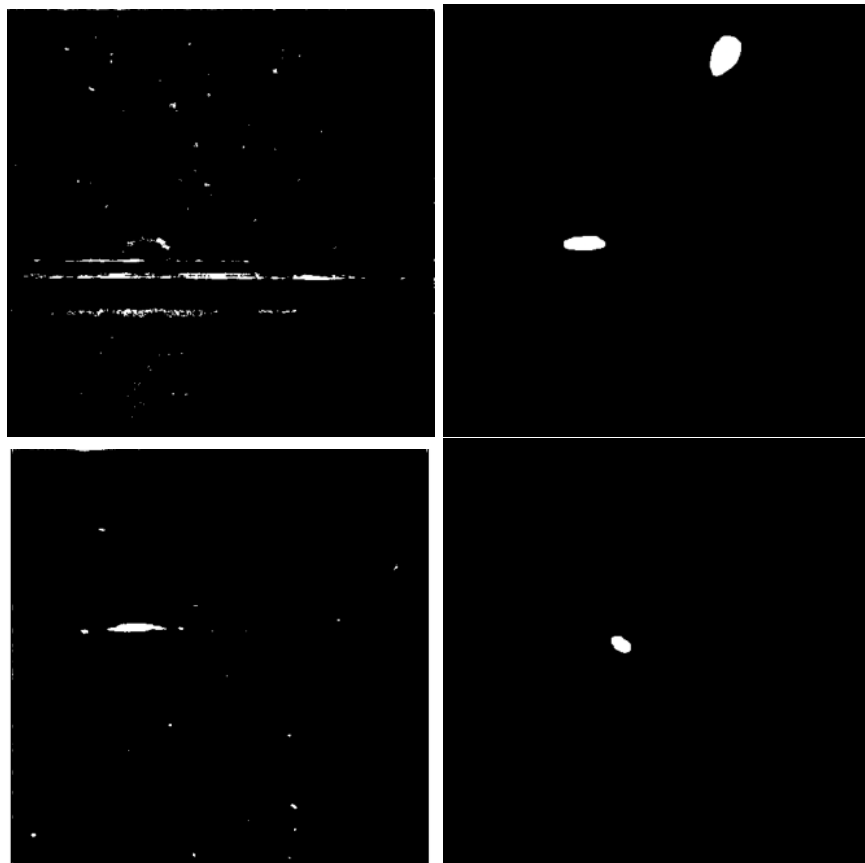




STFPM

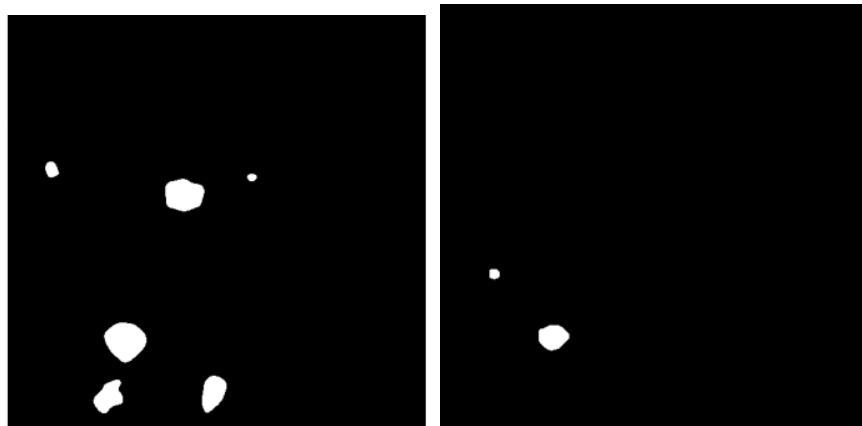


DRAEM



PatchCore

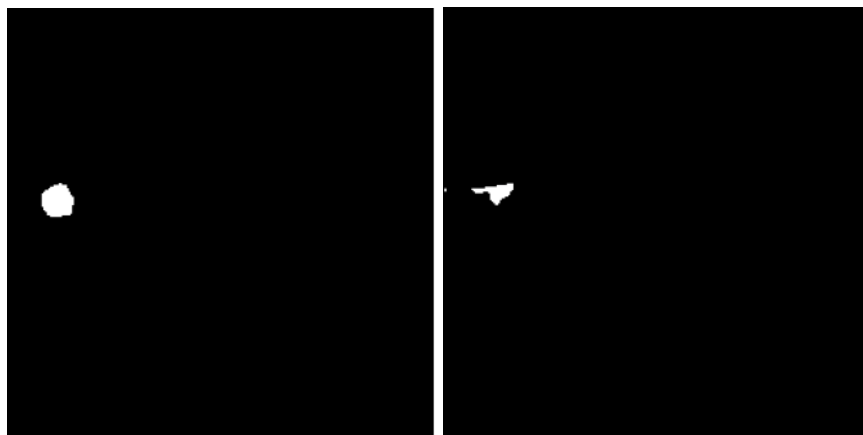




8.3 AutoAD

Here are the binary results of the methods on the *AutoAD* dataset. On the left is the threshold of the output of the certain method. On the right is the ground truth.

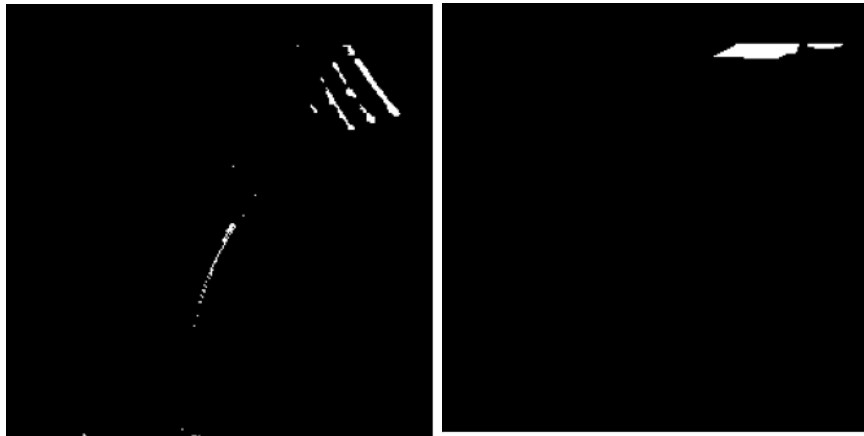
FastFlow



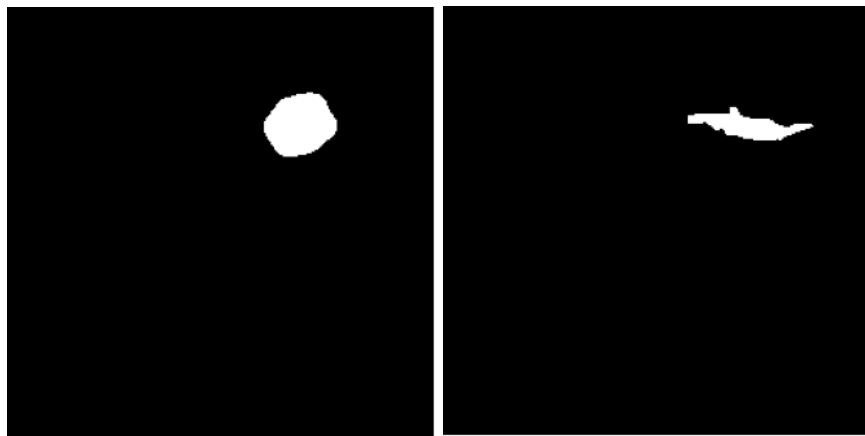
STFPM



DRAEM



PatchCore



Bibliography

- [Abati et al., 2019] Abati, D., Porrello, A., Calderara, S., and Cucchiara, R. (2019). Latent Space Autoregression for Novelty Detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision and Pattern Recognition*.
- [Akçay et al., 2018] Akçay, S., Abarghouei, A. A., and Breckon, T. P. (2018). Ganomaly: Semi-supervised anomaly detection via adversarial training. *CoRR*, abs/1805.06725.
- [Akçay et al., 2022] Akçay, S., Ameln, D., Vaidya, A., Lakshmanan, B., Ahuja, N., and Genc, U. (2022). Anomalib: A deep learning library for anomaly detection.
- [Bergmann, 2022] Bergmann, P. (2022). *Unsupervised Anomaly Detection and Localization for Visual Quality Inspection*. PhD thesis, Technische Universität München.
- [Bergmann et al., 2019] Bergmann, P., Fauser, M., Sattlegger, D., and Steger, C. (2019). Mvtec ad — a comprehensive real-world dataset for unsupervised anomaly detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- [Chalapathy and Chawla, 2019] Chalapathy, R. and Chawla, S. (2019). Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407.
- [Deecke et al., 2019] Deecke, L., Vandermeulen, R., Ruff, L., Mandt, S., and Kloft, M. (2019). Image anomaly detection with generative adversarial networks. In *Machine Learning and Knowledge Discovery in Databases*.
- [Dhamija et al., 2018] Dhamija, A. R., Günther, M., and Boulton, T. E. (2018). Reducing network agnostophobia. *CoRR*.
- [Dinh et al., 2014] Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *CoRR*.
- [Eskin, 2000] Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, page 255–262.

- [Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- [Hu et al., 2020] Hu, W., Gao, J., Li, B., Wu, O., Du, J., and Maybank, S. (2020). Anomaly detection using local kernel density estimation and context-based regression. *IEEE Transactions on Knowledge and Data Engineering*, 32:218–233.
- [Jiang et al., 2022] Jiang, X., Xie, G., Wang, J., Liu, Y., Wang, C., Zheng, F., and Jin, Y. (2022). A survey of visual sensory anomaly detection. *arXiv*.
- [Jingkang et al., 2021] Jingkan, Zhou, K., Li, Y., and Liu, Z. (2021). Generalized out-of-distribution detection: A survey. *CoRR*, abs/2110.11334.
- [Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv*.
- [Kobyzev et al., 2021] Kobyzev, I., Prince, S. J., and Brubaker, M. A. (2021). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979.
- [Leys et al., 2018] Leys, C., Klein, O., Dominicy, Y., and Ley, C. (2018). Detecting multivariate outliers: Use a robust variant of the mahalanobis distance. *Journal of Experimental Social Psychology*, 74:150–156.
- [Li et al., 2021] Li, C., Sohn, K., Yoon, J., and Pfister, T. (2021). Cut-paste: Self-supervised learning for anomaly detection and localization. *CoRR*, abs/2104.04015.
- [Liang et al., 2022] Liang, Y., Zhang, J., Zhao, S., Wu, R., Liu, Y., and Pan, S. (2022). Omni-frequency channel-selection representations for unsupervised anomaly detection. *arXiv*.
- [Lin et al., 2017] Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P. (2017). Focal loss for dense object detection. *CoRR*, abs/1708.02002.
- [O’Connor, 2012] O’Connor, B. (2012). F-scores, dice, and jaccard set similarity. <https://brenocon.com/blog/2012/04/f-scores-dice-and-jaccard-set-similarity/>. Accessed: 2023-10-27.
- [Reiss et al., 2020] Reiss, T., Cohen, N., Bergman, L., and Hoshen, Y. (2020). PANDA - adapting pretrained features for anomaly detection. *CoRR*, abs/2010.05903.
- [Ruff et al., 2020] Ruff, L., Kauffmann, J. R., Vandermeulen, R. A., Montavon, G., Samek, W., Kloft, M., Dietterich, T. G., and Müller, K. (2020). A unifying review of deep and shallow anomaly detection. *CoRR*, abs/2009.11732.

- [Ruff et al., 2018] Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. (2018). Deep one-class classification. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4393–4402.
- [Sabokrou et al., 2018] Sabokrou, M., Khaloeei, M., Fathy, M., and Adeli, E. (2018). Adversarially learned one-class classifier for novelty detection. *CoRR*, abs/1802.09088.
- [Salehi et al., 2021] Salehi, M., Mirzaei, H., Hendrycks, D., Li, Y., Rohban, M. H., and Sabokrou, M. (2021). A unified survey on anomaly, novelty, open-set, and out-of-distribution detection: Solutions and future challenges. *CoRR*, abs/2110.14051.
- [Wang et al., 2021a] Wang, G., Han, S., Ding, E., and Huang, D. (2021a). Student-teacher feature pyramid matching for unsupervised anomaly detection. *CoRR*, abs/2103.04257.
- [Wang et al., 2021b] Wang, S., Wu, L., Cui, L., and Shen, Y. (2021b). Glancing at the patch: Anomaly localization with global and local feature comparison. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 254–263.
- [Yan et al., 2021] Yan, X., Zhang, H., Xu, X., Hu, X., and Heng, P.-A. (2021). Learning semantic context from normal samples for unsupervised anomaly detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:3110–3118.
- [Yu et al., 2021] Yu, J., Zheng, Y., Wang, X., Li, W., Wu, Y., Zhao, R., and Wu, L. (2021). Fastflow: Unsupervised anomaly detection and localization via 2d normalizing flows. *ArXiv*.
- [Zavrtanik et al., 2021] Zavrtanik, V., Kristan, M., and Skocaj, D. (2021). Dræm - A discriminatively trained reconstruction embedding for surface anomaly detection. *CoRR*, abs/2108.07610.
- [Zong et al., 2018] Zong, B., Song, Q., Min, M. R., Cheng, W., Lumezanu, C., Cho, D., and Chen, H. (2018). Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*.