# Corrosion Detection with Computer Vision and Deep Learning

Achilleas Matthaiou

**Diploma Thesis**


School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Assistant Prof. George Papalambrou

Committee Member : Prof. M. S. Samuelidis
Committee Member : Prof. G. Zarafonitis

2021 March

# Acknowledgements

This thesis is the final requirement for the completion of my studies at the National Technical University of Athens (NTUA) School of Naval Architecture and Marine Engineering (SNAME). I would like to thank my supervisor Assistant Professor George Papalambrou whose support and guidance were instrumental to the completion of this study. Through his courses and later with our cooperation for this study, he introduced me to the field of artificial intelligence and inspired me to follow it as a carrier path. I would also like to thank Professor Manolis Samuelidis for his continued support and collaboration for the purposes of this study. I express my gratitude to Professor Panagiota Vasiliou from the School of Chemical Engineering for her instrumental help in the creation of the datasets used by this study. A thank you to Mr. Ritsakis and KA-ENERGY Shipbuilding Studies for providing images of bulk carriers' inspections. Finally, I thank my family for their continued support throughout the years my of studies. They are always by my side helping me achieve my goals and supporting me to be the best version of myself.

# Abstract

This study investigates and test solutions for automated corrosion detection processes that focus on the visual attributes of corrosion. Artificial intelligence methods are employed to extract information from images. Corroded surfaces have two visually identified attributes color and texture. To detect corrosion based on color, a color tracking algorithm is created and tested using images from different compartments of vessels. To detect corrosion based on texture, deep learning algorithms are used, and two approaches are tested. The first approach is a binary classification model trained using a Convolutional Neural Network (CNN) architecture employing transfer learning. The model is also used by a sliding algorithm to allow detection and localization in large corroded plates. The second approach treats corrosion detection as an object detection problem. A Single Shot Detector (SSD) is trained using transfer learning to detect corrosion on real world images. To support training and testing of all models two datasets are created. The first dataset consists of images of metals corroded in a laboratory environment, while the second dataset from real world images of corroded compartments from bulk carriers' inspections. The study finds all three methods capable to perform corrosion detection with the deep learning approaches yielding better results. Comparing the two deep learning approaches, object detection is found to be more suitable for real world examples.

# Contents

**1 Introduction** **9**

**2 Literature Review** **11**

**3 Theoretical Background** **13**

3.1 Computer Vision . . . . . 14

3.1.1 Digital Image . . . . . 14

3.1.2 Image Filtering . . . . . 17

3.1.3 Open CV Library . . . . . 18

3.2 Deep Learning . . . . . 19

3.2.1 Artificial Non-Linear Neurons . . . . . 19

3.2.2 Neural Network Architecture . . . . . 22

3.2.3 Neural Network Training . . . . . 22

3.2.4 Optimization Algorithms . . . . . 25

3.2.5 Neural Network's Performance Evaluation . . . . . 27

3.2.6 Neural Networks and Deep Learning . . . . . 30

3.2.7 Convolutional Neural Networks . . . . . 31

3.2.8 Convolutional Neural Network Architectures . . . . . 34

3.2.9 Transfer Learning . . . . . 38

3.2.10 Object Detection . . . . . 39

3.2.11 TensorFlow Deep Learning Framework . . . . . 41

**4 Corrosion Datasets** **42**

4.1 GUI Application . . . . . 42

4.2 Laboratory Dataset . . . . . 42
4.3 Surveys Dataset . . . . . 47

**5 Methodology** **50**
5.1 Color Tracking for Corrosion Detection . . . . . 50
5.2 Deep Convolutional Neural Networks for Corrosion Detection 53
5.3 Corrosion Detection - Object Detection Approach . . . . . 55

**6 Results** **57**
6.1 Corrosion Color Tracking Results . . . . . 57
6.2 Corrosion Detection with Deep Convolutional Neural Networks Results . . . . . 67
6.2.1 Corrosion detection model results . . . . . 67
6.2.2 Model's deployment with sliding algorithm results . . 75
6.3 Corrosion Detection with Object Detection Algorithm Results 78

**7 Conclusion - Future work** **90**

**Bibliography** **93**

# List of Tables

6.1 Model's summary . . . . . 67
6.2 Model's classification report on test data . . . . . 69

# List of Figures


3.1 Venn diagram of artificial intelligence . . . . . 13
3.2 Image matrix representation . . . . . 14
3.3 Grayscale range intensity . . . . . 15
3.4 RGB image representation and colour space . . . . . 16
3.5 HSV colour space . . . . . 17
3.6 Artificial neural neuron representation . . . . . 20
3.7 Plotted activation functions . . . . . 21
3.8 Feed-Forward Neural Network graph . . . . . 22
3.9 Error function represantation as a surface of the weight space 24
3.10 Learning rate values movements on the error function . . . . 26
3.11 AdaGrad versus Gradient decent movements . . . . . 27
3.12 Total Error and model complexity relationship for the MSE function . . . . . 28
3.13 Bias-variance tradeoff . . . . . 29
3.14 Bull eye diagram . . . . . 29
3.15 Machine and deep learning performance with regards to the amount of data . . . . . 30
3.16 Connections between layers with zero padding . . . . . 32
3.17 Dimensionality reduction . . . . . 32
3.18 Convolutional layers with multiple feature maps and RGB input image . . . . . 33
3.19 MaxPooling with a 2x2 kernel and stride 2 . . . . . 34
3.20 Typical CNN architecture . . . . . 35
3.21 AleNet architecture . . . . . 36

3.22 VGGNet architecture . . . . . 37
3.23 Residual Learning . . . . . 37
3.24 ResNet architecture . . . . . 38
3.25 Transfer learning strategies . . . . . 39
3.26 Single Shot Multibox Detector example with MobileNet . . . 40
3.27 TensorFlow dataflow graph . . . . . 41

4.1 Gui application layout . . . . . 43
4.2 Laboratory dataset sample of original images . . . . . 43
4.3 Laboratory dataset sample of dropped images . . . . . 44
4.4 Laboratory dataset sample of Rust class . . . . . 45
4.5 Laboratory dataset sample of NoRust class . . . . . 46
4.6 Sample of images provided by KA-ENERGY . . . . . 47
4.7 Surveys dataset sample of Corrosion class . . . . . 48
4.8 LabelImg tool . . . . . 49

5.1 Data flow diagram of color tracking method . . . . . 52
5.2 Data flow diagram of the method employing CNN architecture 54

6.1 Color tracking results image 1 - Fuel Oil Tank . . . . . 58
6.2 Color tracking results image 2 - Fuel Oil Tank . . . . . 59
6.3 Color tracking results image 3 - Fuel Oil Tank . . . . . 59
6.4 Color tracking results image 4 - Fuel Oil Tank . . . . . 60
6.5 Color tracking results image 5 - Ballast Tank . . . . . 60
6.6 Color tracking results image 6 - Ballast Tank . . . . . 61
6.7 Color tracking results image 7 - Ballast Tank . . . . . 61
6.8 Color tracking results image 8 - Ballast Tank . . . . . 62
6.9 Color tracking results image 9 - Cargo Hold . . . . . 63
6.10 Color tracking results image 10 - Cargo Hold . . . . . 64
6.11 Color tracking results image 11 - Cargo Hold . . . . . 65
6.12 Color tracking results image 12 - Cargo Hold . . . . . 66
6.13 Model's accuracy and loss graphs . . . . . 68
6.14 Model's confusion matrix on test data . . . . . 70

6.15 Model's AUC - ROC curve on test data . . . . . 71
6.16 Images used in feature maps visualization . . . . . 71
6.17 ResNet50 - 2nd layer's 64 feature maps . . . . . 72
6.18 ResNet50 - 7th layer's 64 feature maps . . . . . 73
6.19 ResNet50 - 10th layer's 64 feature maps . . . . . 74
6.20 Sliding algorithm result 1 . . . . . 75
6.21 Sliding algorithm result 2 . . . . . 76
6.22 Sliding algorithm result 3 . . . . . 77
6.23 Loss graph of SSD Mobilenet v1 . . . . . 79
6.24 SSD Mobilenet v1 result 1 . . . . . 80
6.25 SSD Mobilenet v1 result 2 . . . . . 81
6.26 SSD Mobilenet v1 result 3 . . . . . 82
6.27 SSD Mobilenet v1 result 4 . . . . . 83
6.28 SSD Mobilenet v1 result 5 . . . . . 84
6.29 SSD Mobilenet v1 result 6 . . . . . 85
6.30 SSD Mobilenet v1 result 7 . . . . . 86
6.31 SSD Mobilenet v1 result 8 . . . . . 87
6.32 SSD Mobilenet v1 result 9 . . . . . 88
6.33 SSD Mobilenet v1 result 10 . . . . . 89

# Chapter 1

# Introduction

The word corrosion is derived by the Latin word corrodere which means "to wear away gradually". Science defines corrosion as the electrochemical reaction of a metal or an alloy with its environment [1]. It constitutes one of the two main causes of metal deterioration alongside mechanical loss by erosion, abrasion, or wear. All metal structures are prone to corrosion. Sometimes it can be prevented but economically it makes more sense to control it. Therefore, engineers choose materials based on the assumption that the structure will be maintained during its life cycle.

In the shipping industry maintenance of ships is regulated through the international maritime treaty SOLAS (International Convention for the Safety of Life at Sea) and individual class rules. Ships are subject to a through-life survey regime to remain class approved. There are class surveys, intermediate surveys, annual surveys, and bottom/docking surveys of the hull that indicate the needed maintenance of the ship's structure. Surveys can be costly and time-consuming processes exposed to human error.

In recent years, the prospect of automated inspections is gaining ground. Advancements in drone technology and Artificial Intelligence promise solutions where drones will inspect ship structures more efficiently for a fraction of the time and cost that currently are needed. These practices meet classification societies and owners' vision for the future of inspections. A drone flying in the ship mounted with a camera identifying structural issues such as corrosion and reporting them is closer to the present than ever.

The purpose of this study is the detection of corrosion in images. This is implemented using algorithms that quantify the two main visual characteristics of corrosion, color and roughness/texture. Three different methods are tested. The first method employs computer vision algorithms to detect corrosion's color. The other methods utilize deep supervised learning techniques, with one of them treating detection as a binary classification

problem and the other as an object detection one.

In the following chapters, an overview of the literature is presented. Key theoretical concepts regarding computer vision and deep learning are discussed and defined. A presentation of the datasets that were created to support the algorithms that are utilized. The methodology is explained and discussed. Results are presented including visual examples of what the study achieved. Finally, the conclusion of this study with proposals for future projects.

# Chapter 2

# Literature Review

Studies regarding visual corrosion detection used to focus mainly on computer vision and machine learning approaches that were tasked with identifying the key characteristics of corrosion in digital images. But breakthroughs in the field of deep learning refocused the interest of researchers to deep learning approaches for corrosion detection. In this section previously published works on automated visual corrosion detection from digital images are presented and discussed.

B. B. Zaidan, A. A. Zaidan, Alanazi, and Alnaqeib (2010) [2] proposed a texture analysis method to detect corrosion on metals. Using a standard deviation filter their method focused on the rough texture of the corroded area, and identified the simple texture as not corroded. Bonnin and Ortiz (2014) [3] used vision-based corrosion detection algorithms developed on the European project MINOAS. The algorithms were a weak classifier color-based corrosion detector (WCCD) and an AdaBoost based corrosion detector (ABCD), which both combine weak classifiers to achieve good performance. The first algorithm is compromised of two stages with the first using the energy of the symmetric gray level co-occurrence matrix to extract pixels with rough texture and the second classifying extracted pixels based on their color information. The second algorithm uses the Adaptive Boosting paradigm to classify corroded areas based on their texture using decision trees as weak classifiers. Then stage two of the WCCD is used to classify the extracted pixels. The authors tested the algorithms using images from vessels and concluded that WCCD outperforms ABCD. Petricca and Moss (2016) [4] compared standard computer vision and deep learning techniques. They created a color tracking application and retrained a model called bvlc-reference-caffenet employing transfer learning to detect corrosion. Their applications were tested using data from various sources and concluded the deep learning approach yields greater results. Bondada, Pratihar, and Kumar (2018) [5] worked on a computer vision approach for detection and

quantitative assessment of corrosion on pipelines. Their method detected corrosion based on the saturation of its color using various threshold values and identified the most damaged regions using the k-means clustering algorithm. Gibbons, Pierce and their colleagues (2018) [6] created a probabilistic process for detecting corrosion in remanufacturing. A Gaussian mixture model was trained using images of used parts achieving a 85% accuracy on the test set.

Following works focused mostly on deep learning approaches. Atha and Jahanshahi (2017) [7] evaluated the use of CNNs for corrosion detection. They tested for different color space input images and multiple architectures. Evaluating both proposed and well-known architectures of the ImageNet, VGG16 architecture with RGB or YCbCr input images was found to perform better. Hoskere and Narazaki (2017) [8] worked on vision-based structural inspection, utilizing data from inspections of civil infrastructure, that included corrosion. They employed two CNNs one to classify the image as damaged or not and the other to classify the type of damage. The study yielded great results achieving 86.7% accuracy across the 7 classes of damages, for whom they trained their model. Yao and Yang (2019) [9] studied an AI based hull structural plate corrosion damage detection approach employing a CCN based on the AlexNet architecture achieving an accuracy of 98.9%.Bastian, Jaspreeth and their collegues (2019) [10] proposed an inhouse CNN architecture combined with a localization algorithm that achieved 98.8% accuracy for detecting corrosion on pipelines. The CNN was trained to classify between 4 classes (no, low-level, medium-level, and high-level corrosion). Finally, Nash and Powell (2020) [11] created a crowd sourced web site, where users labeled and uploaded images with whom an original CNN model was retrained. This led to an increase of the initial 66% accuracy of the model to 93% after 37 days.

This study will test both computer vision and deep learning approaches. First, the accuracy of a color-based approach on images from vessels employing standard computer vision techniques will be explored. Then a texture-based approach employing a CNN for a binary classification of corroded and clean metals. Finally, another texture-based approach treating the corroded regions found on images from vessels' compartments, as objects that need to be detected and located.

# Chapter 3

# Theoretical Background

Alan Turin in his paper "Computing Machinery and Intelligence" (1950) set the foundation of what we call today Artificial Intelligence (AI). The term can be described as the field of computer science that answers the Turin test question "Can a machine exhibit intelligent behavior equivalent to that of a human?". There is not a clear definition of the field that is universally accepted. Notable definitions are given by Russel and Norvig in their book "Artificial Intelligence: A Modern Approach", where they define AI as "the study of agents that receive percepts from the environment and perform actions", and by Professor Patrick Winston of MIT, who defines it as "algorithms enabled by constraints, exposed by representations that support models targeted at loops that tie thinking, perception and action together". AI incudes a variety of subcategories, as shown in figure 3.1, including computer vision, machine, and deep learning.

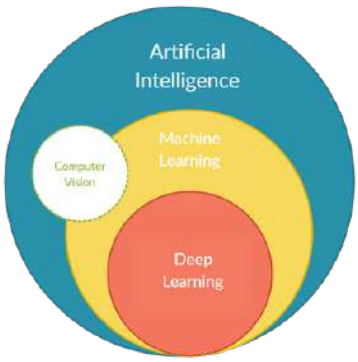



Figure 3.1: Venn diagram of artificial intelligence

In recent years, advancement in Graphics Processing Unit (GPU) technology have provided researchers the computational power needed to

create the one-layer artificial neural network in the 60s, three-layers in the 80s and 50 plus layers today. Deep neural networks promise to solve several issues and have successfully dealt with problems previously regarded to be of the computer vision domain.

This study utilizes relevant research for the purpose of corrosion detection. In this chapter computer vision and deep learning concepts are discussed and defined.

## 3.1 Computer Vision

### 3.1.1 Digital Image

Computer vision is a scientific field concerned with the undertesting of digital images by computers. According to Richard Szeliski [12] "computer vision tries to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions". To do that an image is described as a function [13]:

$$f : R^2 \rightarrow R \tag{3.1}$$

where f(x,y) is the intensity of position (x,y). Computers operate on digital (discrete) images that sample the 2D space of regular images on a grid and quantize (round to the nearest integer) each sample. The equation expressing this process is:

$$f[i,j] = Quantize f(i\Delta, j\Delta) \tag{3.2}$$

where $\Delta$ is the distance of samples. This way an image is described as a matrix of integers, as shown in figure 3.2.

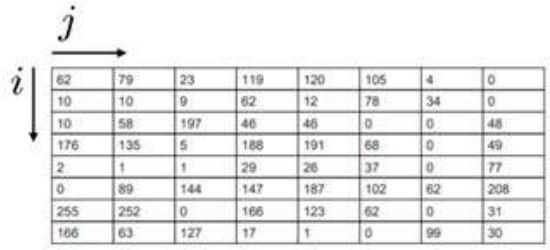



Figure 3.2: Image matrix representation

#### Grayscale Image

The image as described above represents the grayscale image. Its element on the image matrix represents a pixel (short for picture element) and

describes the intensity of gray to be displayed. The range of gray intensity, figure 3.3, has 256 possible values with 0 being black and 255 white.


Figure 3.3: Grayscale range intensity

### Coloured Image and Colour Models

Coloured images include colour information in each one of their pixels. Colour is used to facilitate further processing of images. Depending on the application different colour models are used.

- RGB model:

  RGB representation [12] was standardized by the Commission of Internationale d'Eclairage (CIE) in 1930 after conducting colour matching experiments using the primary colours red, green, and blue. This standardization quantifies the tri-chromatic theory of perception by reproducing all monochromatic colours as a mixture of three chosen primaries, therefore RGB is considered as an "additive" model. It has found applications on colour images on screens such as TVs, computer, and smartphone monitors.

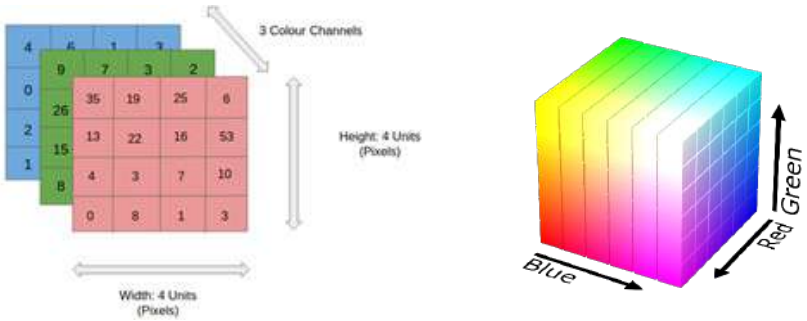
  An RGB image is an $M \times N \times 3$ array of colour pixels, as shown in figure 3.4. Every pixel holds three values (3 channels) from the RGB space corresponding to the red, green, and blue intensity. The RGB space is represented graphically as cube, figure 3.4. Assuming a common colour depth of 24 bits, also known as "true colour", this model provides 256 possible values for red, green, and blue or $256 \times 256 \times 256 = 16777216$ possible colours. Mixing large quantities provides light colours with 100% mixture of all colours creating white light, while small quantities provide dark colours with a 0% mixture created black.

  Colour cameras integrate light according to the spectral response function of its sensors. For RGB the equations describing this process are:

$$R = \int L(\lambda)\, S_R(\lambda)\, d\lambda \tag{3.3}$$

$$G = \int L(\lambda)\, S_G(\lambda)\, d\lambda \tag{3.4}$$

$$B = \int L(\lambda)\, S_B(\lambda)\, d\lambda \tag{3.5}$$




(a) RGB Image (b) RGB cube

Figure 3.4: RGB image representation and colour space

where L($\lambda$) the incoming spectrum of light of a pixel and $S_R, S_G, S_B$ spectral sensitivities of red, green, and blue sensors, respectively.

- HSV model:

  Hue, Saturation, Value colour model (HSV) [12] is a projection of the RGB cube to a non-linear chroma angle, radial saturation percentage and luminance value. Hue defines the colour range from 0 to 360 degrees. Saturations is a measure of dilution by white light with a range [0,1]. Value is the brightness of the colour with a range [0, 1]. The HSV space is represented graphically as cylinder, figure 3.5. It is useful to applications such as colour tracking.

  HSV parameters can be calculated by converting RGB values. The first step to this process is the normalization of R, G, B to [0,1] and calculation of $max = max(R, G, B)$ and $min = min(R, G, B)$. Then the following procedure is applied[14]:

$$1.\ H = \begin{cases} \text{undefined or } 0 & \text{if max=min} \\ 60 \times (0 + \frac{G-B}{max-min}) & \text{if R=max} \\ 60 \times (2 + \frac{B-R}{max-min}) & \text{if G=max} \\ 60 \times (4 + \frac{R-G}{max-min}) & \text{if B=max} \end{cases}$$

$$2.\ if\ (H < 0)\ then\ H = H + 360$$

$$3.\ S = \begin{cases} 0 & \text{if max=0} \\ 100 \times \frac{max-min}{max} & \text{else} \end{cases}$$

$$4.\ V = 100 \times max$$

  Notable conditions are when R=G=B where no Hue value can be assigned and R=G=B= 0 where no saturation can be assigned and is set to zero. It is important to remember that for every HSV colour

there is an RGB equivalent but HSV only describes 1/256th of RGB colours.

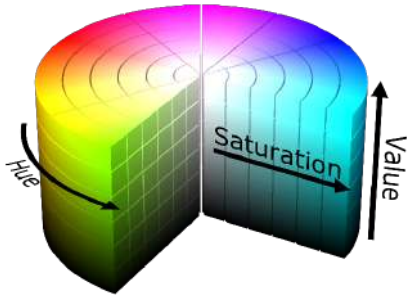



Figure 3.5: HSV colour space

### 3.1.2 Image Filtering

Filters are used to perform a variety of image transformations such as accentuating edges, sharpening details, reducing noise etc. There are two types of filters, linear and non-linear.

#### Linear Filters

Neighbourhood operator[12] is a linear filtering process that outputs a pixel as the weighted sum of input pixel values. This is described by the correlation operator:

$$g = f \bigotimes h \tag{3.6}$$

$$g(i,j) = \sum_{k,l} f(i+k, j+l)\, h(k,l) \tag{3.7}$$

where the matrix h is called kernel or mask and its entries filter coefficients. By reversing the sing of the offsets in f we get:

$$g(i,j) = \sum_{k,l} f(i-k, j-l)\, h(k,l) = \sum_{k,l} f(k,l)\, h(i-k, j-l) \tag{3.8}$$

$$g = f * h \tag{3.9}$$

This gives us the convolution operator where h is called the impulse response function. It provides the addition of shifted impulse response functions $h(i-k, j-l)$ multiplied by the input pixel matrix $f(k,l)$.

Correlation and convolution are linear shift invariant operators meaning that they obey:

- The linear superposition principle, where the net output of two or more operations is the sum of the outputs if every operation is performed individually.
- The shift invariance principle, where shifting the input entries of the vector then the output is shifted by the same amount.

Well known and commonly used linear filters are:

- The moving average or box filter that averages the pixel values in a $K \times K$ frame.
- The Bartlett filter that provides a smoother image. The filter is also known as a piecewise linear "tent" function.
- The Gaussian filter used to reduce noise. A combination of two of those filters can be used for edge detection by subtracting their outputs.

**Non-Linear Filters**

Non-linear filters [12] tend to perform better than linear ones. They follow an alternative approach by thinking the filter as a statistical estimator. Their goal is to estimate the value of a pixel in the presence of noise. Commonly used non-linear filters are the median filter, which selects the median value from each pixel's neighbourhood and the bilateral filter which drops pixels whose values are significantly different than that of the central pixel.

Non-linear filters are also used in the processing of binary images. These images have pixels that output only two colours, usually black and white. They are created by thresholding operations:

$$\theta\left(f,t\right)=\begin{cases}1 & \text{if f} \geq \text{t}\\ 0 & \text{else}\end{cases} \tag{3.10}$$

Binary image operations are called morphological operations as they alter the shape of objects. They take as an input the binary image and a structuring element and convolute them. Then the output is the threshold result of the convolution. Standard operations include dilation, erosion, majority, opening and closing.

### 3.1.3 Open CV Library

OpenCV or Open-Source Computer Vision Library [15] is an open-source computer vision and machine learning library. It provides infrastructure to build computer vision applications with more than 2500 optimized

computer vision and machine learning algorithms. It is developed using C++ and it has C++, Python, Java, and MATLAB interfaces. This study employs OpenCV for the task of corrosion colour tracking and processing of an image dataset.

## 3.2 Deep Learning

Deep learning (deep structured learning) is part of machine learning methods for feature learning based on neural networks.

To discuss deep learning, we first must explain what a neural network (NN) is. NNs are computing systems that try to copy the working process of biological neural networks find in brains of humans and animals. They manage to learn through their interaction with data during a process called training.

There are three main approaches to NN learning. The most common is supervised learning, where the task of learning a function that maps an input to an output is achieved by using input and output pairs (labeled data) for training. Semi-supervised learning that is performed using labeled and unlabeled data simultaneously. Finally, there is unsupervised learning that approaches the task by trying to identify patterns in unlabeled data with minimum human supervision. This study focuses on supervised learning techniques.

NNs solve two main categories of problems, regression, and classification. Regression is the process in which data are mapped into continues real values. Classification is the process of mapping data into multiple discrete values (categorical classes).

Below key concepts of artificial neural networks will be discussed.

### 3.2.1 Artificial Non-Linear Neurons

A neuron is an information-processing unit fundamental to the design of an artificial neural network [16]. A representetion of the neuronal model is shown in figure 3.6. Its basic elements are:

- The synapses (or connection links) that accept the input signal xj and multiply it with their individual weight (or strength) $w_{kj}$.
- The summing junction that adds all signal and weight products.
- The externally applied bias $b_k$ that is added to increase or lower the net input of the activation function.

- The activation function that limits the amplitude of the neuron's output. Typically, the range of the amplitude is normalized to be [0,1] or [-1.1].

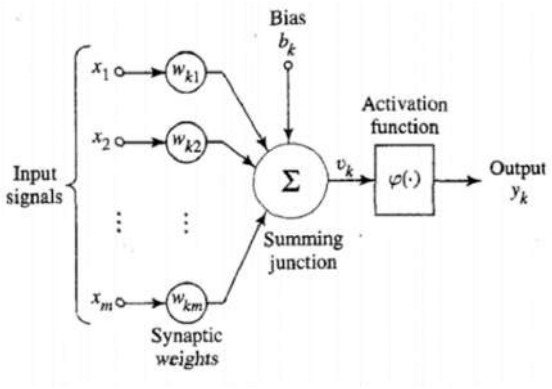



Figure 3.6: Artificial neural neuron representation

The equations describing a neuron are the following:

$$u_k = \sum_{j=1}^{m} w_{kj}\, x_j \tag{3.11}$$

$$y_k = \varphi\left(u_k + b_k\right) \tag{3.12}$$

where $x_j$ the input signals; $w_{kj}$ the weights of neuron k; $u_k$ the linear combination of summing junction; $b_k$ the bias; $\varphi$ the activation function; $y_k$ the output of neuron k. The bias applies an affine transformation $v_k = u_k + b_k$ to the output that increases the activation potential of the neuron.

We may add the synapse $x_0 = 1$ and $w_{k0} = b_k$ to introduce the bias as an internal parameter. So, the equivalent equations are:

$$v_k = \sum_{j=0}^{m} w_{kj}\, x_j \tag{3.13}$$

$$y_k = \varphi\left(v_k\right) \tag{3.14}$$

Activation function $\varphi\left(v_k\right)$ limits the output of the neuron as already mentioned, but it also provides non-linearity to the neuronal model which is needed to solve nontrivial problems. Here we discuss the most common activation functions, and plot them in figure 3.7:

- **Sigmoid Function**: The most widely used activation function that provides great balance between linear and non-linear behaviour. It is

described by the logistic function:

$$\varphi(v) = \frac{1}{1+e^{-\lambda v}} \quad \forall v \in R \rightarrow \varphi \in [0,1] \tag{3.15}$$

where $\lambda \in R$ the slope of the function. The main drawback of the function is the "vanishing gradient". Near the edges of the function the gradient tends to minimize and changes in the Y do not respond to changes in X axis, the neuron stops learning.

- **Tanh Function**: The hyperbolic tangent function is a scaled sigmoid defined by:

$$\varphi(v) = \frac{2}{1+e^{-2v}} - 1 \quad \forall v \in R \rightarrow \varphi \in [-1,1] \tag{3.16}$$

It presents the same useful non-linear properties but has a steeper derivative amplifying the "vanishing gradient" issue.

- **ReLU**: The rectified linear unit (ReLU) is defined by:

$$\varphi(v) = \max(0, v) \quad \forall v \in R \rightarrow \varphi \in [0, \infty) \tag{3.17}$$

It is mainly used in CNNs that will be discussed further below. It converges fast, requires little computational power, and is activated sparsely since its zero for all negative inputs. However, being zero for negative inputs comes with a drawback called "Dying ReLU". If a neuron is stuck to negative values, it always outputs zero. This makes the neuron passive and does not contribute to the learning process. Several variants are found in the bibliography that deal with this issue like Leaky ReLU, Scaled ELU (SELU) and Concatenated ReLU (CReLU).

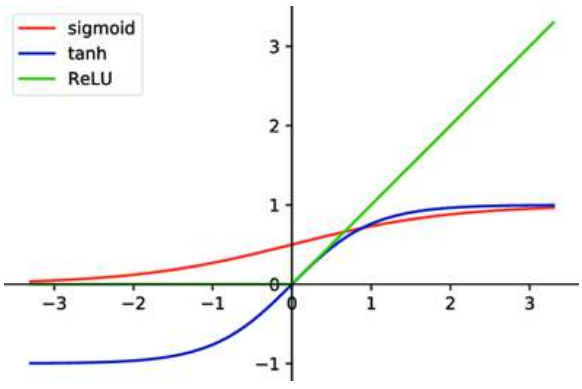



Figure 3.7: Plotted activation functions

Choosing the correct activation function is a complex task that requires experience. The main idea is to try and find one that better approximates the characteristics of the function the neuron is tasked to describe.

### 3.2.2 Neural Network Architecture

After discussing the neuron, we understand the fundamental block of neural networks. NNs can be described as architectural graphs where information is transmitted through neurons towards the output and is used to learn the desired function.

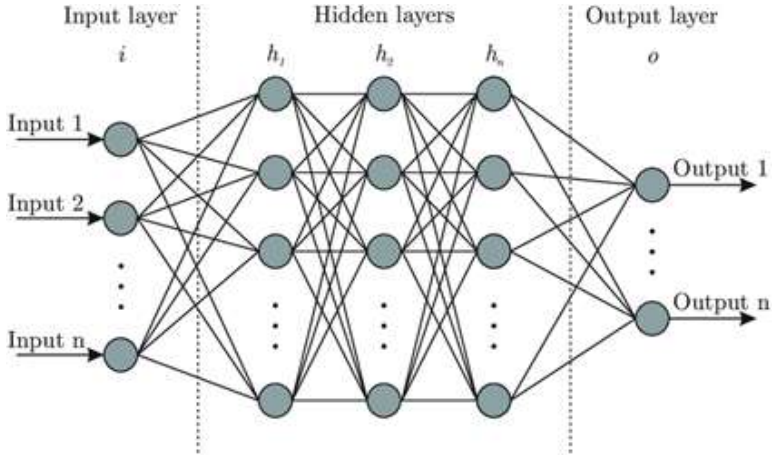



Figure 3.8: Feed-Forward Neural Network graph

Neurons, or computation nodes as they called, are organized in layers [16]. The simplest implantation is the single layer feed-forward network that has an input layer of source nodes that provides data to the output layer of computation nodes. To increase performance, we can add layers between the input and output layers, as shown in figure 3.8. They are called hidden layers or hidden units and enable the network to extract higher order statistics, an attribute especially valuable when the size of the input layer is large. They sequentially process the input signal and project onto the next layer until it reaches the output layer. NNs that their layers nodes are connected to every node in the adjacent and forward layer are called fully connected. If some links are missing, then they are called partially connected.

There are many variations to the feed forward neural network. One of them is the recurrent neural network (RNN). It has at least one feedback loop that provides extra input to a layer's neurons the output of a forward layer. This study focuses on another variant the Convolutional Neural Network (CNN) that will be discussed in depth in later sections.

### 3.2.3 Neural Network Training

So far, we have described NNs as a series of functional transformations where parametric non-linear functions provide an output vector $\mathbf{y}$ given

an input vector $\mathbf{x}$. A NN has "learned" the function in question when its parameters are tuned to the desired values. This happens during a process called training.

Neural networks are trained by dynamically calibrating their parameters through the minimization of an error function. Let us discuss a regression problem to show why we minimize the error function[17]. Assume we have an input vector $x_n$ and a target vector $t_n$ (n =1,. . . ,N) that has a normal distribution with mean y($\mathbf{x}$,$\mathbf{w}$). The probability that the NN output describes the relationship of the two variables is given by:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = N(\mathbf{t}|y(\mathbf{x}, \mathbf{w}), \sigma^2) \tag{3.18}$$

Assuming that $\mathbf{x}$ and $\mathbf{t}$ are independent and identically distributed observations we get the likelihood function:

$$p\left(\mathbf{t}\middle|\mathbf{x}, \mathbf{w}, \frac{1}{\sigma^2}\right) = \prod_{n=1}^{N} p\left(t_n\middle|\, x_n, w, \frac{1}{\sigma^2}\right) \tag{3.19}$$

and the logarithmic function

$$\ln\left(p\left(\mathbf{t}|\mathbf{x}, \mathbf{w}\right)\right) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{\left(y\left(x_n, w\right) - t_n\right)^2 + \frac{N}{2} ln\left(\frac{1}{\sigma^2}\right)\right\} - \frac{N}{2} ln(2\pi) \tag{3.20}$$

Our goal is to calculate the optimum weights w that maximizes the probability of $\mathbf{x}$ describing $\mathbf{t}$. By maximizing the likelihood function (p=1) we take the error function

$$\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{\left(y\left(x_n, w\right) - t_n\right)^2 - \frac{N}{2} ln\left(\frac{1}{\sigma^2}\right)\right\} + \frac{N}{2}\ln\left(2\pi\right) = 0 \tag{3.21}$$

and by ignoring constant values, we get the mean square error function

$$E\left(\mathbf{w}\right) = \frac{1}{2}\sum_{n=1}^{N}\left(y\left(x_n, w\right) - t_n\right)^2 \tag{3.22}$$

So, when the error function minimizes, $E\left(\mathbf{w}\right) \to 0$, probability of the independent variable being significant to describe the dependent variable maximizes. The same conclusion can be drawn from a classification problem such as a binary classification that is solved by minimizing the cross-entropy function

$$E\left(\mathbf{w}\right) = -\sum_{n=1}^{N}\left(ln\left(y_n + \left(1 - t_n\right)\ln\left(1 - y_n\right)\right)\right) \tag{3.23}$$

where t the target variable, y($\mathbf{x}$,$\mathbf{w}$) output of a sigmoid function and the conditional probability of targets described by the Bernoulli distribution.

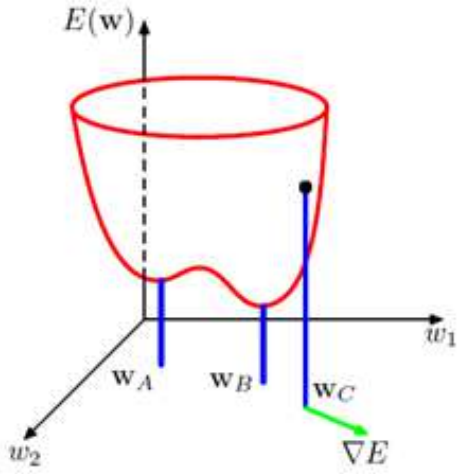



Figure 3.9: Error function represantation as a surface of the weight space

As we already mentioned our goal is to find a weight vector $\mathbf{w}$ that minimizes the chosen error function. It is common in the bibliography to visualize the error function as a surface of the weight space, like the one in figure 3.9. We can change the value of $\mathbf{w}$ by $\delta\mathbf{w}$ to change the error function by $\delta E \cong \delta\mathbf{w}^{\mathbf{T}}\nabla E(\mathbf{w})$. Error is a smooth continuous function; it minimizes when the gradient vanishes so that:

$$\nabla E(\mathbf{w}) = 0 \tag{3.24}$$

It is not always possible to find a vector $\mathbf{w}$ that returns a zero gradient. As a result, we try to identify one for whom E($\mathbf{w}$) takes its smallest value. This task is difficult since the error function has a high non-linear dependence on weights and biases. There will be many points at which the gradient vanishes where we will have multiple inequivalent minima. The optimum result is to find the global minimum but in practice we will not be able to know if it has been found. To deal with this, we compare several local minima and chose the one with the smallest value as a sufficient solution.

There is no analytical solution to the gradient equation of the error function. We use iterative numerical procedures to solve it and tune our NN's parameters. Optimization of continuous nonlinear function is a widely discussed issue and many algorithms and techniques exist to solve it. It is common to initialize the weight vector to an initial value $w^{(0)}$ and then make use of the gradient information to update the weight vector. Important optimization algorithms for NNs like gradient decent, AdaGrad, RMSProp will be discussed further below.

For the network to use the gradient, information should be able to travel forwards and backwards through it. This happens during a process called error backpropagation or backprop. First the derivative of the error function is evaluated and then it is used to compute the adjustments to be made to weights. This process happens for every iteration.The steps of the error backpropagation are the following:

- Apply an input vector $x_n$ and compute the activations for all hidden and output neurons.
- Calculate the error of all output neurons $\delta_k = y_k - t_k$.
- Backpropagate output errors using the backpropagation formula:

$$\delta_j = \varphi'(v_j) \sum_{k=0}^{n} w_{kj}\ \delta_k \tag{3.25}$$

  to obtain the error $\delta_j$ of each hidden neuron.

- Evaluate the required error function derivatives using:

$$\frac{\partial E_n(w)}{\partial w_{kj}} = \delta_j\ y_k \tag{3.26}$$

  where $\delta_j = \frac{\partial E_n(w)}{\partial v_j}$ and $y_k = \frac{\partial v_j}{\partial E_n(w)}$.

For batch methods, the derivative of the total error is calculated as the sum of all derivatives.

### 3.2.4 Optimization Algorithms

#### Gradient Decent

Gradient decent or batch gradient decent[18] is an iterative optimization algorithm used to minimize the objective function J($\theta$) where $\theta$ the model's parameters. The algorithm updates the parameters $\theta$ in the opposite direction of the gradient of the objective function. The equation describing the relationship is:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{3.27}$$

where $\eta$ the learning rate that determines the step size at each iteration while moving toward the minimum. The tuning of the learning rate is crucial as small values will require much time for the model to train and large values will cause the model to converge too quickly to a suboptimal solution, as

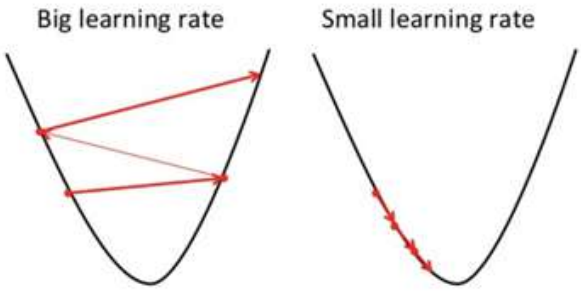



Figure 3.10: Learning rate values movements on the error function

shown in figure 3.10. The gradient of the objective function is calculated for the entire dataset.

There are many variations of this algorithm that improve its output. Like stochastic gradient decent (SGD) that instead of performing calculations for the whole dataset, it performs one update at a time for each training example $x^{(i)}$ and $y^{(i)}$ label:

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right) \tag{3.28}$$

This allows the optimizer to fluctuate on the surface describing the parameter space and potentially find a better local minimum. Another variation is the Mini-batch gradient where updates happen for a batch of n training examples:

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right) \tag{3.29}$$

It reduces the variance of the parameters making the convergence more stable and is computationally efficient.

### AdaGrad

The AdaGrad algorithm [19] is an improvement of the gradient decent. Gradient decent tends to go quickly down towards the steepest slope making it difficult later to find the global minimum since the original trajectory does not point to it, as shown in figure 3.11. This algorithm corrects its direction earlier by scaling down the gradient vector along the steepest dimensions.

The process is performed on two steps. Firstly, $s = s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$ is computed that accumulates the squares of the gradients of all iterations where $\otimes$ is the element wise multiplication. Secondly, the parameter $\theta$ is calculated

$$\frac{\theta = \theta - \eta \nabla_\theta J(\theta)}{\sqrt{s + \varepsilon}} \tag{3.30}$$

where $\varepsilon$ is a smoothening term to avoid division with zero and $\oslash$ is the element wise division. One of the main benefits is that this optimizer decays the learning rate requiring less tuning of this parameter. But this holds the risk of the algorithm stopping early since the learning rate is scaled down so much.

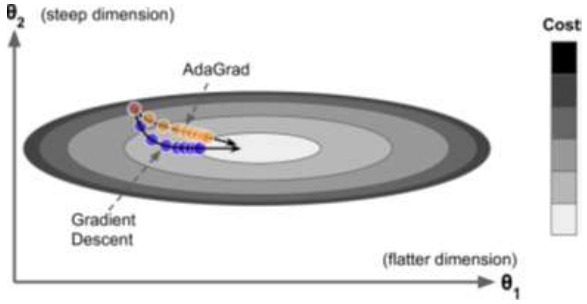



Figure 3.11: AdaGrad versus Gradient decent movements

#### RMSProp

RMSProp [20] deals with the issue of adaGrad that slows down to fast and misses the optimum converges, by only accumulating gradients of the most recent iterations. It achieves that by using exponential decay, so the equations are:

$$s = \beta s + (1 - \beta) \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta) \tag{3.31}$$

$$\theta = \theta - \frac{\eta \nabla_\theta J(\theta)}{\sqrt{s + \varepsilon}} \tag{3.32}$$

where $\beta$ is the decay rate.

### 3.2.5 Neural Network's Performance Evaluation

#### Residuals Evaluation

As we discussed the main goal when training a NN is to minimize the error function. But the performance of a network is based on its correct predictions during training and how well it predicts previously unseen data. Creating an optimized network (model) requires us to monitor its bias and variance.

When we created the error function for the regression example, we ignored two constant values which refer to the variance and irreducible error of the mean square error function. Therefore, a form that better represents

the total error of NN is:

$$Total\ Error = Bias + Variance + Irreductible\ Error \tag{3.33}$$

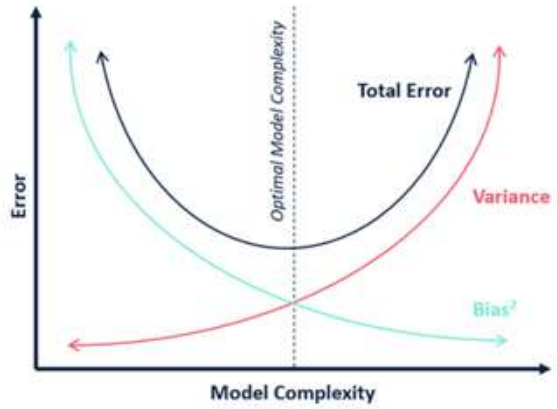



Figure 3.12: Total Error and model complexity relationship for the MSE function

Bias is the target value minus the average predicted value the model is tasked to map and it is inherent to the optimizer we used during training. Models with high bias make incorrect assumptions about the data and are oversimplified. They are unable to predict the target value on the training data.

Variance is the error due to sensitivity to small fluctuations of the training dataset. Models with high variance tend to map the noise of the data rather than the intended function. They fail to generalize meaning that they have great performance of the training data but high error on test data.

Irreducible error represents the amount of noise in our training data. It cannot be reduced by any algorithm as it is the natural variability in a system.

Monitoring bias and variance is about finding a good balance between them. The two variables have an inverse relationship, as shown in figure 3.12. When bias minimizes the variance is increased in relation to our model's complexity and vice versa. We want to tune our model, so the sum of bias and variance terms minimizes the total error. In case we fail our model will either under-fit or over-fit. Under-fitting happens when a model fails to map the underlying pattern of the training data and over-fitting happens when a model maps the noise of the training data. Under-fitting is associated with high bias and low variance while over-fitting with low bias and high variance. In the field of machine learning, the inverse relationship of the

two variables its consequences are called bias-variance tradeoff [21], and is visually identified as shown in figure 3.13.

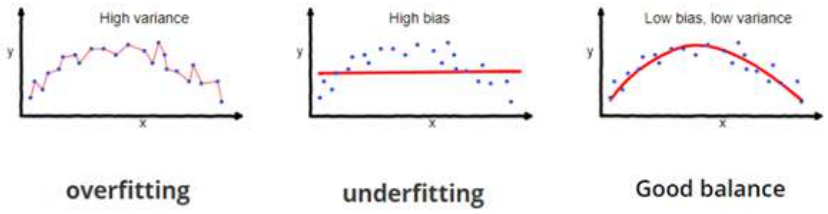



Figure 3.13: Bias-variance tradeoff

A graphical illustration of the relationship between bias and variance is shown in the bull eye diagram, figure 3.14, where the center target is a model that adequately maps the depended variables.

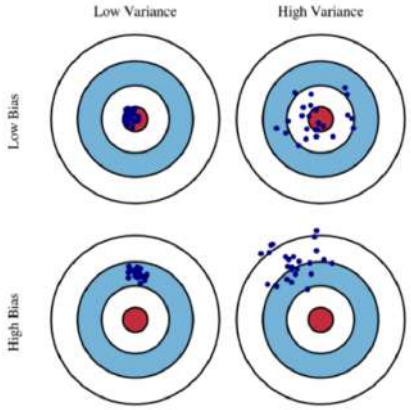



Figure 3.14: Bull eye diagram

### Cross Validation

The main problem with residuals validation is that it does not give an indication on how well the model will perform when exposed to previously unseen data. The idea is that part of the dataset removed before training then when training is finished the model is evaluated on the remaining data.

A widely used method is hold out [22]. In this method data are separated into two sets, called training and testing. The neural network is trained only using the training set and then the model is used to predict the output values for the test data it has never seen before. The errors are

accumulated and used to evaluate the model. The issue with the method is that it can have a high variance. The evaluation is heavily depended on how data are split. If data points in the training set and test set are significantly different then the results will vary. It is best for the data to be shuffled randomly and then split. A common split is 80% to 70% training and 20% to 30% test data.

An improvement of the holdout method is K-folds cross validation [22]. Data are divided into k subsets and holdout method is repeated k times. For every fold, one of the k subsets is used as test set while the other k-1 are joined to form the training set. Then the average error of all trials is computed. K value is preferred to be between 5 and 10. This method deals with the issue we mentioned about hold out since it matters less how the data are divided. Every data point is used k-1 times as training data and one as testing data. The disadvantage is that the model must be trained k times from scratch meaning that the method is slower and requires more computational power.

### 3.2.6 Neural Networks and Deep Learning

Having discussed neural networks, we now understand the key component of deep learning. Deep learning can be described as large neural networks, architectures with large numbers of layers, that are trained on huge amounts of data. In contrast to other machine learning algorithms the more the data the greater the performance of the deep neural networks [23], as shown in figure 3.15.

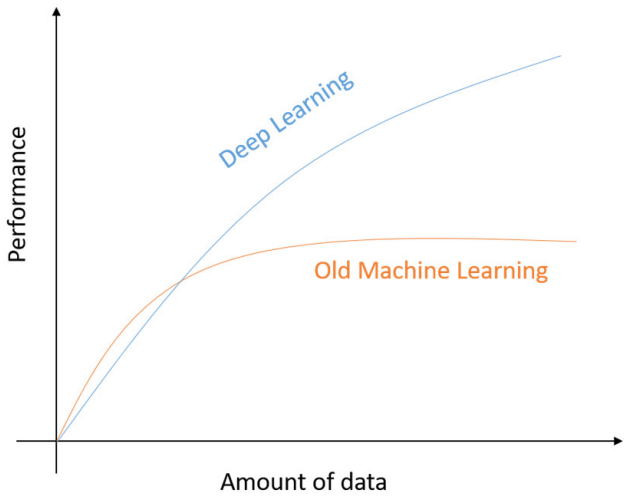



Figure 3.15: Machine and deep learning performance with regards to the amount of data

### 3.2.7 Convolutional Neural Networks

Convolutional neural networks [24] emerged while studying the visual cortex. David Hubel and Torsten Wiesel in their studies found that neurons in the visual cortex have a small local receptive field. They react to visual stimulations in a limited region of the visual field. Different neurons overlap their receptive field and combined they create the visual field. Also, they argued that neurons only react to images with a specific orientation of lines like horizontal or vertical. They noticed that neurons receptive field varies with some having large ones that can detect complex patterns as combination of low-level patterns. Their studies and the idea that neurons may be able to detect complex patterns using the output of neurons detecting simple patterns led to what we now call Convolutional Neural Network.

They create models that are invariant to transformations of the input by building invariance properties to their architecture. Also, they are preferred to deep fully connected neural networks for a variety of tasks, because they are partially connected and require less parameters to be trained. Therefore, they are widely used to computer vision tasks. They employ a key attribute of images, pixels close together are highly correlated in contrast to the distant ones. So, in the first stages they detect local features on small sub regions of the image and then merge them to obtain higher order features that yield information about the input image. This happens with the use of two new building blocks the convolution layer and pooling layer.

#### Convolution Layers

The convolution layer[24] is the most important building block of the CNN. The input image, one-layer image like grayscale or multi-layer like RGB, is connected to a convolution layer and every neuron receives pixels corresponding to its receptive field. Neurons of those layers maintain this architecture of partial connections throughout the network.

They receive data from a rectangular receptive fielding in the previous layer. This architecture allows the network to identify low-level features in the first hidden layers and then assemble them into larger higher-level features in the last ones.

Convolution layers can be visualized as a 2D vector of neurons. As shown in the figure 3.16, a neuron in row i, column j is connected to neurons in rows i to $i + f_h$–1, columns j to $j + f_w$–1 of the previous layer, where $f_h$ and $f_w$ are the height and width of the receptive field. This process raises dimensionality issues between layers. To deal with them it is common to add a border of pixels with value zero around the edges of the image, as shown in figure 3.16, in a technique known as zero padding. This allows a layer to have

the same height and width as the previous layer. However, occasionally it is useful to reduce the dimensions of layers to reduce computational complexity of the model. This is possible by spacing out receptive fields while connecting large layers to smaller ones. This is achieved by tuning the stride of the layer where stride is the shift from one perceptive field to the next one, as shown in figure 3.17.

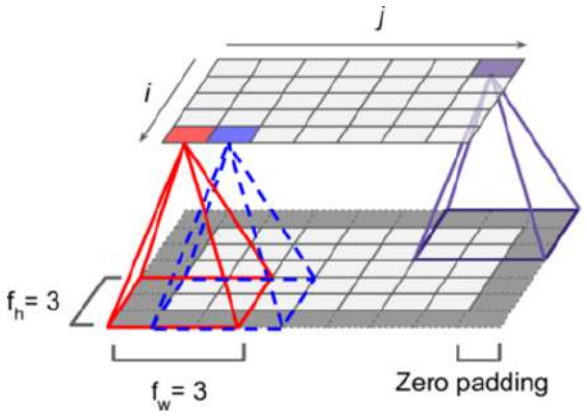



Figure 3.16: Connections between layers with zero padding

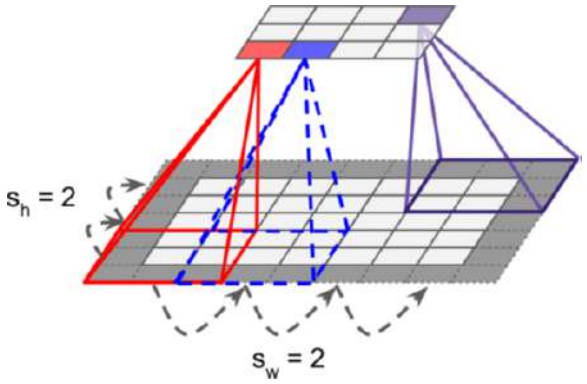



Figure 3.17: Dimensionality reduction

Weights of neurons are represented as 2D vectors that have the size of the receptive field. These vectors are called convolution kernels or filters. A layer full of neurons returns a feature map that indicates the areas in an image which activated the filter the most. During training the convolution layers tune their filters to create the best filters for the task of the model.

In general, convolutional layers have multiple filters and output one feature map per filter, as shown in figure 3.18. A feature map's neurons share the same parameters such as weights and biases and every pixel correspond to a neuron. Neurons in different feature maps use different parameters. A neuron's receptive field contains all the feature maps of the previous layers.

In short, the layer applies at the same time multiple trainable filters to its input. This gives CNNs one of their key capabilities, to detect multiple features regardless of where they are in the input image.

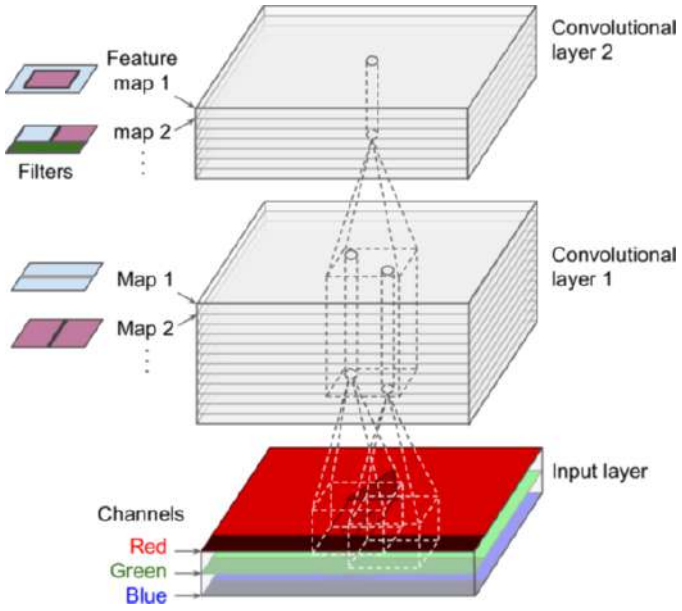



Figure 3.18: Convolutional layers with multiple feature maps and RGB input image

The equation describing the output of a neuron in feature map k of a convolution layer is:

$$y_{i,j,k} = \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \; w_{u,v,k',k} + b_k \tag{3.34}$$

where $s_h$ and $s_w$ are the vertical and horizontal strides. $f_h$ and $f_w$ are the height and width of the receptive field. $f_{n'}$ is the number of feature maps in the previous layer. $x_{i',j',k'}$ is the output of the neuron located in the previous layer. $b_k$ is the bias term for feature map k. $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer and its input located at row u, column v (relative to the neuron's receptive field), and feature map k'.

### Pooling Layers

The goal of a pooling layer [24] is to reduce the number of parameters. It shrinks the image to reduce computational load and make the model less prone to overfit. Its architecture is similar to the one of the convolution

layers. Its neurons receive data from a receptive field in the previous layer and it has its own filter/kernel size, stride, and padding parameters. It works as an aggregation function, like max or mean and its neurons have no weights meaning it is not trained during backpropagation. For example, a max pooling layer holds the max value of each kernel and drops the rest. Other common max pooling layers are the min pooling that holds the min values, the average pooling that calculates and hold the mean value of each kernel and L2 pooling that applies the Euclidean norm to each kernel and holds its value.

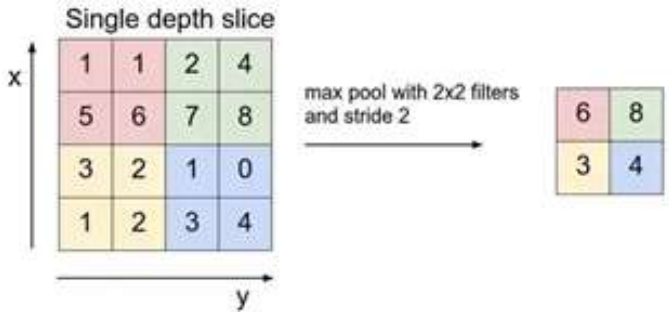



Figure 3.19: MaxPooling with a 2x2 kernel and stride 2

Max pooling layers are commonly used in CNNs and provide the model with some level of invariance to certain small distortions. Transformations like shifting, rotation and scaling of an object in an image can be dealt with because the layer will output an identical or similar feature map, since it keeps only the pixel with that has the biggest activation from the filter of the previous layer. This attribute is useful in classification tasks that the prediction should not be depended on these transformations. However, it is important to keep in mind that by using this technique we cast out huge amounts of information that could be useful to certain tasks, for example with a max pooling layer of a kernel $2 \times 2$ and stride 2 we drop 75% of the variables, as shown in figure 3.19.

### 3.2.8 Convolutional Neural Network Architectures

CNNs are compromised from a convolution base and at the top a classifier, as shown in figure 3.20. The convolution base consists of the input layer that gives the image to a first block of stacked convolution layers, whose neurons are activated by an activation function (commonly ReLU for computer vision tasks). Then a pooling layer, another stack of convolution layers and so on. During this phase features are extracted, and useless parameters are dropped. Finally, at the top of the base there is a regular neural network that receives the data and tunes its parameters to make

correct predictions.


Figure 3.20: Typical CNN architecture

In recent years there have been great advancement in the architectures of CNNs. Academia and industry come together and test their advancements in the field in the ImageNet Large Scale Visual Recognition Challenge or ILSVRC [25]. They compete on a dataset of millions of images and thousands of classes on tasks such as:

- Image classification: Predict the classes of objects present in an image.
- Single-object localization: Image classification and drawing a bounding box around one example of each object present.
- Object detection: Image classification and drawing a bounding box around each object present.

Competitors have achieved to reach a less that 2.3% error to classification problems, giving us notable CNN architectures. The most important are:

**AlexNet:**

This architecture [26], shown in figure 3.21, won the 2012 ImageNet ILSVRC challenge by achieving an error rate of 17%. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. It was the first CNN architecture to stack convolutional layers on top of one another.

To avoid overfitting, two regularization techniques were used. Firstly, dropout was applied with a 50% dropout rate during training to the outputs

of the two fully connected layers. Dropout is a process where randomly different neurons in each epoch are turned off at a predetermined probability rate. It allows the model to use different sample of parameters at every iteration making the model more robust. Secondly, data augmentation was performed. Images where randomly shifted by various offsets, flipped horizontally, and their lighting conditions were altered.

It also used a competitive normalization step after the ReLU activations of the first and third convolution layers, called local response normalization. This process improves generalization by allowing the most activated neurons to constrain other neurons located at the same position in neighboring feature maps. This makes feature maps to specialize by forcing them to explore a wider range of features.

The model was trained across two GTX 580 GPUs hence the split into two partitions of the architecture illustration below.

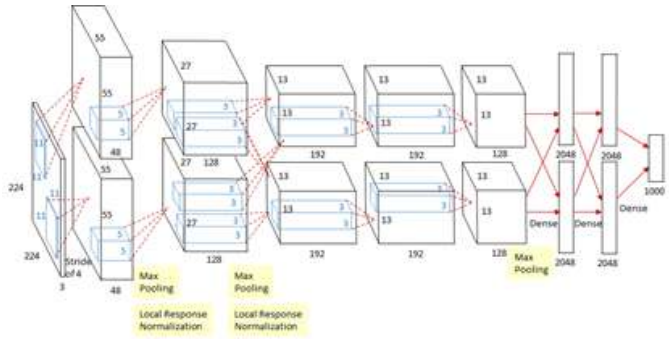



Figure 3.21: AleNet architecture

**VGGNet:**

This architecture [27], shown in figure 3.22, was the runner-up at the ILSVRC 2014 competition. It was developed by the community and was developed by Karen Simonyan and Andrew Zisserman of University of Oxford in 2014. Images are passed through a stack of convolution layers. There are 13 convolutional layers and 3 fully connected layers. It has small filters $3 \times 3$ with more depth instead of having large filters. It is currently the most preferred choice in the community for extracting features from images. However, the model has 138 million parameters making it slow to train from scratch. The original required 2-3 weeks of training on four NVIDIA Titan Black GPUs.

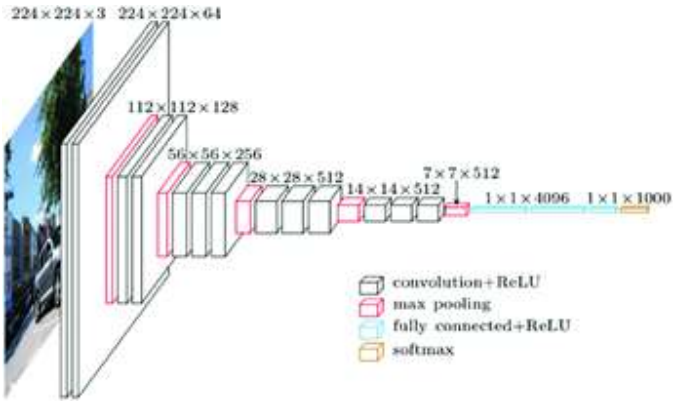



Figure 3.22: VGGNet architecture

**ResNet:**

The Residual Neural Network [28] won the ILSVRC 2015 challenge and it was developed by Kaiming He. It managed to beat human level performance on the trained dataset achieving a 3.57% error. Its success is based on the use of skip connections. These connections instead of only feeding the output of a layer to the next one, they also feed it as input to layers higher in the architecture.

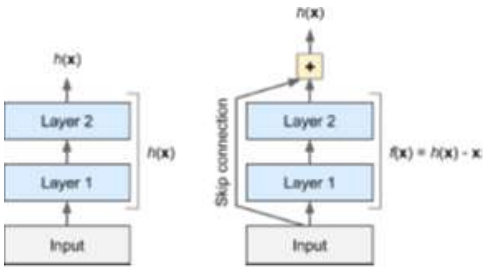



Figure 3.23: Residual Learning

Skip connections force the network to model a new function f(x) = h(x) – x, as shown in figure 3.23, where h(x) the original excepted output and x the original input. This practice is called residual learning. This enables us to carry important information in the previous layer to the next layers, preventing vanishing gradient problem. Also, it allows the network to treat the number of layers as a hyperparameter to tune, since layers that do not help the overall accuracy are skipped. Networks employing it can be though as dynamic neural network.

ResNet architecture, shown in figure 3.24, is like that of other networks but it also has a deep stack of residual units. Each residual unit has two convolutional layers (filters $3 \times 3$ and stride is one) with batch nor-

malization (used to increases stability, it normalizes the input of a layer by subtracting the mean and dividing by the standard deviation) and are activated using ReLU. Every few residual units the number of feature maps is doubled but their dimensions are cut in half. This creates a shape compatibility issue with the output of the residual units as they cannot be added to the original inputs of the layers. It is dealt by using a convolutional layer with a filter of $1 \times 1$ and stride 2 for the inputs.

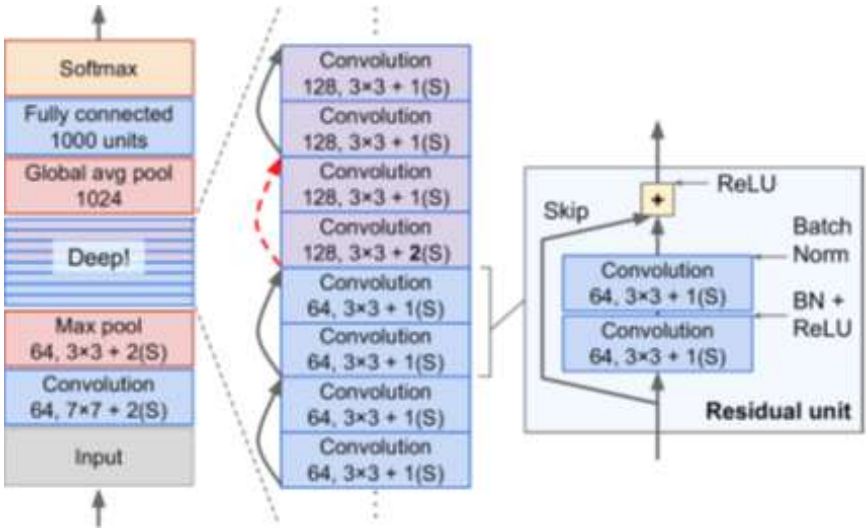



Figure 3.24: ResNet architecture

### 3.2.9 Transfer Learning

Transfer learning [24] in computer vision is a method that allows us to build accurate models efficiently and fast. Usually, instead for the model to be trained from scratch, it inherits pieces of its architecture and parameters from previously pretrained models. These models where trained on large benchmark datasets to solve several problems requiring various patterns to be detected. This way the new model leverages previous trainings and avoids learning already known patterns. The computational power and time needed to train models exploiting this technique is minimum since only a few parameters need to be tuned. Pretrained models are most of the times published models like the ones from the ImageNet competition previously discussed.

When transfer learning is employed the classifier of the pretrained model is removed and the convolution base is fitted to the new model. There are three strategies, figure 3.25, to repurpose the convolution base:

- Train all the model's parameters both on the convolution base and

classifier. Doing this only the original architecture of the base is reused, and the model will need a large dataset to learn the needed task.

- Freeze the whole convolution base and only train the classifier. This strategy is helpful when the dataset is small and there is shortage of computational power or the pretrained model solves a similar problem.
- Freeze part of the convolution base and retrain the rest. This strategy is based on the ability of CNNs to detect general features on the lower convolution layers and specific to the problem features on higher layers. So, we only adjust higher layers to identify problem related features. This practice is helpful because it allows to train efficiently on large datasets while training few parameters, making the model less susceptible to overfitting.

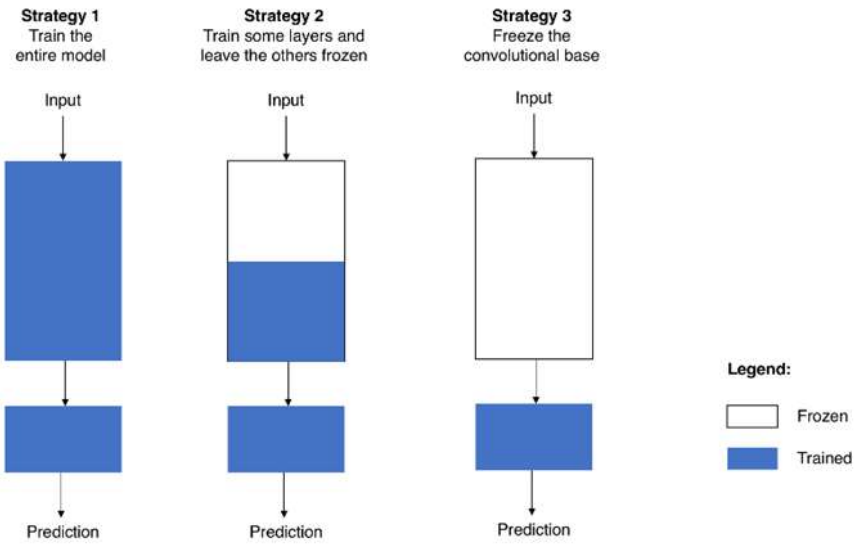



Figure 3.25: Transfer learning strategies

### 3.2.10 Object Detection

Object detection is the task of classification and localization of multiple objects in an image. It allows the detection of different objects and classes and assigns a bounding box in the region they are located. It is possible to use object detection technics to solve a classification problem more efficiently. An example is the classification of an image into a certain category, when the characteristics required to be detected are small with

respect to the image. In this case object detection would yield greater performance even if there is no interest in identifying the exact location of the characteristics.

As all supervised learning procedures, objects detection requires label data. The dataset is compromised of the images with corresponding bounding box coordinates (x, y) = (bottom left coordinate, top right coordinate) and labeling for each box.

There are many architectures used for object detection such as the Faster R-CNN, You Only Look Once (Yolo) and the Single Shot Multibox Detector (SSD). This study uses SSD architecture [29] for object detection, like the one shown in figure 3.26.

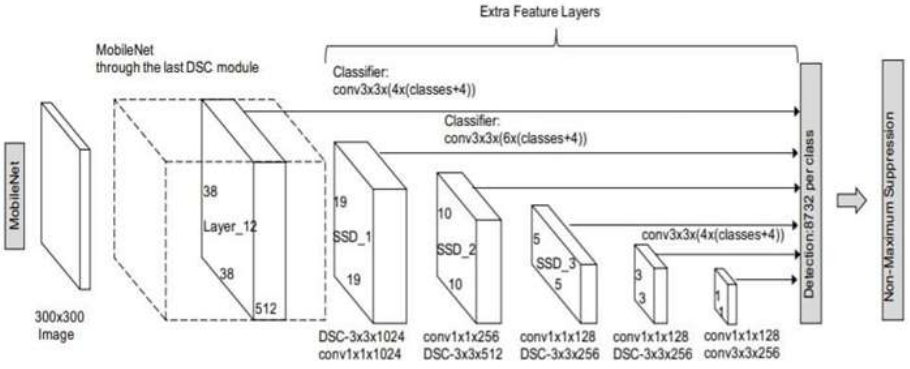



Figure 3.26: Single Shot Multibox Detector example with MobileNet

The SSD was published in 2016 from Google and uses a single deep neural to predict bounding box coordinates and classify their regions in one pass. The detector has a base model that extracts features followed by additional convolutional layers that detect the objects. The convolution layers decrease their size gradually to facilitate detection at different scales. Also, progressively feature maps depth is increased. As a result, deeper layers have larger receptive fields and more abstract representations are constructed. In shorts, initial layers detect small objects and deep layers larger objects. Finally, non-maximum suppression is used to reduce the number of detections to the actual number of objects present.

Non-maximum suppression (NMS) compares all detected boxes and returns those that have detected the objects better. All boxes are gathered in a list. The algorithm first selects the box with the highest accuracy and adds it to a list of filtered boxes. Then for the previously selected box it calculates the Intersection over Union ratio (IOU is the area of intersection of

boxes divided by the area of their union) with all other boxes in the original list. All boxes with ratio greater than a threshold value are removed. The process is repeated by choosing the box with the highest accuracy from the original list and performing the same calculation and tests. This is an iterative process repeated until the original list is empty.

### 3.2.11 TensorFlow Deep Learning Framework

TensorFlow [30] is an end-to-end open-source platform for machine learning. It was developed by researchers and engineers from Google to conduct machine learning and deep neural networks research. It employs Python for a convenient front-end API used to build applications, while executing them in high-performance C++.

It allows developers to represent computations without performing them until asked. This happens by creating dataflow graphs, like the one in figure 3.27, that describe how data moves through a graph, or a series of processing nodes. Nodes in the graph represent a mathematical operation, and connections between nodes multidimensional data arrays, or tensors. The computations of the graph are executed in a session that places the graph operations on hardware and provides methods to execute them.

This study employs TensorFlow and its high-level API keras to create deep convolutional neural networks used for image classification. It is also used with its Object Detection API to create models able of localizing and identifying objects in a single image.

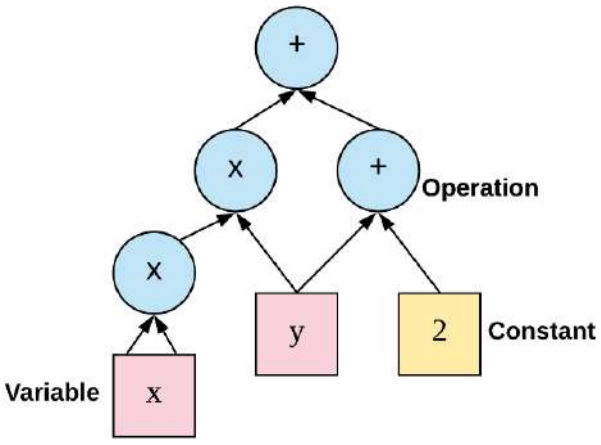



Figure 3.27: TensorFlow dataflow graph

# Chapter 4

# Corrosion Datasets

To implement corrosion detection two image datasets were created. The first one was created in a laboratory environment using metals from previous experiments. Its purpose is to facilitate the creation of models able to distinguish clean from corroded metals. The second dataset is made of real-life inspection images of bulk carriers. This one is used to create models able to detect corrosion from real life images. It treats corrosion detection as an object detection problem.

## 4.1 GUI Application

To handle large numbers of images a graphical user interface (GUI) application was created. Its layout can be found in figure 4.1. The app allows the user to iterate fast through images and allocate them in folders. It was created using python and Qt [31], which is a set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems, like traditional UI development. Also, Qt Designer is used to simplify the GUI creation. It includes GUI layout and forms designer that allow the user to design the desired layout and retrieve the code.

## 4.2 Laboratory Dataset

This dataset was created using corroded and clean metals from the materials laboratory of the Department of Materials Science and Engineering at the School of Chemical Engineering at the National Technical University of Athens. The metals were corroded during various lab experiments. Three hundred and sixty-six (366) metals were photographed on white background


Figure 4.1: Gui application layout

(figure 4.2) using a NIKON D3400 mounted on a tripod. The original dimensions of the images are 6000 × 4000 pixels with vertical and horizontal resolutions of 300 dpi. The images have 24bit depth, sRGB color representation and are JPG formatted.

(a) Corroded metal (b) Clean metal (c) Corroded metal

(d) Clean metal

Figure 4.2: Laboratory dataset sample of original images

As discussed in previous chapters, deep learning algorithms, like those that will employ this dataset, require large amount of data to be efficient.

Hence, the number of images needs to be increased artificially. Firstly, parts of the original images that included the metal plates were extracted. Then the number of images on the dataset is increased by cropping the extracted images to $300 \times 300$ pixels pieces. This process returned approximately 25000 images depicting corroded and clean metals. Using the GUI app, the cropped images where classified to two classes Rust and NoRust. Images that included metals with corroded and not corroded parts were dropped (figure 4.3). This choice was made to avoid confusion and noise in the data as the issue we want to solve using the dataset is a binary classification between clean and fully corroded metals. Therefore, in the Rust class we include images of metals that are 100% corroded and in the NoRust class we include images of metals that are 100% not corroded. Finally, the dataset is balanced as it includes 13200 images (dimensions $300 \times 300$) from which 6600 are in the Rust class and 6600 are in the NoRust class. Sample images can be found in figures 4.4 and 4.5.

Figure 4.3: Laboratory dataset sample of dropped images

Figure 4.4: Laboratory dataset sample of Rust class

Figure 4.5: Laboratory dataset sample of NoRust class

## 4.3 Surveys Dataset

This dataset was created using images from bulk carriers' inspections. Four thousand one hundred and one (4101) images (figure 4.6) were provided to the school of Naval Architecture and Marine Engineering of the National Technical University of Athens from KA-ENERGY shipbuilding studies. Original images dimensions and resolutions vary as they were created using multiple sources such as smartphones and a NIKON COOLPIX S210. But all images have 24bit depth, sRGB color representation and are JPG formatted. The GUI app was used to iterate though the images and identify those that depict corrosion. Finally, 352 images were categorized as corrosion depictions, a sample of the class can be found in figure 4.7 .

The purpose of this dataset is to be used from object detection algorithms. As it is already mentioned datasets for object detection require the image and coordinates of where the object is in the image. To create this information a graphical image annotation tool called LabelImg[32] is used (figure 4.8). The tool is open-source and is created using python and Qt. Annotations that include the coordinates and the class information are saved as XML files in PASCAL VOC format. Images were reshaped to $800 \times 800$ pixels to decrease memory requirements per image and annotations were created for every image.


Figure 4.6: Sample of images provided by KA-ENERGY

Figure 4.7: Surveys dataset sample of Corrosion class


Figure 4.8: LabelImg tool

# Chapter 5

# Methodology

In this chapter, three methods that will be used to detect corrosion are defined and analyzed. This study focuses on visually detecting corrosion. Corrosion is the deterioration of a metal and can be visually identified by its color and texture. Therefore, the first method uses traditional computer vision techniques to extract parts of images that include corrosion, based on their color. The second method employs deep learning. While humans pay attention to the shape of an object to identify it, deep learning computer vision algorithms focus on their texture instead. Based on this attribute of deep learning algorithms and the distinct texture of corrosion contrary to other metals, deep convolutional neural networks are used to detect corrosion on a laboratory environment. The third method is also approaching corrosion detection as a deep learning problem. It treats corrosion as an object in an image where there are many other objects and backgrounds. Using appropriate object detection algorithms, it tracks corrosion in real world images.

## 5.1 Color Tracking for Corrosion Detection

This method tracks corrosion based on its color. The process is created combining various traditional computer vision algorithms and is implemented using python and OpenCV.

The method considers as input an RGB image. The first step is to transform the color space of the image from RGB to HSV. The HSV color model is preferred for color detection tasks as it is more robust to external lighting changes. This characteristic is attributed to the separation of the intensity-brightness (Value) from the color information (Hue). There are many other models that separate intensity from color but HSV is favored because transforming RGB to HSV is easily implemented as described in

chapter 3. The next step is to define the HSV color range that describes corrosion by setting a lower and upper limit of HSV values. Pixels containing these values will be extracted from the image and be categorized as corrosion. The extraction is performed by thresholding all values in the designated range. This returns a mask which is a binary image, where the pixels containing values in the designated color range are white and the rest black. To deal with holes in the regions containing corrosion a non-linear filter is used to perform the morphological operation of dilation. The operation is described mathematically by:

$$dilate(f, s) = \theta(c, 1) \tag{5.1}$$

$$c = f \bigotimes s \tag{5.2}$$

where $\theta$ the threshold operation as described by equation 3.10, f the mask of the input image, s the structural element that for our process is a $5 \times 5$ filter with all its values being 1s.

The mask we created contains the information needed to find regions in the image containing corrosion. To visualize the corroded areas in the mask, the per-element bit-wise conjunction of the input image and the mask is calculated. This process returns the pixels of the input image that their coordinates correspond to pixels in the mask whose values are 1 (visually identified as white) and keeps all other pixels' values to 0 (visually identified as black). To visualize the same areas in the input image the contours are identified and drawn on it. Contours are defined as a curve joining all continuous points, along the boundary of an object in an image, that have the same color or intensity. The contours are detected from the mask using the algorithm of Satochi Suzuki and Keiichi Abe, that is described in their paper "Topological Structural Analysis of Digitized Binary Images by Border Following" [33]. They are retrieved without establishing any hierarchical relationship and to preserve memory only the two edge points of every contour line are saved. Then the contours are drawn on the input image pointing out the regions containing corrosion. Using the Green's formula, the area surrounded by contours is calculated to extract the percentage of corrosion in the image. Results are returned on a .txt file.

Color detection applications sometimes require two HSV color ranges to be extracted. This happens when tracking an object that its Hue range is a combination of the upper and lower values of the HSV space (ex. Hue is in the range of [350, 10]). In these cases, the threshold operation of the input image takes place twice, once for each range, returning two masks. Then the masks are added, and the process continues as explained above.

The steps of the process can be summarized in the following manner:

1. Input an RGB image

2. Convert image to HSV color space
3. Define the color range to be tracked
4. Threshold the image to create a mask containing regions with the previously defined colors
5. Dilate the mask's detected regions to deal with missing pixels
6. Visualize the corrosion areas detected in the mask
7. Find the contours and draw them on the input image
8. Calculate the total corroded area of the image
9. Return the input image marking corroded regions

In figure 5.1 the data flow diagram of the method is presented.

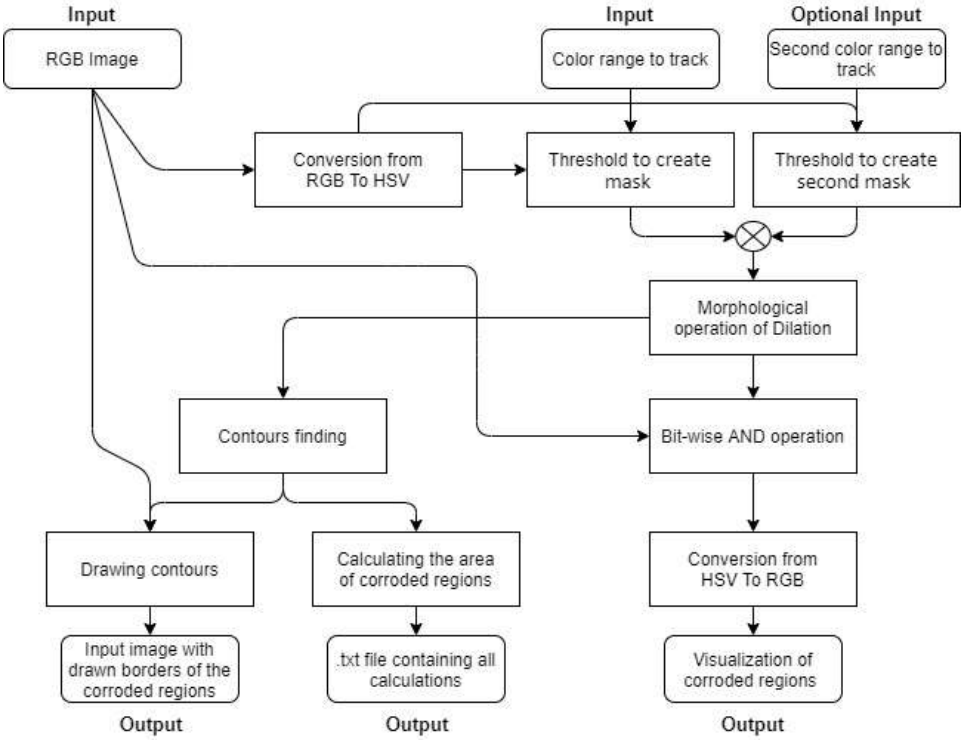



Figure 5.1: Data flow diagram of color tracking method

## 5.2 Deep Convolutional Neural Networks for Corrosion Detection

This method employs Convolutional Neural Networks to detect corrosion(rust). The goal is to determine which parts of a metallic surface are corroded or not. Deep learning algorithms like CNNs focus on the texture of the image. Corroded metals usually have rough surfaces, while clean ones tend to be smooth. Hence, the method assumes that a CNN will detect the texture of corroded regions and correctly classify them. The process is implemented using python and the TensorFlow deep learning framework.

The CNN was trained using the laboratory dataset, where classes are labeled as Rust and NoRust. The hold out method was used to split the dataset into training, validation, and test sets. Due to the small number of images and to reduce the time needed for training, transfer learning was employed. Different pretrained models were tested, like VGGNet and ResNet, and it was decided to use parameters of ResNet50 for the convolution base. The classifier consists of a flatten layer (arranges features extracted from the convolution base to a 1-D array so they can be received by a layer of neurons), one fully connected layer with 256 neurons using the ReLU activation function, and an output layer with one neuron using the sigmoid activation function. The sigmoid activation of the final neuron is appropriate since the model is tasked with resolving a binary classification problem. When the final neuron's output is 0, it means the model classified the input image to the NoRust class, while when the output is 1 it means the model classified it to the Rust class. Results regarding the training process and the metrics used to test the model are discussed in chapter 6.

For the model to be used with metals containing both corroded and clean regions, a sliding algorithm to feed data to the CNN was created. The algorithm assumes a window of $M \times N$ pixels that slides over an image returning the pixels it slides over as an image with the dimensions of the window. In our case, the window is $300 \times 300$ and slides over the image with a horizontal and vertical step of 300-pixels returning the extracted part of the image. Horizontal steps have priority meaning that if there is a 300-pixel step for the algorithm to make horizontally, it will not take a vertical step.

The method combines the model with the sliding algorithm. An input image is given to the algorithm and gets broken into pieces. Every piece, once extracted, is fueled into the deep learning model and is classified. Then if a piece is found to be corroded, it is marked accordingly to the input image. Finally, the image is returned with the corroded regions annotated. The dataflow diagram of the method is presented in figure 5.2.

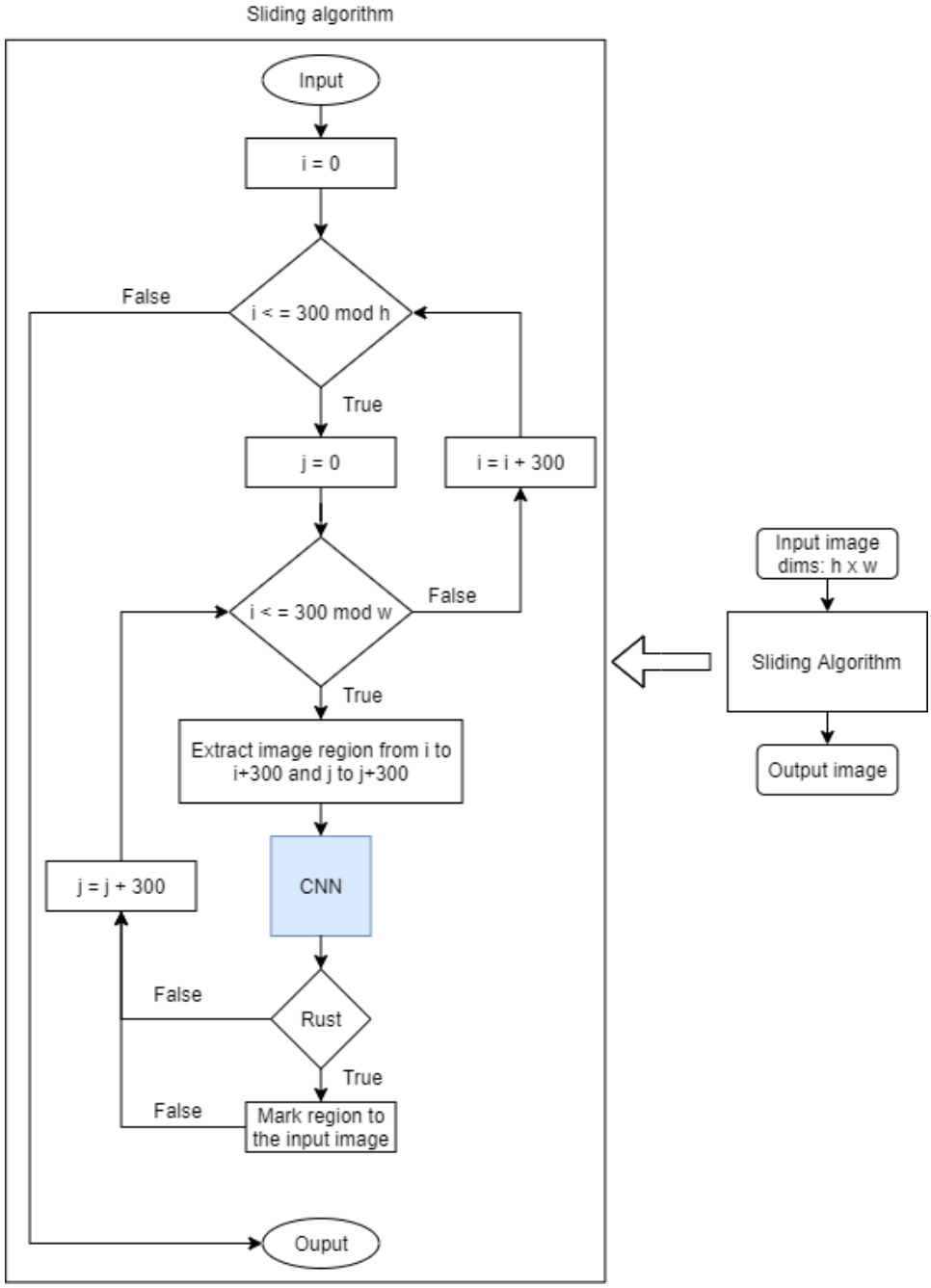



Figure 5.2: Data flow diagram of the method employing CNN architecture

## 5.3 Corrosion Detection - Object Detection Approach

This method employs the Single Shot Multibox Detector to detect corrosion in real world environments. By employing deep convolutional neural networks for corrosion detection (section 5.2), the problem of misclassifying metallic surfaces due to their color was resolved as these algorithms focus on the texture of the image. However, in real life applications the binary classification approach of the previous method holds the risk of misinterpreting other faults as corrosion. This happens because the model has learned what corrosion is and what is not. But the class containing not corroded images can not cover every state a metal may have and not be corroded. Therefore, it is assumed that corroded regions are objects in the image that need to be tracked. The advantage of this approach is that the model does not train on a particular no rust class and assumes all background regions to be not corroded. The method is implemented using python and the TensorFlow Object Detection API.

As discussed in chapter 3, the SSD architecture is comprised of 3 parts. A base model tasked with feature extraction from the input image. Stacked convolution layers that detect the position of object in the image. A non-maximum suppression algorithm that drops detected boxes whose content has already been detected.

To minimize the time needed to make a prediction and the number of trainable parameters, the Mobile Net architecture [34] was used as base model. The Mobile Net architecture uses depth-wise separable convolutions to build light weight CNNs. Depth-wise separable convolutions are made from two operations. First, a depth-wise convolution is performed without changing the depth of the image. For example, assume a 3-channel image with dimensions $12 \times 12$ for which 3 filters with dimension $5 \times 5 \times 1$ are used. Each filter convolves with one channel returning an image with one channel. Stacking the products of all three convolutions we get a 3-channel image, in our example assuming no padding and stride 1 we get an $8 \times 8 \times 3$ image. Then, a point-wise convolution is used to increase the number of channels. It uses a $1 \times 1 \times N$ kernel for the operation, where N is the number of channels of the image returned from the depth-wise convolution. The operation transforms the image to a 1-channel image. Multiple kernels are used with their number determined by the depth we want to achieve using the convolution layer. In our example, if 256 kernels were used, we would get an image with dimensions $8 \times 8 \times 256$. This way we get the same result we would get performing normal convolutions, but the model requires less computation.

For this application, the SSD MobileNet v1 is used to initialize our

model using transfer learning and resume training for the model to learn the "corrosion" object. This detection model is pre-trained on the COCO 2017 dataset. Its performance on the COCO dataset is a mean average precision (mAP) of 21 and speed 30 ms. It outputs boxes to the detected regions.

# Chapter 6

# Results

In this chapter, results of the methods menthioned in chapter 5 will be presented and discussed.

## 6.1 Corrosion Color Tracking Results

To test the color tracking method for corrosion detection images from the Surveys Dataset (chapter 4.3) were used. For results presentation, 12 images are being displayed. The images were chosen to represent different compartments of a bulk carrier to show the method's performance under various backgrounds. The 3 compartments described in the images are fuel oil tanks, ballast tanks, and cargo holds.

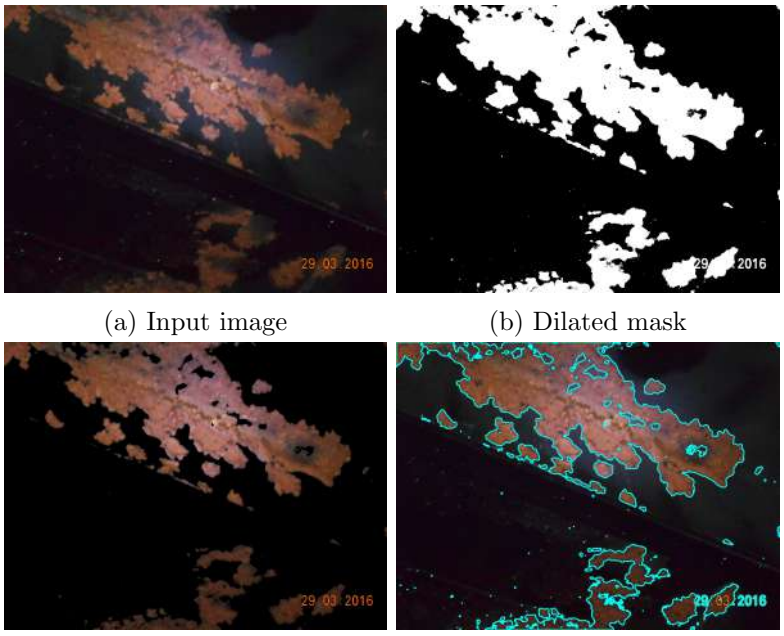
The trial-and-error method was employed to decide the appropriate HSV color range to be extracted from the images. The range needed to be in the OpenCV format as this library was used for the implementation of the process. Thus Hue takes values from [0, 180], Saturation from [0, 255] and Value from [0, 255]. Finally, range's upper limit was set to [H, S, V] = [140, 255, 255] and the lower limit to [H, S, V] = [100, 50, 50].

The results are presented in the figures below. In figures 6.1, 6.2, 6.3, 6.4 corrosion is tracked from fuel oil tanks' images. The precentages of corroded areas in relation to the total area of the image for images 1 to 4 are 27.10 %, 48.06 %, 25.55 %, 38.27 % respectively. In figures 6.5, 6.6, 6.7, 6.8 corrosion is tracked from ballast tanks' images with corrosion precentages 85.27 %, 83.99 %, 93.09 %, 90.34 % respectively. In figures 6.9, 6.10, 6.11, 6.12 corrosion is tracked from cargo holds' images with corrosion precentages >95.00 %, >95.00 %, 79.90 %, >95.00 % respectively.

The efficiency of the method is closely related to the background surrounding corroded areas and the dispersion of corrosion in the image.

In images where the shade of corrosion is significantly different form the background colors the color tracking process yields great results, like those shown in the images from fuel oil tanks. In contrast, when the shade of corrosion is similar to the background, the method assumes large parts of the background to be corroded regions. This is shown in the cargo hold images, where the red/maroon coating is misinterpreted by the process as corrosion. In cases where corrosion is widely and densely dispersed over the image, like the ballast tank images, small regions of clean metal on the corroded borders tend to be characterized as corrosion. This diminishes efficiency, but the process still successfully tracks corrosion.

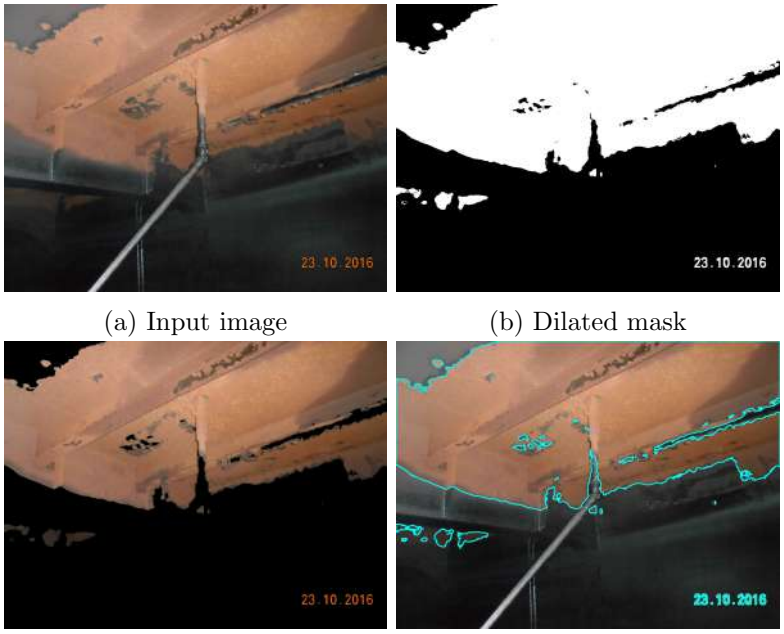
Overall, the color tracking method can track corrosion on simple cases but fails on demanding ones, where color is not the main characteristic of identification. It also lacks generalization since the user should change the color range to be extracted from the image for cases, where the color of corrosion may differ (ex. The rust of ferrous materials and stainless steel is visually identified with black color). Hence, solutions that consider the texture of the corroded metals are needed. This approch is examined by the deep learning methods of chapter 5.



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

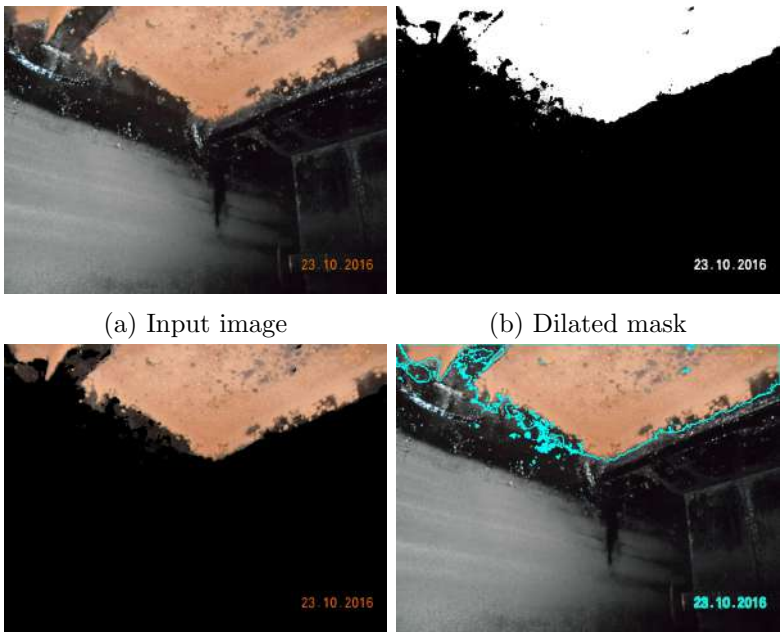
Figure 6.1: Color tracking results image 1 - Fuel Oil Tank




(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

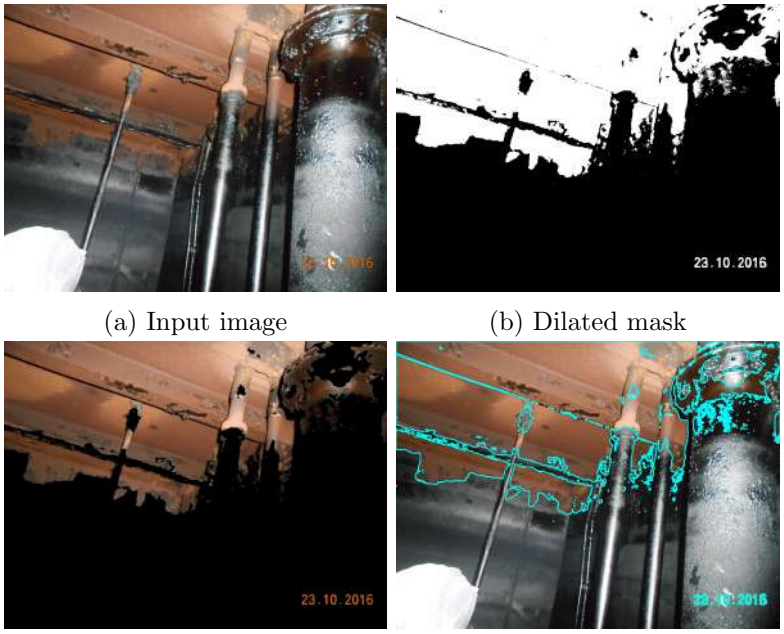
Figure 6.2: Color tracking results image 2 - Fuel Oil Tank




(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

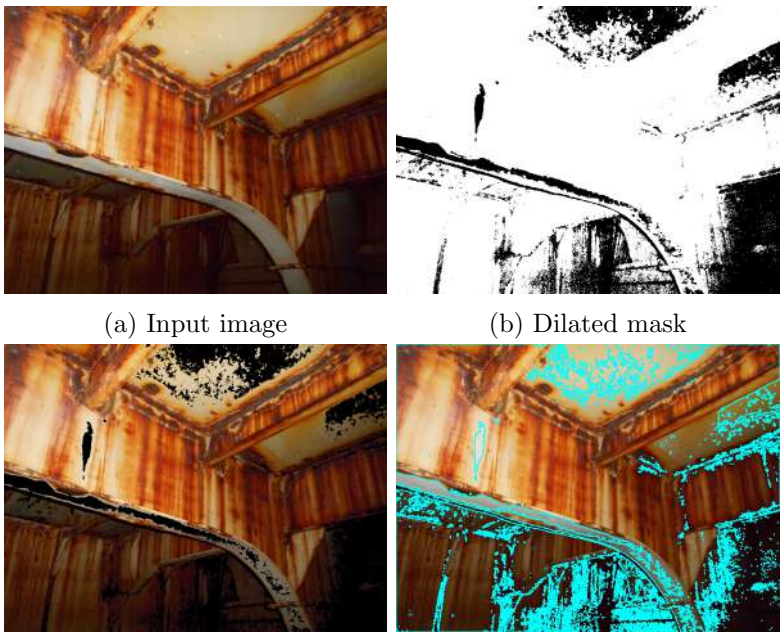
Figure 6.3: Color tracking results image 3 - Fuel Oil Tank




(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

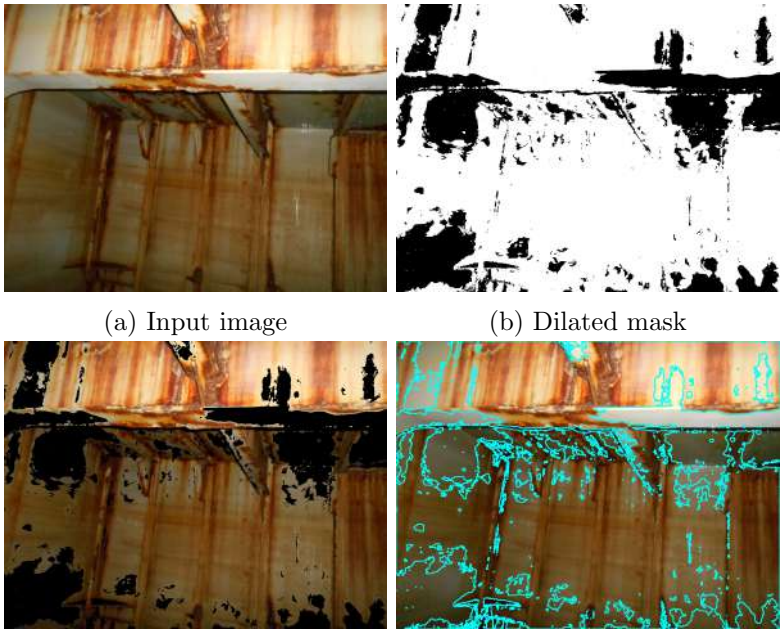
Figure 6.4: Color tracking results image 4 - Fuel Oil Tank



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

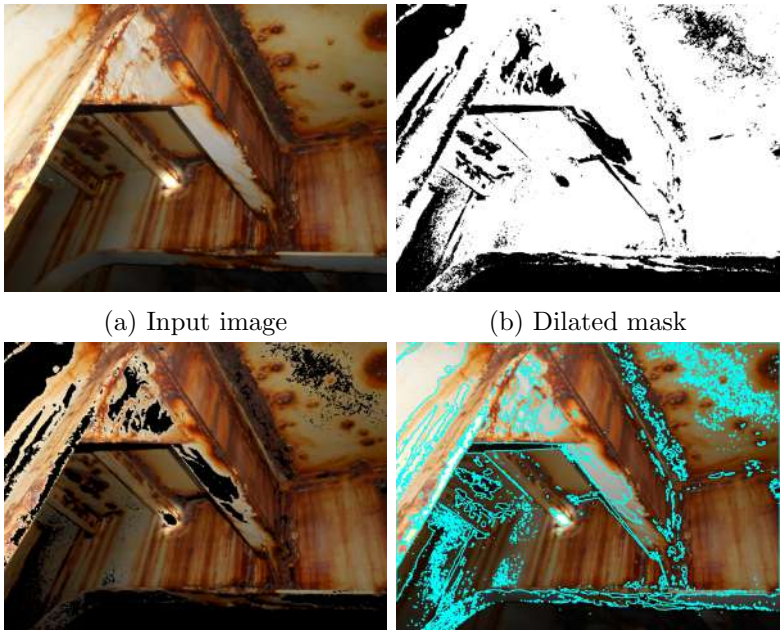
Figure 6.5: Color tracking results image 5 - Ballast Tank



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

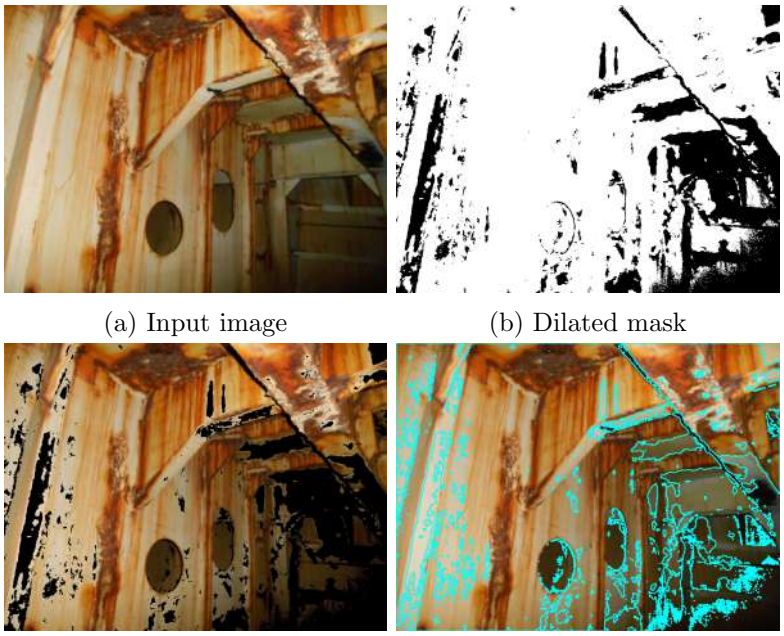
Figure 6.6: Color tracking results image 6 - Ballast Tank



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

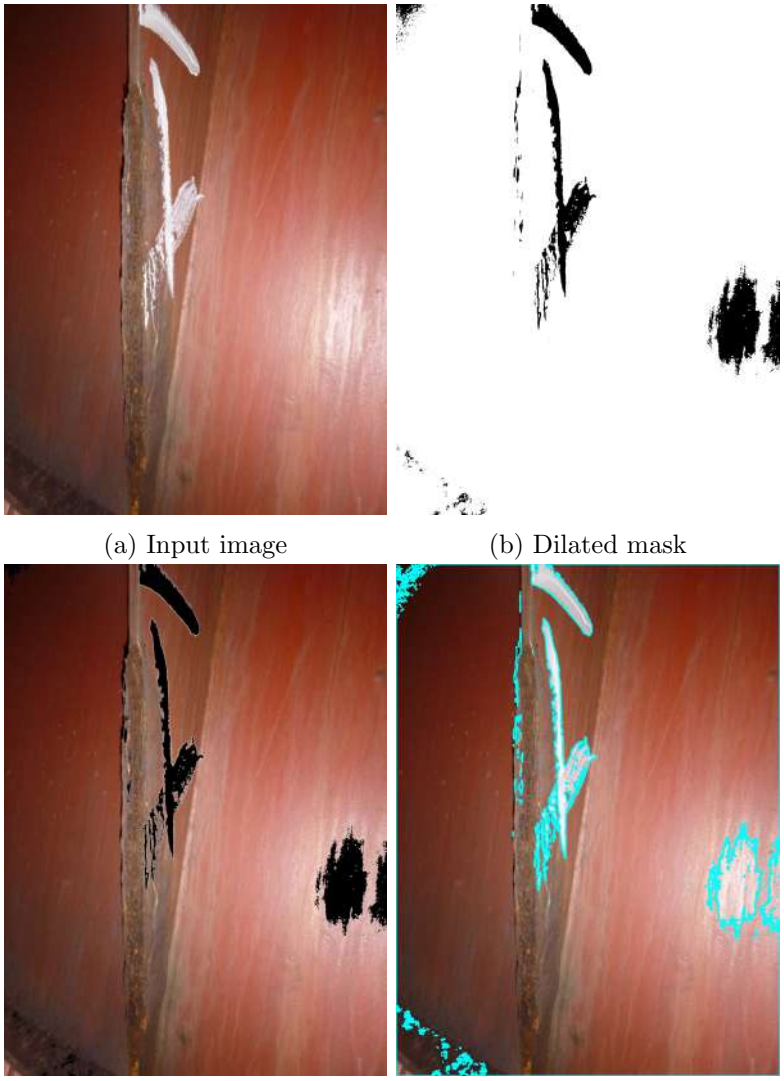
Figure 6.7: Color tracking results image 7 - Ballast Tank



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

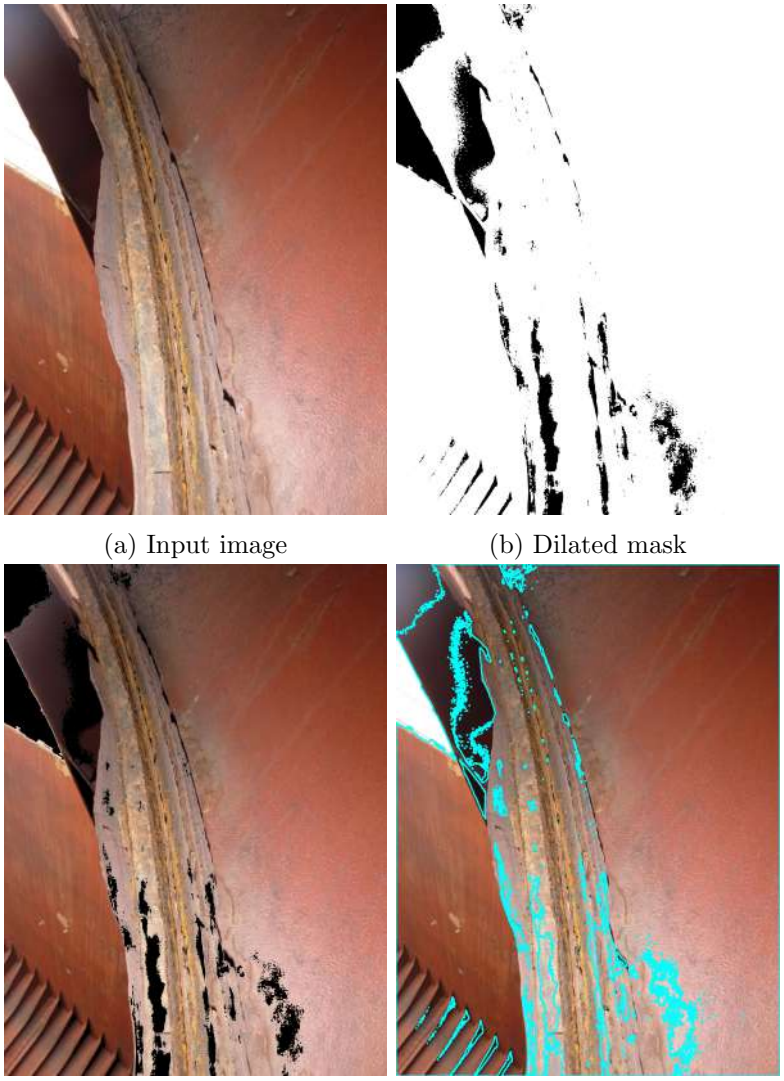
Figure 6.8: Color tracking results image 8 - Ballast Tank



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

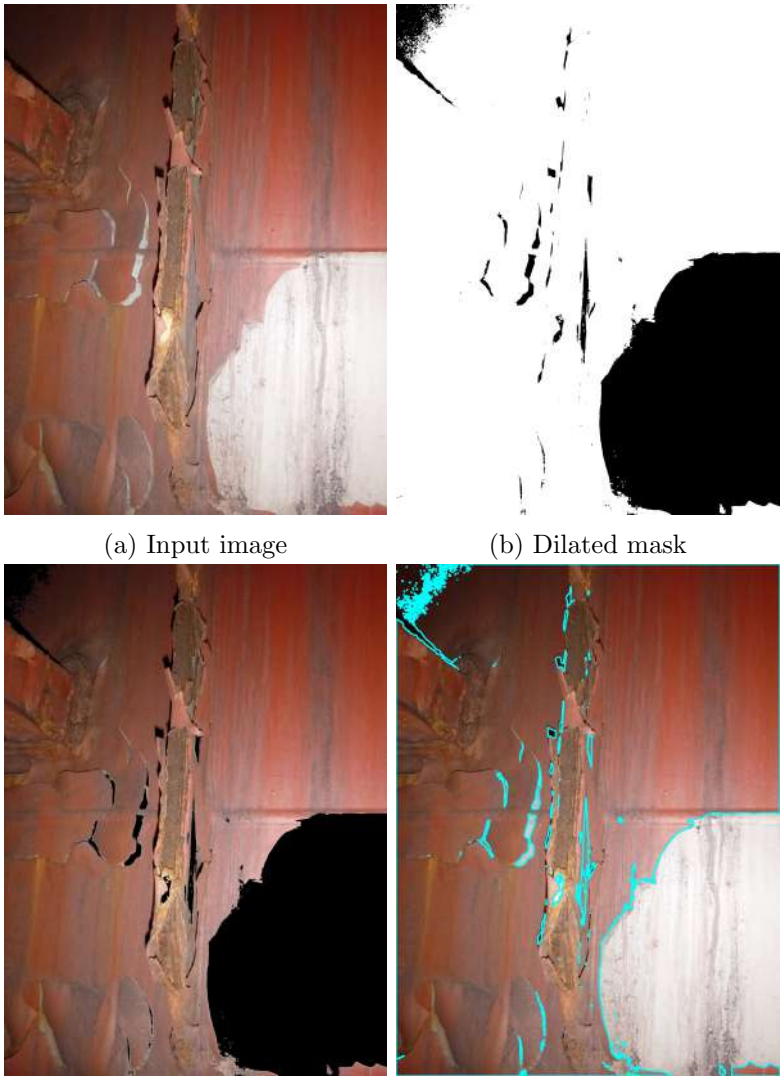
Figure 6.9: Color tracking results image 9 - Cargo Hold



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

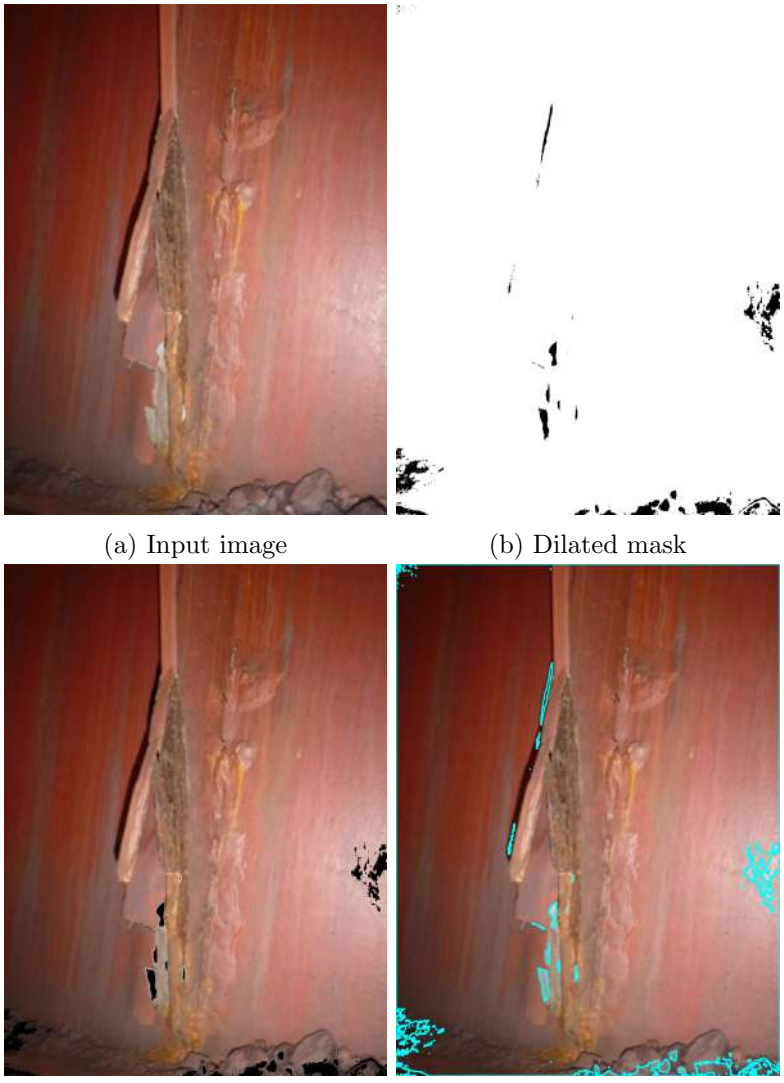
Figure 6.10: Color tracking results image 10 - Cargo Hold



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

Figure 6.11: Color tracking results image 11 - Cargo Hold



(a) Input image (b) Dilated mask

(c) Extracted Rust (d) Image with drawn contours

Figure 6.12: Color tracking results image 12 - Cargo Hold

## 6.2 Corrosion Detection with Deep Convolutional Neural Networks Results

### 6.2.1 Corrosion detection model results

The model was trained using the laboratory dataset. The hold out method was employed to spit the data into training, validation, and test datasets containing 9760 images (4880 per class), 2440 (1220 per class) and 1000 (500 per class) respectively. The input pipeline reshaped the images to $150 \times 150$ dimensions and scaled pixels to [0,1] by dividing each pixel with 1/255. Neural networks use small values for processing data, so normalization (scaling) of the inputs improve convergence speed and accuracy. Since the dataset was created in a laboratory environment no image augmentation operations were performed. Transfer learning was employed freezing the convolution base of ResNet50 and the classifier was trained after its weights were randomly initialized. The summary of the model including the number of trainable and non-trainable parameters is presented on table 6.1. To train the model faster and reduce memory requirements a batch size of 16 is selected for the training dataset and 32 for the validation dataset. The model was trained for 100 epochs using the RMSProp optimizer and the cross-entropy loss function. It was trained on an Intel Core i7 6700 CPU for 7 hours.

| Layer (type) | Output Shape | Parameters |
|---|---|---|
| ResNet50 (Functional) | (None, 5, 5, 2048) | 23587712 |
| Flatten (Flatten) | (None, 51200) | 0 |
| Dense Layer (Dense) | (None, 256) | 13107456 |
| Output Layer(Dense) | (None, 1) | 257 |
| Total params: 36,695,425<br>Trainable params: 13,107,713<br>Non-trainable params: 23,587,712 | | |

Table 6.1: Model's summary

The model achieved a training accuracy of 95.75% with loss 0.1193 and a validation accuracy of 95.10% with loss 0.1257. Training results are shown in figure 6.13.

The model was tested using the test dataset. To measure performance the main classification metrics were calculated. Note that predicted values are described as Positive and Negative and actual values as True and False. Recall measures out of all positive classes; how many were predicted

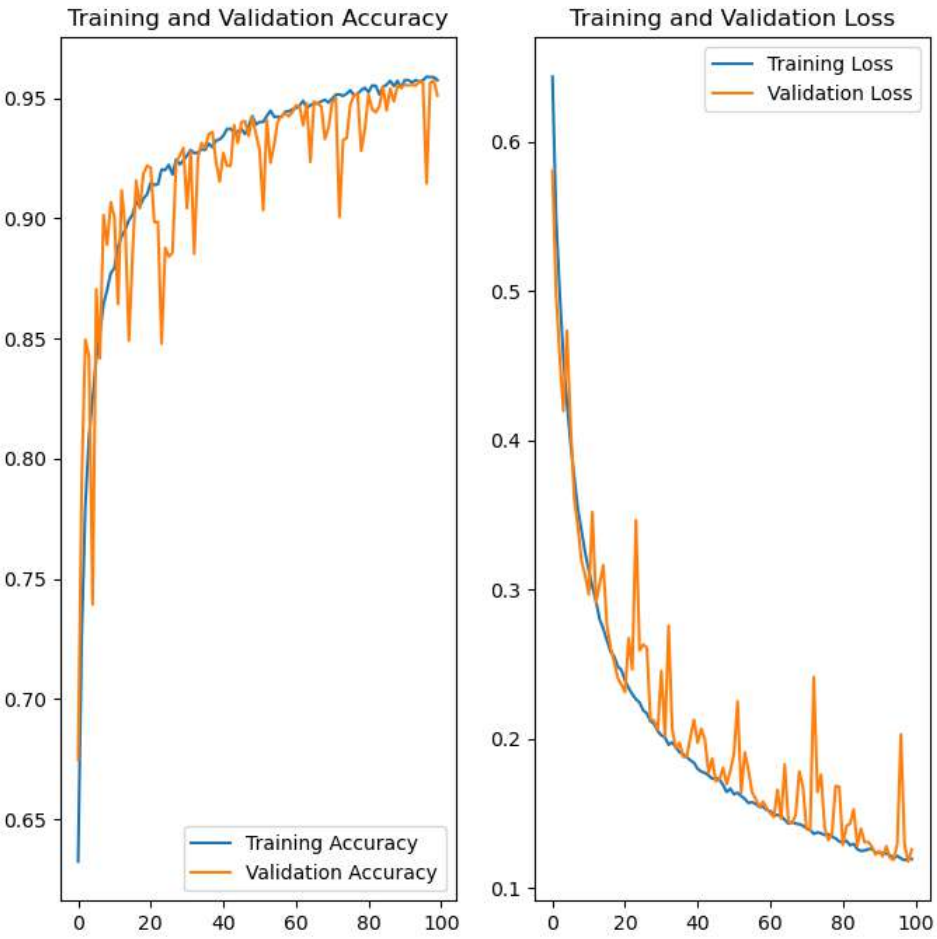



Figure 6.13: Model's accuracy and loss graphs

| Class | Precision | Recall | f1-score | Support |
|---|---|---|---|---|
| Rust | 0.94 | 0.94 | 0.94 | 500 |
| No Rust | 0.94 | 0.94 | 0.94 | 500 |
| Accuracy | | | 0.94 | 1000 |
| Macro Avg | 0.94 | 0.94 | 0.94 | 1000 |
| Weighted Avg | 0.94 | 0.94 | 0.94 | 1000 |

Table 6.2: Model's classification report on test data

correctly. Precision measures out of all the positive classes predicted, how many were positive. Accuracy measures out of all classes; how many were predicted correctly. F-score measures the harmonic mean of recall and precision to make them comparable. The equations of the above metrics for binary classification are:

$$Recall = \frac{TP}{TP + FN} \tag{6.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

$$Accuracy = \frac{TP + TN}{Total} \tag{6.3}$$

$$F - score = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{6.4}$$

where TP is True Positive (predicted positive and it is true), TN is True Negative (predicted negative and it is false), FP is False Positive (predicted positive and it is false), FN is False Negative (predicted negative and it is false). The results between predicted and actual values are presented on the confusion matrix of figure 6.14 and the classification metrics results are presented on table 6.2.

The AUC (Area Under the Curve) – ROC (Receiver Operating Characteristics) curve is also plotted, on figure 6.15,to measure the performance of the model. It measures the ability of the model to distinguish between classes. The ROC curve is a probability curve plotted with TPR (True Positive Rate) against FPR (False Positive Rate) at different classification thresholds, where TPR is related to the sensitivity and FPR to the specificity. The equations for TPR and FPR are:

$$TPR = Sensitivity = Recall = \frac{TP}{TP + FN} \tag{6.5}$$

$$FPR = 1 - Specificity = 1 - \frac{TN}{TN + FP} = \frac{FP}{TN + FP} \tag{6.6}$$

The area under the ROC curve (AUC) provides a measure of the performance across all possible classification thresholds. It takes values from 0

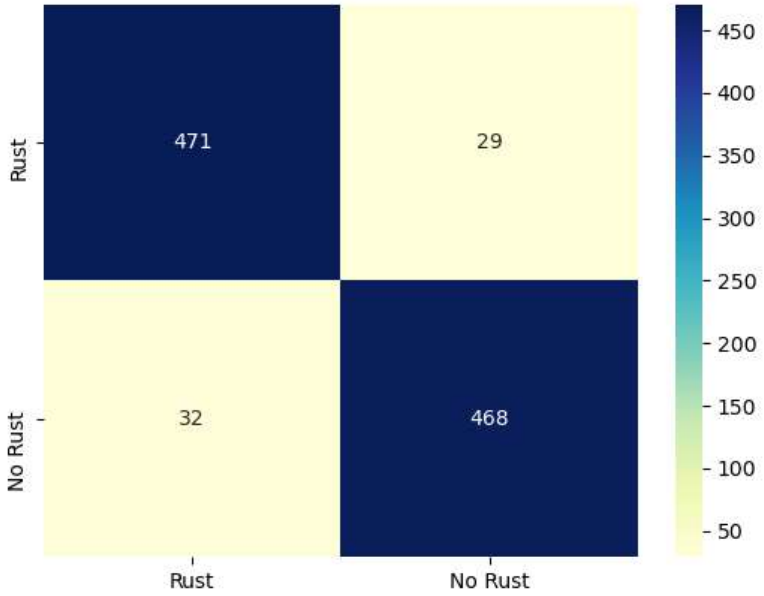



Figure 6.14: Model's confusion matrix on test data

to 1 with models close to 1 having a good measure of separability between classes.

Overall, the model performs well on both the training and validation datasets. There is not any indication of overfitting based on their accuracy and loss curves. Normally we would expect the training and validation curves to deviate slightly more, but results are accepted as most of the images in the dataset are quite similar, since they were created on a laboratory environment. Also, the model performs well on previously unseen test data achieving an accuracy of 94%. Finally, it has excellent separability with the AUC value being close to 0.99.

According to the above metrics, we managed to implement a model capable of detecting corrosion using a CNN. The hypothesis is that the CNN detects the texture of the corroded metal. As it is already discussed the convolution base extracts the features used by the classifier to make the prediction. To test the hypothesis, we extract the feature maps of the first convolution layers of images that the model has predicted as rusted or not rusted. Visualizing the results, we get the low-level features detected by the model.

The images shown in figure 6.16 are given as input to the model

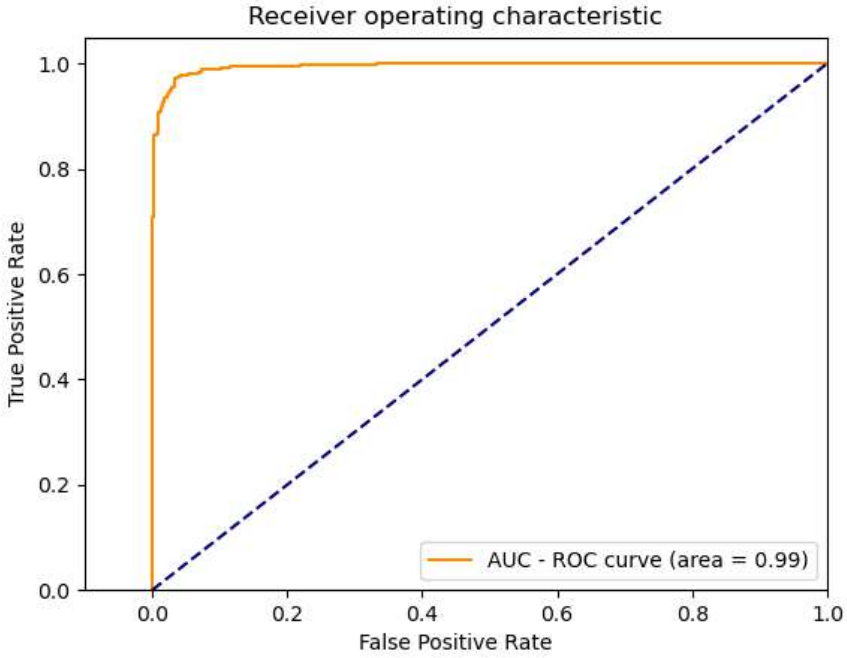
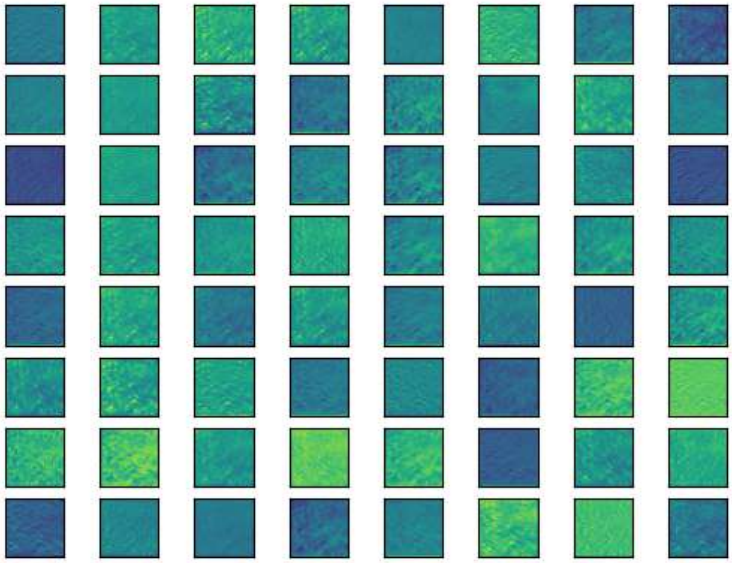



Figure 6.15: Model's AUC - ROC curve on test data

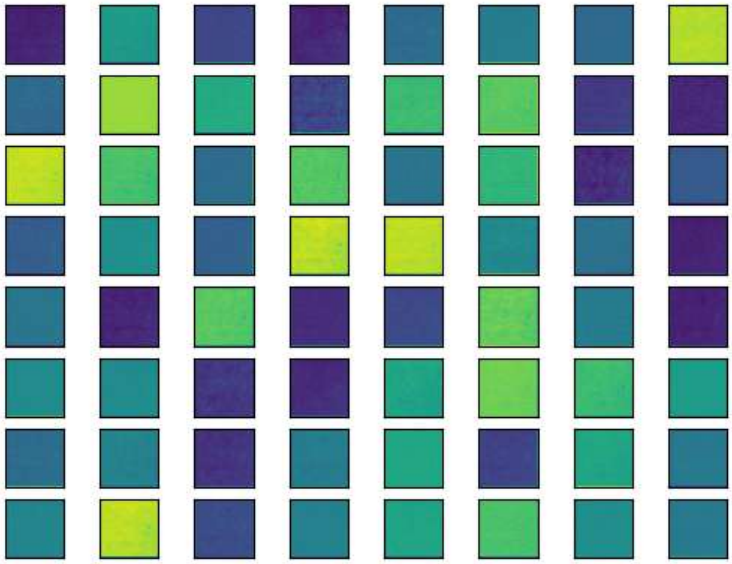
and the feature maps of the 2nd, 7th, and 10th layers, which are the first 3 convolution layers, are extracted. The feauture maps are visualized in figures 6.17, 6.18, 6.19. Note that the input layer has dimensions of $150 \times 150$, while the 2nd layer $75 \times 75$, the 7th $38 \times 38$, and the 10th $38 \times 38$. We see that the texture of the rusted image is activated as expected. The texture is also found on the output of 7th and 10th layer, meaning that it travels as information between layers and is not dropped. Therefore, by comparing the feature maps of Rust and NoRust images our hypothesis that the model does tracks the texture of the corrosion is supported.

(a) Rust (b) NoRust

Figure 6.16: Images used in feature maps visualization

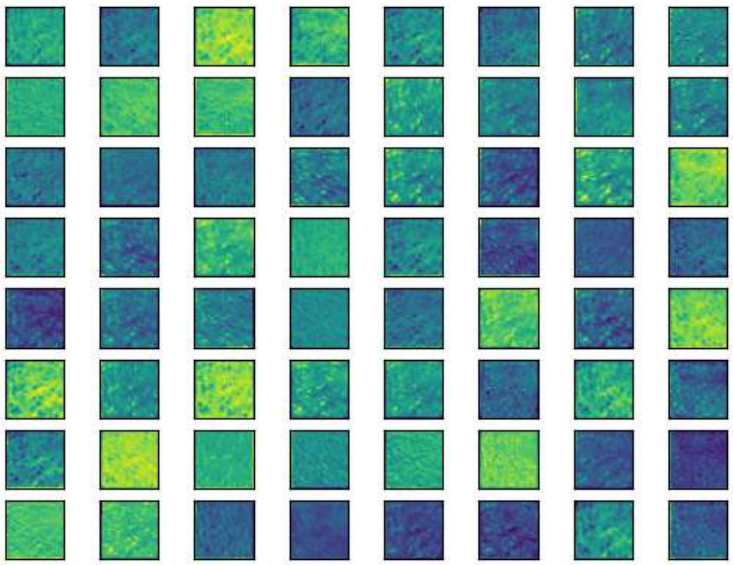


(a) Applied on Rust image

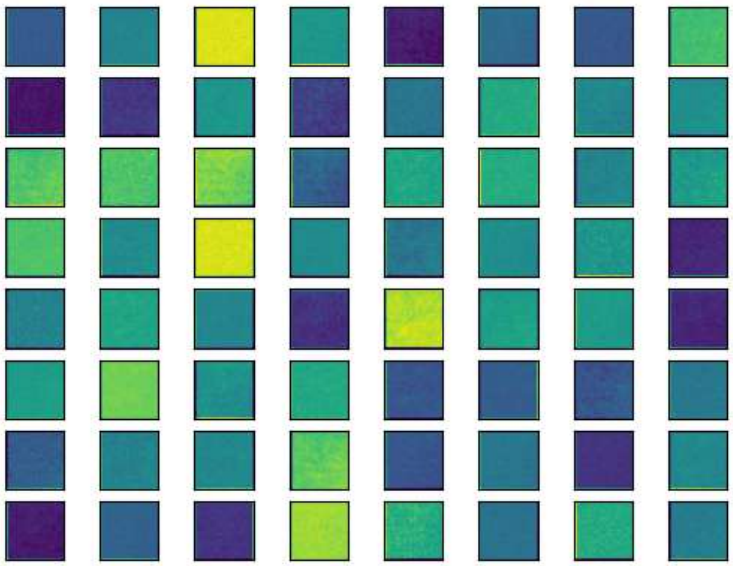


(b) Applied on NoRust image

Figure 6.17: ResNet50 - 2nd layer's 64 feature maps

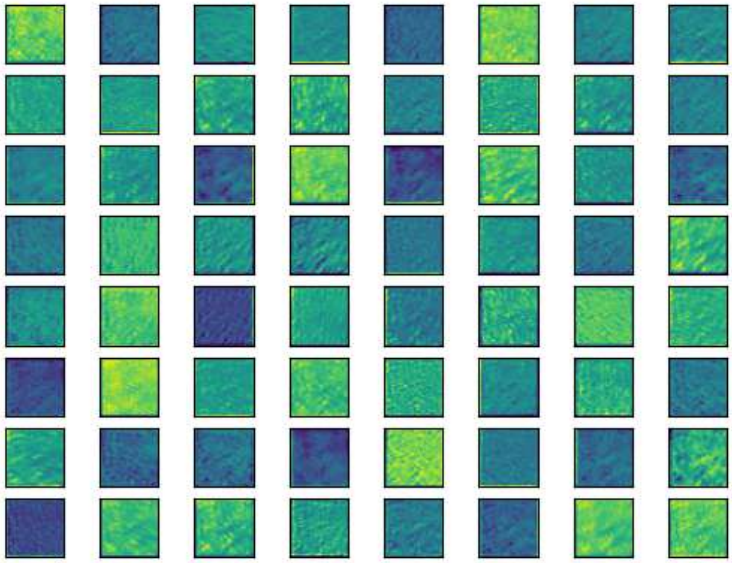


(a) Applied on Rust image

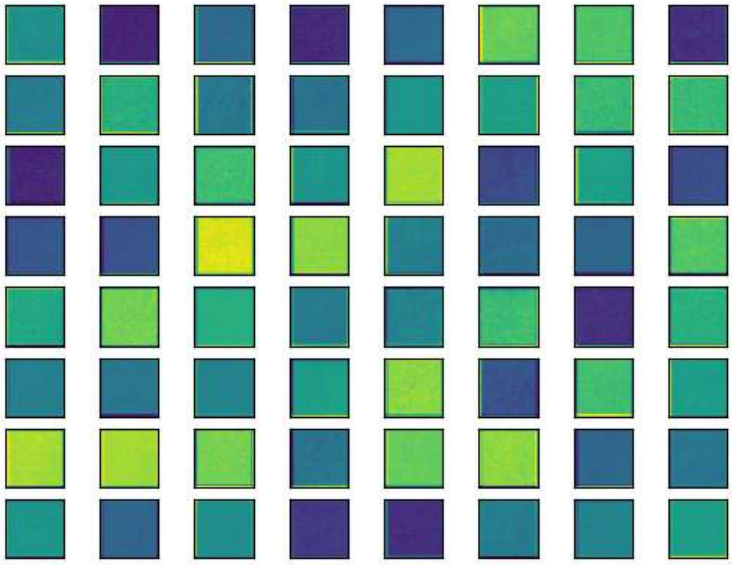


(b) Applied on NoRust image

Figure 6.18: ResNet50 - 7th layer's 64 feature maps



(a) Applied on Rust image



(b) Applied on NoRust image

Figure 6.19: ResNet50 - 10th layer's 64 feature maps

### 6.2.2 Model's deployment with sliding algorithm results

In this section the model is deployed utilizing the sliding algorithm described in chapter 5.2. It is tested on previous unseen images of the laboratory dataset. These images include both corroded and clean regions. The results are presented in figures 6.20, 6.21, and 6.22. The areas included in the blue rectangles are categorized by the model as corroded, while the rest of the image as not corroded. In the result, there is a margin on the right and bottom borders that is not tested, since their dimensions are smaller than the window the algorithm extracts.

The combination of the model and the sliding algorithm achieves detection and localization of corrosion in images containing both corroded and clean regions. However, in cases where the window of the algorithm does not land exactly on the corroded region it gets broken into pieces for other windows to track. Hence, parts of the region are usually misclassified. For this reason, other algorithms are needed that can perform a better search of the image, like an object detection algorithm. This approach is tested from this study as discussed in section 5.3.

Figure 6.20: Sliding algorithm result 1

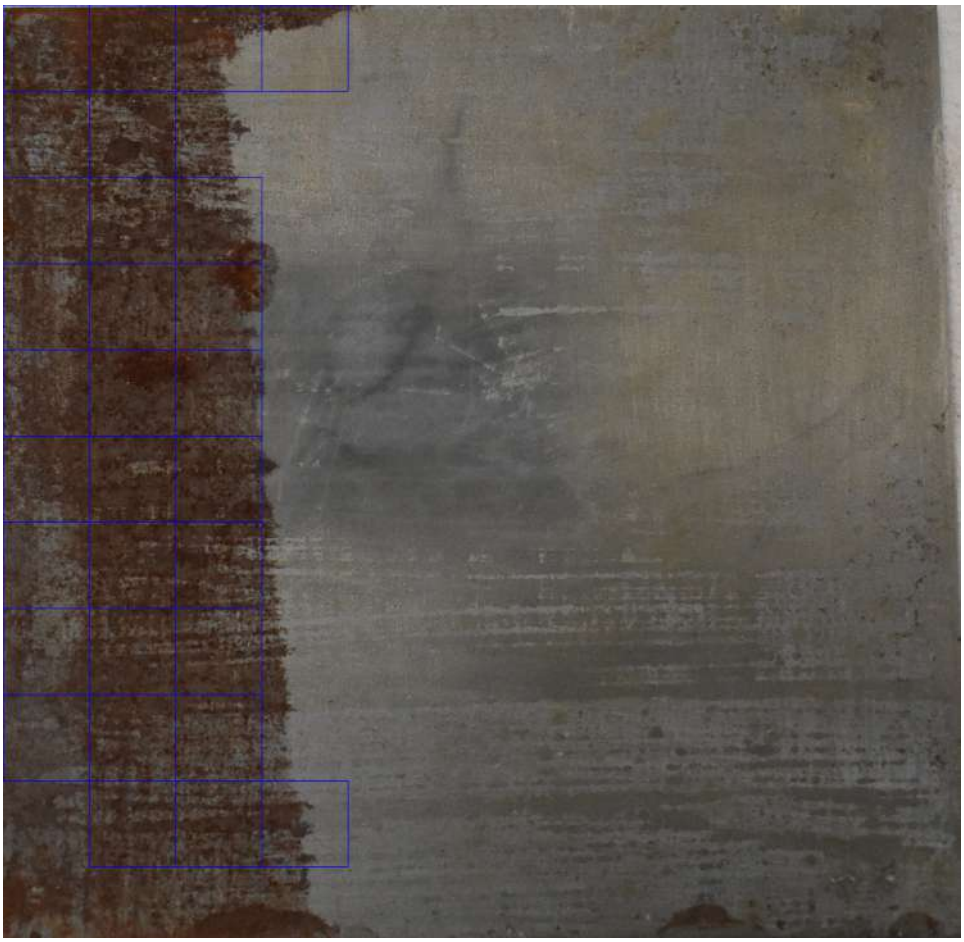


Figure 6.21: Sliding algorithm result 2

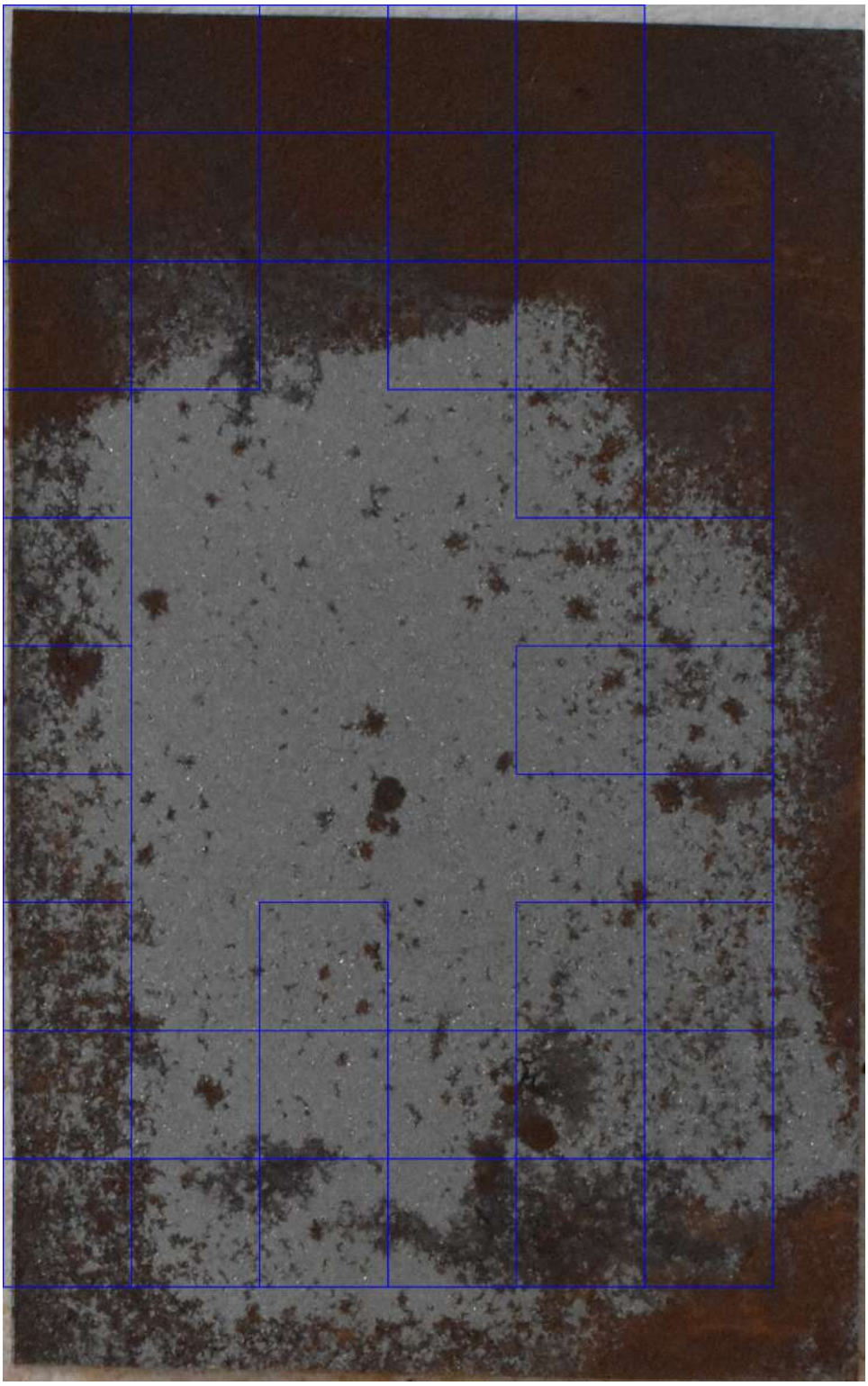


Figure 6.22: Sliding algorithm result 3

## 6.3 Corrosion Detection with Object Detection Algorithm Results

The model was trained using the surveys dataset. As all object detection algorithms, the Single Shot Detector with Mobilenet v1 requires great computational power. The model was trained on an Intel Core i7-7500U CPU. Hence, to reduce the load for the CPU from the 352 images of the dataset only 155 were used. They were split into training dataset with 130 images and test dataset with 25. The RMSProp optimizer was used for training the model with an exponential decay learning rate and a batch size of 10. The loss function is compromised of two terms. The first is a sigmoid cross entropy loss function that measures the confidence level of the prediction of the bounding box and is called classification loss. The second, the location loss, is a L1-norm that measures the offset of the bounding box from the ground truth. The total loss is the sum of the two losses with weights 1. To inflate the number of images since the training dataset is small, image augmentation operations were employed. The images were randomly flipped and cropped. Only one class was tracked, named "Corrosion". The model was trained for 173614 epochs for 9 days 6 hour 34 minutes and 25 seconds.

The model achieves a total loss of 1.793175 with a classification loss of 1.608378 and a localization loss of 0.143491. In figure 6.23 the losses are presented versus iterations. With total loss being constantly below 2 the model is considered acceptable. In the 25 images used for testing the model detected corrosion in 19 (76%) of them. The mean accuracy score in the images that corrosion was detected is 85%. Testing results are presented in figures 6.24 to 6.33. Overall, although the model is trained with a small amount of data it performs well.

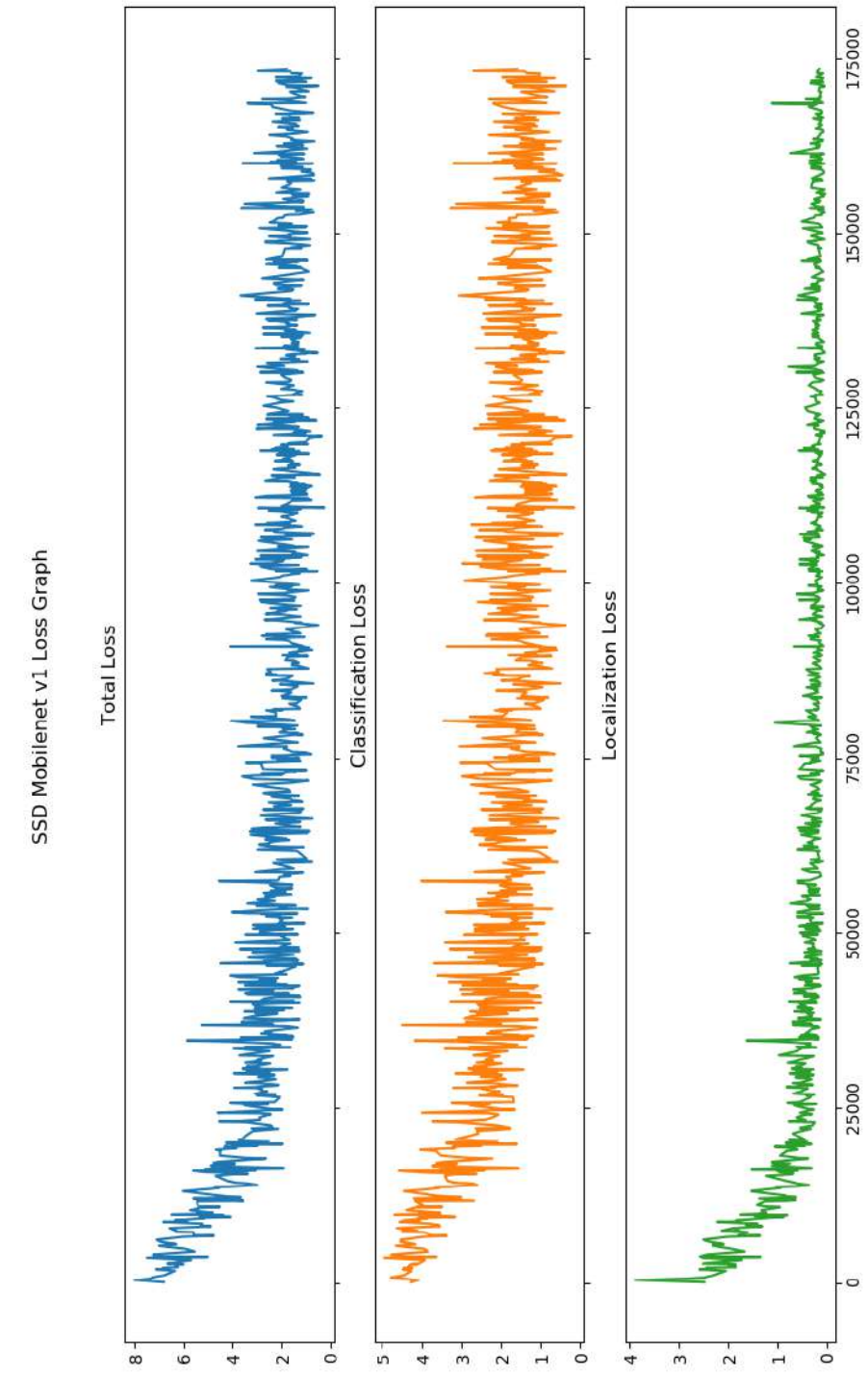



Figure 6.23: Loss graph of SSD Mobilenet v1

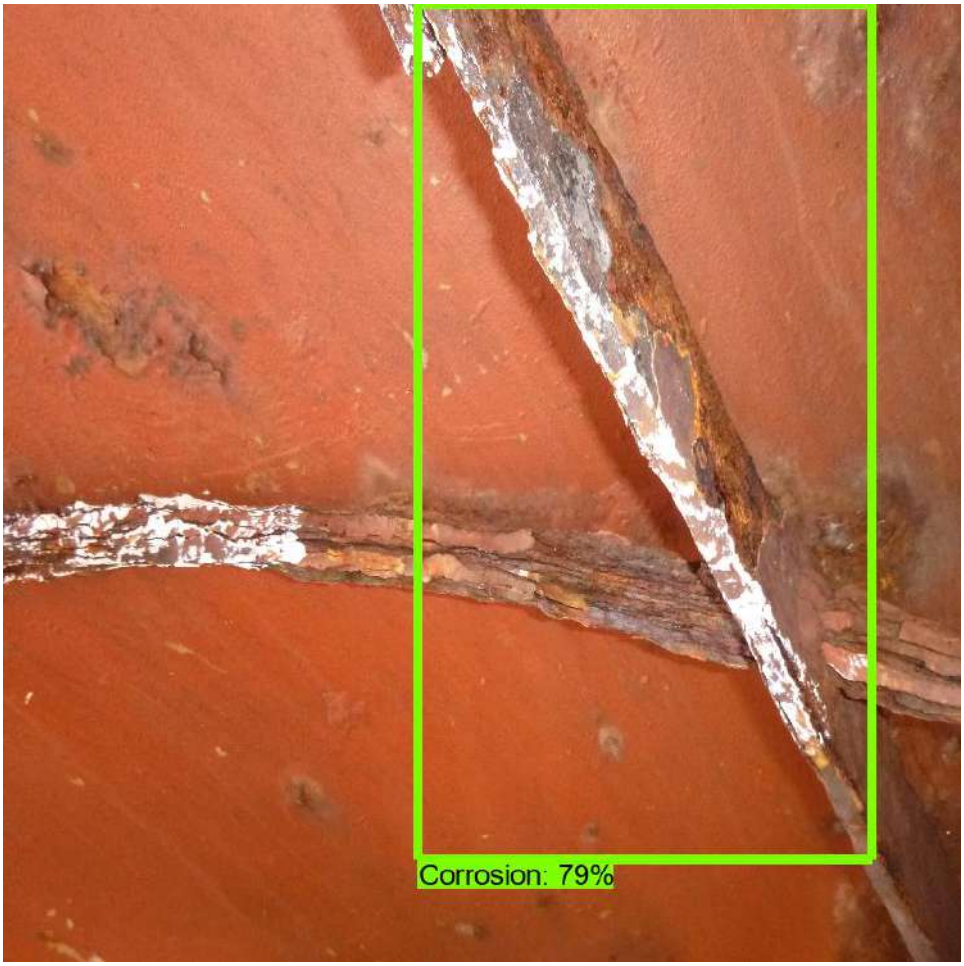



Figure 6.24: SSD Mobilenet v1 result 1


Figure 6.25: SSD Mobilenet v1 result 2

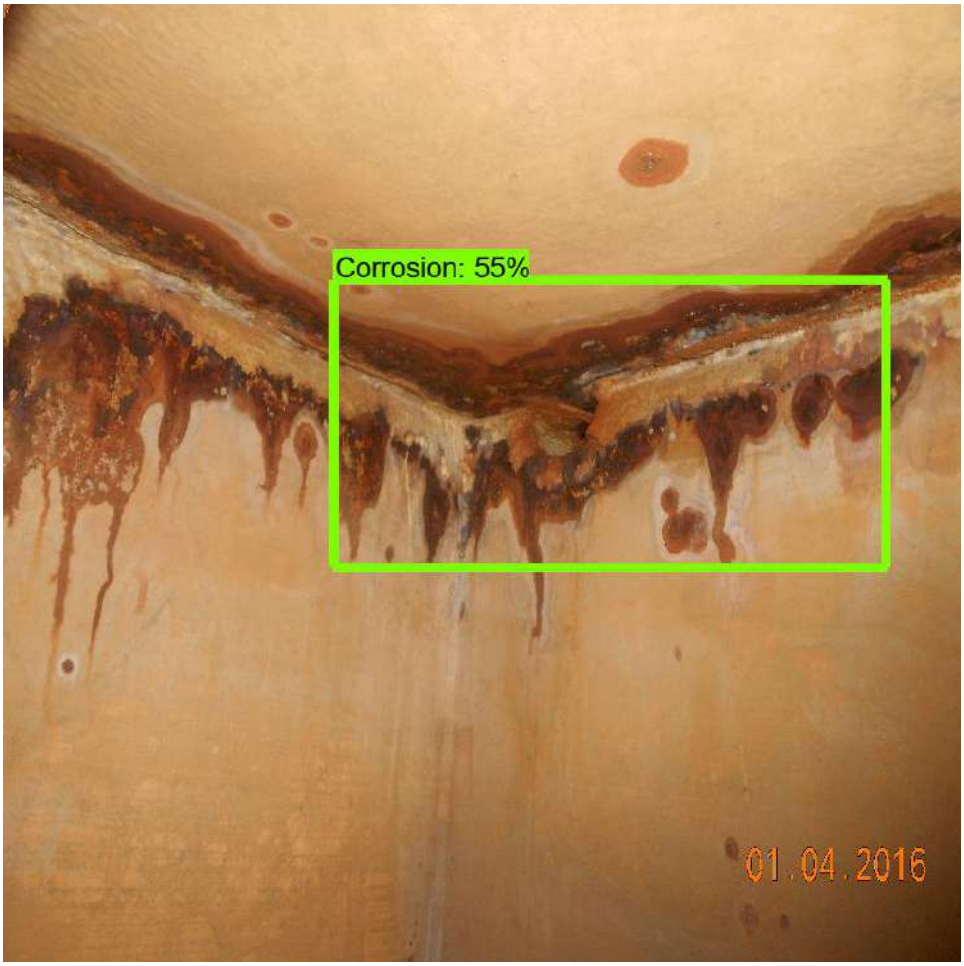



Figure 6.26: SSD Mobilenet v1 result 3


Figure 6.27: SSD Mobilenet v1 result 4


Figure 6.28: SSD Mobilenet v1 result 5


Figure 6.29: SSD Mobilenet v1 result 6


Figure 6.30: SSD Mobilenet v1 result 7


Figure 6.31: SSD Mobilenet v1 result 8


Figure 6.32: SSD Mobilenet v1 result 9


Figure 6.33: SSD Mobilenet v1 result 10

# Chapter 7

# Conclusion - Future work

This study focused on the topic of corrosion detection. The issue was approached taking into consideration the visual attributes of corrosion, which were identified as color and texture. Computer vision and deep learning algorithms were utilized to test three different solutions to the problem.

Computer vision processes were used to create a color detection algorithm that detects corroded metallic surfaces based on their color. The algorithm was tested on real world images depicting a bulk carriers' compartments. The images were taken during an inspection by seasoned surveyors. The results vary depending on which compartment the images depict. For example, in images of fuel oil tanks, the algorithm performs great, while on cargo holds that are usually painted maroon, a color similar to the one of corrosion, it fails to differentiate between the corroded region and the background. This means that the process is overly sensitive to the noise surrounding the region of interest. Therefore, the color tracking method can not be used as a universal solution to the problem. But due to its straightforward implementation and low computational power requirements, it can be applicable to certain compartments of a vesses yielding adequate results.

Considering the drawbacks of a color-based solution, the study also tested textured based solutions. For this purpose, deep learning algorithms were employed, who are well known for their ability to detect the texture of images. To support training of supervised learning algorithms two datasets were created. The first solution employed a Convolution Neural Network that used transfer learning from ResNet50. The CNN was trained on laboratory images for a binary classification problem, trying to classify rusted and not rusted metals. The model achieved great performance of 94% on test data. Its results supported the hypothesis that the texture of corrosion was detected. To scale up the abilities of the model a sliding algorithm was created that cropped images in pieces and fed them to the model. This way

corrosion was classified and located in the image. However, this approach failed to scale to real world images of vessels due to the noisy backgrounds surrounding corroded regions. The laboratory dataset that was used for the binary classification, assumed that a not corroded region is a clean shiny metallic surface. The problem with this approach is that corrosion is usually surrounded by other structural and general failures, such as cracks, deformation and coat failing, that could confuse the model. Hence, a better algorithm is needed that does not assume the nature of a non-corroded region. It is preferred to assume during training that everything surrounding the corroded area is not corrosion. Towards this end, corrosion detection was approached as an object detection problem. A Single Shot Detector with Mobilenet v1 was used and it achieved a loss of 1.6, 76% detection success on the test data and a mean accuracy score of 85% on the detection boxes of test data.

Overall, texture-based approaches are found to perform better than color-based ones. Deep learning algorithms achieve great performance on color detection with the object detection approach being superior to the binary classification approach. A key take away of this study is the importance of training data for deep learning models that are to be used in real world problems. Algorithms like the ones used are fine tuned in achieving great performance but this only solves one part of the problem. With corrosion being a state of the metallic surface, its attributes differ over different metals, conditions under which rust is created, the location of the rust in the vessel etc. Thus, a great generalization of image examples is required to consider all corrosion depictions that can be found onboard a vessel. Otherwise, a model that performs perfectly on certain vessel would fail to generalize on others.

Based on the above findings the following approaches are proposed for future studies on the visual detection of corrosion. First, the same methods could be used on different expanded datasets to test their validity on more complex examples. The models used for transfer learning and the transfer learning strategies that were employed could be altered. Also, an image segmentation approach is proposed. This operation ties every pixel in an image to an object. In the case of corrosion detection every pixel would be categorized to three classes, pixels belonging to the corroded region, pixels bordering the corroded region and surrounding pixels. As a result, the region will be located, and the shape would be extracted. This could be helpful in cases where small corroded regions are widely dispersed in the image and bounding boxes of traditional object detection algorithms fail to include them. Finally, it would be useful to extract the reduction of thickness that has occurred to the metal due to corrosion. With this information the corrosion could be classified as pitting or general corrosion, which are the main corrosion phenomena on ships, and based on CSR rules

a risk analysis of the corroded region could be performed. This is possible following approaches like the unsupervised monocular depth estimation [35] from researchers of the University College of London, where using data from stereo cameras algorithms learn to predict the depth of the image.

# Bibliography


[1] K. A. Chandler, *Marine and Offshore Corrosion: Marine Engineering Series*. Butterworth-Heinemann, 1985, ISBN: 0408011750.

[2] B. Zaidan, A. Zaidan, H. O. Alanazi, and R. Alnaqeib, "Towards corrosion detection system," *International Journal of Computer Science Issues (IJCSI)*, vol. 7, no. 3, p. 46, 2010.

[3] F. Bonnin-Pascual and A. Ortiz, "Corrosion detection for automated visual inspection," in *Developments in Corrosion Protection*, IntechOpen, 2014.

[4] L. Petricca, T. Moss, G. Figueroa, and S. Broen, "Corrosion detection using a.i : A comparison of standard computer vision techniques and deep learning model," vol. 6, May 2016, pp. 91–99. DOI: 10.5121/csit.2016.60608.

[5] V. Bondada, D. K. Pratihar, and C. S. Kumar, "Detection and quantitative assessment of corrosion on pipelines through image analysis," *Procedia Computer Science*, vol. 133, pp. 804–811, 2018.

[6] T. Gibbons, G. Pierce, K. Worden, and I. Antoniadou, "A gaussian mixture model for automated corrosion detection in remanufacturing," *Advances in Manufacturing Technology XXXII*, vol. 8, pp. 63–68, 2018.

[7] D. J. Atha and M. R. Jahanshahi, "Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection," *Structural Health Monitoring*, vol. 17, no. 5, pp. 1110–1128, 2018. DOI: 10.1177/1475921717737051. eprint: https://doi.org/10.1177/1475921717737051. [Online]. Available: https://doi.org/10.1177/1475921717737051.

[8] V. Hoskere, Y. Narazaki, T. Hoang, and B. Spencer Jr, "Vision-based structural inspection using multiscale deep convolutional neural networks," *arXiv preprint arXiv:1805.01055*, 2018.

[9] Y. Yao, Y. Yang, Y. Wang, and X. Zhao, "Artificial intelligence-based hull structural plate corrosion damage detection and recognition using convolutional neural network," *Applied Ocean Research*, vol. 90, p. 101 823, 2019.

[10] B. T. Bastian, J. N, S. K. Ranjith, and C. Jiji, "Visual inspection and characterization of external corrosion in pipelines using deep neural network," *NDT and E International*, vol. 107, p. 102 134, 2019, ISSN: 0963-8695. DOI: https://doi.org/10.1016/j.ndteint.2019.102134. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096386951930060X.

[11] W. T. Nash, C. Powell, T. Drummond, and N. Birbilis, "Automated corrosion detection using crowdsourced training for deep learning," *Corrosion*, vol. 76, no. 2, pp. 135–141, 2020.

[12] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[13] U. of Washington. (). "Image filtering," [Online]. Available: https://courses.cs.washington.edu/courses/cse576/11sp/notes/Basics2_white.pdf.

[14] P. Cattin. (). "Introduction to signal and image processing," [Online]. Available: https://miac.unibas.ch/SIP/pdf/SIP-02-Fundamentals.pdf.

[15] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[16] S. Haykin, *Neural Networks A Comprehensive Foundation Second Edition*. Pearson Education, 1999, ISBN: 81-7808-300-0.

[17] C. M. Bishop, *Pattern Recognition and Machine Learning*. Cambridge, UK: Springer, 2006, ISBN: 978-0387-31073-2.

[18] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[19] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[20] G. Hinto, N. Srivastava, and K. Swersky. (). "Neural networks for machine learning. lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude," [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[21] K. Q. Weinberger. (). "Machine learning for intelligent systems. lecture 12: Bias-variance tradeoff," [Online]. Available: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html.

[22] D. Berrar, "Cross-validation," in Jan. 2018, ISBN: 9780128096338. DOI: 10.1016/B978-0-12-809633-8.20349-X.

[23] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, and V. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, p. 292, Mar. 2019. DOI: 10.3390/electronics8030292.

[24] G. Aurélien, *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow Second Edition*. O'Reilly, 2019, ISBN: 978-1-492-03264-9.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21–37.

[30] *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[31] The Qt Company. [Online]. Available: https://www.qt.io/.

[32] Tzutalin, *Labelimg*, Free Software: MIT License, 2015. [Online]. Available: https://github.com/tzutalin/labelImg.

[33] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985, ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(85)90016-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0734189X85900167.

[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[35] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.