

## MASTER

### Visual anomaly detection using deep learning

Komar, T.L.

*Award date:*  
2018

[Link to publication](#)

#### **Disclaimer**

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

#### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

# Visual anomaly detection using deep learning

*Master Thesis*

Tomas Lukas Komar

Public Version

Supervisors:

Joaquin Vanschoren

Niels ten Dijke

Examination committee:

Joaquin Vanschoren

Niels ten Dijke

Nikolay Yakovets

Eindhoven, October 2018

# Abstract

Anomaly detection in the industrial sector is an important problem as it is a key component of quality control systems that minimize the chance to miss a defective product. Most often, the anomaly detection is done through analysis of the images of the products. Because the products, or their designs change and quality data is hard to obtain, this problem is approached in an unsupervised manner.

There are many different anomaly detection approaches, but most of them deal with low dimensional data and do not work well with the images. Other approaches, that can perform well with images, require intensive parametrization and lack robustness. They fail when ambient brightness changes or camera gets slightly moved, thus resulting in changes in the images and outcome of the model. Also, it is important for the model that it is able to point out where the defect in the image is. In the past decade, deep learning techniques have been proven to work very well with high-dimensional data like images. We examine deep learning techniques that utilize convolutional neural networks which can extract meaningful image representations to a lower-dimensional space. It allows the models to learn the important features of an image, regardless of some small changes in the input. In the study, we propose several approaches to the anomaly detection problem. Firstly, we look at the supervised approach with convolutional neural network classifier. Then we focus on the unsupervised approaches. At first, we extract meaningful image representations with several feature extractors - not-neural network based KAZE algorithm as a baseline, several pre-trained convolutional neural networks and an encoder, taken from the autoencoder trained on the images of products without defects. Then we use these extracted features to train standard anomaly detection algorithm - one class support vector machine to detect anomalous image features. The next approach is to use the trained autoencoder itself for the anomaly detection by measuring the reconstruction error of an encoded-decoded image, that should be higher for the anomalous images. The last approach is an anomaly detection with the generative adversarial networks. The network learns a mapping from the latent space to a representation of a healthy data and is able to produce new and unseen data samples from random latent vectors. Then, given a testing image, we try to find a latent variable  $z$  from which a network can generate the most similar image to the one being tested. When such  $z$  is found and generated image differs a lot from the test image - the image is labelled as an anomaly.

The findings show that using these techniques we can build robust anomaly detection models. However, different approaches have also different drawbacks. The supervised approach requires lots of labelled training data. One class support vector machines perform poorly with linear kernel and good with RBF, but we did not succeed in finding a way to examine which features of an image exactly contribute to the classification, hence there is no defect localization. The autoencoder and the generative adversarial networks show good, promising results and provide an easy way to do the defect localization. The autoencoder approach has been proven to be fast and reliable, however only for the bigger anomalies. We also argue that training generative adversarial networks is a very hard and unstable process and even though they do work well for anomaly detection problem, they might be suited better for a different data sets, that have more deviation of features among the images.

# Preface

The past two years have provided me with many great experiences, new knowledge and a strong academic ground for my future professional career. This Master thesis marks the last step in my Master's project of the international EIT program Data Science. The first year spent in the Universidad Politecnica de Madrid in Spain has laid out the basics of Data Science to me in various fields. The second year I have spent in the Eindhoven University of Technology, where I have continued my studies and gained deeper knowledge in the domain. In the second semester, I have started my internship project in the company called "Prodrive Technologies" that has laid the groundwork for this thesis. The internship had a duration of 5 months and the sixth month has been dedicated to writing the thesis. It has been a great opportunity to work with the newest technologies and research an interesting and hot topic of deep learning.

The research question has been formulated by me together with Niels Ten Dijke, who was my internship supervisor in the company. I would like to thank Niels for helping me to define a project, problem and come up with possible solutions. During his supervision, Niels has provided me with guidance that helped me to navigate among and search for various research papers. Moreover, he has shared his insights as a mentor and taught me how to connect academic research to the business objectives.

Also, I want to express my gratitude to my supervisor in the university Joaquin Vanschoren. Joaquin has provided me with valuable feedback, especially from an academic perspective - how to perform and write a scientific work. Weekly meetings we were having with Joaquin and other Master students were beneficial and interesting, and I have learned many new things.

In addition, I want to show appreciation for my friends and colleagues from the university and the company, that have helped me in various ways during this project. Without their support, this journey would have been more difficult.

# Contents

<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Problem definition</b>	<b>3</b>
<b>3 Data definition</b>	<b>5</b>
3.1 Raw data . . . . .	5
3.2 Datasets . . . . .	5
3.3 Artificial anomalies . . . . .	7
3.4 Data augmentation . . . . .	8
3.5 Data preprocessing . . . . .	9
<b>4 Theory background</b>	<b>10</b>
4.1 Transfer learning . . . . .	10
4.2 Feature extractors . . . . .	11
4.2.1 Autoencoders . . . . .	11
4.2.2 KAZE algorithm . . . . .	12
4.3 Support vector machines . . . . .	13
4.3.1 Kernel function . . . . .	14
4.3.2 One class support vector machines . . . . .	14
4.4 Batch normalization . . . . .	15
4.5 Generative adversarial networks . . . . .	15
4.5.1 Deep convolutional generative adversarial networks . . . . .	16
4.5.2 Evaluation of generative models . . . . .	17
4.5.3 Instability of training GAN's . . . . .	17
4.5.4 Mode collapse . . . . .	18
4.5.5 WGAN . . . . .	18
4.5.6 WGAN-GP . . . . .	18
4.5.7 Checkerboard effect . . . . .	19
<b>5 Approaches</b>	<b>20</b>
5.1 Current approach . . . . .	20
5.2 Supervised convolutional neural network approach . . . . .	20
5.3 Unsupervised learning . . . . .	21
5.3.1 Anomaly detection with autoencoder . . . . .	21
5.3.2 Anomaly detection with OCSVM . . . . .	22
5.3.3 Anomaly detection with GAN's . . . . .	23
5.3.4 Architecture of GAN's . . . . .	24

<b>6</b>	<b>Results</b>	<b>28</b>
6.1	Supervised convolutional neural network approach . . . . .	28
6.1.1	Transfer learning . . . . .	29
6.1.2	Class activation maps . . . . .	30
6.2	Unsupervised approaches . . . . .	30
6.2.1	Feature extraction and OCSVM . . . . .	30
6.2.2	Hyperparameters of OCSVM . . . . .	31
6.2.3	Dataset B1 . . . . .	32
6.2.4	Dataset B4 . . . . .	34
6.2.5	Dataset C1 . . . . .	35
6.2.6	Autoencoder . . . . .	37
6.2.7	Generative adversarial networks . . . . .	38
<b>7</b>	<b>Discussion</b>	<b>42</b>
7.1	Results . . . . .	42
7.1.1	Supervised classification . . . . .	42
7.1.2	Feature extraction and OCSVM . . . . .	42
7.1.3	Autoencoder . . . . .	43
7.1.4	Generative adversarial network . . . . .	43
7.2	Conclusions . . . . .	44
<b>8</b>	<b>Future work</b>	<b>45</b>
8.1	Improvements of researched approaches . . . . .	45
8.2	Other approaches . . . . .	45
	<b>Bibliography</b>	<b>47</b>

# List of Figures

3.1	Examples of good (a) and defective (b) images . . . . .	5
3.2	Examples of artificially created anomalous images . . . . .	7
3.3	Examples of augmented images . . . . .	9
4.1	Transfer learning . . . . .	11
4.2	Abstract architecture of autoencoder's . . . . .	12
4.3	Scale-space representation levels . . . . .	13
4.4	Mapping data to a higher dimension with the SVM's kernel . . . . .	14
4.5	One class support vector machine . . . . .	14
4.6	The overview of the generative adversarial networks . . . . .	16
4.7	Figure illustrating checkerboard effect [27] . . . . .	19
5.1	Simple convolutional neural network . . . . .	21
5.2	Autoencoder . . . . .	22
5.3	Training and testing flow of the OCSVM with the extracted features . . . . .	23
5.4	Architecture of DCGAN . . . . .	24
5.5	Architecture of WGAN . . . . .	24
5.6	Overview of the training and testing approach . . . . .	27
6.1	Class activation maps . . . . .	30
6.2	Examples of B1 dataset classified images . . . . .	33
6.3	Examples of B4 dataset classified images . . . . .	35
6.4	Autoencoder's anomaly detection confusion matrix . . . . .	37
6.5	Autoencoder's loss distribution . . . . .	37
6.6	Autoencoder's training loss history . . . . .	38
6.7	Autoencoder's input, output and defect localization, respectively . . . . .	38
6.8	WGAN's generator and discriminator losses . . . . .	39
6.9	Examples of $G(z)$ (generated) images during training . . . . .	39
6.10	WGAN's anomaly detection confusion matrices . . . . .	40
6.11	WGAN's anomaly detection loss distribution . . . . .	40
6.12	WGAN (from left to right): Test image, Generated image $G(z)$ , defect localization . . . . .	41

# List of Tables

3.1	Dataset definitions . . . . .	6
3.2	Augmentation parameters . . . . .	8
6.1	Supervised classifier results . . . . .	28
6.2	Extracted features . . . . .	31
6.3	B1 dataset results with RBF kernel . . . . .	32
6.4	B1 dataset results with LINEAR kernel . . . . .	33
6.5	B4 dataset results with RBF kernel . . . . .	34
6.6	B4 dataset results with LINEAR kernel . . . . .	35
6.7	C1 dataset results with RBF kernel . . . . .	36
6.8	C1 dataset results with LINEAR kernel . . . . .	36
6.9	Autoencoder anomaly detection results . . . . .	37



# Chapter 1

## Introduction

Developments in the computing machines and the information sharing infrastructure (aka the internet), together with various algorithms in the past 70 years have paved the way for a new digital revolution. It has affected almost every aspect of human lives, but also in multiple industries - healthcare, manufacturing, transportation, etc. Recent developments in a computational power and new methods have started new trends in manufacturing, what some people call “Industrial revolution 4.0” [19]. It is being characterized as having a trend in smart automation [6], intensive data sharing not only among people, but also between machines themselves (Internet of Things) and cloud computing.

One of the tasks that is very important manufacturing sector is quality control. It is a process in which companies that manufacture goods check whether the produced product does match all the quality requirements. Traditionally, it has been done by human professionals who would do the visual inspection of a physical goods manually, simply by looking at it. However, humans tend to get tired or distracted and make mistakes, which can lead to a waste of scrapped materials, as well as lost production time [21]. It is a very important process in any manufacturing company, also because the cost of selling a defective product is high and leads to many consequences like warranty claims, customer disappointment and the loss of sales [26]. But employing knowledgeable professionals that can perform quality control for every product manually is very resource exhausting.

The popularity of automating quality control is on the rise [6], as such automation allows this task to be performed more efficiently and with fewer costs. It reduces the costs of quality control as less or no inspection employees are needed, while the performance of automated models often is comparable or even better. Automated visual inspection models do not suffer from human-related limitations like tiredness and can be employed to work without any breaks. This inspection is most often done by inspecting the images of a produced product sample which can be good (“healthy”) or bad (“anomalous”).

Finding the defective samples in images among the pool of “healthy” images translates to the problem of finding unusual or *anomalous* data patterns that do not conform to the expected behaviour is defined as the anomaly detection. Anomaly detection does find itself among a very broad range of domains like fraud detection, insurance, healthcare, cyber security, etc. [8], [18]. Visual inspection of a product in the quality control system is essentially an anomaly detection, because we want to detect any product that has any deviation from the “healthy” products, while defective products are also much rarer than the good ones.

In the context of this project, the quality control is performed on the products that have had a sealant (a plastic / gum “lines” put, so another part of the product could be placed and stick to it). As no other information is available, the defective samples have to be detected solely from image data.

Prodrive Technologies is a company that manufactures various goods and is aiming to become a fully-automated manufacturer. However, the current approach used in Prodrive’s production environment lacks robustness and requires a lot of parametrization. It means that the right parameters have to be found (for example - distance from the camera to the product) and often they have to be calibrated, if a camera, for example, shifts. If the parameters are not changed when the environment changes, the model fails. So a research into the new methods which are not that sensitive to changes in the images (as long as an important part, the sealant, can be seen) is desired, to make quality control more robust and automated. So, me together with Prodrive Technologies, have started this case study on a robust automated visual quality control system.

There is a long history of trying to find unusual or *anomalous* samples among the corpus of data. These approaches work well for the low-dimensional data like customer transactions, however, in the context of image data, they suffer from the curse of dimensionality (problems that occur working with high-dimensional data, because of the big increase in the dimensions and the volume of the space. Then data becomes sparse and more data and more training is needed to find a solution) and thus, are not suited for such a task [1].

So a more suitable approach to analyse images is needed. In recent years deep convolutional neural networks became a state of the art technique for classification and object detection with high dimensional data like images as they work very well with high-dimensional data. These networks trained on image data are able to learn good representative features at different levels, that can be extracted from images. For example, in a face recognition task, at lower levels an edges or curves are detected. Then these features are used in a higher layers to detect features like eyes or nose and subsequently passed into a higher layers, until a face can be detected.

These features that are extracted from the images can be used in different ways in the context of this problem. Firstly, we can take a bunch of images that would be labelled into one of two categories - pass or fail. Then we extract the deep features from these images and use them to train the model to classify between “pass” and “fail” classes (supervised approach, because we train on labelled data). Alternatively, we use classical anomaly detection algorithms like one class SVM 4.3 on these features to detect anomalous samples in an unsupervised way [28]. We also use more complex generative model approaches, like deep convolutional generative adversarial networks, that have been shown to perform well in high-dimensional anomaly detection problems [29], [31], [38].

In the subsequent chapters these approaches together with theoretical background are explained in more detail. Later, data definition and the experimental setup of this project is described. After proposed models have been implemented, tested and their results discussed in the Results chapter (6), the differences between approaches and how do models relate to the requirements are discussed in the Discussion chapter 7.

## Chapter 2

# Problem definition

The problem in the current approach for visual inspection used in Prodrive technologies manufacturing lacks robustness and requires extensive parametrization. It means that the model is not able to automatically adjust to changes in ambient lighting or camera shifts. Moreover, if some change in camera's position happens, an operator has to re-calibrate the parameters of the model, like a distance to the product from the camera or markup of an area where a sealant has to be put. Moreover, looking at the outputs of the model, one can easily detect many false positives or false negatives, so it also has fairly low performance in general.

The requirements that are desired for the quality control model are:

1. The model does not require a large amount of labelled data
2. The model can show where the suspected defect is in the image (defect localization)
3. The model is robust with respect to changes in lighting, slight rotations, zoom in / out or horizontal / vertical shifts (because the camera can slightly move in any direction)

In the case of this project, we assume a large amount is more than 50 original train images (as we do augment these images later).

Standard anomaly detection techniques work well with the low-dimensional data but are not suited for the high-dimensional data like images. Deep convolutional neural networks are state-of-the-art models that can reduce this dimensionality by transforming an image into meaningful feature representations. Researching how to use such representations to build a robust visual anomaly detection model is a logical direction that has been followed in this project.

So the research question in this project is:

**Can a deep learning based framework be used to build a robust visual anomaly detection model with a limited amount of training data?**

To address this research question we need to propose a model that utilizes deep learning techniques, implement it and test with the real data taken from the manufacturing. Then we evaluate it with respect to the real and artificial (more details in Chapter 3) anomalies. We evaluate it with the F1 score, which involves precision and recall metrics, and the requirements listed above. Moreover, we define the limitations of each approach, like the amount of needed data, the time needed to train the model and any drawbacks that the approach has.

In this project, several models have been proposed, implemented and tested. Firstly, a supervised approach has been tried with transfer learning. Transfer learning has been used, because there was not enough data to properly train such a neural network from scratch. In this approach, a

model learns to classify between "pass" ("healthy") and "fail" ("anomalous") classes. Then, two unsupervised approaches (more details in section 5.3) were proposed. First is to use traditional machine learning classification algorithm support vector machine (explained in the section 4.3) in an unsupervised one-class manner. As explained in section 4.3.2, we transform a multi-class classification problem into a one-class novelty detection problem. This algorithm was used on the features extracted from the images. These features have much fewer dimensions compared to a full image where an image has  $n \times m$  dimensions, where  $n$  = width and  $m$  = height of the image, so the problem becomes lower-dimensional. These features were extracted with pre-trained convolutional neural networks that are publicly available or autoencoders that we train ourselves, but also using a simple feature extraction algorithm KAZE, to demonstrate differences in the performance of the algorithm between features extracted with the KAZE algorithm and those features that are extracted using deep learning methods. Another approach proposed uses autoencoder (section 5.3.1) trained on the positive samples. It learns to encode and decode "healthy" images. The model trained in such a way can reconstruct "healthy" images with small reconstruction error, whereas this error for anomalous images is higher. The last model proposed uses generative adversarial networks that learn to generate new, unseen, but "healthy" samples. Then this trained network is used to build an anomaly detection model. These approaches are explained more in detail in the Approaches chapter (5).

In the next chapter (3) we will define the data with which we have done the experiments and the type of augmentations of the data that have been performed. Then we describe the relevant theoretical background in the chapter 4 and the approaches in the chapter 5 which show how do we try to solve the problem of this project. After that, we present the results of each approach in the chapter 6. In the following chapter (7), a discussion and a comparison of these results is presented. Finally, in the last chapter (8) we write the future work that can be done to further research the problem.

## Chapter 3

# Data definition

### 3.1 Raw data

Datasets consist of an images of a product that had a sealant put by a robot in the manufacturing process. The images are mostly big and differ in sizes among clusters (for example 864x628, 928x1320, 5120x3840 pixels in width and height). During the experiments, they have been downsized to a more computationally friendly size - 512x512 px (or less, for faster evaluation). The results reported are for the experiments conducted with the images of size 512x512 px.

The images have been made by the cameras placed in the manufacturing facility at a different assembly lines.

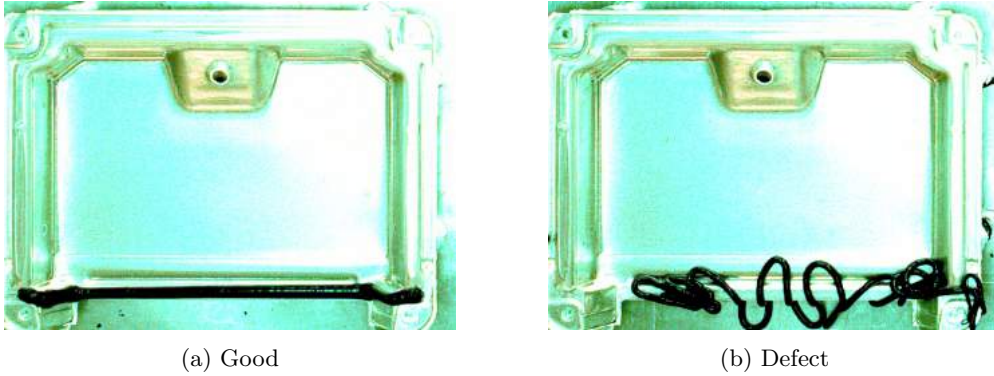


Figure 3.1: Examples of good (a) and defective (b) images

### 3.2 Datasets

In total there were 9 datasets that can be clustered into 3 clusters - A, B, and C. The main difference between the datasets is that each of them consists of images of a different products.

Datasets in the cluster A differ from datasets in the clusters B and C in a way that cluster A datasets have been collected from the outputs of the current visual inspection models. It means that there are a lot of labelled images, however, in many cases these labels are incorrect. This has been observed by doing an empirical analysis of the datasets by looking at the labels and the images, where many cases of incorrect labels could be easily identified.

Cluster B datasets (except the dataset labelled as B1 - see table 3.1) are correctly labelled by a professional, but do not have many images (B1 has a bit more than others in cluster B).



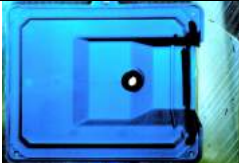




Dataset	Training	Testing	Example image
A1	Pass: 1609; Fail: 629	Pass: 400; Fail: 196	
A2	Pass: 3760; Fail: 16312	Pass: 930; Fail: 4089	
B1	Pass: 501; Fail: 0	Pass: 43; Fail: 93	
B2	Pass: 72; Fail: 0	Pass: 7; Fail: 0	
B3-1	Pass: 54; Fail: 0	Pass: 3; Fail: 3	
B3-2	Pass: 54; Fail: 0	Pass: 3; Fail: 3	
B4	Pass: 75; Fail: 0	Pass: 18; Fail: 58	
C1	Pass: 61; Fail: 51	Pass: 13; Fail: 70	[No-image]
C2	Pass: 26; Fail: 48	Pass: 5; Fail: 9	[No-image]

Table 3.1: Dataset definitions

It is important to note, that the labelling in the dataset B1 has been done by a person who is not a professional, so the dataset is a bit noisy (in the sense that some manual labelling of images with minuscule or ambiguous defects is often not done correctly). The defects in the images of this dataset are often very minor and hardly distinguishable even for a (non-professional) human. Cluster C consists of two datasets that are from a different manufacturing machine and has a low number of images (but correctly labelled).

### 3.3 Artificial anomalies

For the datasets A1 and B4, there was a lack of anomalous images among the data, so some defective sample images were created. These anomalous samples have defects that have been introduced artificially, by taking a good (non-anomalous) image and in-painting some random black lines or making the sealant wider / shorter, thicker / thinner. This also allows to test the robustness of the different models to a different types of defects, ones that are not present in the current datasets of the products, but still might occur in the future. Creation of artificial anomalies in the healthy images has been done with a simple image editing tool “Paint”. Example of such images can be found below in the Figure 3.2 with the artificial defect place circled in red:

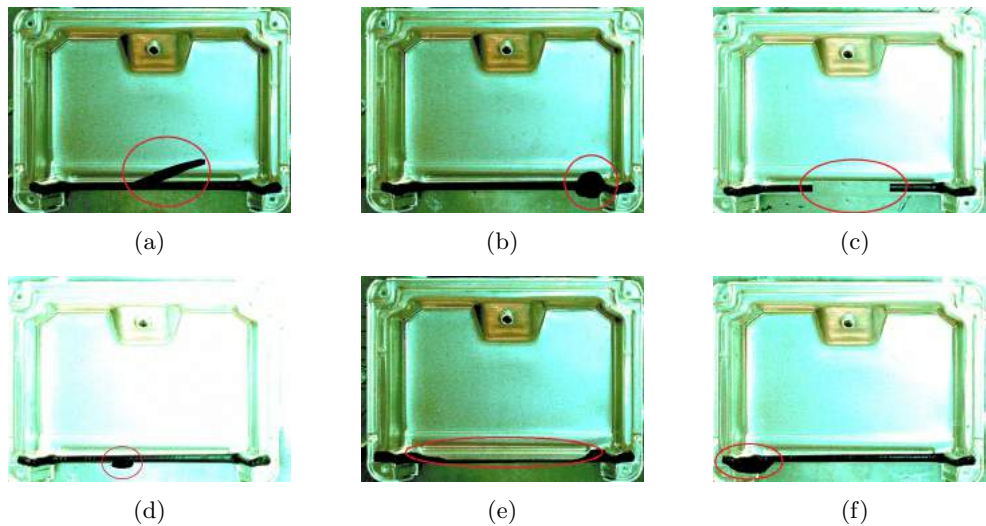


Figure 3.2: Examples of artificially created anomalous images

As we see, expected defects involve some path discrepancies while the sealant is being put (example a), having too much sealant put at some place (examples b, d, f), or not enough - having some skips of the sealant or having too thin sealant (examples c, e). This has been done for datasets A1 and B4 (as these two datasets are inter-related, the only difference that A1 is big and unlabelled while B4 is smaller and labelled by a professional).

Dataset B4 has been the main dataset that was used in the most of the experiments, while other datasets have been used after already achieving a concrete approach and good results with B4 dataset. Mainly to check how well that approach and / or parameters work on another dataset with real anomalies (not artificially created and labelled by a professional). This choice has been based on the fact that this dataset has been properly labelled (we have several hundred “healthy” samples and we are sure that they are “healthy”). Moreover, the dataset contains images of the same product but with different brightness and positional shift changes. That is why this dataset can be considered representative as it easily can be tested for robustness using the real data.

### 3.4 Data augmentation

During all of the experiments, the augmented data has been used. By using data augmentation, we can greatly increase the size of our training samples covering more and different situations. We make our models more robust by creating much more training data with different parameters and use it to train our models. Some augmentation, like brightness change or moving horizontally or vertically is very important in our problem setting, as we want the model to be robust to the lighting and movement (of a camera) changes. In this way, we generate brighter / darker samples or samples that are shifted / rotated, etc., so our models also learn much wider healthy data representation in the unsupervised approaches.

The concrete values and ranges that should be used for augmentation should be decided for each dataset separately and depend on the images themselves. In general, we would like to have higher values, but not too high, so that, for example, in all horizontally or vertically shifted images we would still have 100% of the sealant seen. Otherwise, the model might learn that partially put sealant is good.

Augmentation itself is done in this way: for each image  $x \in X_{train}$ , we randomly select one of the augmentation parameters (independently - meaning that no parameter or all parameters can be selected for each augmentation) and randomly select its value and apply the transformation for an image  $x$ . We repeat this process  $n$  times. After the augmentation, the dataset size becomes  $n \times len(X_{train})$ , where  $n$  is a number of augmentation iterations and  $len(X_{train})$  is a number of training images in the training dataset  $X_{train}$ .

Parameters that have been used in this project setting for augmenting images are shown in the table 3.2 below:

Action	Values	Explanation
Rotation	5°	Max. rotation in deg (degrees)
Width shift	5%	Max. percentage of width to shift horizontally
Height shift	5%	Max. percentage of height to shift vertically
Zooming	5%	Max. percentage to zoom in or out the image
Brightness	[0.5, 2]	Range of values to apply brightness change. Value 1 is baseline (current brightness). Values $v > 1$ make image brighter, while lower values $v < 1$ reduce brightness (makes them darker).

Table 3.2: Augmentation parameters

As we can see from the example augmented images in the Figure 3.3, they are slightly rotated and shifted. The empty space that appears if we rotate or shift an image is filled by taking the nearest "true" image's pixel value and using that for the new pixels.

The differences in different augmented images are illustrated in (b) and (c) of Figure 3.3. Here images are rotated in a different directions and have a different brightness setting set. So by having only the darker images, we produce many brighter ones (and vice versa) to increase the robustness of the models.



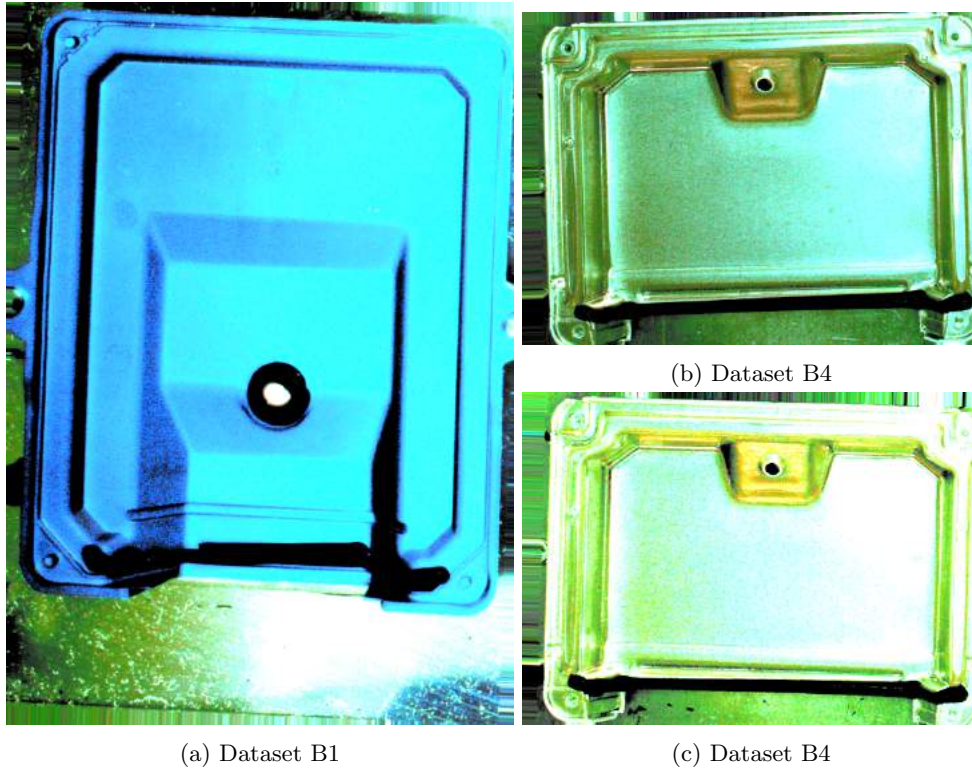


Figure 3.3: Examples of augmented images

### 3.5 Data preprocessing

Training machine learning models sometimes take a lot of time and computational resources, especially when training on high-dimensional data like images. To support faster and easier training some data augmentation has been done prior to the training. Then, the data before being used in the models is rescaled. For algorithms it is harder to constantly “work with” colour values that can take value in the range of 0 to 255, than with smaller values, for example, in the range -1 to 1.

The pixel values have been rescaled using a custom function to either the range (0, 1) or (-1, 1), depending on the loss function used in the model.

Pre-trained models that have been used, do provide their own preprocessing function that actually rescales all the values to the range (-1, 1).

## Chapter 4

# Theory background

### 4.1 Transfer learning

The idea behind transfer learning is that we can train a neural network model on some task and then, later reuse the learned weights of the model (or parts of it) for a related, but different task. In the object recognition tasks, these models learn to detect objects, but the classification and / or detection is done on the features extracted through convolutional layers. These features are general, especially on the lower level convolutional layers. Training these layers require a lot of data and the pre-trained models that are available are actually trained on big datasets. The learned weights of the network convolutional layers that can extract good features from an image can be reused in other tasks, as many (especially lower level) features like edges and curves are rather general among many different image tasks. This has been explored and discussed by Yosinski et. al. [37]. The whole process of the transfer learning is illustrated in the Figure 4.1. Common practice in the transfer learning is to freeze some transferred layers. That means that the layer weights are not changed anymore during the training process, in a way that we do not allow back-propagation on these layers. This is most effective when the features learned are very similar and relevant across domains. Often transferred layers are frozen to speed up the training process as it reduces the number, of computations to compute new weights. Usually, convolutional layers would be frozen, to train only the fully-connected layers that have been attached on top of the pre-trained model. However, in many situations, relevant features are only partly related across different problems and are more similar in the first convolutional layers than the latter ones. Sometimes no layers at all are frozen, in such a way allowing the network to fine-tune the already pre-loaded weights to adapt more to the new problem / domain.

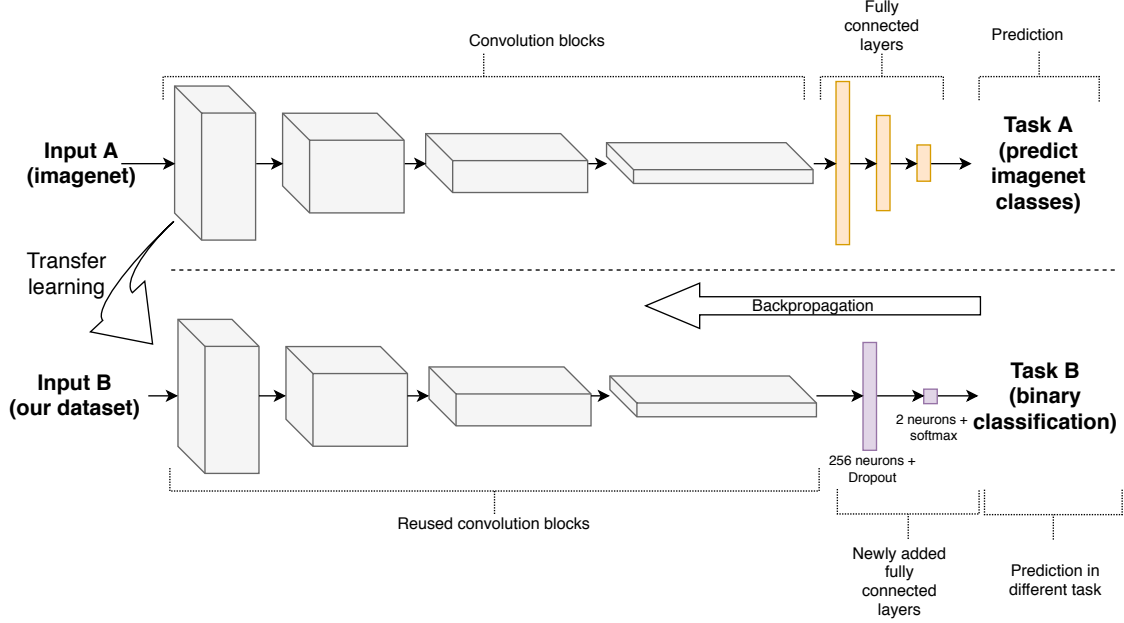


Figure 4.1: Transfer learning - Upper part shows a possible architecture of a neural network that is trained on Task A with input A. The part below in the figure (separated by a dashed line), shows the architecture of the neural network used for our task B. Note that the convolution blocks are reused, while the other layers are changed.

## 4.2 Feature extractors

A feature of a data point is defined as being a piece of information that is relevant for solving some computational task for a certain application. A feature of an image can be a numerical structure that represents points, edges, objects or areas. The numerical values of a feature describe properties of an object that it describes. Here I present the theoretical background behind the models or algorithms that have been used to extract such features from the images to be used later for anomaly detection.

### 4.2.1 Autoencoders

Autoencoders are a type of neural networks that learn to encode data to efficient representations and then reconstruct the original data from such encoded representations. The goal is to learn the weights of such encode-decode pipeline (see Fig. 4.2) so that the reconstruction of the encoded image has a very small loss. Moreover, this training is done in an unsupervised manner. Typically autoencoders are used for dimensionality reduction (feature encoding or extraction), but recently have also been used in generative models [22].

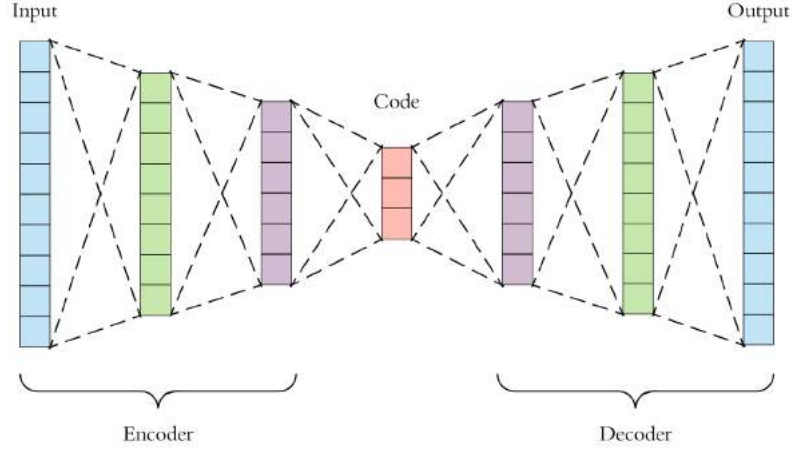


Figure 4.2: The autoencoder consists of the encoder and the decoder neural networks that are connected to each other. The encoder takes data as an input (for example an image) and learns to encode it to an efficient data representation. Then the decoder network learns to reconstruct the image from that data representation. *The image has been taken from [12]*

#### 4.2.2 KAZE algorithm

KAZE is a relatively fast (one of the most efficient among free image feature detectors as shown in comparative analysis in [33]), free and novel feature detection and extraction algorithm [5]. The algorithm works by finding points of interest in an image that are repeatable and then for each of those detected features it calculates a distinctive descriptive vector that can be matched and compared efficiently between the two images [3]. Novelty of a KAZE algorithm comes from the fact that both feature detector and descriptor work in a non-linear scale space as compared with other algorithms like SIFT or SURF that use linear (Gaussian) kernel function. Scale space  $L(x; y; t)$  is a signal representation to handle image structures at a different scales and is used in computer vision, image processing and signal processing. It consists of several images smoothed at a different ( $n$ ) scales  $t_{0..n}$  that are calculated from the same image. An image at the scale  $t_0 = 0$  looks like an original image. The next scale space representation at scale  $t = 1$  is a product of a kernel function  $k(x; y; t)$  and a convolution  $f(x; y)$ . Note that convolution is performed only on variables  $x, y$  and parameter  $t$  only defines the scale level. So the scale space becomes a collection of the same image transformed into a different scales. The higher the scale  $t$ , the more blurry and smoothed an image looks. (see Figure 4.3) It allows to construct image feature descriptors on a different object abstraction levels, hence making the whole collection of such descriptions more rich and informative. For example, looking at the image consisting of several trees, the algorithm would calculate many descriptors of the leaves for both trees. Then, at the higher scales, the leaves would get blurred (smoothed), so that the descriptors calculated at this scale would be of the trees themselves, rather than their leaves.

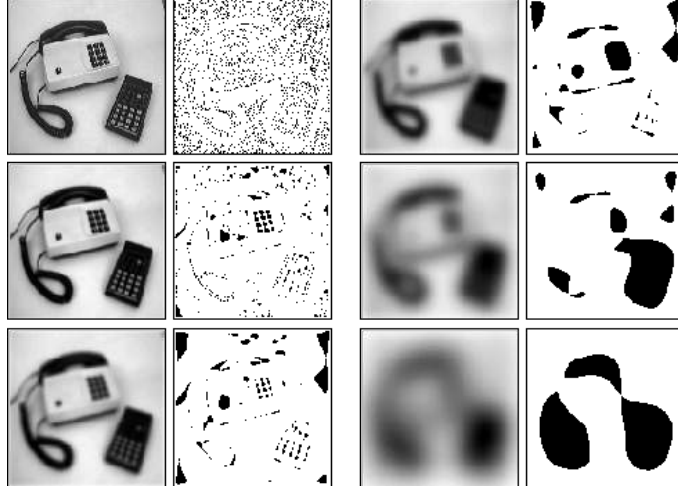


Figure 4.3: Scale-space representation at levels  $t = 0, 2, 8, 32, 128$  and  $512$  with gray blobs showing local minima. The image has been taken from [25]

### 4.3 Support vector machines

Support vector machines (SVM's) are a supervised learning models that are defined by an optimal hyperplane that separates data into different classes. For example, in a two dimensional space, this hyperplane would be a line separating a plane into the two parts, each for either of the two classes.

SVM's can create a decision boundary through a function  $\phi$  or a non-linear boundary with a non-linear function  $\phi_{nl}$  (kernel function) by projecting data to a feature space that can have linear relationships among the features. So when data can not be separated by a straight line, they are "lifted" from their original space (lower dimensional)  $I$  to a feature space "F" (higher dimensional), where there is a higher chance that such a straight hyperplane would exist, as shown in the Figure 4.4. This feature space  $F$  can be of an unlimited dimension and hyperplane separating points can be very complex, and that what gives SVM's such a great power. The hyperplane is defined by the formula  $\mathbf{w}^T \mathbf{x} + b = 0$ , where  $\mathbf{w} \in F$  and  $b \in R$ . Constructed hyperplane determines the *margin* between the classes, which has equal distance from the closest data point from both classes. That means, that the hyperplane maximizes the margin between the two classes.

The objective function for a SVM classifier is:

$$\min_{\mathbf{w}, b, \xi_i} \frac{\|\mathbf{w}\|^2}{2} + C \sum_{i=1}^n \xi_i \quad (4.1)$$

With constraints:

$$\begin{cases} y_i(\mathbf{w}^T \mathbf{x} + b) \geq 1 - \xi_i, \forall i = 1, \dots, n \\ \xi_i \geq 0, \forall i = 1, \dots, n \end{cases} \quad (4.2)$$

Here the parameter  $C$  determines the trade-off between maximising the margin and how many data points lie within that margin, while the error (slack) variable  $\xi_i$  (that can take any positive number) allows some data points to lie within that margin.

Solving the SVM's minimization problem (as defined in 4.1) using the Lagrange multipliers, the decision function becomes:

$$f(x) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(x, x_i) + b\right) \quad (4.3)$$

Where  $\alpha_i$  are Lagrange multipliers, that are weighted in the decision function;  $x$  is a data point;  $K(x, x_i)$  is a kernel function that is more described in the following section 4.3.1.

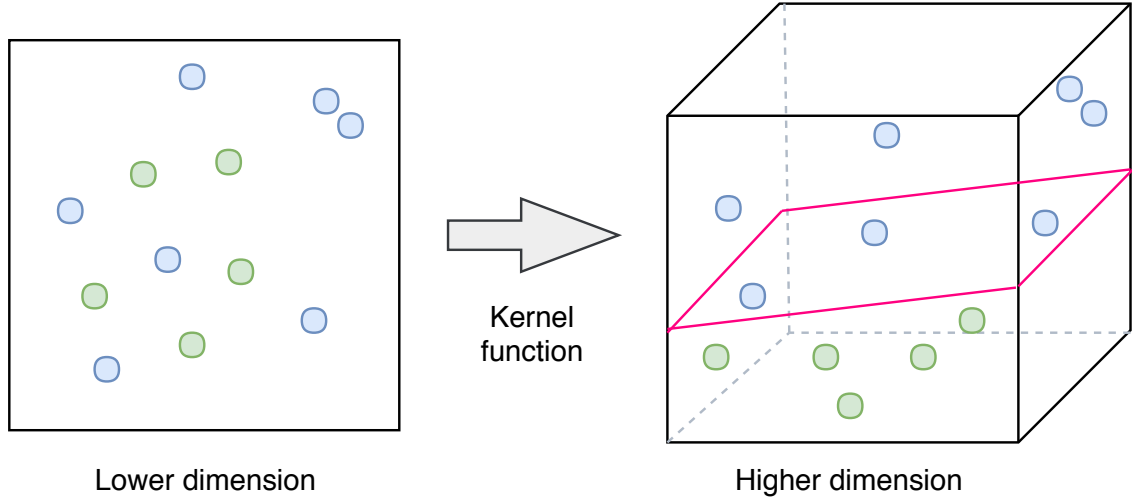


Figure 4.4: Mapping data to a higher dimension with the SVM's kernel

### 4.3.1 Kernel function

Popular choices for the kernel functions include linear, polynomial, sigmoidal or RBF (Radial Base Function). Gaussian RBF is the most popular kernel function used in SVM's and is defined by an equation:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \quad (4.4)$$

Where  $\sigma \in R$  is a parameter of the RBF kernel.

### 4.3.2 One class support vector machines

In 1999 Schölkopf et al. [32] has published a work where the method of transforming multi-class SVM to a one-class novelty detection model is proposed. It learns a binary decision function that captures all of the regions of the features in an input space, thus capturing and encompassing the density of the data. After projection into a feature space, a decision function is found (hyperplane) (see Fig. 4.5). Then the function returns 1 for data points that lie within a learned region (of training data) and -1 for data points lying outside the learned decision function.

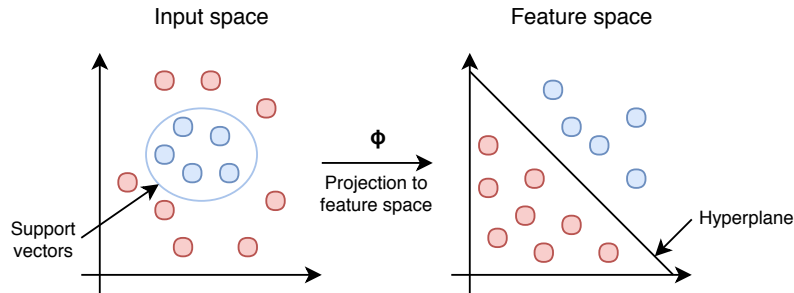


Figure 4.5: One class support vector machine

The minimization function is slightly different from the multi-class one:

$$\min_{w, \xi_i, \rho} \frac{1}{2} \|w\|^2 + \frac{1}{\nu n} \sum_{i=1}^n \xi_i - \rho \quad (4.5)$$

Previously, parameter  $C$  decided decision boundary smoothness, whereas here a parameter  $\nu$  performs similar job:

1. It sets an upper bound on the fraction of data points that lie outside decision function (errors)
2. It is a lower bound on the fraction of the support vectors (number of training examples used as a Support Vector)

This method is widely used as an anomaly detection algorithm in many contexts like fraud detection or defect diagnosis. It is an effective unsupervised method to detect anomalies (as it has been shown in [32] to outperform state of the art).

We focus on the OCSVM as it is considered to be a novelty detection algorithm, in comparison with Isolation Forest, local outlier factor, etc., that are outlier detection algorithms. The difference here is that in outlier detection algorithms, the training is done in a completely unsupervised manner, which means that we do expect some percentage of outliers to be in the training dataset.

## 4.4 Batch normalization

Batch normalization has been introduced in 2015 by S.Ioffe and C. Szegedy [20]. It is a technique to increase performance and stability of the artificial neural networks. It achieves that by taking the outputs of previous activation layer and normalizes it by subtracting the batch mean and dividing that by the batch standard deviation. Then the output of a batch normalization layer has zero mean/unit variance.

## 4.5 Generative adversarial networks

Goodfellow *et al.* in 2014 [16] have introduced a new generative unsupervised learning neural network. GAN's are specifically trained to learn a mapping  $G: z \rightarrow X$  to data distribution  $P_{real}$  on which they are trained, so that the network models and approximates the distribution  $P_{model}$  and is able to produce new data points that fall in the data distribution  $P_{model}$ .

Generative adversarial networks is a probabilistic generative model that consists of two neural networks, a **generator** and a **discriminator** which compete with each other in a min-max game. The discriminator takes a data point  $X \leftarrow X_{real} \vee X_{fake}$  as an input and predicts whether this data is real (coming from training data) or generated by a generator ( $G(z)$ ), where  $G(z)$  has a label 0 and  $X$  has a label 1. Generator  $G$ , in turn, takes random noise vector  $z$  from some predefined latent space  $z \subset Z$  as an input, generates a synthetic data point  $G(z)$  such that the discriminator would label it as coming from the real data distribution  $P_{real}$ . The high level overview is illustrated in the Figure 4.6. Because these two networks are connected, during the training both are improving in their tasks through back-propagation. The generator becomes better at producing synthetic data in a way that the discriminator gets fooled into predicting it as real, while the discriminator gets better at distinguishing real from synthetic data.

During the training, a generator and a discriminator play the min max game with a value function  $V(G, D)$ , which is an objective of the GAN, as formulated in [16]:

$$\underset{G}{Min} \underset{D}{Max} \{V(G, D) = \mathbb{E}_{x \sim p} [\log(D(x))] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(x)))]\} \quad (4.6)$$

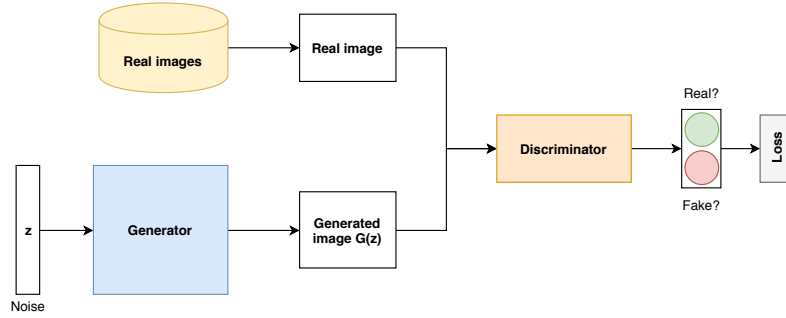


Figure 4.6: The overview of the generative adversarial networks

### 4.5.1 Deep convolutional generative adversarial networks

There are many different applications for such networks, but the most popular one is to train generative adversarial networks to produce images. In 2015 [29] DCGAN has been introduced, which is a generative adversarial network that utilizes deep convolutional layer architectures as in CNN's (Firstly introduced by Y. Lecun et al. [24]) to produce much more realistic and meaningful images as generator learns deep features from data distribution  $P_{real}$ , rather than utilizing just plain pixels. So, for example, a successfully trained DCGAN on the data containing images of different dogs, learns mapping  $G : z \rightarrow X$  where  $z$  is latent noise variable and  $X$  is an image of a dog. Then, we can explore latent space  $Z$  and select other vectors  $z_i$ , such that  $z_i \in Z$ . Then, by using the generator part of the network, generate outputs  $G(z_i) \rightarrow X$ , where  $X$  is an image generated utilizing the learned deep features. If the training was successful then  $X$  is an image of a dog, that has not been seen during the training. The discriminator then is able to return a probability of whether given image  $X$  belongs to the training data manifold  $P_{real}$ .

As presented in [29], the noise vector  $z$  is connected to the dense layer which has  $n$  number of neurons, followed by a reshaping layer that reshapes input into the form suitable for the convolutional layers. Then a fractionally-strided (convolutional layer with parameter  $strides = 2$ ) convolutional layers are added, each of which reduces feature space and increases the first two dimensions (size of an image), until a desired size of an image is obtained. Normally it would be the size of the real images that are used for training. While designing an architecture of a generator one has to consider how many  $n$  neurons a dense layer should have and how many convolutional blocks containing the fractionally-strided convolutional layers (which are also referred as transpose convolutions or deconvolutions) to put.

It is also recommended to use the batch normalization (section 4.4) after each convolutional layer [36], because in many cases it improves the training stability and can help to address problems like mode collapse and checkerboard effect. This technique achieves it by reducing covariance shift by normalizing the output of the previous activation function.

In the original DCGAN paper [29] some recommendations are given for building deep convolutional generative adversarial networks:

1. Instead of using pooling layers, use strided convolutions for the discriminator and fractional-strided convolutions for the generator
2. Use batch normalization in both the generator and the discriminator
3. Avoid fully-connected layers, except the first one in a generator, which is used to shape noise input  $z$  into a form suitable for convolutions
4. Use ReLU in the generator (except the last layer which uses tanh), and use LeakyReLU with slope=0.2 for the discriminator.



### 4.5.2 Evaluation of generative models

The evaluation and comparison of generative models is an active research area and no universal approach exists. As the objective function of GAN's is a binary cross-entropy, the default measure is the likelihood [15]. The key idea is to have a function that would correlate with the quality of the generated images. The approach defines the likelihood function based on the estimation of the distribution of the real images embeddings in the discriminator. Then this function can measure the likelihood that an image belongs to the learned data distribution  $P_{data}$  based on the embeddings of the discriminator. Nevertheless, models can have a good likelihood but suffer from the problems like mode collapse (section 4.5.4) or checkerboard effects (section 4.5.7).

But there are some other approaches to it. For example, in the work of [30] they did perform an experiment with the human annotators, so that they would look at generated and real images and try to distinguish them. But the downside was that human annotators are likely to make mistakes when they get tired or unmotivated. Moreover, this approach also becomes more difficult for the humans if the images contain fuzzy or very ambiguous objects or patterns.

An alternative presented in [30] is to use the Inception score. This score is obtained by using the pre-trained Inception neural network model on the generated images to obtain a conditional label distribution  $p(y|G(z))$ . If the image has meaningful objects in it (such that Inception model is more confident about what it detects in it), this label distribution  $p(y|G(z))$  should have low entropy (so small uncertainty). In addition to the images having meaningful objects, we also want to have a diversity among generated images. That means that among different generated samples, the label distribution has to have high entropy. By combining these two criteria, researchers in [30] introduce a metric that is:

$$\exp(\mathbb{E}_x KL(p(y|x)||p(y))) \quad (4.7)$$

In this metric they use the Kullback-Leibler divergence to measure the distance between these two label probability distributions to know how well these two criteria of diversity and label certainty are met. In their work, they state that using this metric the results resembled the results of the quality assessment experiment with human annotators.

### 4.5.3 Instability of training GAN's

Training of GAN's is a hard and unstable process with no formal approach on how to solve this problem. There are many reasons behind this, most prominent being that the generator and the discriminator are being trained separately without unified loss function and either of them can get trained too much in contrast with the other. Another reason is the lack of proper evaluation metric. Because the actual loss does not provide us with much knowledge about how good the network is performing. For example, low loss of a generator can mean both, that the generator is good or that discriminator is bad which results in a bad generator training while it fools the discriminator with bad images. Moreover, there is no way to know when exactly to stop learning procedure or compare different models. However, there are several techniques collected and presented in [9] that can help to stabilize the training:

1. Normalize the inputs between -1 and 1
2. Modified loss function. Use  $-\mathbb{E}_{z \sim p_z} [\log(D(G(x)))]$  instead of  $\mathbb{E}_{z \sim p_z} [\log(1 - D(G(x)))]$  in the second term of the equation 4.6.
3. Sample a noise vector  $z$  from the normal, rather than uniform distribution [35].
4. Train the discriminator in separate batches of a real and synthetic data.
5. Stop training the discriminator or the generator if their loss becomes too small

6. Add some noise to the inputs of the discriminator and / or add Gaussian noise to the layers of the generator.
7. Use dropout (50%) in the generator.
8. Use Adam as an optimizer

#### 4.5.4 Mode collapse

Mode collapse is an effect when for any given noise vector from latent space  $z$  the generator produces the same or almost the same output. This happens because generator learns to produce one output that always fools the discriminator. Partial mode collapse is when the output is different but only by a very small deviation. Usually, we would want to have outputs as diverse as possible. However, it is often observed that sometimes there is a trade-off between image diversity and quality. One technique that has been proposed in [30] is to add label smoothing. That means, that instead of using the labels  $l = \{0, 1\}$ , use, for example, new labels  $l_n = \{0.1, 0.9\}$ .

#### 4.5.5 WGAN

WGAN stands for Wasserstein generative adversarial network, that is a type of GAN introduced in the paper written in 2017 by M. Arjovsky et al. [4] that tackled the problem of stability of training and interpretability of the loss function. Moreover, it has been observed, that WGAN's are very prone to the mode collapse.

In GAN's, the objective is essentially to learn the distribution of the data  $P_{data}$  by minimizing f-divergence (the distance between two distributions  $= P_{model}$  and  $P_{data}$ ) over region  $D$ . This convergence in classical GAN is interpreted as minimizing Jensen Shannon (JS) or KL divergence in the min max game. In the paper [4] authors argue about the drawbacks of such metrics and propose to use Earth's Mover's/1-Wasserstein distance as an alternative. This distance metric in a physical world metaphor could be formulated as finding how much work has to be done to transport one distribution to another one which is amount (mass) multiplied by the distance moved. In other words, it tries to find the smallest / simplest way to transport from one pile to another (learned distribution  $P_{model}$  to the real distribution  $P_{data}$ ). In the paper authors have demonstrated that there are cases of distributions where KL or JS divergence do not find a solution while EM distance does.

Major differences in the architecture of the gan are as follows:

1. The discriminator output is no longer sigmoid and does not represent the probability
2. The discriminator loss is the difference between the generated and real samples
3. The discriminator is trained  $n$ -times more than the generator (the paper suggests  $n = 5$ )
4. Weight clipping: Enforcing 1-Lipschitz in the discriminator weights to make them small values centered around zero

The paper has proved that using Earth's mover's distance for the divergence of the two distributions is a very strong candidate. However, there are some problems, like that weight clipping makes the weights to be pushed in one direction and clipping them sometimes introduces loops. This problem has been addressed in [17] by introducing an improved WGAN.

#### 4.5.6 WGAN-GP

Wasserstein generative adversarial network with the gradient penalty [17] (in other words - improved WGAN) addresses the problem of weight clipping in the discriminator by removing weight clipping and introduces the gradient penalty as an alternative. It penalizes the gradient of the discriminator with respect to the input instead of clipping the weights. The total loss function

of an improved GAN is:  $\mathcal{L}_{total} = \mathcal{L}_{real} + \mathcal{L}_{fake} + \mathcal{L}_{GP}$ . As explained in [17], gradient penalization happens by itself, because it is included in the general loss function. That is why authors encourage not to use batch normalization.

#### 4.5.7 Checkerboard effect

The task of the generator is to generate an image and it does it by using the layers that perform upscaling task and is referred usually as "deconvolution" or transposed convolution layer. Often these generated images come out with artefacts that resemble the checkerboard pattern (a repetitive grid of squares). Such an effect is undesired as it lowers the quality of an image. This effect happens among different types of neural networks that do increase the resolution of an image in their architecture. This effect was researched in [2] and few approaches are suggested to deal with this effect:

1. In the transposed convolution layer use a kernel size that is divisible by the stride, because otherwise, a number of lower features that contribute to higher level features is not constant. That means that some upscaled pixels receive more attention than others.
2. Separate upsampling and convolution of features into two layers. That means, for example, firstly to resize the image (using nearest-neighbour interpolation or bilinear interpolation) and then convolute the features.



(a) Image with checkerboard effect



(b) Image without checkerboard effect

Figure 4.7: Figure illustrating checkerboard effect [27]

## Chapter 5

# Approaches

In this chapter we describe the approaches to the research problem of this project. We start by describing the current approach used in the company "Prodrive technologies" and what drawbacks it has. Then we introduce a supervised classification approach. After that, we will explore approaches that work in an unsupervised manner. Firstly we will see anomaly detection technique using autoencoder. Then we look at how we can extract meaningful image features and use them with OCSVM. Lastly, we introduce an approach that uses GAN's and its modified version - an improved Wasserstein GAN.

### 5.1 Current approach

The blueish/white paste is the sealant and it must be put on the product according to some design in CAD. Currently, the sealant inspection algorithm takes a sample (the golden sample) and compares all subsequent images according to the sealant it detected in the golden sample. There are several problems with this approach. Firstly, it requires a lot of knowledge in the selection of the parameters to do the segmentation and to find the sealant on the product, which means the operators often have to correct the sealant inspection. Furthermore, this is not always done correctly, as employees that set up the parameters sometimes make errors in setting these parameters. Next, the inspection algorithm is not robust to light changes or if the product gets slightly shifted in the images. Lastly, using a golden sample is not a good way of checking if the product is similar to the design, the CAD data should be leading in deciding whether the sealant is correct or not, because the golden sample does not cover all the variability within distribution what is considered "good" samples.

### 5.2 Supervised convolutional neural network approach

The most straightforward deep learning technique is to train a convolutional neural network classifier on a dataset containing labelled pass / fail images. Such a network should consist of several convolutional blocks, where one block consists of a convolutional layer followed by an activation function and a maximum pooling layer. Alternatives, such as Average pooling layer are possible, however, in the experiments that have been performed, average pooling proved to be worse than maximum pooling.

An example of such network architecture (and the one that I have tried in the experiments) is:

For the last dense layer, there are two options for an activation function - 'softmax' with two output neurons, which assigns probabilities for the each class that add up to 1. Or 'sigmoid' with the one output neuron.

However, such a neural network has to learn the important visual features and that requires a very big amount of training data, which is not available in this project context. To solve this issue,

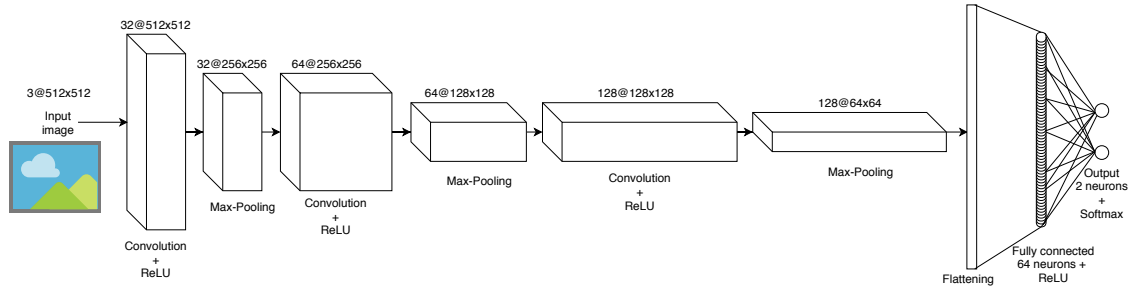


Figure 5.1: Simple convolutional neural network

a transfer learning approach offers a method to reuse publicly available pre-trained convolutional neural network weights. In the next section the transfer learning approach is explained.

In the experiments, few pre-trained networks that are available in the Keras repository have been used. Particularly - VGG16, InceptionNetV3 and Xception. They have been trained with 'imagenet' dataset and loaded without their top fully-connected layers (as those are trained to classify the 10 classes in 'imagenet' dataset). That means that only the weights of the convolutional part have been used. On top of this backbone of the convolutional feature extraction layers, some new custom layers were added on top. These layers were:

1. **Flattening layer**

It flattens the output from previous convolutional layers so that we can connect it to a fully connected layer

2. **Fully connected layer with 256 neurons**

We add this hidden dense layer so that a network could learn how to make a decision (classification) from the features extracted by the convolutional layers

3. **Dropout layer**

We add the dropout layer to prevent over-fitting

4. **Fully connected layer with 2 neurons (with softmax activation function)**

Output layer to make a prediction

The optimizer function used in supervised learning experiments was SGD (stochastic gradient descent) with a small learning rate of 0.00001 and momentum - 0.9, based on the observations and the recommendations from various tutorials and research papers.

## 5.3 Unsupervised learning

One of the problems in this project context is that in the casual scenario there is a very limited amount of an unlabelled data, so the supervised classification approaches do not fit in this context as they do require enough labelled data to be trained. However, it is relatively easy for operators to collect a small amount (20 - 50) of positive samples for a training data. So the approaches further this section are essentially a special case of unsupervised learning, because during the training process only the one-sided label (positive class) is known.

### 5.3.1 Anomaly detection with autoencoder

In the experiments an autoencoder with following architecture was used (see Figure 5.2):

*Important note:* The convolutional layer in the encoder part has  $strides = 2$ , while convolutional layers in decoder part have  $strides = 1$ , as it has Upsampling layer to increase the size of the first

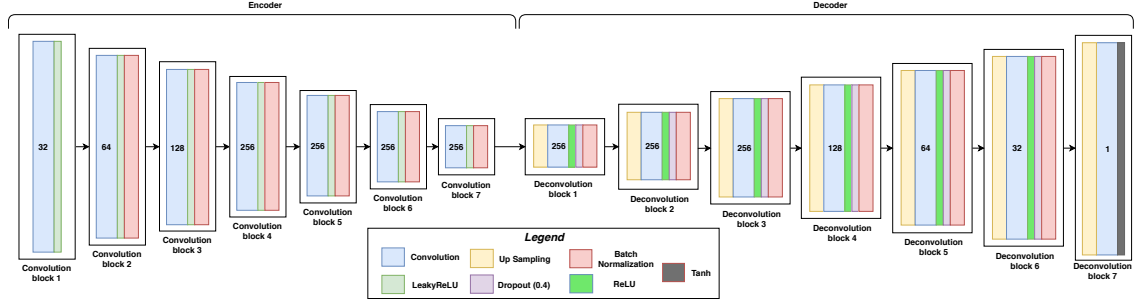


Figure 5.2: Autoencoder

two dimensions (height and width) while convolution reduces the the third dimension of features. Model is compiled with loss function of “Mean Absolute Error” and optimizer Adam with learning rate 0.001, based on the observations and the recommendations from various tutorials and research papers.

In the context of anomaly detection, autoencoders can be used solely or in hybrid models. In this project, it has been used both as a feature extractor (for OCSVM) and as encoder-decoder network for anomaly detection by itself. First and a very simple idea is to train autoencoder on good samples of images, so it learns a good representation of such images. Then, after the training is done, we can use this autoencoder to encode / decode good or anomalous images. The idea is that the reconstruction error of the anomalous images should be higher than of the images similar to the ones on which the autoencoder has been trained on. The reconstruction loss is calculated as a L2-norm (or mean squared error) as authors P. Vincent et. al have chosen in their successful work about denoising autoencoders [34]. It is defined by the formula 5.1:

$$\frac{1}{n} \sum_{i=1}^n (x_t - \hat{x}_t)^2 \quad (5.1)$$

Where  $x_t - \hat{x}_t$  denotes the difference (error) between two pixels, and  $n$  denotes the number of pixels.

In the hybrid approach (section 4.3.2) the autoencoders are trained in a similar manner, but only the encoder part is used as a feature extractor and then the classification or anomaly detection algorithms, like OCSVM are used are used.

### 5.3.2 Anomaly detection with OCSVM

In high-dimensional problems like visual inspection, features extracted from a positive class samples with deep convolutional neural networks (for example, the output of an encoder part in the autoencoder scenario) are used to train OCSVM. Then, during testing, the same feature extractor is used to get features of a test sample (as shown in Figure 5.3). If this item is anomalous, its features should differ to some extent from the features of the positive samples that have been used to train OCSVM and the model will classify this item as an anomaly, because the projection of those features should fall outside the hyperplane decision boundary.

In the experiments, we compare the OCSVMs trained with the features extracted from different feature extractors. These include a traditional KAZE feature extraction algorithm, the encoder part of the deep convolutional autoencoder and several pretrained neural networks on imagenet dataset. Additionally, for dataset B4, I have also included the features extracted using the discriminator from the training of GAN’s. KAZE algorithm was used with the 2048 features extracted

from every image.

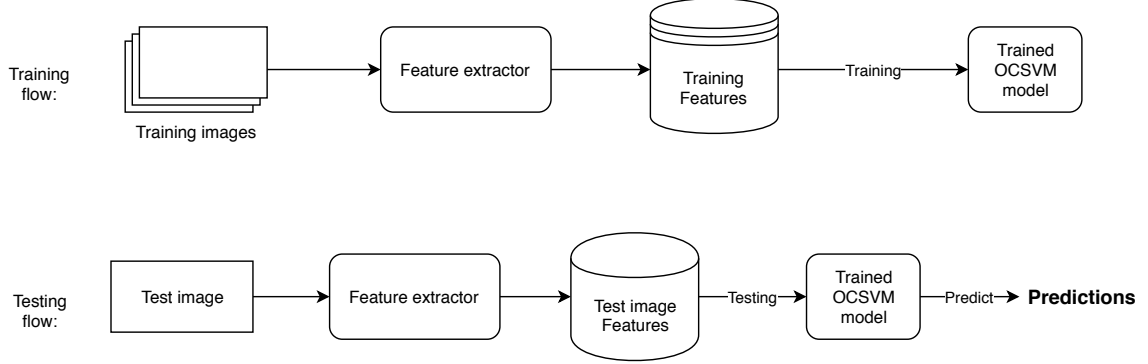


Figure 5.3: Training and testing flow of the OCSVM with the extracted features

Hyperparameters of the one-class SVM were obtained by trying out few different values and testing how they perform on the test set. Then best parameters were chosen and fixed for the experiments that use other feature extractors. In the few cases, some alternative parameters were tried as well and reported. Performing the parameter search with GridSearch I was able to find better hyperparameters that has led to better results, however, with a very high chance of overfitting, as there is not enough data to make quality train / validation / test sets. Moreover, in the real case scenario, we would not have any testing data, only the positive training samples. That means that there is no proper way to do parameter search over OCSVM, as it requires some evaluation of performance, which is not possible without test data.

### 5.3.3 Anomaly detection with GAN's

Over the past years there has been an increasing amount of research done on the use of the generative adversarial networks for anomaly detection problems [31] (AnoGAN), [11] (ADGAN), [10] (GANomaly). These models can learn data distribution  $P_{model}$   $P_{data}$  and generate new realistic samples. Moreover, it also learns to distinguish whether a given image belongs to learned data distribution  $P_{model}$  or not.

This is done by making the generative model learn over one class, or, in other words, the "normal" or "healthy" distribution  $p$  and use this learned distribution to decide whether a new data point belongs to this distribution or not, based on the ability of the generative network to generate a similar image to the test image  $G(z) \sim x$ . This is done as follows: given a test image  $x_{test}$ , we try to find a latent vector  $z$ , from which a generated image  $G(z)$  is the most similar to the  $x_{test}$ . If the  $x_{test}$  belongs to a healthy data distribution  $p$ , the generator will be able to generate  $G(z)$  that is similar (by some metric) to  $x_{test}$  and if the image is anomalous, the difference between  $x_{test}$  and  $G(z)$  will be bigger.

There are few works that use a similar approach to the one presented here to do the anomaly detection in images using GAN's. Schlegl et al. [31] use such approach to detect the markers of the disease in the tomography images of the retina. They split each tomography image in the training dataset into the  $k$  patches of the size  $m \times n$  and train GAN on these patches. Deecke et al. [11] also have done work very similar to [31], with slight differences in the way they compute anomaly score as they do not use an intermediate discriminator layer for that.

In our work we follow the AnoGAN approach described in [31], except that we train a network to generate a full image, instead of image patches. If we would split our images into small patches and train a network on those, we would also need to do the anomaly detection which would be patch-based. Then imagine a test image that has only 50% of the sealant put. Then for the patches that contain the sealant we could generate a similar patch, but also for the patch that has

no sealant where it should be. This is because the parts of the image containing no sealant (and it should not be there) could look the same as the patch where the sealant had to be put. In other words, we can not distinguish the healthy part of the image without the sealant, from anomalous part of an image, also without a sealant. It is different in, for example, tomography images used in [31], where no lesions should appear in any place (or patch) of the image.

### 5.3.4 Architecture of GAN's

In this project, both DCGAN and GP-WGAN have been explored and tested. The convolutional layers in the generator part (in both DCGAN and WGAN) have *strides* = 1 and *strides* = 2 in the discriminators convolutional layers.

The DCGAN architecture is shown in Figure 5.4. We implement similar architecture to the one presented in the original DCGAN paper [29]. Additionally, we add Minibatch discrimination layer, as suggested in [30]. It penalizes generator more if it has low entropy within a batch (generated samples are similar).

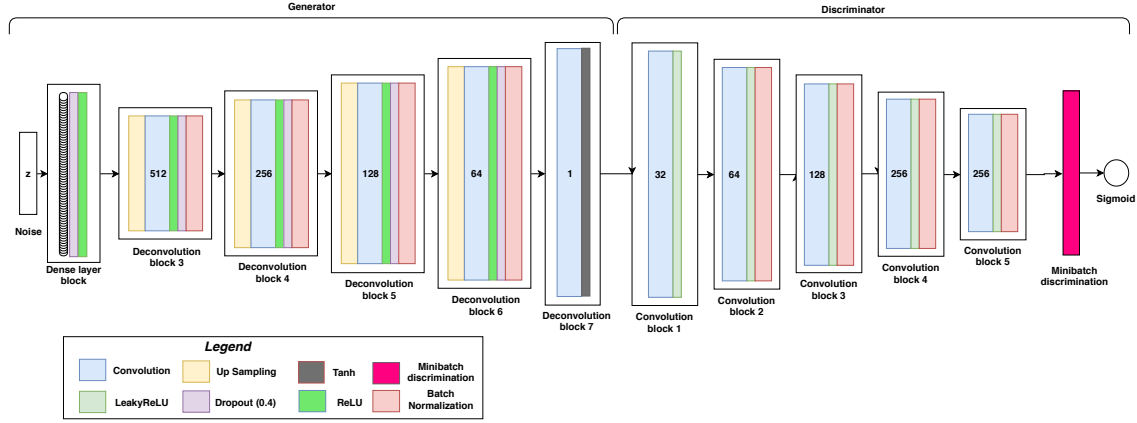


Figure 5.4: Architecture of DCGAN

The architecture of the GP-WGAN we implement is similar (Figure 5.5), but modified in a way how it is suggested in the original paper - [4]. These differences are - no batch normalization layers, added dropout layers in the generator, the first layer of the discriminator has *strides*=1, no MinibatchDiscrimination layer, the output neuron does not have an activation function. And, of course the model is built with a different loss function (more details in section 4.5.5).

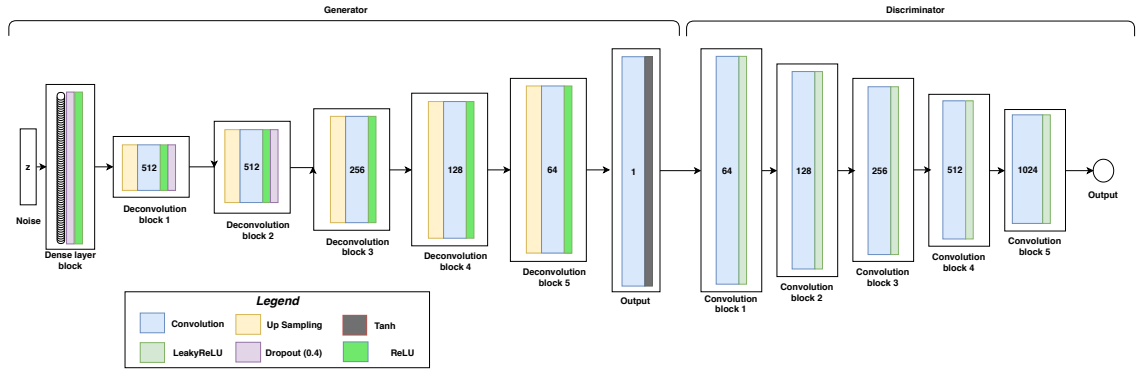


Figure 5.5: Architecture of WGAN



### Unsupervised manifold learning of healthy samples

To learn a healthy data distribution  $P_{real}$  we train a GAN with an architecture similar to the one proposed in [29]. During the training, the generator learns to generate realistic images from the latest space vector  $z$  while the discriminator gets better at distinguishing real from synthetic data. A very important feature of this approach is that our generator learns to not only generate samples as it has seen during training but also new, unseen, but still 'healthy' samples that fall in the learned data distribution  $P_{model}$ . This feature is important in the context of this project because we might not have the training data that covers all the possible variability of the images that still would be considered as non-anomalous. For example, let us say that we are given a training set of "healthy" images  $X_t$  that contains images  $x_1, x_2 \in X_t$ , where  $x_1$  has a sealant with width  $w_1 = 10px$  and  $x_2$  has sealant put with width  $w_2 = 15px$ . We want our model to learn that a "healthy" width of sealant is  $10px \leq w \leq 15px$ , and that if  $w < 10px$  or  $w > 15px$  the test image with such sealant is anomalous. This can be learned, because during the training the discriminator would see real images with varying sealant width  $10px \leq w \leq 15px$ . If the generator generates an image with sealant's width outside this range, the discriminator would correctly guess that it was generated and backpropagate this information. The generator, in turn, learns to generate images in this range, as otherwise the discriminator distinguishes that this data was generated.

This allows us to have smaller training sets as long as we have images that define the boundaries of what is considered "healthy" data. Then the model can properly work with unseen but still "healthy" data.

### Mapping to the latent space

When the training has finished, the generator  $G$  has learned the mapping  $G(z) = z \rightarrow x$  from latent space to a new healthy and realistic images, so we can generate many new images that are non-anomalous. However, the inverse mapping  $\vartheta(x) = x \rightarrow z$  is not readily available. So to find such  $z$ , we start by sampling random  $z_0$  from the latent space distribution  $Z$  and generate  $G(z_0)$ . Then, for  $l = 0, \dots, k$  steps we backpropagate the loss between first generated image  $G(z_0)$  and  $x_{test}$ , making the subsequent generated image  $G(z_{l+1})$  more like  $x_{test}$ , by updating the gradients of the coefficients of  $z_l$ , that results in an updated position in the latent space. After  $k$  steps, we have found the most similar image  $G(z_k)$  to the test image  $x_{test}$ . We have used  $k = 500$  steps, the same as in [31].

The work in [11] defines a loss function consisting of only the reconstruction loss, while Schlegl et al. [31] defines a loss function for the mapping  $x_{test} \rightarrow z$  that uses two components - the reconstruction (residual) 5.2 loss and the discriminative loss 5.3. The reconstruction loss pushes images  $G(z_k)$  and  $x_{test}$  to be more visually similar, while discriminative loss pushes generated image  $G(z_k)$  to lie on a learned manifold. So both, discriminator and generator are used to adapt the coefficients of  $z$  via backpropagation.

**Reconstruction loss** The *reconstruction loss* measures the visual difference between  $x_{test}$  and  $G(z_k)$  and is defined by:

$$\mathcal{L}_R(z_k) = \sum |x_{test} - G(z_k)| \quad (5.2)$$

Where  $G(z_k)$  is an image generated by a generator after  $k$  steps and is the closest image that generator  $G$  could generate. If  $x_{test}$  and  $G(z_k)$  are visually similar, the reconstruction loss is close to zero.

**Discriminative loss** The *discriminative loss* is based on the feature matching between images and this enforces the generated images  $G(z_k)$  to have similar intermediate feature representations as the training data. This loss is defined as follows:

$$\mathcal{L}_D(z_k) = \sum |f(x_{test}) - f(G(z_k))| \quad (5.3)$$

Here  $f(\cdot)$  is an output of some intermediate layer of the discriminator. In this way the approach utilizes the discriminator as a feature extractor.

So the overall loss is a weighted sum of both components:

$$\mathcal{L}(z_k) = (1 - \alpha) \cdot L_R(z_k) + \alpha \cdot L_D(z_k) \quad (5.4)$$

### Detecting anomalies

From the previous steps we have trained the generator  $G$  to produce healthy images and after some parameter optimization we have obtained a latent vector  $z_k$  for a test image  $x_{test}$ . Now we need to define a way to classify test image as an anomaly or not. We do this by defining an Anomaly Score. This score can be defined in a different ways and few of them have been explored and compared. Authors Schlegl et. al in [31] define this score to be similar to the loss function they use (equation 5.4) and is defined like this:

$$A(x) = (1 - \alpha) \cdot R(x) + \alpha \cdot D(x) \quad (5.5)$$

Where  $R(x)$  and  $D(x)$  are the *reconstruction*  $L_R(z_k)$  and the *discriminative*  $L_D(z_k)$  losses of the last ( $k^{th}$ ) iteration of the mapping to the latent space procedure.

The rationale is that  $A(x)$  is high for the anomalous images and low for the similar ones, as a similar image  $x_t$  has been seen during the training and the model could find a  $z$  from the the latent space from which a similar image  $x_s \sim x_t \sim x_{test}$  could be found. So, we need to define a threshold  $\epsilon$  such that:

$$f(x) = \begin{cases} 1, & \text{if } A(x) \geq \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (5.6)$$

Where 1 is an anomaly and 0 is not.

There is no proper mathematical way to determine the value of  $\epsilon$ , so a more empirical approach has to be used. In my work, after completing the training I have repeated the second step of mapping test images to latent space many times using the some of the images from training data together with some  $k$  randomly generated images  $G(z_{i..k})$  to compute anomaly scores of these images. Then, I took the upper bound of these anomaly scores  $A(x)$ , and added a small margin ( $\gamma = 5\%$ ) so that  $\epsilon = \lceil A(x) \rceil \cdot (1 + \gamma)$ . Note: Having gamma parameter is not necessarily needed and it just defined how conservative an anomaly detection model should be.

### Anomaly score

After our model can produce  $G(z)$  we want to compute the anomaly score  $A(x)$ , which rationale is explained in the subsection above. However, there are multiple alternative metrics we can choose from to base our anomaly score on. Work in [31] use their loss function that basically combines mean absolute error and the discriminative (based on features) losses, while work in [38] uses only the generator loss (mean absolute error). In this work, we use loss as defined in AnoGAN [31], discriminative loss and Mean absolute error.

After the similar image has been generated by the generator after fitting for the latent variable  $z$ , we use a non local fast denoising algorithm [7] to remove noise from the images. This algorithm removes the noise and smoothens an image while keeping the structural information, like lines or curves untouched. We do this to make them more consistent in the pixel values for the metrics like the mean absolute error. Moreover, after calculating the residual image (between  $x$  and  $G(z)$ ) we apply custom threshold function  $T(x)$  that is defined as follows:

$$\forall_{c \in X} f(c) = \begin{cases} 255 & \text{if } c \geq 150 \\ c & \text{if } 100 < c < 150 \\ 0 & \text{if } c \leq 100 \end{cases} \quad (5.7)$$

In this way we "filter out" the most of noisy values after image subtraction that are close to 0 and emphasize more on places with a bigger absolute error.

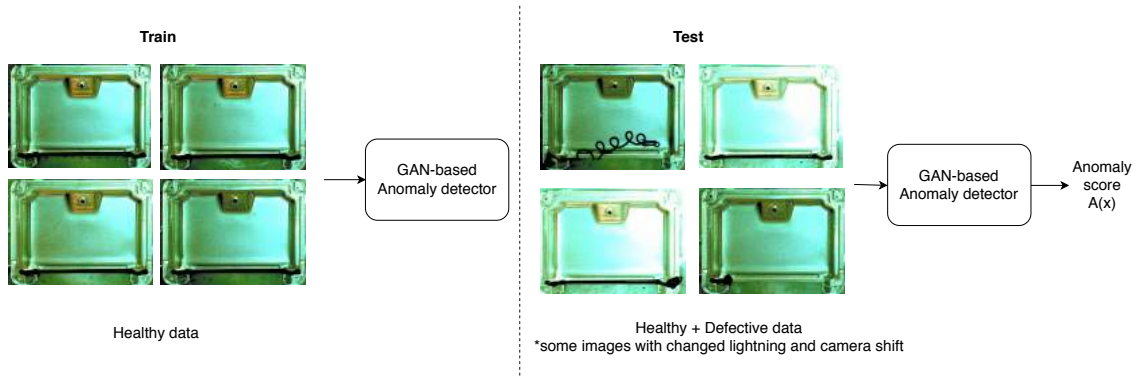


Figure 5.6: Overview of the training and testing approach

In the results section, we present the results of the anomaly detection using these methods using all these metrics and see if there is a difference and how big is among these metrics.

## Chapter 6

# Results

### 6.1 Supervised convolutional neural network approach

The results obtained from the supervised approach are promising, however, requires a big amount of labelled training data. During experiments, such data was available, but it was not completely correctly labelled (labels taken from current visual inspection algorithms), so the results have to be taken with extra care. Moreover, the A1 dataset have been created, by using samples of "pass" class, but the "fail" class has been created from the samples without sealant, combined together with around 25 definite anomalies with sealant put defectively (labelled by a non-professional). So, this dataset provides a good testing ground for the supervised approach as it is very easy for a model to learn that one class is with sealant and another is without.

In the table 6.1 summary of results obtained is shown.

ID	Data	Conv. layers	Loss	Accuracy	ROC AUC	TN / FP / FN / TP
1	All	InceptionV3	0.7715	0.8952	0.9568	340 / 59 / 0 / 164
2	All	InceptionV3	0.0870	0.9655	0.9920	2031 / 73 / 351 / 9862
3	A1	InceptionV3	0.0172	0.9946	0.9998	399 / 0 / 3 / 161
4	A1	Own	0.0180	0.9928	0.9993	399 / 0 / 4 / 160
5	A1	VGG16	0.0207	0.9982	0.9997	399 / 0 / 1 / 163
6	A2	VGG16	0.1726	0.9460	0.9581	696 / 234 / 37 / 4052
7	A2*	VGG16	0.2362	0.9456	0.9027	698 / 232 / 41 / 4048

Table 6.1: Supervised classifier results

*\*Images used here were grayscale*

The models with ID  $\in$  1, 2 have been fine tuned on the data taken from all the clusters. The model with ID=1 has then been tested on dataset A1, while the model with ID=2 has been tested with all raw data. Models with ID  $\in$  3, 4, 5 are trained and tested with cluster A1 data. The result is satisfiable - ROC AUC scores are close to 1 and only a few false negatives are present. However, from this cluster, only 39 are "true" anomalies (including artificially created ones) and

others are just images without any sealant put at all (in some cases it can also be considered a defect).

If we look at the models with the ids 5 and 6, we see that the difference between them is that one was trained with grayscaled images and another with 3-channel RGB images. Experiments show that under the same setup, in the supervised approach RGB images are better in terms of ROC AUC. However, the difference between actual predictions (validation accuracy) is not that different. That means that colour channels make the model more “confident”, that is, its predictions are closer to the true value (because the output is probabilistic softmax), but this improvement is not big enough to noticeably change the predictions.

As we see, there is no significant difference between VGG16 and InceptionV3 backbones. The ROC AUC scores are high for all the settings and we can see that the supervised approach can show good performance, but requires lots of training data from both “pass” and “fail” data and is prone to overfitting.

All the above experiments have been performed using the pre-trained neural network back-bones. However, when performing the same experiments on these datasets with these architectures (VGG16, InceptionV3, etc.) but without the pre-trained imagenet weights, but using random initialization, almost the same results can be achieved. However, the difference being that it takes more time to achieve similar performance as models with pre-trained weights. For example, using VGG16 network in the supervised experiment on dataset Cluster4\_corr without imagenet weights in 3-4 epochs achieves the same performance as the network with the preloaded imagenet weights achieves in 2 epochs (in the experiment, one epoch took around 10 minutes).

This means that it is not necessary to pre-load the imagenet weights if the dataset is big enough so that the network can learn, but can speed up things. Moreover, as we will see in the next approach, we will use pre-trained networks as feature extractors on relatively small unlabelled or semi-labelled (only positive class) training dataset, where training such a network is not possible, supports the claim that features extracted from such pre-trained networks are representative and can be used to encode the image of an arbitrary domain into the feature space.

### 6.1.1 Transfer learning

In our experiments the transfer learning with all convolutional (reused) layers frozen did not work very well (the model immediately overfits - low training loss but high validation loss).

When not freezing transferred layers and allowing back-propagation, model learns fast (in just a few epochs) with validation loss  $< 0.2$  and validation accuracy  $> 90\%$ .

Freezing the first convolutional layers makes model worse than without freezing (although converges to similar performance after a number of epochs), but is much better than models with all layers frozen.

A probable reason is that the actual features learned from imagenet dataset images are very different in nature than the features defining defects in this domain. However, in my understanding, the most basic features learned in first convolutional layers are still similar (lines, curves, etc.) and because of that allowing model to fine-tune itself enables us to achieve working model.

Training transferred model without freezing, then using this model as a transferable model, while freezing the convolutional layers and adding new fully-connected layers - produces the same results as an initial model.

VGG16 fine-tuned model with imagenet weights preloaded trains 20% slower than InceptionV3 and achieves very similar results. However, InceptionV3 does not work with a `backprop_modifier` option when trying to visualise CAM or saliency maps and, hence, produces inaccurate attention maps. This is important as these maps should show a defect localization. It is further explained in the next section 6.1.2.

### 6.1.2 Class activation maps

The basic idea behind the class activation maps is simple: we compute the gradient based class activation map, which shows parts of the input that maximise the outputs of *filter\_indices* in *layer\_idx*. By doing this, we want to mark the pixels in the image that contribute the most to the output of the model, which can be one of the two classes - pass or fail. The intuition behind this is that a model should predict an image being an anomaly because of the features extracted from specific parts of the image, parts where anomaly should be. And this is done by using the penultimate convolutional layer's (pre-dense) gradients, so that the spatial information of an image would not get lost in the dense layers.

Using `backprop-modifier='guided'` returns much better and meaningful (at least what one would expect) results. InceptionV3 model does not support this option, so VGG16 proved to be a better choice in this context.

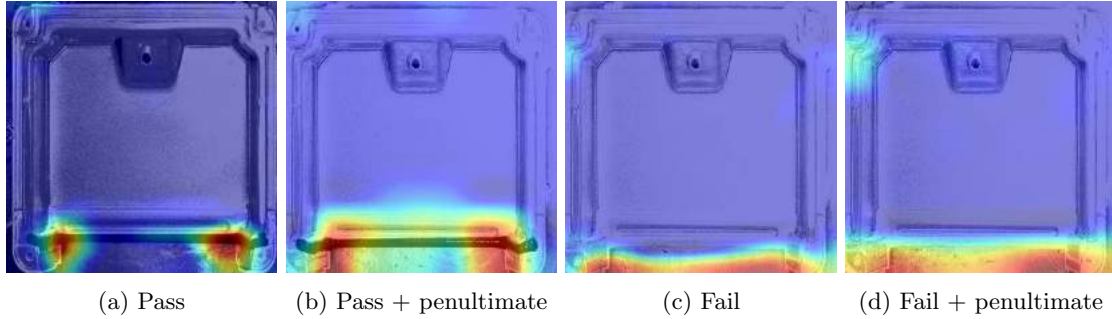


Figure 6.1: Class activation maps

By setting a penultimate layer, we define a layer from which feature maps are taken and used to compute the gradients with respect to the output. If no penultimate layer was set, the nearest (to the output layer) convolutional or pooling layer is taken. In my experiments, I have tried setting one convolutional layer below, to get 'lower level' feature maps, but that did not result in different outcome.

Grad-cam for correct cluster 4 more or less makes sense for PASS predictions, although it “marks” only part of the sealant and not all of it, as it would be expected. This might be because the model has learned from the data it had that the full sealant usually is not important and only parts of it.

But for the fail cases, it does show class activation from the pixels that are nearby, but not exactly in a place where the sealant is missing. That is not exactly what has been expected, as the pixels that cover sealant area should contribute to class “fail” neuron in the last layer of the classifier.

## 6.2 Unsupervised approaches

### 6.2.1 Feature extraction and OCSVM

During the experiments a choice to use the RBF kernel has been made, because it is the default and most popular kernel used in OCSVM. Moreover, the linear kernel has always shown poorer results (see linear kernel table of cluster c1), so this option has been discarded. Other kernels did not differ from RBF, so the choice to stick with the RBF kernel has been made.

The choice of a layer from which features were extracted was chosen to have (at least in pre-trained models) similar shape. This is summarized in table 6.2:

Features	Shape	No. of features	Layer name
KAZE	(n, 2048, 1)	2048	-
Autoencoder	(n, 64, 64, 128)	524'288	encoder_pool
VGG16	(n, 32, 32, 512)	524'288	block5_conv3
InceptionV3	(n, 30, 30, 192)	172'800	conv2d_40
Xception	(n, 16, 16, 1024)	262'144	conv2d_3

Table 6.2: Extracted features

### 6.2.2 Hyperparameters of OCSVM

The performance has been sensitive to the “gamma” hyperparameter of OCSVM with RBF kernel. The choice which parameters to use has been made by trying out few experiments and looking at the test results. In general, small gamma values have shown to work good, while changing the “nu” parameter had little to no influence. However, very small NU values (  $< 0.05$  ) have failed to produce good results.

For the dataset B1, the hyperparameter of gamma has been left to default option, which is  $1 / n\_features$ , so it differs for different feature extractors. This choice has been left to default, as trying smaller or bigger values resulted in poorer performance.

OCSVM with these parameters performed similarly when applied to pre trained feature extractors. However, using our own trained autoencoder, smaller gamma values showed better results. The comparison between  $\gamma=1e-8$  and  $\gamma=1e-3$  while using an autoencoder as a feature extractor can be seen in the figures summarising results below.

Choosing the right hyperparameters might be a challenge, because one has to decide which is the most important metric on which the decision which hyperparameters are the best will be based. In these experiments, the hyperparameters which maximize ROC AUC score have been chosen as the best ones. However, sometimes some other metric might be more important, like F1 score, or just plain ratio (or just single value) or False negatives and False positives.

In the real case scenario this hyperparameter choice would be even harder to make, as we would not have any testing data. So there would be no way to check whether given hyperparameters perform well or not.

In this approach, several feature extraction models or algorithms have been used on three different datasets. These feature extractors are:

1. Classical features extracted with KAZE algorithm
2. Different pretrained models with the imagenet dataset
  - (a) InceptionV3
  - (b) Xception
  - (c) VGG16
3. Autoencoder trained on only the positive (and augmented) samples

The datasets with which models have been trained in these experiments were chosen to be more different from each other, because that supports the claim of the generality of this approach to different datasets. The datasets can be considered different as they have a different number of instances, the difficulty of defects (smaller/bigger in relative to the image size, more or less distinct, etc.).

These datasets used in this approach are number B1, B4, and C1.

### 6.2.3 Dataset B1

Features	F1 score	ROC AUC	TN / FP / FN / TP	Parameters
KAZE	0.2075	0.5358	41 / 2 / 82 / 11	gamma=0.0005, nu=0.5
Autoencoder	0.4179	0.5854	50 / 5 / 75 / 25	gamma=1.9e-6, nu=0.5
VGG16	0.4651	0.5915	37 / 6 / 63 / 30	gamma=1.9e-6, nu=0.5
InceptionV3	0.3846	0.5795	50 / 5 / 75 / 25	gamma=5.8e-6, nu=0.5
Xception	0.7946	0.6268	20 / 35 / 11 / 89	gamma=3.8e-6, nu=0.5

Table 6.3: B1 dataset results with RBF kernel

*\*gamma= 1/n\_features*

The first thing to note is that this dataset was pretty noisy and many anomalies were minor, and as such, features not too different from each other between the healthy and defect samples. The testing data has been labelled by a non-professional inspector, so many samples that are among healthy training data can be defective. Or samples that are among anomalous data can have some healthy samples mixed in. This is due to the fact that the difference between anomalous and not sometimes is very small and labelling them manually can result in many errors.

From the result tables of both kernels - table 6.3 and table 6.4, we see that none of the feature extractors provided good and distinct features to fit OCSVM model for this dataset. Even when parameters were found with the grid search, which often leads to overfitting, did not produce good results. This is probably due to the fact that this dataset has not been labelled correctly, as explained in the end of the section 3.2.

To fix this problem few approaches can be tried. Firstly, better and more distinct (between healthy and anomalous samples) data can be used. This can also be achieved by trying to do unsupervised clustering, like k-means clustering algorithm with k=2 (bigger number of clusters should be considered as well). The rationale behind this is that healthy images should have more similar features than anomalous samples and thus, should be clustered together. However, if this approach would also fail to produce a good cluster of healthy samples, then it can be concluded that the problem lies in the data itself.

Below, in the Figure 6.2 examples of few images are shown: As we see from the false positive case, the label (defect or not) of an image is not clear even for a human inspector. And there are many pictures like that, as they have been labelled by me for the testing purposes. Also looking at false positives we see a very similar situation - it is unclear whether the sealant is anomalous or not for a human inspector, and the algorithm classified it as an anomaly.



Features	F1 score	ROC AUC	TN / FP / FN / TP	Parameters
KAZE	0.6455	0.6114	29 / 14 / 42 / 51	gamma=0.0005, nu=0.5
Autoencoder	0.5764	0.4378	15 / 28 / 44 / 49	gamma=1.9e-6, nu=0.5
VGG16	0.2372	0.4473	32 / 11 / 79 / 14	gamma=1.9e-6, nu=0.5
InceptionV3	0.5	0.4601	22 / 21 / 55 / 38	gamma=5.8e-6, nu=0.5
Xception	0.4769	0.5968	37 / 6 / 62 / 31	gamma=3.8e-6, nu=0.5

Table 6.4: B1 dataset results with LINEAR kernel

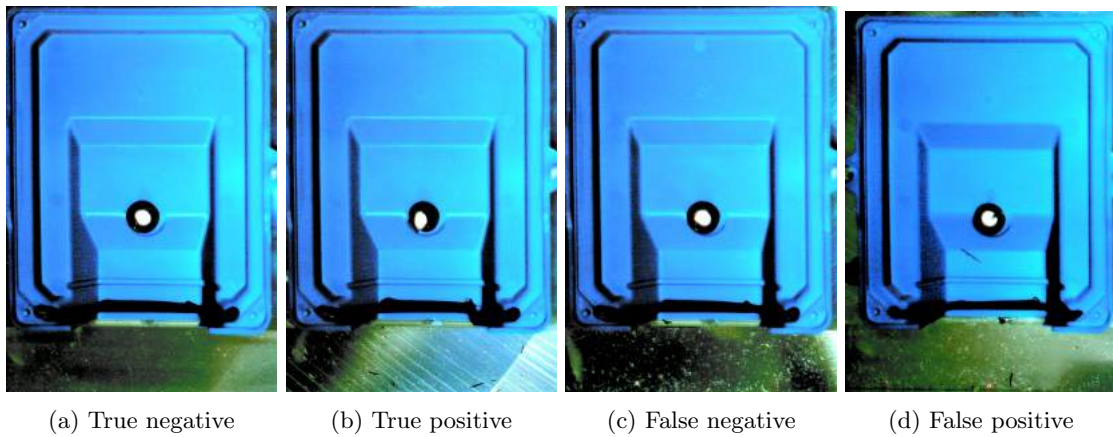


Figure 6.2: Examples of B1 dataset classified images

### 6.2.4 Dataset B4

B4 dataset has few “naturally” occurred anomalies together mixed with anomalies that have been introduced superficially by in-painting black lines that resemble the “sealing” in a way that it would be considered a definite anomaly in real life setting.

It is important to note, that this dataset consists of two types of images (same product) - one type is darker and another is lighter, also horizontally and vertically shifted (see Fig. 6.3 (a) and (c) are darker, while (b) and (d) are lighter and shifted), and these changes are from a real world scenario.

It is very important, because the models have been trained only using the darker (and augmented from them) images, while testing has been performed using both types. This is important, because this example exactly shows the robustness of a model to lightning changes and horizontal / vertical shifts.

Features	F1 score	ROC AUC	TN / FP / FN / TP	Parameters
KAZE	0.6804	0.6265	13 / 6 / 25 / 33	gamma=1e-8, nu=0.5
Autoencoder	0.5238	0.5843	15 / 4 / 36 / 22	gamma=1e-8, nu=0.5
Autoencoder	0.8644	0.7028	10 / 9 / 7 / 51	gamma=1e-3, nu=0.1
VGG16	0.9075	0.7813	11 / 8 / 5 / 53	gamma=1e-8, nu=0.5
InceptionV3	0.9105	0.7459	10 / 9 / 2 / 56	gamma=1e-8, nu=0.5
Xception	0.8869	0.7817	13 / 6 / 7 / 51	gamma=1e-8, nu=0.5
GAN Discriminator	0.7586	0.5108	5 / 14 / 14 / 44	gamma=1e-8, nu=0.5

Table 6.5: B4 dataset results with RBF kernel

In this dataset deep feature extractors, all proved to be better than using the features derived from KAZE algorithm by a noticeable margin. The best feature extractor for this dataset was InceptionV3 considering the F1 metric (or false negative / false positive ratio), or the VGG16, if we consider ROC AUC score.

Looking at the false negatives, the few pictures that get misclassified as a healthy sample, are those, which have only visually small anomaly in-painted. Moreover the anomaly itself has smooth, round edges (example of one such image is in the table below).

Using the features extracted from the autoencoder, with the chosen parameters, the model did not perform well. However, trying smaller gamma and nu parameter values of OCSVM had improved

the results, as can be seen in the table 6.5.

Features	F1 score	ROC AUC	TN / FP / FN / TP	Parameters
KAZE	0.5584	0.6016	15 / 4 / 34 / 24	nu=0.1
Autoencoder	0.6279	0.7064	18 / 1 / 31 / 27	nu=0.1
Autoencoder	0.7874	0.4310	0 / 19 / 8 / 50	nu=0.5
VGG16	0.9075	0.7064	18 / 1 / 31 / 27	nu=0.1
InceptionV3	0.6279	0.7064	18 / 1 / 31 / 27	nu=0.1
Xception	0.6279	0.7064	18 / 1 / 31 / 27	nu=0.1
GAN Discriminator	0.5842	0.5925	14 / 5 / 32 / 26	nu=0.5

Table 6.6: B4 dataset results with LINEAR kernel

The table 6.6 above shows the results when using the linear kernel. As we see, the results of OCSVM with the linear kernel for different deep features extractors are the same, except for the KAZE algorithm's features which shows similar, but worse performance.

In the Figure 6.3 a few examples of the images are shown.

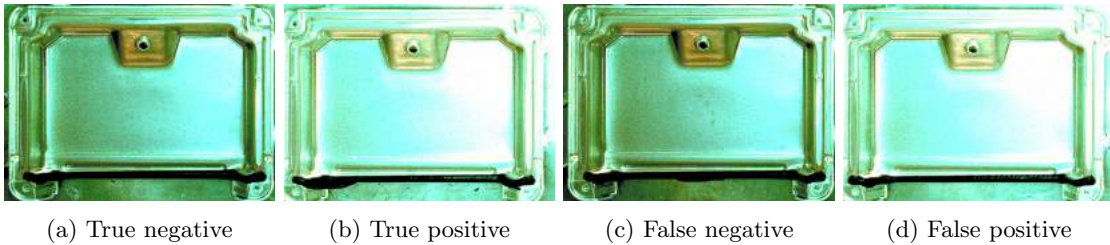


Figure 6.3: Examples of B4 dataset classified images

### 6.2.5 Dataset C1

RBF kernel of OCSVM using different deep feature extractors has performed well in these experiments with dataset C1, while autoencoder's features had produced the best results - all anomalies have been detected while only three false positives have been raised.

Features extracted from the KAZE algorithm with the parameters chosen for pre-trained models or autoencoder did not work and predicted that all images follow the healthy distribution. Trying bigger gamma value (gamma=0.1) had improved the result. Results are shown in the table below:

In comparison with the linear kernel (Table 6.8), the results were bad in the terms of F1 and ROC AUC scores, but also that it had a very high false negative rate. Trying nu=0.5 values has improved results slightly, but not significantly enough.

Features	F1 score	ROC AUC	TN / FP / FN / TP	Parameters
KAZE	0.4421	0.4961	9 / 4 / 49 / 21	gamma=0.01, nu=0.1
KAZE	0.6164	0.8244	6 / 7 / 16 / 54	gamma=0.1, nu=0.1
Autoencoder	0.9790	0.8846	10 / 3 / 0 / 70	gamma=0.01, nu=0.1
VGG16	0.9271	0.5769	2 / 11 / 0 / 70	gamma=0.01, nu=0.1
InceptionV3	0.9523	0.7307	6 / 7 / 0 / 70	gamma=0.01, nu=0.1
Xception	0.9589	0.7692	6 / 7 / 2 / 68	gamma=0.01, nu=0.1

Table 6.7: C1 dataset results with RBF kernel

Features	F1 score	ROC AUC	TN / FP / FN / TP	Parameters
KAZE	0.3736	0.4675	9 / 4 / 53 / 17	nu=0.1
KAZE	0.7394	0.6219	8 / 5 / 26 / 44	nu=0.5
Autoencoder	0.3023	0.4774	10 / 3 / 57 / 13	nu=0.1
VGG16	0.7868	0.6890	9 / 4 / 22 / 48	nu=0.1
InceptionV3	0.5904	0.5675	9 / 4 / 39 / 31	nu=0.1
Xception	0.7079	0.6703	10 / 3 / 30 / 40	nu=0.1

Table 6.8: C1 dataset results with LINEAR kernel

### 6.2.6 Autoencoder

Here are the results for the simple autoencoder approach of anomaly detection based on the reconstruction loss (mean squared error). Anomaly score here is defined as explained in section 5.3.1 and is based on the reconstruction loss (mean squared error) between test image and image that goes through autoencoder.

The threshold anomaly value has been selected following the described approach in section 5.3.4. It involves testing the model on the training data, selecting the biggest anomaly score (in this case - mean squared error), adding 5% margin and classifying an item as an anomaly if its anomaly score is higher than this threshold.

However, because these values (and classification as well) are based on the threshold that is selected based on observations of the training, it is important to also look at how these values are distributed (for the test set). The loss (Mean squared error) distribution is presented in the boxplot (Figure 6.5).

Precision	Recall	F1	TN	FP	FN	TP
0.8852	0.9642	0.9230	13	7	2	54

Table 6.9: Autoencoder anomaly detection results

As we see from the results table 6.9 this approach achieves good results on the dataset B4 with F1 score-metric as well as recall, which is an important metric in this project scenario, as the cost of False Negative is the biggest.

		True values	
		Fail	Pass
Predicted	Fail	True Positive 54	False Positive 7
	Pass	False Negative 2	True Negative 13

Figure 6.4: Autoencoder's anomaly detection confusion matrix

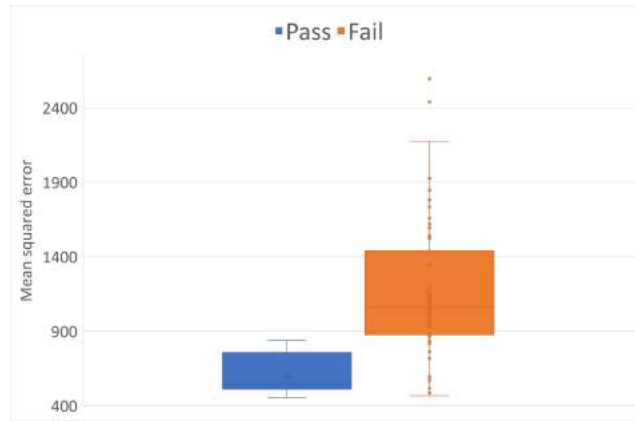


Figure 6.5: Autoencoder's loss distribution

### Training

Training an autoencoder is relatively fast and easy. On the Nvidia Tesla K80 GPU, one epoch takes only 10 seconds to complete. During this training, we have trained the model for 100 epochs. Only augmented images from few healthy images were used for training, while other images of positive, "healthy" class were used for validation. In the Figure 6.6 training loss history is presented.

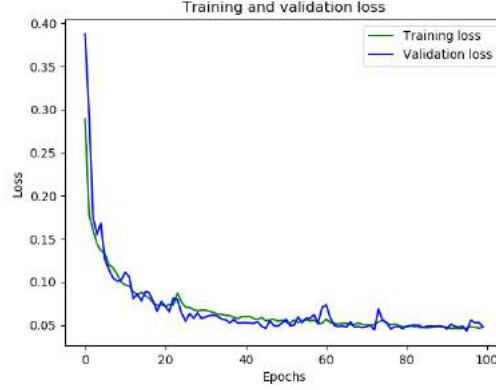


Figure 6.6: Autoencoder's training loss history

### Defect localization

Images after passing the encoder-decoder pipeline in the autoencoder come out pretty blurry (as seen in - 6.7). Nevertheless, the positional orientation and brightness levels are kept and the sealant that is placed correctly is drawn (because autoencoder has been trained only on healthy samples and decoder has learned only weights corresponding to correctly put sealant).



Figure 6.7: Autoencoder's input, output and defect localization, respectively

As we see, the images with defects (anomalies) have that place marked in a red color. It shows the missing sealant and / or place where sealant has been put too much. However, some noise is also detectable - in many places algorithm has marked pixels as being anomalous that are not in an "area of interest" (which is the area of where a sealant is put). This issue could be tackled by applying a mask that would define the "area of interest" and ignore anomalous pixels elsewhere.

### 6.2.7 Generative adversarial networks

Training of the GAN's has been a difficult task because of an unstable training and hard to interpret losses. As noticed during the experiments, this was especially true while training DCGAN, even after applying all the stability tricks recommended among other papers [9], [29], [36]. During the training, the DCGAN network would always fall into partial mode collapse and would not be able to generalize to images of different brightness and / or positional parameters.

This problem has been overcome with the improved WGAN, which would always show affinity to

converge and was not sensitive to hyper-parameters like a number of features in the convolutional layers or optimisers.

### WGAN training

In the Figure 6.9 generated images  $G(z)$  at a different timing (no. of epochs) are shown. As we see, network learns to generate healthy looking images without mode collapse (as there are positional and brightness differences among the pictures) .

In the figure 6.8 we plot the training loss of these two modules. An important note is that we train the discriminator 6 times more than the generator, hence the difference in the number of the iterations made. We can see that both losses tend to converge to the similar values. In the case of a generator, where loss often is negative (because it is a WGAN), the loss converges to have small values around 0. However, the image quality does increase even when this balance has been reached in 400 iterations. The discriminator loss starts small but quickly jumps to be very high, and with time it tends to decrease towards 0.

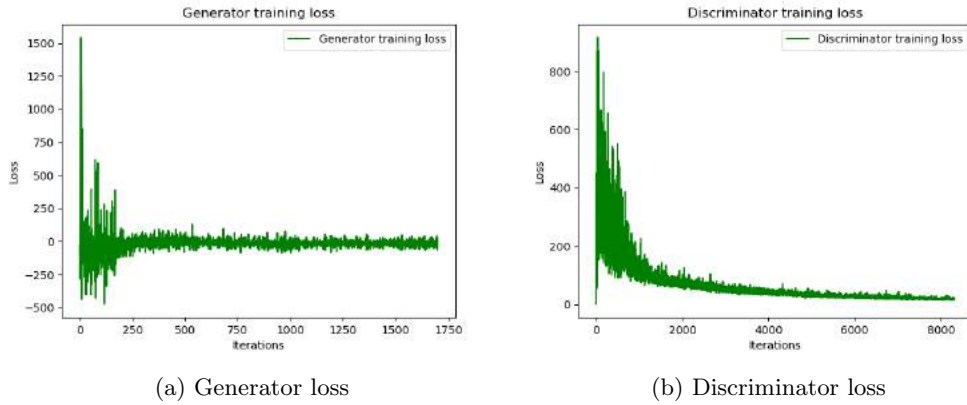


Figure 6.8: WGAN's generator and discriminator losses

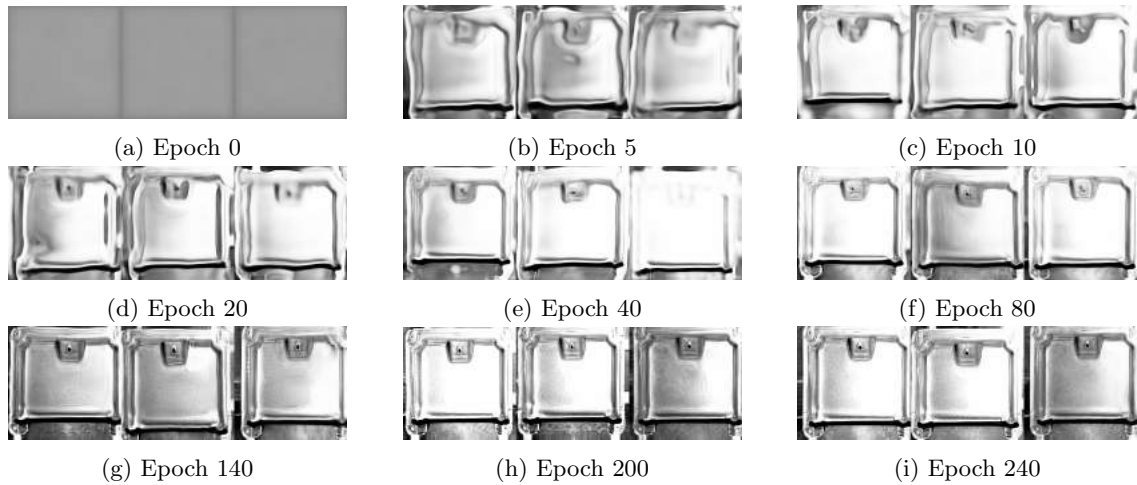


Figure 6.9: Examples of  $G(z)$  (generated) images during training

### Anomaly scores

As we can see in the Figure 6.10 the GAN approach also performs good, for this dataset it achieves the best result with F1 score of **0.95** when using only the discriminative loss as Anomaly score  $A(x)$ . Using combined (discriminative and residual losses with  $\lambda = 0.1$ ), F1 score is also high - **0.92**. In the Figure 6.10 we can see two confusion matrices that have been built by using different anomaly scores. In a) the discrimination loss has been used as a metric for defining Anomaly score  $A(x)$ , while in b) combination of discrimination and reconstruction (mean squared error) based.

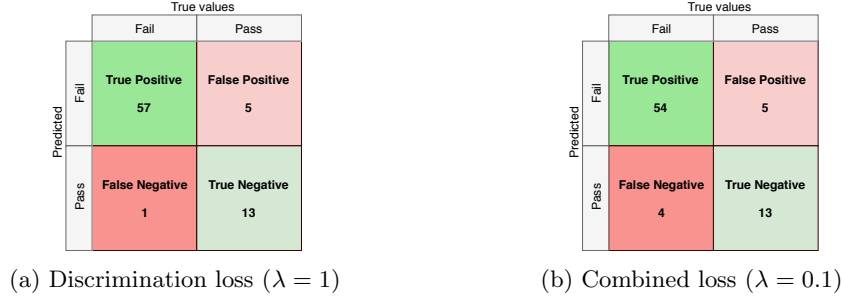


Figure 6.10: WGAN's anomaly detection confusion matrices

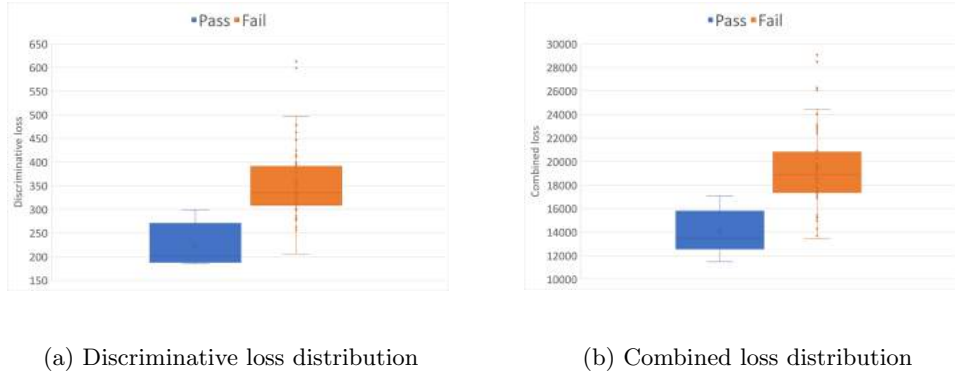


Figure 6.11: WGAN's anomaly detection loss distribution

### Defect localization

Using the approach described in 5.3.4 we present the results of defect localization by comparing the test image and the generated image after the latent variable  $z$  such that  $G(z)$  has minimal loss compared with the test image, has been found. In the Figure 6.12 few examples of images tested are shown. Each block in Figure 6.12 consists of three images - a testing image, a generated image  $G(z)$  and an image with defect localization, which is obtained by taking the mean absolute error between the images, as described in 5.3.4.





Figure 6.12: WGAN (from left to right): Test image, Generated image  $G(z)$ , defect localization

# Chapter 7

## Discussion

### 7.1 Results

In this chapter, we discuss the results obtained for each of the approaches tried. We evaluate the approach in the terms of performance, speed, amount of training data needed, robustness to environmental changes and its ability to localize the defect in the image.

#### 7.1.1 Supervised classification

Building a model in a supervised manner in the context of this project is effective, but not very well suited. Mainly, because it requires a lot of labelled training data. Even with transfer learning techniques applied, so that the model does not need to train itself from scratch, still needs labelled data. Moreover, there are no guarantees that it will perform when new types of anomalies appear. Training it does not require much time, assuming transfer learning has been used as it achieves good accuracy within few epochs.

The approach does provide out of the box defect localization with class activation maps or saliency maps, however the output in the experiments done in this project did not provide very meaningful results (Figures 6.1a, 6.1b, 6.1c, 6.1d).

#### 7.1.2 Feature extraction and OCSVM

Extracting features from the images and training a classifier (or novelty detection model like OCSVM) has proved to be an effective approach. It performed well on two out of three datasets (however the performance on the dataset B1 should be taken with a grain of salt because of the not correctly labelled data which confuses the model). It can detect most of the anomalies and only the anomalies that are very small get undetected, and that could be probably solved with an increased resolution of the images.

In this approach, if we look at the result tables of the linear kernel (6.4, 6.6, 6.8,) and compare the results there with the results of RBF kernel (6.3, 6.5, 6.7) we see that only the RBF kernel has shown good performance and linear kernel of the OCSVM has failed to do so. It is also a logical outcome, as the features extracted from the images are rather complex and not linearly separable. Moreover, it has also clearly demonstrated that features extracted with deep learning methods (convolutional layers) significantly outperform classical feature extraction algorithms like KAZE. It can be seen by comparing the results of the feature extractor KAZE and others, based on the deep learning.

The training of the model does not take long and very depends on the number of features fed into the OCSVM. Testing whether an image contains anomalies or not is done almost instantaneously, within milliseconds.

This approach does require a lot of training data - at least 500 images, because training model on

less data has produced bad results. However, these images are augmented from only a few positive samples, so from an application point of view it does not require much initial training data as data augmentation helps to build bigger training sets.

The model is pretty robust because of the training on the augmented images.

However, the approach has a drawback of not being able to localize the defect. The hyperplane can be obtained and features contributing for an image being classified as an anomaly or not can be obtained. Then this features backpropagated and class activation map could be built. But the problem is that it can be obtained using the linear kernel and not the others, like RBF. Considering that the approach performed well only with RBF kernel, we can conclude that the model does not provide a way to localize the defects.

### 7.1.3 Autoencoder

This approach has proved to be the best one in all the metrics. The performance is high and comparable with OCSVM approach performance with RBF kernel. It is rather simple (simple reconstruction loss metric) and fast to train as well as to test an image. Moreover, it also does not require much initial training data as augmented images work well to train the model. By encoding-decoding a test image and then taking the difference and making residual image provides a way to have a detailed defect localization. Moreover, the model is robust and learns to correctly encode-decode images in a changing environment and fails to encode-decode properly images with anomalous sealant. However, the drawback of this approach is that the decoded images look blurry (so it is harder to build residual image and detect smaller anomalies).

### 7.1.4 Generative adversarial network

This approach is more complex than the previous ones and harder to train. We have tried different techniques, architectures and losses. GAN training takes also a very long time to train - it takes 12hours to (almost) reach the Nash equilibrium (which is when losses of both generator and discriminator converge to some small value range and the network stops learning).

Speaking about the performance of this approach in the context of anomaly detection, we can say that it definitely can compare with above-mentioned approaches and perform better in some cases than OCSVM. It also depends greatly on how we calculate the difference (loss) between test and generated image. Also looking at the values of different metrics, we can see that sometimes one metric works well and in other cases another one, so some kind of ensemble approach here probably could increase the performance even more. The approach is very general in the sense that it easily deals with new, not seen anomalies and is robust to changes in the environment.

It does require at least 500 images to train on (training of smaller datasets showed worse results), but that is not the problem given that the data augmentation is being done.

Defect localization here is rather complicated. Even if it does correctly label image being an anomaly, the defect localization fails in some cases. This is mostly the case where the defect is that the sealant has not been put in some place where it should have. On the contrary, it always works well with defects where a sealant has been misplaced, is too thick or has some structural deviations (sealant is where it should not be). This is probably because the model has not reached true Nash equilibrium (or has not been trained completely), as the model is able to find a latent variable  $z$  from which generated image  $G(z)$  looks anomalous itself (for example, without a sealant).

Another problem with the approach is that it takes a long time to find the latent variable  $z$ , in this case - more than a minute for a test image. It would not be suitable in the production environment, because visual inspection systems should detect the defective products (almost) instantly, as the product immediately goes to the next manufacturing step on the assembly line. However, there are alternative architectures of such GAN's that tackle this problem by learning inverse mapping

during the training of the network [13], [14].

Generally speaking, this approach is powerful and can be used in anomaly detection. However the scope of the problems in which this approach is suitable is rather limited. Anomaly detection with GAN's performs very well when there is high (unseen) variability among the healthy data while the amount of training data is limited and can not be compensated with image augmentation. Moreover, it works well when we want to detect something that is there when it should not be and not vice versa. For example, when building a model for anomaly detection of brain tumours in brain MRI scans. After training a model on a bunch of healthy MRI brain scans, the generator could easily produce images of brain scans that are artificial but look real. The point here is that every brain scan will be different to some extent as every person have slightly different anatomy. Whereas anomaly detection of rather static products in the manufacturing regards always almost the same looking images and has only meta-differences, like brightness / rotation / etc. changes. That means that the main advantage of GAN's is not exploited enough in this setting, while the complexity and training time of this approach are high.

## 7.2 Conclusions

Visual anomaly detection problem has many different approaches and each approach has it's particularities. From the results we conclude that using deep learning techniques in anomaly detection performs well and better than the approaches that do not, because convolutional neural networks can extract meaningful representations from an image.

The best approach for this context we have found to be the autoencoder, because of its robustness, speed, simplicity and ability to do the defect localization, all in an unsupervised manner.

Using deep learning and data augmentation we enable the robustness to the extent where models really learn the important features and ignore the ones that describe rotations, colours, position, etc. However, the solutions are still not perfect and there is space for improvements in all - better feature extraction, generation of images and detecting the anomalies, as well as defect localization.

Another thing to mention is that this problem is a very context specific. That means that the approach that works for this particular data can not be suitable for the different kind of data. It is very important to understand what variability exists in the healthy data and what kind of anomalies one would expect.

# Chapter 8

## Future work

There are several approaches more or additions to the existing approach that should be considered to try for solving this problem.

### 8.1 Improvements of researched approaches

While the GAN approach does produce results comparable to other approaches, finding the latent variable  $z$  in the latent space  $Z$  is a long process, taking up to one minute for higher resolution images. This is because we need to fit the model to find the variable  $z$  that produces image  $G(x)$  most similar to  $x$ . However, there exists few architectures of GAN's that do take this into account and learn an approximate mapping  $x \rightarrow z$  during its training. In this way, given a test image  $x$  a latent variable  $z$  can be found in a very fast and efficient way with only very small changes to the predicted variable needed. In practice, it reduces time to find this variable  $z$  from seconds to milliseconds and in a rapid production environment this timing is essential.

One approach to learn this inverse mapping as a training process is to use a variational autoencoder in the generator part of the GAN, as is described in [23]. J. Donahue et. al in 2017 have introduced a variation of GAN called BiGAN [13]. In this network, in addition to standard GAN framework, authors include another neural network called *encoder* which learns to map data to their latent representations  $z$ . In this way discriminator is trained to distinguish not only between  $x$  and  $G(z)$ , but between tuples  $(G(z), z)$  and  $(x, E(x))$ . After training has finished, the encoder can map new images to their latent representation  $z$ . A very similar approach called AliGAN has been introduced in [14].

### 8.2 Other approaches

Firstly, supervised classification approach has shown to be able to produce very good results if there is enough labelled data to train from. So the first thing to consider is if it is possible to produce such labelled data, and if yes - train models on that data. For example, in this project we have augmented positive examples. So it would be easy to produce even thousands of healthy samples. It would be interesting to find a way to produce some images of products with defects - by, for example, automatically in-painting in the image the sealant that would be too thick/thin, of a different shape, in a wrong place or not full.

Another approach worth attention is done through segmentation. Building a segmentation model that could detect and mark the sealant put, then applying template matching or any other anomaly detection technique should prove to work very good, as no other parts of the image that are unimportant (could be considered as noise) would interfere with the prediction. However, using unsupervised or transfer-learning approaches for segmentation models would not work in every

case given datasets of this project, as some of them have shadows that fall over the sealant area, and, thus, the model would segment parts of the shadow as a sealant. But, there exists supervised segmentation approaches, in which model could learn how sealant looks like and segment it in different lighting and other environmental conditions correctly. But this approach again, would need labelled dataset with sealant area correctly segmented already.

# Bibliography

- [1] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. *SIGMOD Rec.*, 30(2):37–46, May 2001. 2
- [2] Andrew P. Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, and Wenzhe Shi. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize. *CoRR*, abs/1707.02937, 2017. 19
- [3] Pablo Fernandez Alcantarilla, Adrien Bartoli, and Andrew J. Davison. Kaze features. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV’12, pages 214–227, Berlin, Heidelberg, 2012. Springer-Verlag. 12
- [4] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017. 18, 24
- [5] Ramkumar B, Ravi S. Hegde, Rob Laber, and Hristo Bojinov. GPGPU acceleration of the KAZE image feature extraction algorithm. *CoRR*, abs/1706.06750, 2017. 12
- [6] Homayoon Beigi. Automation in manufacturing. 02 1997. 1
- [7] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Volume 2 - Volume 02*, CVPR ’05, pages 60–65, Washington, DC, USA, 2005. IEEE Computer Society. 26
- [8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009. 1
- [9] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to train a gan? tips and tricks to make gans work. 17, 38
- [10] Maria J. M. Chuquicusma, Sarfaraz Hussein, Jeremy Burt, and Ulas Bagci. How to fool radiologists with generative adversarial networks? A visual turing test for lung cancer diagnosis. *CoRR*, abs/1710.09762, 2017. 23
- [11] Lucas Deecke, Robert Vandermeulen, Lukas Ruff, Stephan Mandt, and Marius Kloft. Anomaly detection with generative adversarial networks, 2018. 23, 25
- [12] Arden Dertat. Applied deep learning - part 3: Autoencoders. 12
- [13] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *CoRR*, abs/1605.09782, 2016. 44, 45
- [14] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Massotripietro, and Aaron C. Courville. Adversarially learned inference. *CoRR*, abs/1606.00704, 2016. 44, 45
- [15] Hamid Eghbal-zadeh and Gerhard Widmer. Likelihood estimation for generative adversarial networks. *CoRR*, abs/1707.07530, 2017. 17

- [16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014. 15
- [17] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. 18, 19
- [18] Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, Oct 2004. 1
- [19] Elvis Hozdić. Smart factory for industry 4.0: A review. 7:28–35, 01 2015. 1
- [20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 15
- [21] Hairulliza Judi, Ruzzakiah Jenal, and Devendran Genasan. Quality control implementation in manufacturing companies: Motivating factors and challenges, 04 2011. 1
- [22] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. 11
- [23] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015. 45
- [24] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. 16
- [25] Tony Lindeberg. Scale-space representation: Definition and basic ideas. 13
- [26] Beata Mrugalska and Edwin Tytyk. Quality control methods for product reliability and safety. *Procedia Manufacturing*, 3:2730 – 2737, 2015. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015. 1
- [27] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. vi, 19
- [28] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448 – 3470, 2007. 2
- [29] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 2, 16, 24, 25, 38
- [30] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. 17, 18, 24
- [31] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *CoRR*, abs/1703.05921, 2017. 2, 23, 24, 25, 26
- [32] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS’99*, pages 582–588, Cambridge, MA, USA, 1999. MIT Press. 14, 15
- [33] S. A. K. Tareen and Z. Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10, March 2018. 12



- [34] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. 22
- [35] Tom White. Sampling generative networks: Notes on a few effective techniques. *CoRR*, abs/1609.04468, 2016. 17
- [36] Sitao Xiang and Hao Li. On the effect of batch normalization and weight normalization in generative adversarial networks. *CoRR*, abs/1704.03971, 2017. 16, 38
- [37] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. 10
- [38] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *CoRR*, abs/1802.06222, 2018. 2, 26