Università degli Studi di Padova

University of Padova

Department of Information Engineering

Master's Degree in
Computer Engineering - Robotics and AI

Visual Anomaly Detection on circular plastic parts using Generative Adversarial Networks

Supervisor:
Prof. Alberto Pretto
Co-Supervisors:
Alberto Gottardi
Nicola Carlon
IT+Robotics

Master candidate:
Nicola Rizzetto
2052417

Academic Year 2022/2023
18 October 2023

Abstract

In recent years, automated quality control systems have been established as the main method for anomaly detection, term which refers to the process of identifying and flagging any abnormality in the condition of the given components. Given their efficiency, many new methods were developed, mainly exploiting Computer Vision algorithms, but they have their limitations. In a similar way, many studies were applied on Neural Networks and Machine Learning algorithms, with the development of Convolutional Neural Networks, Transformers and Generative Adversarial Networks (GANs). The objective of this thesis is to develop an automated quality control system exploiting the generative and adversarial qualities of the current state-of-the-art methods based on Neural Networks. The main tool used for this task is the capability of the GANs to learn how a flawless input should look, so that the pipeline can identify inputs with anomalies. The developed solution was tested on a real world problem, aiming to identify cracks and anomalies in plastic motor covers.

Contents

Chapter 1

Introduction

Quality control is an indispensable aspect of industrial operations, encompassing a spectrum of techniques and methodologies aimed at ensuring the consistent production of high-quality products. Many companies have a human operator perform quality control, manually examining each component. This is a repetitive and stressful task for the operator, with the addition of slowing down the production. For this reason many studies have gone on automated quality control systems, which are usually faster and more accurate than a human operator. The automated quality control systems can exploit Computer Vision, for example by processing the image of the component and identify darker areas which may be relative to a crack, or Machine Learning, by training the model to separate compliant and anomalous samples. The modern industrial landscape is characterized by heightened consumer expectations, stringent regulations, and intense global competition. In this context, maintaining high-quality products is paramount for industrial success. Industrial quality control, as a systematic and proactive approach, plays a pivotal role in achieving and sustaining product excellence.
Also, industrial processes are becoming increasingly complex, relying on intricate machinery and systems for efficient operation, reasons why anomalies down the production line can cost a lot to the company, causing equipment failures or safety hazards. Thus, a fast and accurate quality control system can be very profitable, by reducing the wrongly compliant components and keeping the same pace as the production line. There is also a big difference in detecting anomalies at the beginning or at the end of a production line: for example, a company produces multiple parts and assembles them in one final component: detecting anomalies on the single parts is the most expensive way, because it needs a separate system for each part, but guarantees a correct assembly and a compliant final component;

detecting anomalies on the final component costs less and guarantees a compliant final component, but has a higher chance of risk, especially during the assembly part, where some anomalous parts may break and can cause safety hazards.
Consequently, the development of effective anomaly detection methods has garnered substantial attention in recent years, exploiting both computer vision algorithms and machine and deep learning techniques.

1.1 Anomaly Detection

The detection of anomalies in a system is a problem found in various fields such as data mining [20], cybersecurity [18], fraud detection [19], and quality control [21]. It involves identifying patterns or instances that deviate significantly from the expected behavior within a dataset or system. These deviations are often referred to as anomalies, outliers, novelties, or exceptions. The detection of anomalies is a critical component of data-driven systems, as it helps identify issues, threats, or opportunities that might otherwise go unnoticed in large datasets or complex systems. The choice of technique and approach depends on the specific problem domain, data characteristics, and the level of tolerance for false positives and false negatives.
A practical example already cited is about cybersecurity, for the detection of cyber attacks [18]. In this case the available data is the network activity between computers, so it is dependent on the connection between users. The approach discussed in the paper uses a Machine Learning model called Isolation Forest, explained in Section 2.4.1, that exploit differences from the majority of data points. This is a typology of the problem that does not tolerate false negatives, because if it fails to detect a cyber attack the affected company may suffer huge losses or shutdown of their internet services.
Another practical example is financial fraud detection [19], with available data that varies from credit card fraud, insurance fraud to money laundering. Between the proposed methods in the paper stand out the Hidden Markov Models, a typology of stochastic processes that can express more complex processes than traditional Markov models. Explained briefly, Hidden Markov Models exploit Markov chains, a mathematical model that represents a system where transitions between states occur based on certain probabilities, and the future state depends only on its current state. The difference with traditional Markov models is the use of both hidden states, that are states not directly observable and represent

the underlying processes that generate the observed data based on the state probability distribution, and observable outcomes, data points generated from the hidden state as depicted in the scheme at Figure 1.1 [19].

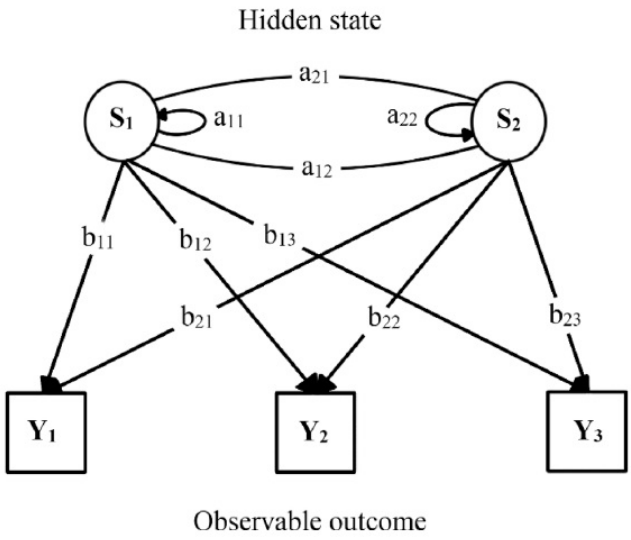


Figure 1.1: Scheme of an Hidden Markov Model.

The paper proposes a credit card fraud detection system where normal transactions of an individual are used to train an HMM for each cardholder, and subsequently incoming transactions are compared with their respective models, and any transaction not accepted by the HMM with sufficiently high probability is considered fraudulent. This typology of system must be careful with both false positives, where an individual may have their credit card blocked from the bank without reason, and false negatives, failing to detect an actual fraud.
The case discussed in this thesis falls in the quality control category, where a company produces circular plastic motor covers and seals them on motors. The sealing phase is where anomalies may form through cracks or deformations, and the proposed solution has to identify them. This type of system must have a low tolerance for false negatives, since a non-detected cracked motor cover can develop in a equipment failure for the motor and possible safety hazard, while also being careful in not having too much false positives, discarding many actually compliant covers.
The solution was discussed and studied in collaboration with IT+Robotics, a company situated in Padua that produces hardware and software solutions for Industrial Robotics, Computer Vision and 3D Quality Control.

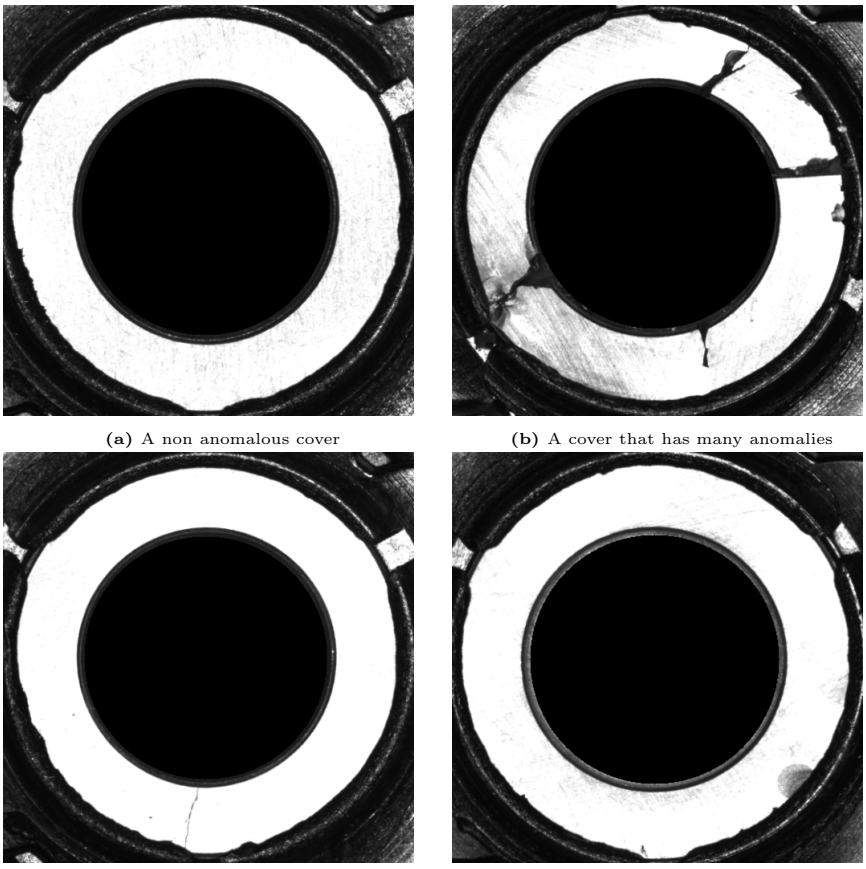
1.2 Proposed approach

A fundamental component for anomaly detection is hardware accuracy, so that the software can work with the best possible input. An example of this is a camera that takes photos of the components to analyze, which should be as invariant as possible to different lighting conditions, object movements and distance from the component.
On the other hand, the software must offer an accurate detection of defective components, highlighting the location of the anomalies if possible, and also ensuring a fast processing time, so as not to slow down the production.
As already introduced, within recent years many methods for quality control have been discovered, mainly using Computer Vision algorithms or Deep Learning models. Since Computer Vision focuses on image processing, it relies on handcrafted optical features, a method that tends to be more domain-specific [22]. Machine and Deep Learning models, on the other hand, can learn which are the important features, and can easily be trained on another dataset if the task changes.
However, these models have some drawbacks, such as dependence on image resolution and requiring a lot more computational power and training time than traditional models. Another difficulty is the lack of behavior interpretability, given the numerous layers and parameters of a Deep Learning model, while traditional models have explicitly defined features.
These are the reasons why many techniques have been improved or replaced with newer Machine Learning and Deep Learning models, which need a substantial number of images of anomalous and non-anomalous components, in order to *train* and *learn* the difference between the two classes of components.
This thesis focuses on giving a solution to a real-world anomaly detection problem based on the latest Deep Learning models, that exploit an input reconstruction capability.
The main reason behind the choice of a Deep Learning model instead of a traditional one is the capability of learning end-to-end mappings from raw data to classification or scores without the need for complex preprocessing or feature extraction pipelines. This also comes with the challenge of needing a lot of computational power and large labeled datasets, and while the first problem can be mitigated through modern GPU hardware accelerators, finding anomalous samples for quality control can be a problem, but the solution will be explained in Chapter 3.

1.3 Real world application

As already introduced in Section 1.1, the proposed solution is going to be applied to an already existing quality control pipeline. The system is composed of a machine that seals circular plastic covers on electric motors, heating up and pressing said plastic covers on the motors, where it may happen that the machine presses a cover that is not heated enough and cracks it. A camera is placed on top of the line and it is programmed to capture one image for each processed cover, and then the anomaly detection system has to evaluate if the current component is defective or not.
The solution has to evaluate each component in less than a second, and is demanded to identify all defective covers.
The biggest challenge on this task is that the class of anomalous components is not defined precisely: cracks and deformations can occur in an unpredictable number and shapes, and on the other hand the anomaly can be so small and undetectable that the component is deemed as compliant. Keeping in mind that a compliant cover is illustrated in Figure 1.2a, which has no cracks and deformations but at best some scratches, some examples of this problem are explained as follows: on Figure 1.2b we have very obvious and numerous anomalies, while also 1.2c is part of anomalous samples, with a much less notable crack. The ambiguity of the definition of anomalous samples can be explained through Figure 1.2d, where the small gray patch can both signify a chipping (then anomalous) or just dirt (compliant).

The dataset used for the training of the model is also very unbalanced, because in this type of industrial applications occur much more compliant components than anomalous ones, with less than 10% of the dataset made up of anomalies.
The ground truth of the dataset is also difficult to establish, because for this type of anomaly detection there are no true labels, just estimated ones through other softwares or human operators. In this case, IT+Robotics has already developed an anomaly detection system in the past, which was used to obtain the ground truth for the dataset, together with further analysis on the most difficult components that even this method had problems classify.
Although the already developed method has a good accuracy, which was helpful for the ground truth, it was decided to try a Deep Learning approach, especially using generation and reconstruction techniques, to identify even the most difficult anomalies.



(a) A non anomalous cover

(b) A cover that has many anomalies

(c) A cover that has a crack difficult to identify

(d) A cover that has a bruise which can mean a chipping or just dirt

Figure 1.2: Example of compliant (a) and anomalous (b,c,d) covers

After the given introductions to the problem and solution, the thesis will be explained in the following structure: Chapter 2 gives a theoretical background on the latest Deep Learning models; Chapter 3 discusses the experimental setup, preprocessing and training of the model; Chapter 4 presents the results of the best obtained models, on both validation and test phases; Chapter 5 will finally suggest some future work on how to improve the current solutions.

Chapter 2

Anomaly Detection

This chapter will explain the main details about the problem of Anomaly Detection and the state-of-the-art solutions using both Computer Vision and Deep Learning.

2.1 Problem Statement

Anomaly detection pertains to the task of discerning unwanted samples in a large dataset or system, these referred to as anomalies. The nature of these sample can be various as such:

- Point Anomalies: individual data points that differ significantly from the majority of the dataset, such as a defective component in a production line;
- Contextual Anomalies: samples that are context-specific and depend on the surrounding data, such as analyzing the temperature in a city (30°C is a high temperature, but it is considered normal if the city is in summer times);
- Collective Anomalies: these anomalies involve a group of data points that exhibit anomalous behavior when considered together but may not be anomalous individually.

Independently from their nature, anomalies are rarely explicitly marked, fact that can make difficult to obtain labeled data, where in many real-world scenarios can be expensive, time-consuming, or even impossible. This also usually leads to a substantial imbalance in the dataset, where the number of anomalous samples is often very limited w.r.t. the normal samples, fact that can be debilitating if the

solution uses a Machine or Deep Learning model. This can lead to models biased towards normal instances and result in higher false negatives, where anomalies are missed.
Given the low number of testing anomalies, an important role is filled by the feature engineering, where selecting the right feature or representation for the data is crucial for an effective detection and efficiency, especially in the case of high dimensional data. This is also important if the solution has to operate in dynamic environments, where the concept of what constitutes an anomaly can change over time (anomalies that were once rare may become more common or vice versa). Models must adapt or be robust to those changes, reason why feature engineering is important.
The chosen model must also be robust to noisy data or data with missing values, which can make it difficult to distinguish true anomalies from normal data. In the real world application discussed in this thesis this is a very frequent problem, where, as illustrated before, a small patch of dirt can be confused for a chipping or, worse, a very narrow crack is deemed as a scratch. But on the other hand, if our solution is too complex it becomes difficult to understand why these samples where mislabeled, due to the many parameters.
Another important point is the correct selection of the threshold that separates normal and anomalous samples, which is often subjective and can impact the detection performance. This threshold is usually extracted from the model results, obtained through particular metrics for anomaly detection, such as the F1-Score or the ROC curve.

2.2 Evaluation Metrics

The most common way to evaluate a solution for Anomaly Detection in general is through the calculation of Precision and Recall, which are bound to the definitions of: False positive, or rejection of a true null hypothesis (e.g.: a normal cover was identified as anomalous); and False negative, or non-rejection of a false null hypothesis (example: a cracked cover was not identified).
With these concepts in mind, Precision is defined as the ratio between True positives and total positives (fraction of relevant instances among the retrieved instances), while Recall is the ratio between True positives and the sum of True positives and false negatives (fraction of relevant instances that were retrieved), definitions explained schematically in Figure 2.1 [25].

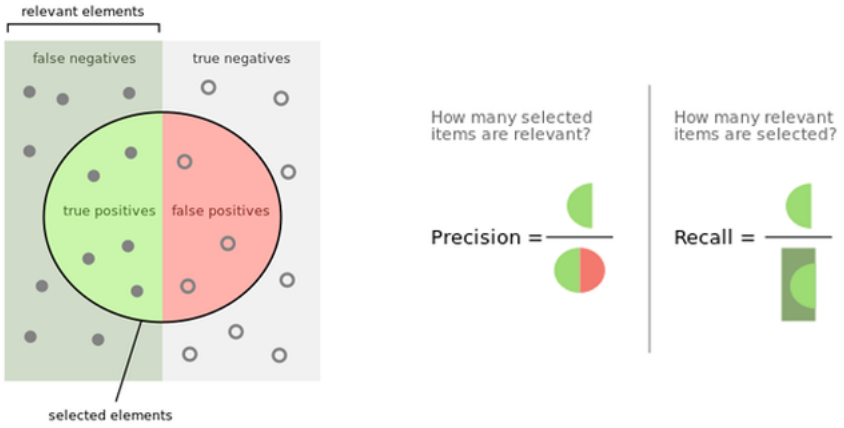


Figure 2.1: Schematic way to explain Precision and Recall

Precision and Recall are then used in different scores that express the different capabilities of the model, where the most important are:

- F1-Score: widely used performance metric in machine learning, particularly in binary classification tasks. It expresses the balance between Precision and Recall, and is particularly valuable when dealing with imbalanced datasets:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.1}$$

- ROC Curve (Receiver Operating Characteristic Curve): a graphical representation that illustrates the performance of a binary classification model across different threshold settings, as seen in Figure 2.2 [24]. It is particularly valuable for assessing a model's ability to discriminate between the positive and negative classes. The ROC curve is created by plotting the true positive rate against false positive rate at various threshold values, where a straight line represents the performance of a random classifier;

- AUC-ROC Curve (Area Under the ROC Curve): a scalar value that quantifies the overall performance of a binary classification model by summarizing the ROC curve. It represents the area under the ROC curve and ranges from 0 to 1, where 0.5 represent a random classifier and 1 a perfect discrimination between the positive and negative classes.

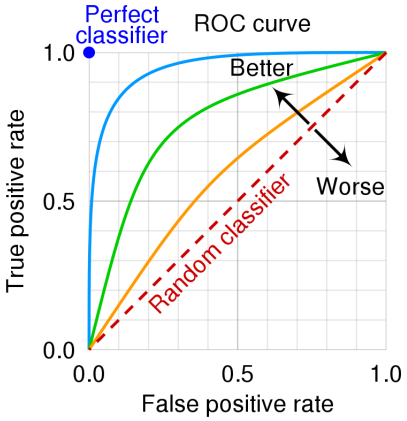


Figure 2.2: Graph of the ROC curve.

2.3 Traditional Methods

This section will explain the main idea behind some methods that exploit traditional algorithms such as statistical analysis, streaming data and computer vision.

2.3.1 Statistical Analysis

Anomalies can be exploited studying the statistical distribution of normal samples and identifying those samples who differ too much from others. One useful metric is the Z-score, that measures how many standard deviations a data point is away from the mean of the dataset. A high absolute Z-score indicates an extreme deviation from the mean, suggesting an anomaly. This method is very useful if the dataset follows a Gaussian distribution [6], otherwise there is a modified Z-score that uses the median and the median absolute deviation (MAD) instead of the mean and standard deviation, which is a robust measure of dispersion.
More complex methods estimate the probability density function of the data [7], where points in low-density regions are considered anomalies, or may need data expressed in time series in order to use moving averages, exponential smoothing, or autoregressive models. These methods are effective for detecting anomalies in temporal data, such as network traffic, sensor readings, or financial data.
Another way can be exploiting the relation between variables, as done in Multivariate Statistical Analysis [8]. This technique uses Mahalanobis distance or Hotelling's T^2 distribution, very useful for high-dimensional data and scenarios where anomalies involve relationships between multiple variables, such as fraud detection or industrial process monitoring.

2.3.2 Streaming Data

Streaming data is data that is continuously generated over time, so the algorithm must adapt to dynamic environments and run automatically and unsupervised, with predictions that must be made online. A useful value for Streaming Data problems is the Moving Average [9], a calculation used to analyze/smooth data by creating a new series of averages from subsets of the data points, most likely the n latest data points. The Moving Average is calculated for every data point, and then the algorithm determines an expected value and calculates the standard deviation using the latest n samples. If the Moving Average is in the range of $expval + std$ and $expval - std$ the sample is considered normal.
Overall, the previous method works well only if the data has small changes or uniformly changes over time, while if it has bigger differences, like seasonal data, it is better to use Exponential Moving Average [10]. The main feature of this method is that the algorithm gives more weight and significance to recent data, decaying the less recent samples.

2.3.3 Computer Vision

When the solution has to deal with images and videos, the main way is to use Computer Vision algorithms, with methods that vary from direct pixel processing to object detection.
A method that directly processes pixels can use: Gaussian Mixtures [11], which models the distribution of pixel values in a scene using a mixture of Gaussian distributions, and it labels pixels as anomalies if they significantly deviate from the modeled background distribution; Temporal Difference [12], that compares the current frame with the previous one to detect changes, so that pixels that show substantial differences are marked as anomalies.
More complex models may use Segmentation [13], which still directly processes pixels but it tries to classify each pixel in a different class, effectively trying to separate the image into different regions. Segmentation can be Semantic Segmentation, which assigns a label to every pixel in the image to indicate what object or category it belongs to, useful for scene understanding (an anomaly can be found if an unexpected class appears in the image), or Instance Segmentation, which combines segmentation and object detection to distinguish individual object instances, which is very useful for object tracking (an anomaly can be found when an unexpected instance appears).

Finally, the most complex models use Object Detection, with the state-of-the-art algorithm being the Viola-Jones algorithm [14], that uses Haar features. These particular features are rectangular filters that are used to compute the difference between the sum of pixel values in adjacent rectangular regions of an image, thus being able to capture basic patterns of light and dark areas in an image. The Haar-like features are defined by their position, size and shape, which can be two, three or four-rectangular, as illustrated in Figure 2.3 [14].

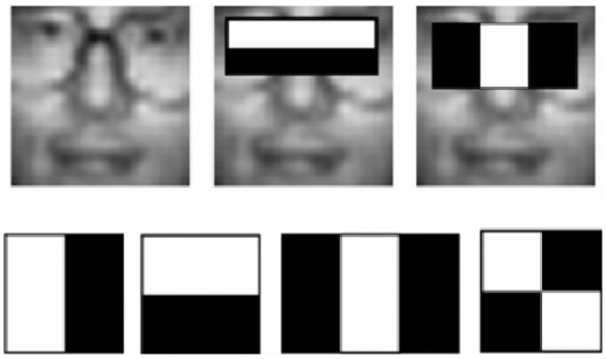


Figure 2.3: Different types of Haar features and how they are applied.

These features are applied to subregions of an image in a sliding window fashion to scan the entire image, and combined in a cascaded structure form a robust object detection system.
The Viola-Jones algorithm combines Haar features and a machine learning technique called Adaboost to obtain a fast and efficient model for Object Detection. The algorithm first chooses a small set of Haar features based on their discriminative power, then Adaboost selects the best features and obtains many weak classifiers, giving more importance to samples that are difficult to classify. The algorithm then organizes the classifiers into a cascade structure, in order to quickly reject regions that do not contain the object, obtaining a final strong classifier. The trained cascade is applied to an image using a sliding window approach, where at each window position the cascade checks the region with a series of classifiers. If a region passes all stages, it is considered a positive detection, and the object is located.
The Viola-Jones algorithm is known for its speed and efficiency, and has been widely used in applications like face recognition in digital cameras and pedestrian detection in autonomous vehicles.

2.4 Artificial Intelligence

This section will explain the main idea behind some methods for anomaly detection that use Machine and Deep Learning.

2.4.1 Machine Learning Methods

When the problem to be faced has a substantial amount of labeled samples and the anomalies are well defined the main way is to use a model trained under Supervised Training, such as Support Vector Machines [27], which maps samples in order to maximize the distance between samples of different labels.
If the labeled samples are scarce or unavailable then Unsupervised Learning models mainly organize data into clusters, which are set of samples that have similar samples close to each other and different samples far from each other. Models that use clusters are K-Means Clustering [28], which computes the optimal amount of clusters to separate the data in, or DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [29], which is very robust to noise and its cluster have arbitrary shape.
Another method is the Isolation Forest [15], where its key point is to isolate anomalies by leveraging the observation that anomalies are typically few and different from the majority of the data points. The algorithm starts by randomly selecting a feature and then randomly selects a split value between the minimum and maximum values of that feature, and splits the dataset into two subsets based on said selected feature and value. The split process is recursively iterated until all data points are fully isolated, and the anomaly score of a data point is how many partitions it took to isolate the sample. However, this method always assumes the separability of data points, and struggles with imbalanced datasets.

2.4.2 Deep Learning Methods

Many methods with Deep Neural Networks were discovered in the recent years, given their capability of adaptation without needing fully detailed samples.
Since Anomaly Detection usually deals with images, one of the most effective networks applied to the problem are the Convolutional Neural Networks (CNN) [26], which are a type of artificial neural network designed specifically for processing structured grid-like data, such as images or data with a grid-like topology. They are characterized by their ability to automatically learn hierarchical features from

the input data, making them particularly suited for tasks where spatial relationships and patterns are important.
These networks are characterized by some key elements:

- Convolutional Layers: the core of CNN. These are learnable filters (or kernels) that are applied to small regions of the input data, as illustrated in Figure 2.4 [23]. Convolution operations involve element-wise multiplication of the filter with the input data, followed by summation, to produce feature maps. These kernels are used in each convolutional layer to capture different features in the data, such as edges, textures, or more complex patterns. This pattern recognition capability is very useful for object classification, especially in anomaly detection;

- Pooling Layers: layers used to downsample the spatial dimensions of feature maps while retaining essential information, for example taking only the maximum value from a small region of the feature map. Pooling reduces the computational burden, decreases the risk of overfitting, and makes the network more robust to translation variations;

- Activation functions: non-linear functions applied to feature maps, usually after the convolution and pooling, which helps to learn complex relationships within the data. The most common function used is ReLU (Rectified Linear Unit), which does not change positive values but transforms to zero negative values;

- Fully Connected (or Dense) Layers: layers that connect all neurons from the previous layer to the current layer, often used for making final predictions, like classifying objects in an image.

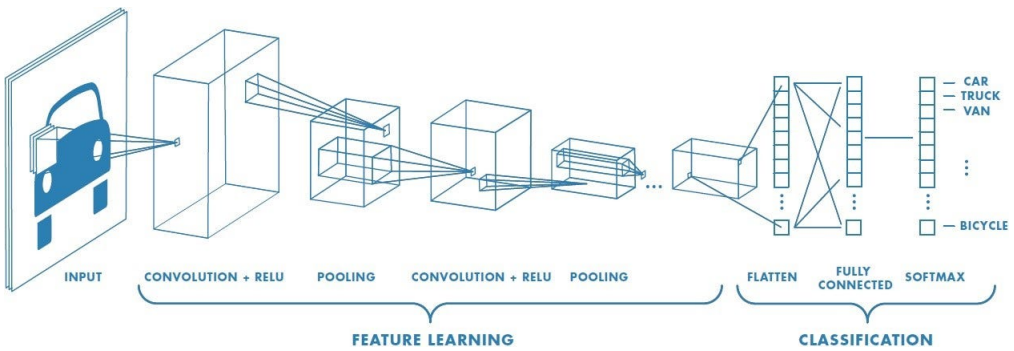


Figure 2.4: Example of Convolutional Neural Network and the use of convolution.

Another type of network is the Recurrent Neural Network (RNN) [30], designed for processing sequences of data. These networks have connections that loop back on themselves, allowing them to maintain a form of memory. At each step, the RNN maintains and updates a hidden state that summarizes the information it has seen so far, where the final hidden state contains information about the entire sequence and is used for various tasks, such as classification or prediction. This ability to capture sequential information makes RNNs well-suited for tasks like natural language processing, speech recognition and time series analysis.
The latest Deep Learning models regarding context-understanding are the Transformers [16], known for their ability to handle long-range dependencies in sequences efficiently, thanks to the self-attention mechanism, illustrated in Figure 2.5 [17]. This mechanism allows each position in the input sequence to focus on other positions, capturing relationships and dependencies between words or elements in the sequence, regardless of their distance from each other. Self-attention is computed using three components: queries, keys, and values. The attention scores are calculated by measuring the similarity between queries and keys and are then used to weight the values. These sets of queries, keys and values are often used in parallel to capture different types of dependencies. Since they do not capture the order or position of elements in a sequence, positional encoding is added to the input embeddings, and to avoid the vanishing gradient problem they also implement skip connections as in Residual Networks, which are special connections that skip one or more layers to pass the information on further layers.

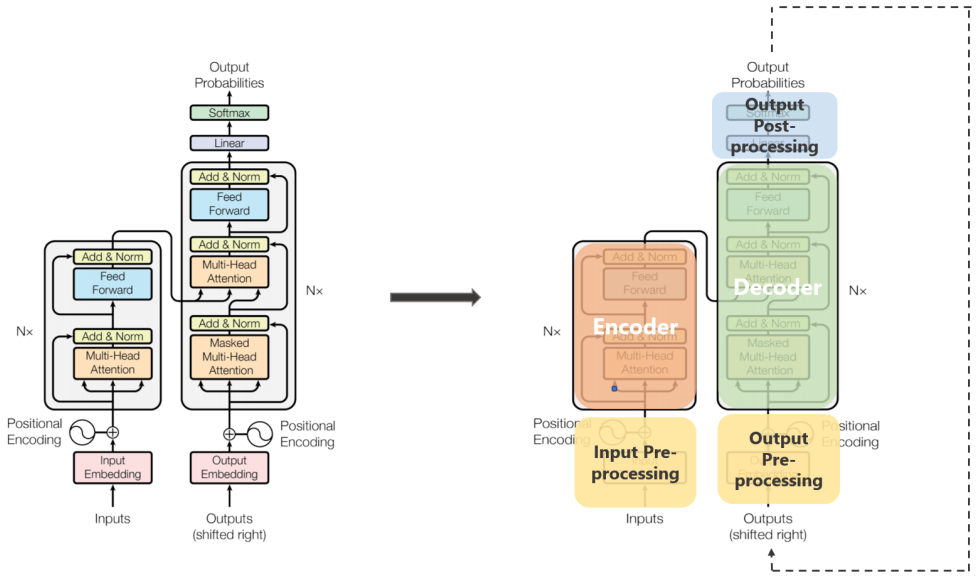


Figure 2.5: Structure of a transformer.

Transformers had a profound impact on the field of deep learning, particularly in NLP, capturing complex relationships between elements, and have led to significant advances in various natural language understanding tasks. However, they need a very large amount of data, that may not be available in some case, reason why they were discarded for this solution.
Other Deep Learning methods exploit the generation and reconstruction capabilities of neural networks, such as Autoencoders or GANs, but will be explained in detail on chapter 3.

Chapter 3

Theoretical Background

This chapter will give a quick theoretical background about Autoencoders and Generative Adversarial Networks, which are the baseline for the proposed solution. The second part of the chapter will explain the main idea behind the state-of-the-art method studied for the development of the proposed solution.

3.1 Generative Networks

In the latest years a new typology of Neural Networks have been discovered: a variant that can reconstruct and generate input samples. This section will explain briefly the idea behind the most common types of Generative Networks.

3.1.1 Autoencoders

Autoencoders are a class of Neural Networks used in Unsupervised Learning modalities, and are particularly useful for dimensionality reduction tasks, feature learning and similar data generation. Their main structure consists of an Encoder and a Decoder, which work together to learn a compact representation of input data, as seen in Figure 3.1 [31].

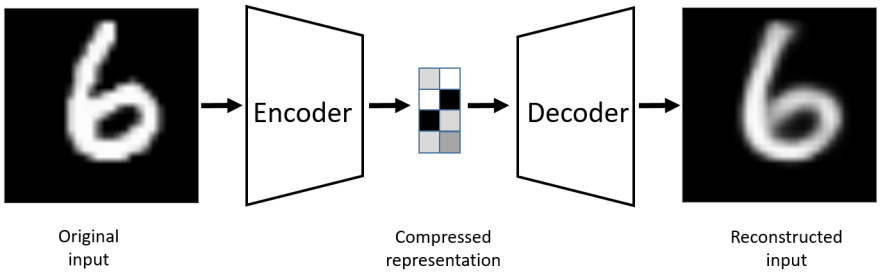


Figure 3.1: Structure of an Autoencoder.

The Encoder, is the first block of the network, which takes the input data and maps it into a lower-dimensional latent space representation through several layers of neurons that progressively reduce the input's dimensionality. Each layer extracts higher-level features from the input data, creating a hierarchical representation.
The compressed data, or encoding, is then passed to the Decoder, which and attempts to reconstruct the original input data. It has the same structure as the Encoder, but with layers of increasing dimensionality instead of decreasing, until reaching the original data dimension. This structure forces the network to learn the most important features of the input data, effectively "encoding" the input and "decoding" the representation from the latent space. The performance of the model is evaluated through the difference between real input and reconstructed input, usually through Mean Square Error (MSE) loss or Binary Cross-entropy loss.
As introduced before, the training is unsupervised because it doesn't require labeled data; the model learns to extract useful features purely from the input distribution.
One recurrent problem of this architecture is the high tendency to overfitting, where the model would just learn the identity matrix as latent vector, which ensures a low loss. This can be avoided by constraining the latent vector on having a much lower dimensionality w.r.t. the input and by adding noise to the input, making almost impossible to learn the identity matrix.
Given their reconstruction capabilities, Autoencoders are very useful for Anomaly Detection tasks [32], where they can be used to detect anomalies or outliers in data by reconstructing input data and identifying data points with high reconstruction errors, reason why they are a fundamental piece in our solution's pipeline.

3.1.2 Variational Autoencoders

The basic concept of Autoencoder can be extended with a probabilistic approach to obtain a wider generative capability. In addition to the data representation learning, Variational Autoencoders can generate new data samples that resemble the input dataset, by modeling the latent space as a probability distribution, allowing for greater flexibility and control over the generated data. The structure is the same as an Autoencoder, with some differences:

- The Encoder does not output a single point in the latent space, but rather the parameters of a probability distribution, typically the mean and variance of a Gaussian distribution;

- The sampling from the learned distribution is not done directly, instead a separate noise variable is sampled from a standard Gaussian distribution, and then this noise is combined with the learned mean and variance from the encoder to obtain a sample from the latent space, allowing the model to be trained using gradient-based optimization;

- The latent space is modeled as a multivariate Gaussian distribution with a mean and variance determined by the encoder's output, while the Decoder has the same function as in a normal Autoencoder;

In addition to the usual reconstruction loss, another component is added to enforce that the learned distribution resembles as close as possible a Gaussian distribution, known as the Kullback-Leibler divergence, which ensures a smooth and continous latent space.
Variational Autoencoder are an overall improvement of classic Autoencoders, and are to be preferred in cases where we have complex data distribution or need to generate similar data, but still output rather blurry samples.

3.1.3 Generative Adversarial Networks (GAN)

GANs are a class of machine learning models that were introduced by Ian Goodfellow and his colleagues in 2014 [4]. Their main advantage w.r.t. Autoencoders is the capability to capture the multi-modality of the latent space distribution, which is not possible using a standardized Gaussian distribution.
They consist of two neural networks, the Generator and the Discriminator, which are trained simultaneously through a competitive process. The Generator takes random noise or a seed as input and attempts to generate data samples that resemble the training data, where the initial output is always random or low quality data. The Discriminator is responsible for distinguishing between real data from the training set and fake data generated by the generator, assigning high probabilities to real data and low probabilities to fake data.
Keeping in mind how the two networks work, the training procedure of a GAN involves a competitive process between the Generator and the Discriminator:

- The Discriminator takes in input both training data and the output data from the Generator, labeling with probabilities each sample;
- After each evaluation, the Generator modifies its network to improve its performance by generating fake data that looks closer to the real data, aiming to "fool" the Discriminator;
- The Discriminator, on the other hand, adapts to better distinguish real from fake data.

This process, better illustrated in Figure 3.2 [4], is iterated an arbitrary number of epochs with both networks getting better at their task. The loss used for this model is an adversarial loss, that measures how well the discriminator can distinguish real from fake data. The generator tries to minimize this loss, while the discriminator aims to maximize it. The network reaches convergence when the generator produces data that is so realistic that the discriminator cannot distinguish it from real data. At this point the network can be used to generate data similar to the real one, to be used for the chosen task.

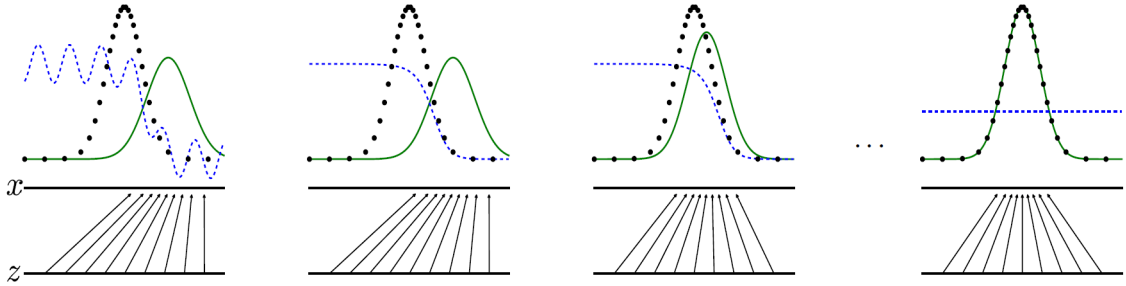


Figure 3.2: At first G (green line) is fixed, and the discriminative distribution (blue dashed line) is trained, having room for improvement. On the second step we obtain D*, the optimal discriminator that separates real data distribution (black line) from G. At the third step, D is fixed and G is trained, pushing its distribution towards the real data one. After many steps, G approximates well the real distribution so D cannot distinguish generated samples from real data.

GANs have revolutionized the field of generative modeling and have the potential to create highly realistic and diverse synthetic data. However, training GANs can be challenging and may require careful hyperparameter tuning and monitoring to achieve desirable results.

3.1.4 Adversarial Autoencoders

Adversarial Autoencoders combine elements of Autoencoders and GANs. Their structure is the same as a Variational Autoencoder, with the latent space following a multivariate Gaussian distribution, with the addition of a discriminator

network, similar to the GANs. The Discriminator has to distinguish from samples taken from the true latent space distribution and the ones from the encoded distribution. Basically, Adversarial Autoencoders are a GAN with a Variational Autoencoder as Generator, as illustrated in Figure 3.3 [33]. The training process is adversarial, as done in a GAN, where the Generator aims to reconstruct (and not generate from noise) the input data, and the Discriminator aims to separate reconstructed data from real data, until the Generator reconstructs data realistically enough to match the assumed latent space distribution.

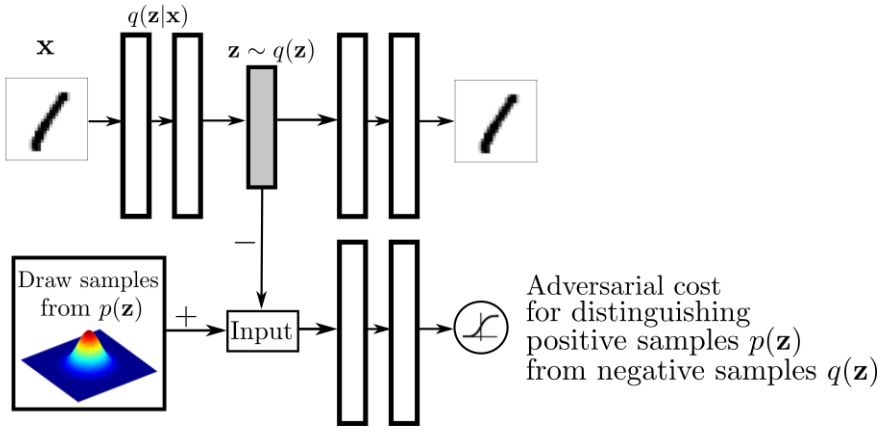


Figure 3.3: Structure of an Adversarial Autoencoder.

Adversarial Autoencoders combine the benefits of autoencoders, which are excellent at data compression and feature extraction, with the adversarial training from GANs, which enables the generation of realistic and diverse data. For this reason they excell in denoising, new data generation and anomaly detection, and are a fundamental component for our solution. Given the proper introduction to the theory behind generative networks, the next section will explain the latest state-of-the-art methods for Anomaly Detection, among which is present the network used in our solution.

3.2 State-of-the-Art

This section will explain the main idea behind the state-of-the-art methods regarding Anomaly Detection using generative networks. These methods exploit GANs and Autoencoders in different manners to achieve a higher accuracy or a faster inference time.

3.2.1 AnoGAN

This method uses a standard GAN, trained only on positive non-anomalous samples, in order to make the generator learn well the distribution of normal samples. Given that, when an anomalous image is encoded its reconstruction will be non-anomalous, since it does not know the distribution of anomalous samples. In this way, the difference between the input and the reconstructed image will highlight the anomalies. The two steps of training and detecting anomalies are summarized in Figure 3.4 [3].

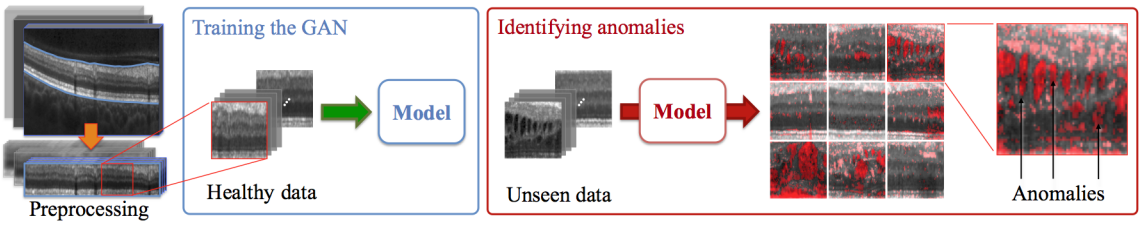


Figure 3.4: Pipeline of the AnoGAN method.

The performance on the training model is determined through two types of loss functions: a *residual loss*, which measures the dissimilarity between the query sample and the generated sample in the input domain space; and a *discriminator loss*, which can be either the result of the sigmoid cross-entropy of the generated image fed to the discriminator, or a feature matching loss using features extracted from a discriminator layer, in order to check if the generated samples has similar features to the input one (as done in the latest release of the paper).
Overall, AnoGAN was one of the first methods to implement GANs for ANomaly Detection, introducing a new mapping scheme from latent space to input data space and using said scheme to define an anomaly score. But this anomaly score is difficult to interpret, since it is not in the probability range, and the network requires a non negligible number of optimization steps at each new input, reason why it is not suitable for real-time application as the problem discussed in this thesis.

3.2.2 EGBAD (Efficient GAN-Based Anomaly Detection)

EGBAD [1] extends the GAN framework using a pipeline called BiGAN, which adds an Encoder to the GAN that learns the inverse of the Generator, in order to learn the mapping from latent space to data and vice versa. Another change

to the GAN pipeline is that the network takes in input pairs of data, composed of a data sample and an auxiliary component, which may be a class label or data from other modalities (feature derived from conditional GANs).
The addition of an Encoder was done to overcome the problems of AnoGAN, since it is now able to inverse map the data and no longer need to optimize at each input to compute the anomaly score.

3.2.3 GANomaly

The first Deep Learning model used in the proposed solution is based on GANomaly [2], which aims to improve the previous works while incorporating their main ideas. As in AnoGAN, the generator network is trained on normal samples to learn their normal latent space. The structure is like a normal GAN, with a generator and a discriminator, but the difference lies in the former.
The generator network consists of three elements in series: an Encoder, a Decoder and a final Encoder, with the two Encoders having the same architecture. The first couple of block work as a normal Autoencoder, encoding the input and reconstructing it (outputting always normal samples, since it does not know the latent space of anomalous samples), and the final Encoder at the end of the generator structure helps, during the training phase, to learn to encode the images in order to have the best possible representation of the input that could lead to its reconstruction.
The Discriminator works as in a standard adversarial training, discerning between real and generated data. The whole pipeline can be seen in Figure 3.5 [2].

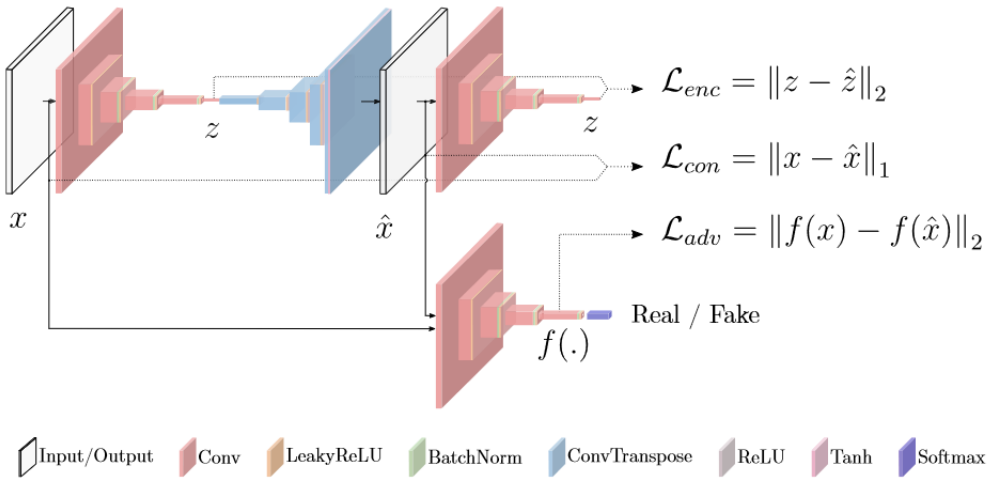


Figure 3.5: Pipeline of the GANomaly method.

The other main contribution of GANomaly is the introduction of the generator loss as the sum of three losses, while the discriminator loss remains the standard one. The generator loss is composed of three components:

- *adversarial loss*: Takes inspiration from the feature matching loss of AnoGAN. Checks if the generated sample and the real one have similar features, using features taken from a discriminator layer;
- *contextual loss*: Loss that permits to the generator to learn contextual information about the input data through the L_1 norm between the input data and the reconstructed data;
- *encoder loss*: Loss used to let the generator learn how to best encode a normal image through the difference of the encoding of the generator and the encoding of the final Encoder.

The resulting generator loss will be the weighted sum of these three losses, with the weights as adjustable parameters. The use of three losses is done to optimize the individual sub-networks of feature matching, reconstruction capability and latent space distribution. The combination of these three losses allows to better adjust the network to the problem through the weights (if the images are simple to reconstruct it may be useful to focus more on the latent vector). The paper also suggests a more interpretable way of computing the anomaly score, by also calculating it for every sample and in the end applying feature scaling to have the scores within the probability range.
Since an Encoder is learned during training, a big optimization process as in AnoGAN is not needed, and also the use of an Autoencoder makes the learning process faster. The final anomaly score is also easier to interpret w.r.t. the previous works (but it is difficult to compare the results with the previous models), and the new contextual loss can be used to localize the anomaly in the input. The addition of the new losses allows to detect anomalies both in the image space and in the latent space, but the results may not match: a higher anomaly score, that's computed only in the latent space, can be associated with a generated sample with a low contextual loss value and thus very similar to the input.

3.2.4 Skip-GANomaly

The structure of GANomaly can be improved by introducing elements present in the most recent networks, such as the Recurrent Neural Networks. Skip-

GANomaly explains this in the paper [5], and was the last Deep Learning model used in the proposed solution as improvement to GANomaly. The pipeline has the same principle as GANomaly, with some minor changes:

- The Autoencoder in the Generator network adopts skip connections between the Encoder layers and the Decoder layers, which contain Convolutional, BatchNorm layers and a modified version of the ReLU called LeakyReLU. This use of skip connections provides substantial advantages via direct information transfer between the layers, preserving both local and global information, and hence yielding better reconstruction;
- The Discriminator, besides working as classificator, is also used as feature extractor, such that latent representations of the input image and the reconstructed image are computed;
- The generator loss is the weighted sum of three losses as in GANomaly, but the encoder loss is substituted by the latent loss, which works the same way but instead uses the latent representations of the Autoencoder and the Discriminator.

The modified pipeline is illustrated in Figure 3.6 [5].

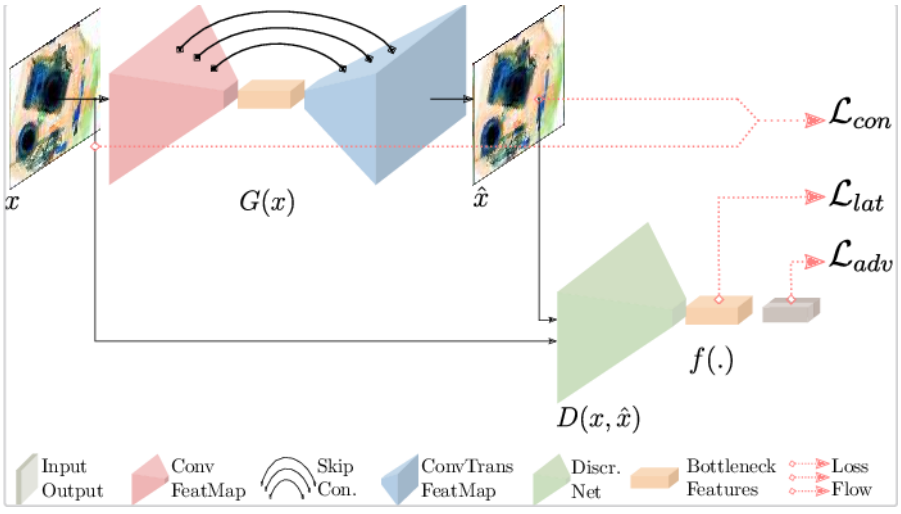


Figure 3.6: Pipeline of the Skip-GANomaly method.

As seen in the results of the paper, the introduction of skip connections enables a much stronger reconstruction capability of the network, reason why this pipeline was selected for our task. Another reason is the removal of one Encoder, which makes the network lighter and permits a faster training time. The choice of this model was dictated by the much faster inference time w.r.t.

the previously discussed methods and the ability to "shift" the attention of the network in the desired sub-network (features, reconstruction or latent space). These advantages are present also using GANomaly, but this model has also the capability to better carry the information from the earliest layers to the latest ones, given the skip connections.
After having fully explained the structure behind the network used, the next section will explain the full pipeline of the proposed method for this anomaly detection task, from input feature engineering to network training. The networks trained for the tests are both GANomaly and Skip-GANomaly, to better compare their performances.

Chapter 4

Proposed Method

In this chapter will be explained the hardware and software used for our proposed method, and then the whole pipeline. The chosen model, GANomaly or Skip-GANomaly, takes in input images that cannot be the raw images coming from the quality control system, due to size and complexity, and therefore need a preprocessing. The image has to keep only the important parts, discussed in Section 4.3.1, and be optimized, as seen in Section 4.3.2 and 4.3.4. These preprocessed images are used as input for the model for training and testing, discussed in Section 4.4.

4.1 Hardware and Software configuration

The hardware employed for this solution is composed of a computer running on Ubuntu 20.04.6, equipped with a 12 GB NVIDIA RTX A2000 GPU used for the model training. All the pipelines, based on GANomaly and Skip-GANomaly, were developed on Python through the *PyTorch* framework, known for its flexibility and dynamic nature. This framework allows us to quickly build and modify a Neural Network and accelerate the training process, for example using the Autograd library to automatically compute gradients of tensors w.r.t. other tensors. Other libraries used were:

- OpenCV 4.7.0, a very useful computer vision and image processing library designed to provide a wide range of tools and functions for tasks related to computer vision and image analysis;
- NumPy 1.24.1, fundamental for numerical and scientific computing, where it provides support for large, multi-dimensional arrays and matrices, along

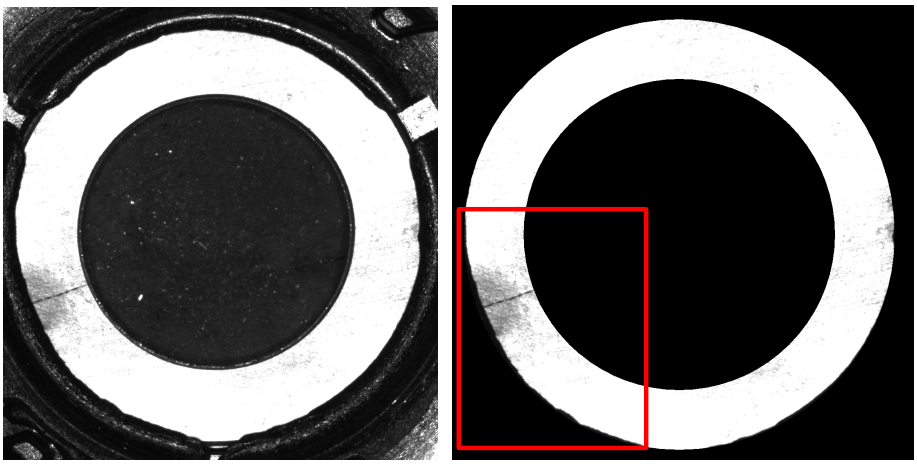
with a collection of mathematical functions to operate on these arrays;

- Pandas 1.5.3, library built on top of NumPy designed for data manipulation and analysis, providing easy-to-use data structures and functions for working with structured data, making it a powerful tool;

- Scikit-learn 1.2.1, which provides a set of tools for machine learning and data mining, as built-in functions for the metrics of Anomaly Detection.

4.2 Dataset management

The company made available a dataset of images that were processed by the previous anomaly detection system, where all the components in the images come from a real-world production line. The process of identifying anomalies with this system can be seen in Figure 4.1. Together with an examination by hand of all processed samples (to avoid having mislabeled images) we obtained a reliable ground truth to use in the training of the model, with a total of 1586 normal samples and 84 anomalous samples, which are not much, but we will use some data augmentation techniques explained in Section 4.3. We can also see that the dataset is highly unbalanced, but our model exploits this imbalance to strengthen the understanding of the normal samples distribution and identify the few anomalous ones. The dataset was then splitted into a training set of 1270/0 samples (compliant/defective), a validation set of 158/42 samples and a test set of still 158/42 samples, without counting the data augmentation techniques.



(a) A sample with a crack on the left side.

(b) The system returns the disk with the highlighted anomaly.

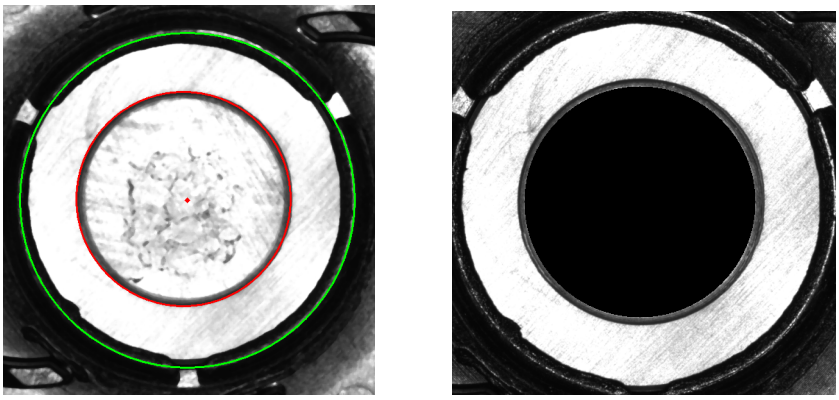
Figure 4.1: Example of processed image using the previous system.

4.3 Image preprocessing

This section will explain the general feature engineering applied to the original images, and then the different types of preprocessing tested. The processed images are used as input for the Deep Learning model, which is GANomaly or Skip-GANomaly.

4.3.1 Image cropping

The original images taken from the company pipeline are a 2591x1944 three-channel matrix, and cannot be used for the chosen Deep Learning model due to their large size. The first step is grayscaling the image, since the disk and the cracks are all in black and white. An important fact is that the image is not limited to the circular part we are interested, but also includes a part of the electric motor, which can be removed without drawbacks. However, the removal operation must be done carefully, in order to preserve all the disk and cut out as much electric motor as possible. A cut too shallow still keeps many unwanted features from the electric motor, which are not related to the presence of an anomaly and are therefore harmful, but on the other hand a cut too aggressive removes parts of the disk that in the worst case contain an anomaly, so it must be avoided. For this reason it was decided to use a Computer Vision tool to detect circles in the image, known as the Hough Transform. This technique is particularly useful to detect lines and other simple geometrical shapes, and it does so by representing points in an image as mathematical entities in a parameter space (in our case as center of a circle (x,y) and its radius r) rather than in the image space itself. After applying an initial edge detection with classic techniques (e.g. Canny), for each edge point found in the edge-detected image the algorithm creates an accumulator matrix where each cell corresponds to a potential center of a circle (x,y) and its radius (r). For every combination of (x,y,r), if the current edge point is consistent (the gradient direction at (x,y) is compared to the expected gradient direction of a circle with radius r) the corresponding cell in the accumulator matrix is incremented (has a "vote"). After accumulating votes in the accumulator matrix, the cells with most votes (relative to a (x,y,r) triplet) are the detected circles, as illustrated in Figure 4.2a (the full image cannot be shown as the original samples are protected by an NDA).

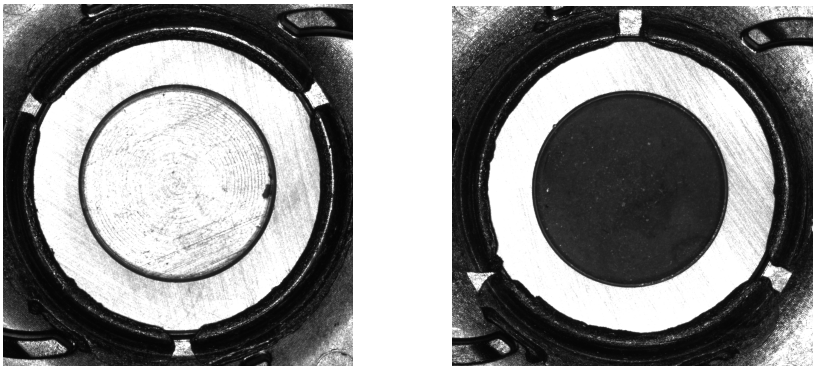


(a) Example of circle detection on a sample.

(b) Sample after cropping and obscuring the central part.

Figure 4.2: Example of use of the Hough Transform and cropping.

The Hough Transform is a very reliable tool for simple shape detection, but is computationally expensive, especially if the input image size is rather large. Another problem of the Hough Transform is the tuning of parameters to avoid multiple detections of the same circle or circles of sizes that do not interest us, but since we are looking for a single circle and roughly know its size we can threshold the results and easily obtain the correct detection. Given the obtained center of the circle and its radius we can crop out the motor parts.
Another part not useful for us (and could make the training more difficult) is the central hollow part of the disk, which is the part of the motor where the disk is placed and is not correlated to the presence of an anomaly or not. To solve the problem we can still use the Hough Transform to also detect the inner circle of the disk and draw a filled black circle where the hollow part is, as done in Figure 4.2b. The difference of the hollow part in different samples can be a problem, because, as illustrated in Figure 4.3, they have different colors and features, as scratches and lighting, reason why obsuring it is so important.



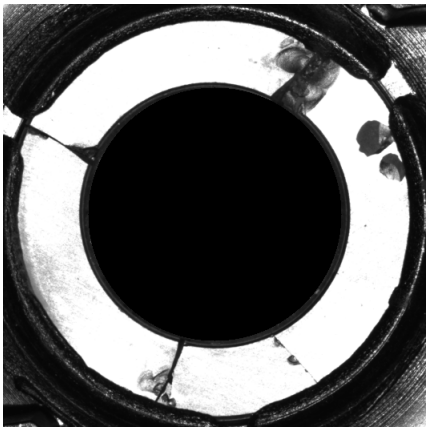
(a) A sample with a white hollow part and some scratches.

(b) A sample with a black hollow part, very different from the left one.

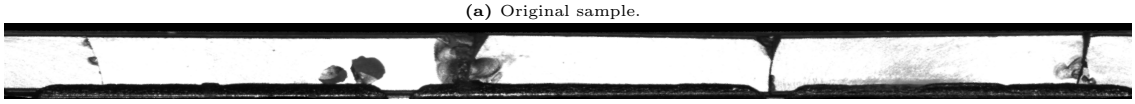
Figure 4.3: Example of different hollow parts in the samples.

4.3.2 Image unwarping

The obtained cropped images contain the strictly necessary features for the model, and therefore can be used for the training. But the obtained image has a dimension of 1200x1200, still too big for GANomaly and Skip-GANomaly, and an excessive compression of the image may cancel out small cracks and deformations in the image. After running different tests, it was found that the largest possible image that can be processed by the model is a 512x512, which is a significant compression and may remove important features. For this reason it was decided to avoid compressing the image and instead extract and use only the parts of the disk. Very little of the image is useful (it has a lot of obscured parts), and therefore the method used to extract the parts of the disk is an unwarping tool, which takes the disk and transforms it from a circular shape to a rectangular shape, optimizing the images by keeping only the useful parts. This transforms the image from a 1200x1200 to a 256x3833, as illustrated in Figure 4.4, which can be directly used to train the model or split into pieces to both have more samples and improve the learning process.



(a) Original sample.



(b) Sample processed with the unwarping tool.

Figure 4.4: Sample transformed from circular shape to rectangular shape.

The splitting process consists in dividing the whole strip by width in non-overlapping different pieces with dimensions more suitable for the model (e.g. splitting in 2 results in pieces of size 256x1792, splitting in 4 results in pieces of size 256x1024 and so on), where the labels of the patches were assigned by hand using the ground truth of the whole image.

This technique introduces a need to check not only the performance on the single patches but also on the whole image, because splitting in many pieces results in a high accuracy on the single patches, but it is enough to mislabel one patch to mislabel the whole image, so it must be done with caution. In this thesis we tested a splitting in 2,4,8,16 and a special case of 27 pieces, where each piece is a 256x256 image and the next image has half overlap with the previous image (to avoid the case where a crack is between two patches). In Figure 4.5 can be seen how an image is splitted according to the split-27, taking the image in Figure 4.4 as example.

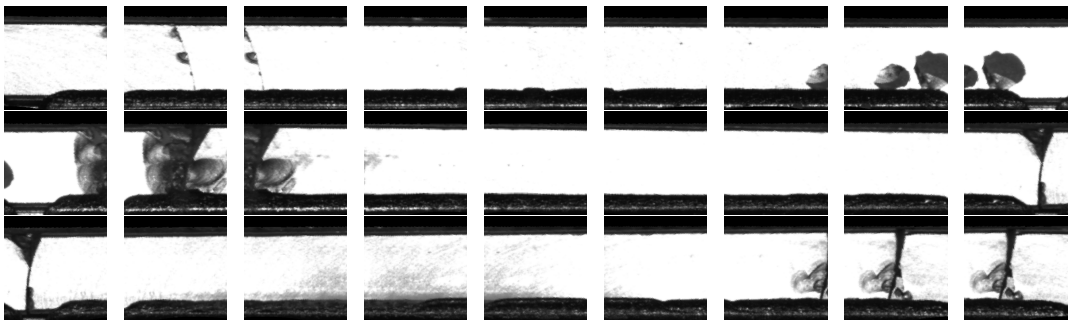


Figure 4.5: Sample splitted into 27 pieces with a half overlap between patches.

4.3.3 Image 4-split

Another technique that was briefly used was the 4-split, which was used before the discovery of the unwarping tool. This technique takes the cropped image, divides it into 4 parts (upper-left quadrant, upper-right quadrant, lower-left quadrant and lower-right quadrant) and rotates each piece to make it look like it is the top-left one, as shown in Figure 4.6. It is an effective way to increase the number of samples and avoid aggressive compressions of the image, since the obtained pieces are 600x600 matrices and can be compressed to 512x512 without major data loss. The main reason why it was discarded is that it still keeps the central part, while the unwarping tool only uses the disk parts.

4.3.4 Image filtering

Another step in the preprocessing phase is the filtering of the image before passing it to the network. This is done to facilitate the learning process of the network via reducing the number of features of the input by keeping the most relevant ones.

The most common and non-invasive method of filtering is using a edge-enhancing

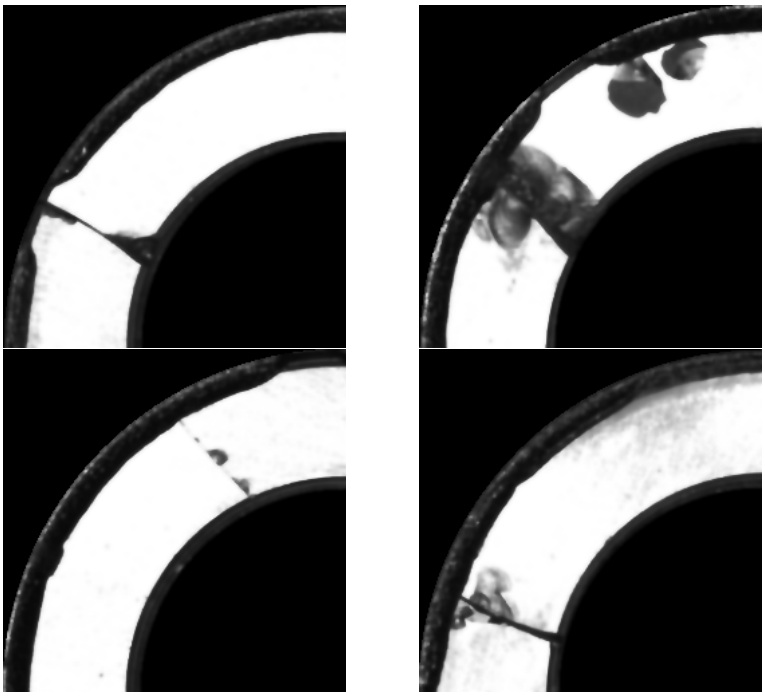


Figure 4.6: Sample processed with the 4-split.

filter, such as Bilateral filter, used in Figure 4.8a, which keeps intact the edges and smooths all the other parts. Enhancing the edges is a way to highlight better the cracks on the disk, while smoothing out the parts that may be dirt or scratches.

A more invasive type of filtering used is a modified version of thresholding, in which all the pixels with intensity above a certain threshold were put to the maximum intensity (pure white). This was done to erase dirt and scratches, leaving only the cracks, as illustrated in Figure 4.8b.

The strongest type of filtering that was used is the region growing algorithm. This method starts from some points called seeds (in our case the points with lowest intensity) that form the initial region, and from these points it iteratively checks the neighboring points, and if they are similar enough they are added to the region. The process is iterated until each point of the regions is processed. These regions are then "printed" into a full white image, resulting in just pure black points (cracks/deformities) and pure white points (normal parts of the disk). For the same reason of the thresholding, it was done to remove dirt and scratches from the disk, and the difference of impact on the image w.r.t. the previous filterings can be seen in Figure 4.8c.

However, these filterings were later discarded, mainly because these are Computer Vision tools that could conflict with the learning of the model and because the

resulting images lack features present in the original images that can be useful for the network. Other reasons are: the aggressiveness of the thresholding and region growing risks to erase parts or all the anomalies in an image (as can be seen in Figure 4.7); the dependence on the image intensity and on the current typology of anomaly (if in the future the anomalies become more gray this filtering completely erases them); the smoothing on the other hand is not general enough for this type of images (a certain tuning of the filter works for some images and not for others), so it was decided to avoid filtering the images.

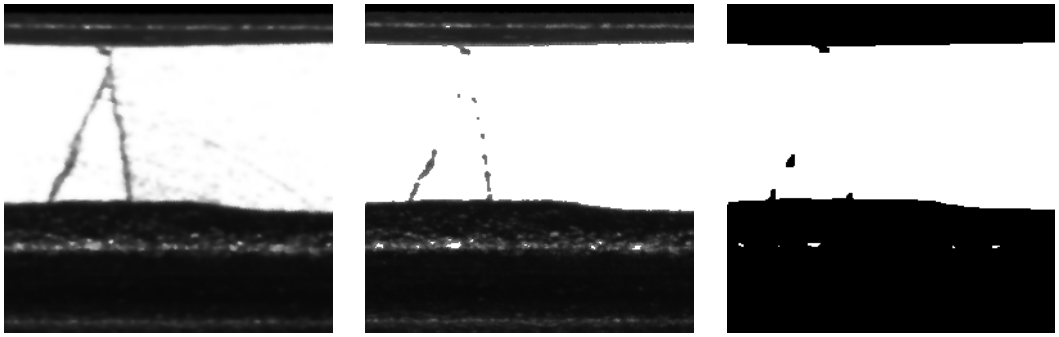
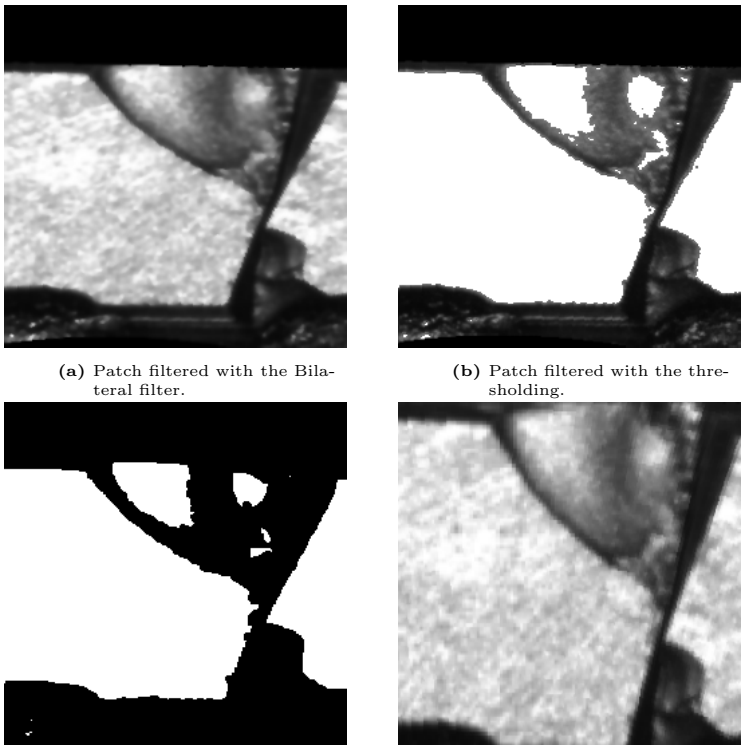


Figure 4.7: Example of a small crack that gets erased by both thresholding and region growing.

Instead it was used another type of preprocessing to improve the unwarping: since the unwarping is not 100% accurate, some useless parts still remain in the image (represented by the black parts on the upper and lower part of the patches). Since they could make the training process more difficult by introducing features not correlated to the presence of an anomaly it was developed an algorithm that erases these black parts, in order to keep only the disk. The algorithm scans the patches row by row, and if a row has an average intensity below a fixed threshold it gets set to 0. After the scan the algorithm keeps only the rows of the image with an average value above zero, as shown in Figure 4.8d.

4.4 Model training

The preprocessed input image (cropped, unwarped and with the black parts removed) is used as input for the chosen network, in this case being GANomaly [2] or Skip-Ganomaly [5]. The process of the image is done according to the scheme in Figure 4.10. Originally the network was trained and tested on the following datasets:



(a) Patch filtered with the Bilateral filter.

(b) Patch filtered with the thresholding.

(c) Patch filtered with the Region growing algorithm.

(d) Patch without the upper and lower black parts.

Figure 4.8: A single patch processed with all the tested filterings.

- CIFAR-10 [5]: This dataset is made from 10 different classes of objects, respectively airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Experiments for this dataset had the one versus the rest approach, where one class between the available was labelled as anomalous, yielding ten different anomaly cases;

- University Baggage Dataset (UBA) [5]: This in-house dataset comprises dual energy X-ray security image patches extracted via a 64x64 overlapping sliding window approach. The dataset contains 3 abnormal sub-classes: knife, gun and gun component;

- Full Firearm vs Operational Benign (FFOB) [5]: this dataset comprises both expertly concealed firearm (threat/anomaly) items and operational benign (non-threat/normal) imagery from commercial X-ray security screening operations (baggage/parcels).

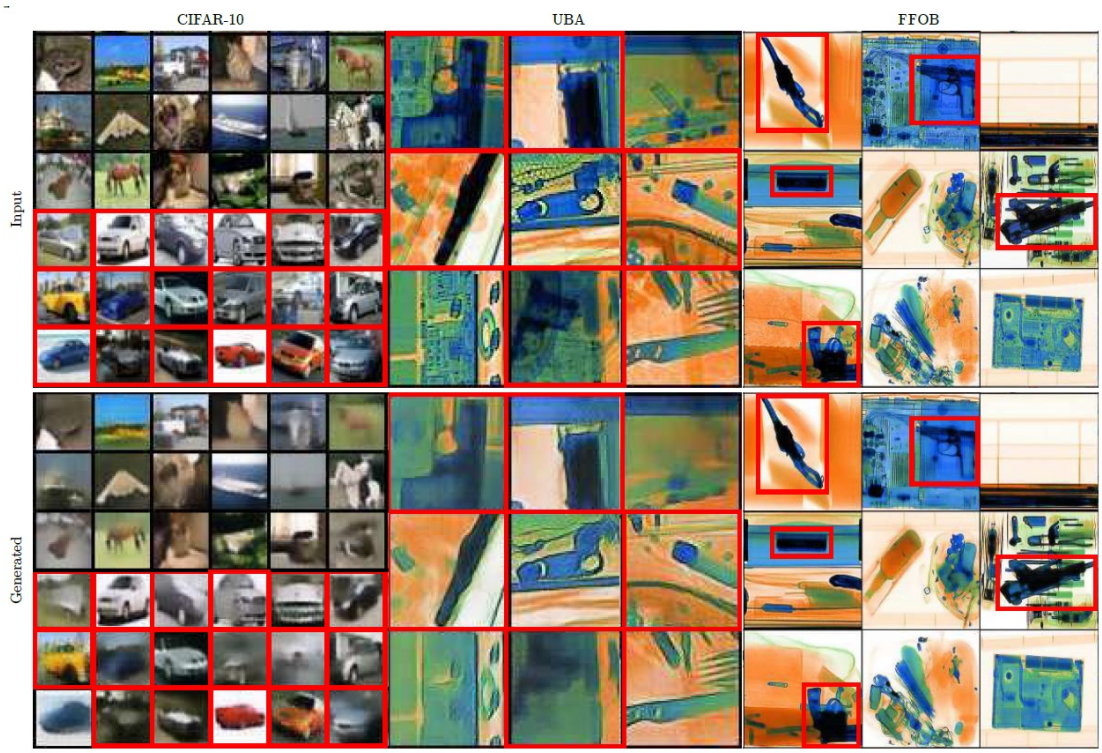


Figure 4.9: Images from the CIFAR-10, UBA and FFOB datasets.

As seen in Figure 4.9 [5], the tested datasets had a very different typology of anomalies w.r.t. to our problem, where in some cases a defective component is almost identical to a compliant one. For this reason some parts of the network were changed to better fit the problem.

For example, the input tensor originally accepted only square inputs (e.g. 256x256 images), but it was modified to accept all kinds of tridimensional matrices. But a non-square input is not ideal for the Discriminator, because in the original network the Discriminator takes in input the reconstructed image and returns a scalar, which represents the predicted class. Since it mainly operates with square operators such as the convolution, a non square input leads to returning a vector instead of a scalar, with size depending on the difference between width and height of the input (e.g., if the input is the whole unwarped image the Discriminator returns a vector of 57 components). This problem is easily fixed by putting a Fully Connected layer at the end of the network, in order to return a scalar.

Another small change is using the number of splittings as batch size (e.g., splitting in 8 pieces results in a batch size of 8), because the hardware could not handle many high dimensional input for batch (like a series of whole unwarped images) and for a better organization of the network, since it gets trained by processing all the patches of a single image at once. In this way the network not only learns

how to classify the single patches but it may also learn a correlation between the patches in the batch, since they all come from the same image. In order to visually keep track of the learning process, the model also saves in a folder the training samples and their reconstructed version.

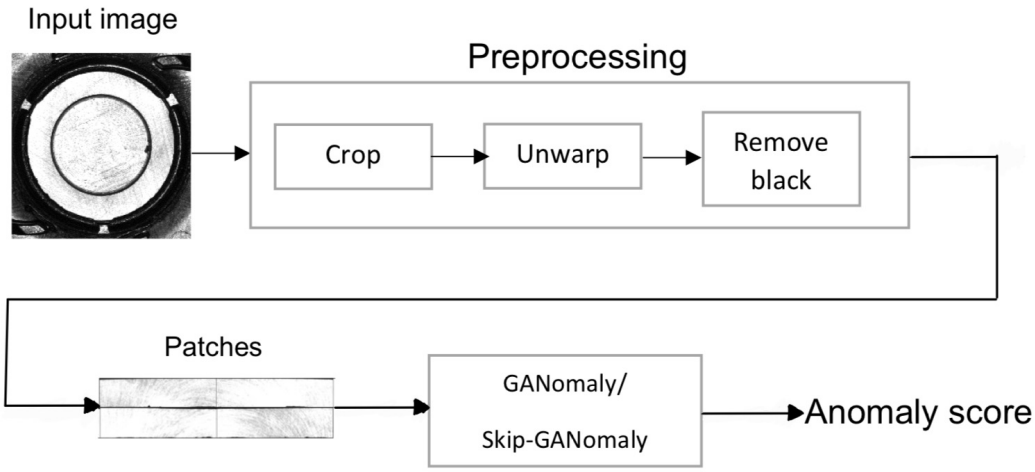


Figure 4.10: Scheme that explains how the main pipeline returns the anomaly score of an image.

At the end of each epoch the model is tested on the validation set (158 normals/42 anomalies), which is composed of also anomalous samples, returns the AUC of the ROC, explained in Section 2.2, and saves in a folder both the original validation samples and the reconstructed ones, in order to have a better understanding of how the network operates. If the AUC is better than the previous epochs the model saves the weights on a folder, so it does not need an early stopping function (a function that stops the training after k epochs if the model is not getting better). At the end of the training (1270 normals) the model is tested on the test set (158 normals/42 anomalies), which was never considered until now, and saves on a folder each batch named with the highest anomaly score of the batch. This last operation represents the classification of the whole image (the batch represents an image and all its patches, the highest score of the batch is the final score of the whole image), where the batches with at least an anomaly and the normal batches are saved in different folders. An important fact is that the original network normalizes the obtained scores on the used dataset, but this was removed due to the fact that this assumes that the dataset has always at least one anomalous sample (if not, the normal sample with the highest anomaly score gets set to 1 and thus always above the anomaly threshold).
To better interpret these images and their score it was developed a simple al-

gorithm, separate from the pipeline, that reads the scores of the images in the two folders and estimates the best threshold to separate normal and anomalous samples. This algorithm allows us to have another performance metric of the model, directly measuring the accuracy of the classified images.
In order to simulate the model function in a real-world production line, it was developed a separate pipeline to test the inference time. The network is not trained, but is only used for testing by loading the desired weights of the network. Instead of using the preprocessed dataset and training, this pipeline takes in input from the original image from the quality control system. The image is preprocessed by cropping, unwarping and removing the black parts and is passed to the model, which returns the anomaly score and saves the image in a folder with the anomaly score as its name. This pipeline is useful to understand the inference time of the model, and to keep it under one second.

Chapter 5

Experimental Setup

In this chapter will be explained the different parameters that have been tuned on the models, and then will be discussed their performances with different pre-processings and tunings. It will then follow a brief examination of the results and the inference times on the test phase. As cited on Sections 3.2.3 and 3.2.4, GANomaly was the first tested model and Skip-GANomaly the latest model.

5.1 Parameters

The parameters discussed in this section are present in both GANomaly and Skip-GANomaly, since they share a very similar structure. The parameters tuned in the tests are:

- Learning rate (lr): weight assigned to gradient-based optimization functions that determines how much the resulting gradient affects the change in the weights of the network. A learning rate too high may induce a zig-zag behavior, wasting time near a minimum point, and on the other hand a learning rate too low may slow down the learning process until never reaching convergence, so it must be tuned carefully;

- Triple loss weights (w_{adv}, w_{con}, w_{lat}): the weights of the combined loss discussed in 3.2.3 and 3.2.4. The tuning of these parameters permits to shift the attention of the network to reconstruction, feature matching or latent space;

- Reconstruction loss and normalization: a typology of loss function and choice of normalizing the input. The original networks used an L1 loss on the comparison between reconstructed input and original input, but it was

decided to also try an L2 loss and test if not normalizing the input led to a better reconstruction;

- Reconstruction error: while in GANomaly the anomaly score is the square difference of the latent vector of the generator and the latent vector of the final Encoder, in Skip-GANomaly the score is calculated through the weighted sum of the reconstruction error and the latent space error, with the sum of the weights amounting to 1. This proportion can be adjusted to focus more on the reconstruction or the latent vector (e.g. a reconstruction error weight of 0.9 indicates a latent error weight of 0.1);

- Extra layers: the structure allows the addition of an arbitrary number of extra layers at the beginning of the Generator Encoder, and to add the same number at the end of the Decoder. These layers are composed of a Convolutional Layer with the output processed by batch normalization and ReLU. However, since the addition of extra layers not only slows down the network but has proven to slightly worsen the performance it was decided to keep it at 0;

- Layer depth (ngf/ndf): the layers of the network have a *width* x *width* x *depth* size, so that every element of the *width* x *width* matrix is represented by a vector of *depth* length. This parameter can be adjusted if the network needs to learn few features, by reducing it, or more features by increasing it;

- Latent vector size (nz): the size of the latent vector is crucial for learning the distribution of the normal data in the latent space, just as for the layer depth;

- Splitting: number of splittings applied to the unwarped image, if the unwarping was adopted.

After the given introduction to the tunable parameters, the next sections will discuss the results obtained on both models with different configurations.

5.2 Skip-GANomaly

In this section will be discussed the results obtained with the Skip-GANomaly model using different settings for the input images, preprocessing and parameters.

The default configuration of the network is [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; *nz*: 100; *ngf/ndf*: 64; *learning rate*: 0.001; w_{adv}: 1; w_{con}: 50; w_{lat}: 1], and all tests were performed on 30 epochs.

5.2.1 Cropped image

The first typology of test was done using the cropped image compressed to a 512x512 matrix, without any further preprocessing. The training and testing of the model took a total of 3.5 hours, using the standard dataset as explained in 4.2. The optimal configuration of the network is the default one with the learning rate set at 1e-06, obtaining the results expressed in AUC of the ROC curve illustrated in Table 5.1.

Input	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
Cropped image	0.518	0.518	0.508	0.508

Table 5.1: Table with the AUC performance using the cropped images as input.

The obtained performances are quite low, almost similar to a random classifier. For this reason, it was decided to explore other solutions.

5.2.2 4-split

As cited in Section 4.3.3, a more optimal way to process the input w.r.t. the cropped image is the 4-split technique. In this way, the compression of the image is much less aggressive, being 600x600 and 512x512 very similar in size. The dataset used is 4 times larger than before, with the training set composed of 5080 normal samples, and the validation and test set composed of 632 normals and 168 anomalies each. The training and testing of the model took 3.5 hours using the default configuration with the learning rate set to 1e-06, and obtained the results in Table 5.2.

Preprocessing	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
4-split	0.393	0.440	0.396	0.421

Table 5.2: Table with the AUC performance using the 4-split preprocessing.

The obtained results prove that, even with less aggressive compressions and more samples, keeping parts of the image other than the disk affect negatively the learning process, reason why all the other tests use the unwarping tool.

5.2.3 Unwarping

In the tests discussed in this subsection all the trials use images preprocessed by the cropping, unwarping and black parts removal. The objective of this subsection is therefore focus on the single parameters, introduced in Section 5.1, and find the optimal value.

Image filtering

The first test that was done using the unwarping tool comprehended also the filtering of the images, which were later discarded as explained in 4.3.4. These tests were performed only on the splitting in 27 patches, because initially it was decided to use only this type of splitting. When the study of the solution explored other numbers of patches, filtering the images was a procedure that was already discarded, reason why it was tested with only one typology. The training of the model was performed in a total of 6 hours, with configuration of the network parameters [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; nz: 100; ngf/ndf: 64; *split*: 27; *learning rate*: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1] and a dataset composed of a training set of 34290 normals and validation/test sets composed of 4266 normals and 1134 anomalies.

Filtering	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
Bilateral filter	0.476	0.492	0.486	0.553
Thresholding	0.671	0.633	0.633	0.664
Region growing	**0.681**	**0.764**	**0.722**	**0.660**

Table 5.3: Table with the AUC performance on different image filterings.

From Table 5.3 can be seen that the region growing algorithm has the best performance, but is still relatively low w.r.t. the results in the next subsections.

Splitting

Since the network uses unwarped input, the most important part is finding the optimal number of splittings, to balance the accuracy on the single patches and the accuracy on the total image. As introduced in Section 4.3.2, the splittings tested were 2,4,8,16 and 27. The training of the model was performed in a total of 4 hours for all variants except the splitting in 27 patches, which took 6 hours. This is due to the fact that the batches were computationally heavier, since the patches are overlapping and therefore the total dimension is larger than the other splittings. The configuration of the network parameters is [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; nz: 100; ngf/ndf: 64; *learning rate*: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1].

Splittings	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	0.692	0.692	0.740	0.740
2	0.661	0.615	0.698	0.684
4	**0.735**	**0.648**	**0.793**	**0.757**
8	0.723	0.624	0.697	0.634
16	0.744	0.652	0.704	0.587
27	0.752	0.553	0.645	0.506

Table 5.4: Table with the AUC performance on different splittings.

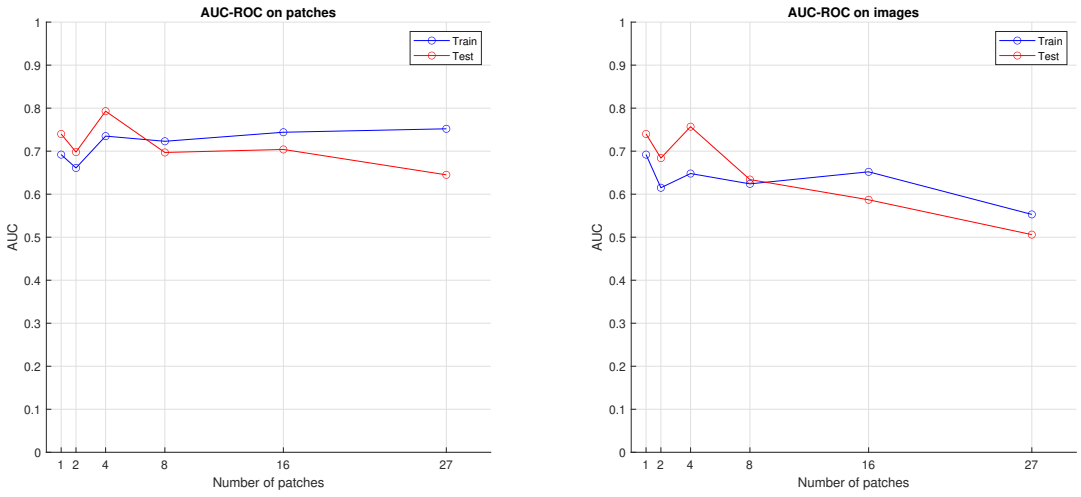


Figure 5.1: Plots of the AUC of different splittings configurations.

As seen in Table 5.4 and Figure 5.1, the optimal splitting number is 4, which had the highest and most balanced performance between patches and image, after the split=1, which has been proven not to be a bad configuration. It must be noted how the split=1 means passing the whole unwarped strip as input, which is equal to passing the whole cropped image. The difference is that the unwarpings of split 1 are composed exclusively of parts of the disk, while the cropped image has many unnecessary parts, and in Table 5.4 can be seen the improvement in performance w.r.t. Table 5.1.

Learning rate

The learning rate can determine if the model reaches convergence or gets stuck in a local minimum, reason why it is important to evaluate the optimal rate of the network. The training of the model took 4 hours, using the configuration [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; nz: 100; ngf/ndf: 64; *split*: 4; w_{adv}: 1; w_{con}: 50; w_{lat}: 1]. An important fact is the use of the split 4, meaning that the dataset is quadrupled and organized as for the 4-split.

Learning rate	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
0.001	0.685	0.574	0.658	0.627
0.0001	0.365	0.483	0.415	0.423
1e-05	0.529	0.586	0.507	0.539
5e-06	0.520	0.548	0.533	0.497
1e-06	**0.735**	**0.648**	**0.793**	**0.757**
5e-07	0.7665	0.670	0.674	0.631
1e-07	0.629	0.588	0.513	0.527

Table 5.5: Table with the AUC performance on different learning rates.

From Table 5.5 can be determined that the optimal learning rate is 1e-06, even if 5e-07 performed better on the training set.

Layer depth

In order to correctly compress and decompress the information of the input distribution, the layer depth must be adequate for the problem. The training of the model variants took a total time that is dependent on the layer depth, and can be

ngf/ndf	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC	Training Time (h)
1	0.413	0.536	0.407	0.525	0.5
2	0.633	0.578	0.694	0.604	0.5
4	0.761	0.733	0.701	0.581	1
8	0.472	0.519	0.398	0.440	2
16	0.481	0.498	0.484	0.563	3
32	0.804	0.731	0.765	0.743	3.5
48	0.671	0.642	0.653	0.605	4
64	**0.735**	**0.648**	**0.793**	**0.757**	**4**
80	0.495	0.498	0.478	0.506	5
96	0.530	0.557	0.447	0.447	6

Table 5.6: Table with the AUC performance on different layer depths.

seen in Table 5.2. The configuration of the network parameters is [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; *nz*: 100; *split*: 4; *learning rate*: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1].

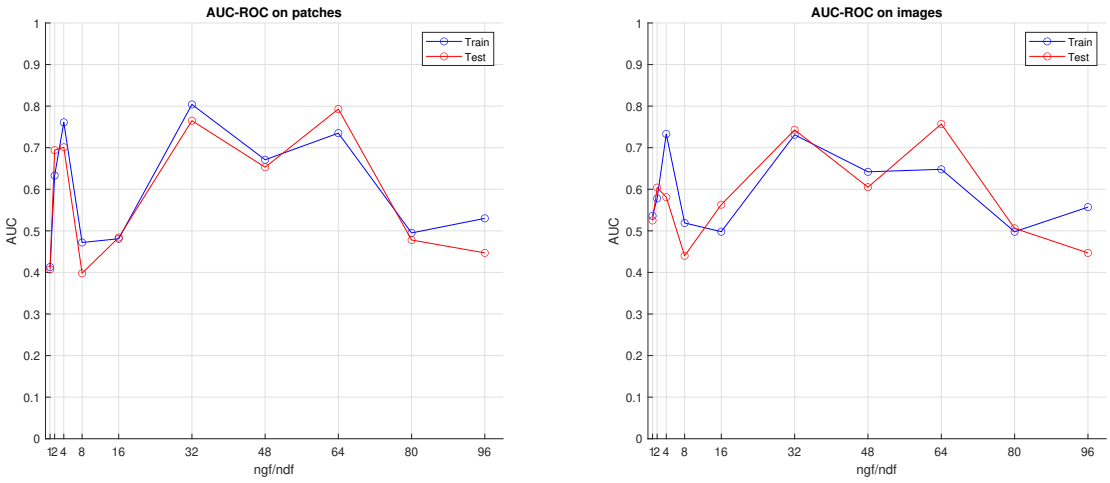


Figure 5.2: Plots of the AUC of different layer depths.

As can be seen in Table 5.6 and Figure 5.2, the best performances are on the configurations using a layer depth of 32 and 64. Since 64 performed better on the test set, it was decided to use that for future tests.

Latent vector size

The size of the latent vector is crucial for the learning of the distribution of the normal samples, and while a small vector may force the network to learn fewer features than the needed number, an unnecessarily large vector does not restrict enough the network to make it learn the important features. The training and testing of the model variants took around 4 hours, with the lightest model taking 3.5 hours and the heaviest taking 5 hours. The configuration of the network parameters is [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; ngf/ndf: 64; *split*: 4; *learning rate*: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1].

nz	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	0.606	0.702	0.686	0.688
5	0.647	0.612	0.593	0.577
10	0.459	0.516	0.404	0.401
25	0.449	0.488	0.412	0.489
50	0.437	0.467	0.511	0.468
100	**0.735**	**0.648**	**0.793**	**0.757**
150	0.469	0.546	0.481	0.452

Table 5.7: Table with the AUC performance on different latent vector sizes.

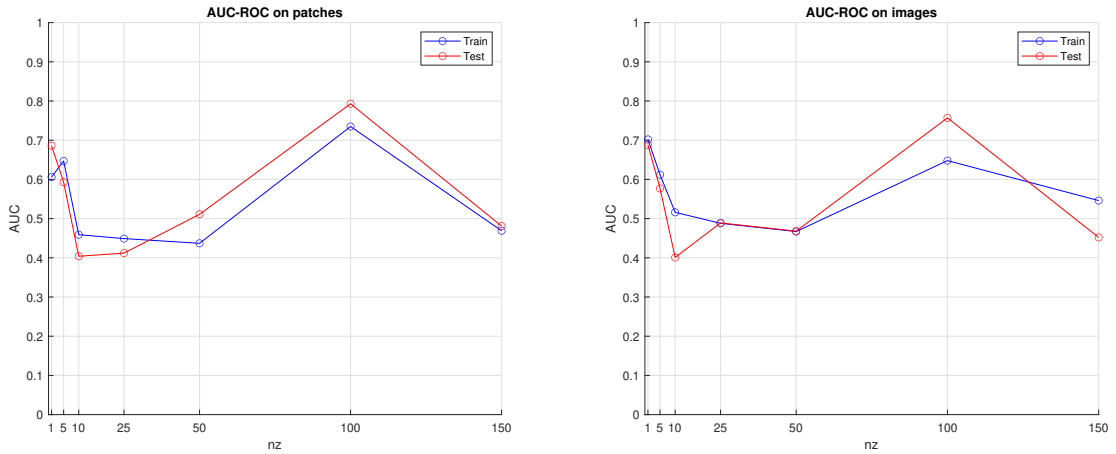


Figure 5.3: Plots of the AUC of different latent vector sizes.

As seen in Table 5.7 and Figure 5.3, the default configuration with size 100 had the best performance, together with 1. This can be explained by the rather

simple distribution of the normal samples, being very similar to one another, but with 100 the model can capture small features that may be crucial to identify anomalous samples.

Triple loss weights

The weights of the triple loss are between the most important parameters, since they can redirect the network attention where needed. Instead of tuning just the weights, it was decided to also change the reconstruction error according to the tested weight. The tests all took 4 hours each, since the change of the weights does not have computational complexity relevance, with configuration [$Loss$: L1; $Normalize$: True; nz: 100; ngf/ndf: 64; $split$: 4; $learning\ rate$: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1].
For the reconstruction weight, it was decided to use the default reconstruction error 0.9, in order to focus on the reconstruction part.

w_{adv}	w_{con}	w_{lat}	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	1	1	0.728	0.656	0.702	0.608
1	2	1	0.555	0.612	0.522	0.533
1	5	1	0.474	0.492	0.494	0.471
1	10	1	0.469	0.540	0.476	0.511
1	**50**	**1**	**0.735**	**0.648**	**0.793**	**0.757**

Table 5.8: Table with the AUC performance on different reconstruction weights.

From Table 5.8 can be seen that the default configuration with weight=50 was the best-performing model, even though the model with all weights set to 1 performed equally well on the training.
The second weight tested was the adversarial weight, responsible for the realistic reconstruction of the samples. Since the weights for the reconstruction and latent space are equal, it was decided to use a reconstruction error of 0.5, in order to have an anomaly score that equally depends on reconstruction and latent space.

w_{adv}	w_{con}	w_{lat}	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	**1**	**1**	**0.742**	**0.658**	**0.718**	**0.633**
2	1	1	0.480	0.487	0.510	0.520
5	1	1	0.444	0.477	0.450	0.472
10	1	1	0.417	0.423	0.507	0.520
50	1	1	0.376	0.450	0.409	0.426

Table 5.9: Table with the AUC performance on different adversarial weights.

Table 5.9 indicates that with all the weights equal the model has a better performance, by balancing latent space and reconstruction.
The last weight tested is relative to the latent space, and thus the reconstruction error is set to 0.1 to entirely focus on the latent vector.

w_{adv}	w_{con}	w_{lat}	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	**1**	**1**	**0.766**	**0.705**	**0.681**	**0.619**
1	1	2	0.415	0.471	0.344	0.395
1	1	5	0.553	0.569	0.613	0.568
1	1	10	0.444	0.516	0.474	0.509
1	1	50	0.438	0.422	0.461	0.501

Table 5.10: Table with the AUC performance on different latent weights.

From Table 5.10 can be seen that the network did not perform well focusing only on the latent space, reason why it was decided to continue with the configuration [*reconstruction error*: 0.9; w_{adv}: 1; w_{con}: 50; w_{lat}: 1]. This demonstrates how, for this particular problem, the network performs better when focusing on input reconstruction.

Loss and normalization

In order to improve the accuracy of the reconstructed inputs, it was decided to test different losses from the default configuration, even by not normalizing the input. The training of the model took 4 hours, with configuration [*reconstruction*

error: 0.9; nz: 100; ngf/ndf: 64; *split*: 4; *learning rate*: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1]

Loss	Normalize	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Testing Images AUC-ROC
L1	**True**	**0.735**	**0.648**	**0.793**	**0.757**
L1	False	0.756	0.645	0.756	0.764
L2	True	0.620	0.551	0.682	0.612
L2	False	0.619	0.571	0.698	0.631

Table 5.11: Table with the AUC performance on different losses and input normalization.

Table 5.11 shows how the default configuration, and in particular the L1 loss, performed better. The non-normalization of the input is a high-risk technique, since the values on the tensors are not bound between 0 and 1. In the early stages of the training, this leads to very high reconstruction losses, because the Generator is not trained enough to reconstruct input-similar data. If the network is not able to recover quickly it can lead to divergence, with the generation of white noise as fake data. However, if the network recovers the reconstruction losses are lower, especially using the L2 norm. The square of the error amplifies little differences between real and fake data, and the network is able to recreate more accurate samples. The use of the L2 loss and non-normalization has proven to be more effective than normalizing, even if by a slight improvement.

5.3 GANomaly

In this section will be discussed the results obtained with the GANomaly model, which was the initial model before it was substituted with Skip-GANomaly. Following the same principles of testing used on Skip-GANomaly, it was decided to do a confrontation analysis of the results of the two networks. All the tests discussed in this section were performed on 30 epochs and by processing the images only with cropping, unwarping and black parts removal, with default configuration [*reconstruction error*: 0.9; *Loss*: L1; *Normalize*: True; nz: 100; ngf/ndf: 64; *learning rate*: 1e-06; w_{adv}: 1; w_{con}: 50; w_{lat}: 1].

5.3.1 Splitting

The training of the model variants took the same time as for Skip-GANomaly, using the default GANomaly configuration.

Splittings	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	0.704	0.704	0.731	0.731
2	**0.838**	**0.864**	**0.872**	**0.873**
4	0.812	0.782	0.832	0.791
8	0.823	0.866	0.838	0.784
16	0.880	0.842	0.908	0.806
27	0.874	0.780	0.832	0.854

Table 5.12: Table with the AUC performance on different splittings.

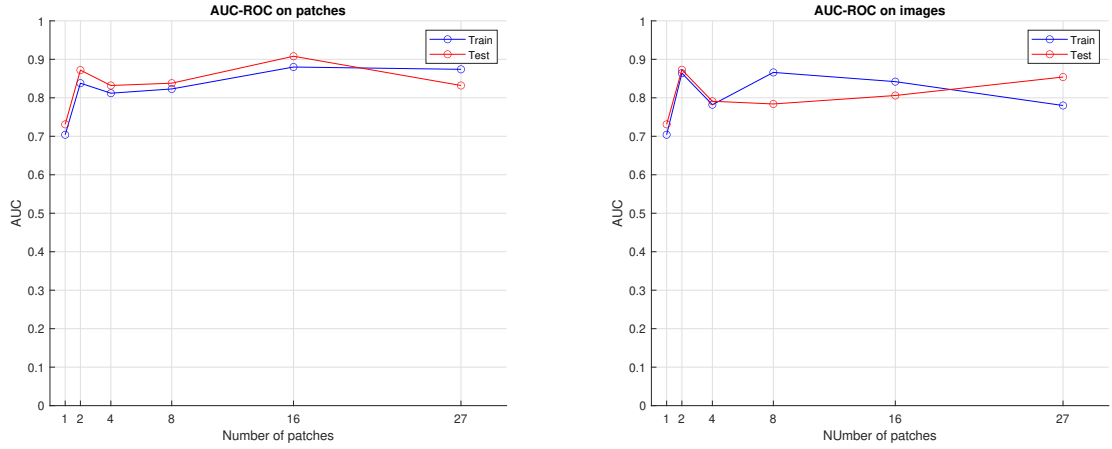


Figure 5.4: Plots of the AUC of different splittings configurations.

Differently from Skip-GANomaly, Table 5.12 and Figure 5.4 illustrate that the best-performing splitting is done with 2 patches. Even though the split=27 performed almost as well, taking into account the time of training split=2 is still to be preferred w.r.t. split=27. Examining the results on both networks, the overlapping of the patches has proven to be not effective enough.

5.3.2 Learning rate

The tests were performed using the default GANomaly configuration and the dataset was obtained with split=2, with a training set of 2540 normals and

validation/test set of 316 normals and 84 anomalies.

Learning rate	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1e-05	0.817	0.761	0.773	0.733
5e-06	0.880	0.865	0.783	0.762
1e-06	**0.838**	**0.864**	**0.872**	**0.873**
5e-07	0.824	0.802	0.858	0.846
1e-07	0.873	0.868	0.839	0.814

Table 5.13: Table with the AUC performance on different learning rates.

From Table 5.13 can be seen that the optimal learning rate is 1e-06, even if 5e-06 and 1e-07 performed better in the training.

5.3.3 Triple loss weights

The testing of the weights of the triple loss on GANomaly was performed by changing only the weights, differently from Skip-GANomaly, since the anomaly score is calculated from a square difference of latent representations. The tests were performed over 4 hours, using the default configuration and split 2.

w_{adv}	w_{con}	w_{lat}	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC
1	1	1	0.746	0.756	0.808	0.808
1	5	1	0.818	0.792	0.852	0.868
1	10	1	0.850	0.819	0.871	0.851
1	**50**	**1**	**0.838**	**0.864**	**0.872**	**0.873**
5	1	1	0.733	0.722	0.732	0.747
10	1	1	0.805	0.841	0.846	0.806
50	1	1	0.741	0.759	0.809	0.798
1	1	5	0.807	0.763	0.789	0.812
1	1	10	0.852	0.819	0.792	0.803
1	1	50	0.839	0.851	0.886	0.856

Table 5.14: Table with the AUC performance on different weights of the triple loss.

Table 5.14 illustrates that the combination (1,1,50) performed better on the patches w.r.t. the default (1,50,1), but a higher performance on the whole images is to be preferred. The good performance of a more latent space-focused configuration was to be expected, since GANomaly is more oriented toward a correct understanding of the sample distribution, but nonetheless, it fits better this problem with a focus on reconstruction.

5.3.4 Layer depth

The tests were performed using the default configuration and the dataset from split 2, with a time of training illustrated in Table 5.13.

ngf/ndf	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Testing Images AUC-ROC	Training Time (h)
2	0.776	0.728	0.786	0.709	1
8	0.664	0.634	0.669	0.678	2
16	0.841	0.839	0.851	0.835	3
32	0.855	0.832	0.812	0.816	4
64	**0.838**	**0.864**	**0.872**	**0.873**	**4**
96	0.756	0.817	0.719	0.716	6

Table 5.15: Table with the AUC performance on different layer depths.

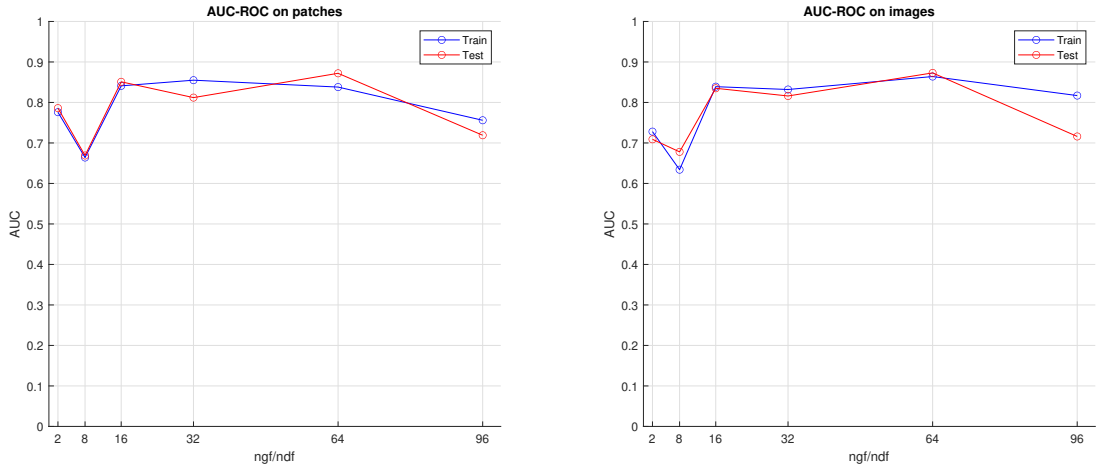


Figure 5.5: Plots of the AUC of different layer depths.

The default configuration is the best performing, even if in Table 5.15 and Figure 5.5 is shown that a depth=16 is slightly worse but trains in less time. A

higher depth value, as seen in both GANomaly and Skip-GANomaly, does not lead to a better performance, as the layer can memorize more features and more likely end up overfitting.

5.4 Results summary

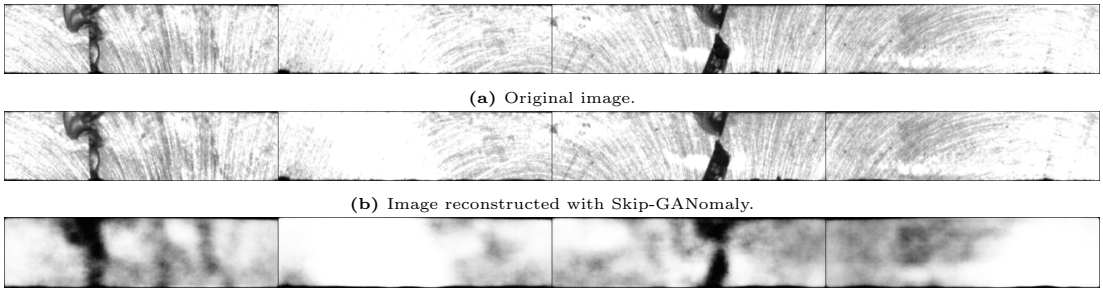
In this section will be discussed the most meaningful results and the inference time of both GANomaly and Skip-GANomaly. To sum up the results obtained, the most important tests were the ones performed on the splittings, respectively in Table 5.2 and 5.10, since they were responsible for balancing patches AUC and whole image AUC. The most effective preprocessing pipeline is the one used for most of the tests, composed of cropping, unwarping and black parts removal, while using only the cropped image or filtering did not lead to a good performance. In Table 5.14 can be seen the best-performing models for GANomaly and Skip-GANomaly, together with the accuracy tested using the algorithm described in Section 4.4. GANomaly managed to reach an AUC of 0.873 in the images on the test set, outperforming Skip-GANomaly by respectively a 0.08 and 0.12 value in the patches and whole images.
It also illustrates the inference time tested with the separate pipeline that takes in input the whole image, preprocesses it and classifies it. The results in this field are not reassuring, since both networks employ far more than one second to process an image. In order to lighten the preprocessing part, it was briefly tested a compression of the image before applying the Hough Transform, since it is the computationally heaviest part of the preprocessing, but the inference time decreased by 1-2 seconds in the most aggressive compressions.

Model	Splits	Training Patches AUC-ROC	Training Images AUC-ROC	Test Patches AUC-ROC	Test Images AUC-ROC	Accuracy (%)	Inference Time (s)
Skip-GANomaly	4	0.735	0.648	0.793	0.757	75.3	6.2
GANomaly	2	0.838	0.864	0.872	0.873	81.0	7.0

Table 5.16: Table with the best performing models.

Regarding the difference in performance between GANomaly and Skip-GANomaly, it can be seen in Figure 5.6 how each network reconstructs an example anoma-

lous sample. Even though the model has not been trained on anomalous samples, Skip-GANomaly reconstruction is almost identical to the original sample, reason why it had difficulties in distinguishing normal and anomalous samples, since it heavily relies on the difference between real and fake data. On the other hand, the reconstruction of GANomaly is blurred and lacks details especially in the cracks, meaning that the network could not efficiently reconstruct the anomalous input, thus leading to a better classification.



(a) Original image.

(b) Image reconstructed with Skip-GANomaly.

(c) Image reconstructed with GANomaly.

Figure 5.6: Example of an anomalous image reconstructed by the two models.

As a final consideration, Figure 5.7 illustrates some of the most difficult anomalous samples to classify. Most of these disks have very little damage, and could not be correctly classified by the networks without heavily compromising the classification of normal samples (by excessively lowering the threshold). This is also due to the fact that a good number of compliant samples present scratches that take a big area of the disk, or are subject to a lower luminosity than the majority of the dataset, both cases that complicate the task of separating anomalous and compliant samples.

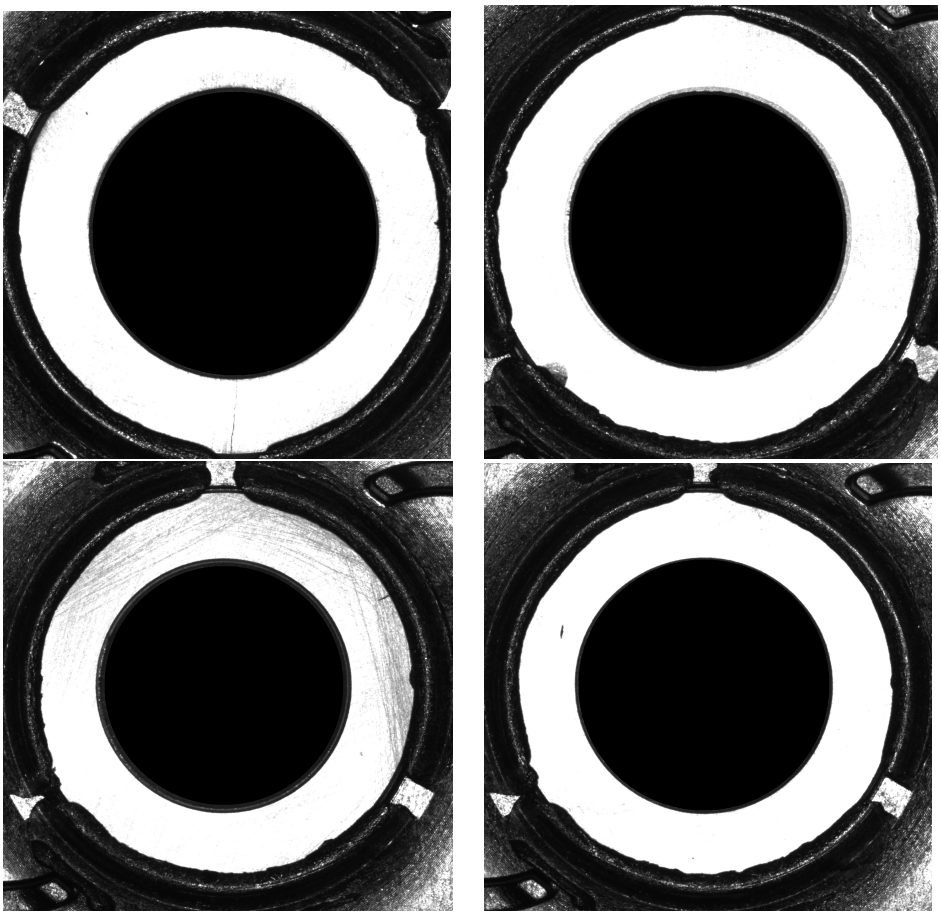


Figure 5.7: Four samples that were misclassified even by the best-performing models.

Chapter 6

Conclusions

In this thesis it was studied the different techniques that, in the latest years, were developed in the field of input reconstruction and generation applied to anomaly detection system. The problem to be faced was a real world production line of circular plastic covers that may present cracks and deformation on their surface, to be detected by the proposed solution. Starting from the basic knowledge of Autoencoders and GANs, the study moved on to the current state-of-the-art methods, until deciding to use GANomaly and Skip-GANomaly, due to their ability to reconstruct input and learn the normal sample distribution via latent space. The images of the plastic parts were processed by different kinds of preprocessings and data augmentation techniques, to better fit the network. The chosen preprocessing optimizes each image by using only the parts of the plastic disks, by transforming it from a circular shape to a rectangular shape and dividing the rectangle in patches. The various tests performed were aimed to find the most effective and balanced number of patches, weights of the particular loss of these networks and other tunings of various parameters.
The overall results show that the best configuration was obtained using GANomaly, with an AUC on the test images of 0.873 and accuracy of 81%. Therefore, the study provided that the use of generative networks and careful data augmentation can lead to good results even in the anomaly detection field applied to real world pipelines, where the class of anomalies is not well defined and thus difficult to separate from normal data.
The proposed solution is however open to improvements and more extensive testing: an interesting tool to be added to the network would be the addition of bounding boxes where the model identifies an anomaly, to better locate cracks and understand how the network works. Another important point is highlighting

that GANomaly, still being less complex than Skip-GANomaly, performed better, meaning that a simpler network could potentially fit better the problem. Therefore are suggested tests on the other less complex state-of-the-art methods, which may correctly classify even the most difficult samples previously illustrated in Figure 5.2. On the other hand, if in the future the dataset will have substantially more samples, the problem could be understood better by exploiting the attention mechanisms of the Transformers, discussed in Section 2.4.2.

Bibliography

[1] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek and Vijay Ramaseshan Chandrasekhar, *Efficient GAN-Based Anomaly Detection*, 2019.

[2] Samet Akcay, Amir Atapour-Abarghouei and Toby P. Breckon, *GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training*, 2018.

[3] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth and Georg Langs, *Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery*, 2017.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio, *Generative Adversarial Networks*, 2014.

[5] Samet Akçay, Amir Atapour-Abarghouei and Toby P. Breckon, *Skip-GANomaly: Skip Connected and Adversarially Trained Encoder-Decoder Anomaly Detection*, 2019.

[6] Zhang, W., Ouyang, L., Zhou, C. and Li L., *Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications*, 2019.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, 1996.

[8] Jianfeng Ma, Yuzhe Tang and Qin Liu, *Anomaly Detection in the Cloud: Challenges and Opportunities*, 2016.

[9] Keogh, E., Lincoln, P. and Wei L., *Real-time Anomaly Detection in Streaming Data*, 2008.

[10] Michael Mathioudakis and George Katsikas, *Online Anomaly Detection in Time-Series Data via EMA-CUSUM Control Charts*, 2010.

[11] Kai-Wen Cheng, Yie-Tarng Chen and Wen-Hsien Fang, *Video anomaly detection and localization using hierarchical feature representation and Gaussian process regression*, 2015.

[12] Mahmoud Afifi and Marco F. Duarte, *UCSD Anomaly Detection Dataset and Benchmark*, 2019.

[13] Ahmed Selim, Mohamed M. Abdelsamea and Mohamed S. M. S. El-Rabaie, *Semantic Segmentation-Based Anomaly Detection in Surveillance Videos*, 2019.

[14] Paul Viola and Michael Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, 2001.

[15] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou, *Isolation Forest*, 2008.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin, *Attention is All You Need*, 2017.

[17] https://towardsdatascience.com/transformers-89034557de14

[18] Md Amran Siddiqui, Jack W. Stokes, Christian Seifert, Evan Argyle, Robert McCann, Joshua Neil and Justin Carroll, *Detecting Cyber Attacks using Anomaly Detection with Explanations and Expert Feedback*, 2019.

[19] Waleed Hilal, S. Andrew Gadsden and John Yawney, *Financial Fraud: A Review of Anomaly Detection Techniques and Recent Advances*, 2021.

[20] Shikha Agrawal and Jitendra Agrawal, *Survey on Anomaly Detection using Data Mining Techniques*, 2015.

[21] Justus Zipfel, Felix Verworner, Marco Fischer, Uwe Wieland , Mathias Kraus and Patrick Zschech, *Anomaly detection for industrial quality assurance: A comparative evaluation of unsupervised deep learning models*, 2022.

[22] Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan and Joseph Walsh. *Deep Learning vs Traditional Computer Vision*. 2019.

[23] https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

[24] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[25] https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021

[26] Keiron O'Shea and Ryan Nash, *An Introduction to Convolutional Neural Networks*, 2015.

[27] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J. and Williamson R. C., *One-Class SVMs for Document Classification*, 2000.

[28] I. Ullah, S. Lee and N. Ahmad, *A Novel Anomaly Detection Scheme Based on k-means Clustering Algorithm*, 2016.

[29] G. Gao, M. Li, A. W.-C. Liew, M. Wu, and Z. Zhao, *Density-Based Clustering over an Evolving Data Stream with Noise*, 2015.

[30] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P. and Shroff G., *LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection*, 2016.

[31] Dor Bank, Noam Koenigstein and Raja Giryes, *Autoencoders*, 2020.

[32] Malhotra Pankaj et al., *Anomaly Detection in Host Logs Using Bidirectional Recurrent Neural Networks*, 2015.

[33] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow and Brendan Frey, *Adversarial Autoencoders*, 2016.