Image Processing for Industrial Structure Analysis

Sérgio Miguel Guerreiro Neves

Thesis to obtain the Master of Science Degree in

Engenharia Eletrotécnica e de Computadores

Supervisors: Prof. Paulo Luís Serras Lobato Correia
Eng. David Jardim

Examination Committee

27 November 2019

Acknowledgments

I would like to thank all my friends that helped me grow as a person and were always there for me during the good and bad times in my life, all of them have inspired me and I could not imagine doing this without them.

I would also like to acknowledge my dissertation supervisors Prof.Paulo Luís Serras Lobato Correia for his insight, support and sharing of knowledge that has made this thesis possible and Eng. David Jardim for being available to answer any questions I had. Also thank you to Axians and Instituto de Telecomunicações for providing me with everything I needed to complete this thesis.

Finally I would like to give thanks to my parents that supported me during the duration of my entire course, especially my mother that made so many sacrifices in order to make this possible, as well as everyone in my family for always encouraging me in my studies.

Abstract

As the world becomes increasingly more dependent on industrial infrastructures, structures like electrical power towers are very important for society to function properly, as the failure of these kinds of structure could affect thousands of people. This makes the inspection and maintenance of these structures a necessity.

The most common way of analysing structures, like electrical towers, is by a human inspector visually assessing the state of the structure, which is time consuming, expensive and potentially dangerous. However, with the increased usage of robotic systems, such as drones, which are unmanned areal vehicles, and high resolution cameras, it is possible to acquire images from structures in a faster and safer way. With this information it is possible to further analyse the images obtained from these structures in order to detect which images contain a significant amount of rust and should be evaluated by an expert.

This thesis proposes the construction of two different methodologies for detecting the amount of rust in structures, the first is an adaptation of a technique based on a sliding window and utilizing a convolution neural network for detecting rust regions, and the second is a methodology based on semantic segmentation for detecting how many pixels in an image are detected as "rust" and how many are detected as "structure". These two approaches will then be compared in order to determine their advantages and limitations, when trying to achieve a good performance for the desired task.

Keywords

Rust Detection; Deep Learning; Convolutional Neural Networks; Semantic Segmentation

Resumo

À medida que o mundo se torna cada vez mais dependente de infraestruturas industrias, estruturas como torres de energia elétrica são muito importantes para que a sociedade funcione adequadamente, podendo o fracasso deste tipo de estrutura pode afetar milhares de pessoas. O que faz com que a inspeção e manutenção destas estruturas seja necessária.

A maneira mais comum de analisar estruturas como torres elétricas é por um inspetor que avalia visualmente o estado da estrutura, o que consome tempo, é caro e potencialmente perigoso. No entanto, com o aumento do uso de sistemas robóticos, como drones, que utilizam veículos aéreos não tripulados e câmeras de alta resolução, é possível adquirir imagens de estruturas de maneira mais rápida e segura. Com estas informações, é possível analisar melhor as imagens obtidas destas estruturas, a fim de detectar quais imagens contêm uma quantidade significativa de ferrugem e devem ser avaliadas por um especialista.

Esta tese propõe a construção de duas metodologias diferentes para detectar a quantidade de ferrugem em estruturas, a primeira é a adaptação de uma técnica baseada numa janela deslizante e a utilização de uma rede neural convolucional para a detecção de regiões com ferrugem e a segunda metodologia é baseada em segmentação semântica de imagens de modo a detectar quantos pixels numa imagem são detectados como "ferrugem" e quantos são detectados como "estrutura". Estas duas abordagens serão comparadas de modo a determinar qual delas apresenta melhor desempenho na tarefa desejada.

Palavras Chave

Deteção de Ferrugem; Aprendizagem Profunda; Redes Neuronais Convulocionais; Segmentação Semântica

Contents

List of Figures

List of Tables

Acronyms

ADAM	adaptive momentum estimation
ANN	artificial neural network
AI	artificial intelligence
CbCr	chroma blue-difference/chroma red-difference
CNN	convolutional neural network
GDP	gross domestic product
FN	false negative
FP	false positive
GLCMs	gray level co-occurrence matrices
HSI	hue, saturation, intensity
HSV	hue, saturation, value
IoU	intersection over union
MSE	mean squared error
NN	neural network
ReLU	rectified linear unit
RGB	red, green, blue
TN	true negative
TP	true positive
YCbCr	luminace, chroma: blue, chroma: red

1 Introduction

Contents

This chapter presents the motivation of this thesis, why corrosion classification is important and how much technology has helped the research in this field. It also identifies the goals of this thesis, summarizes the thesis contributions and explains how the rest of the document is organized.

1.1 Motivation

Rust development in industrial infrastructures, such as electrical towers for energy transportation, telecommunication towers and wind turbines, is responsible for deterioration and subsequent failure (such as power outages) of these types of structures. The corrosion of structures also poses a large economic cost, being estimated that the global cost for corrosion in 2013 was US$2.5 trillion, which is 3.4% of the global gross domestic product (GDP) [9].

Besides the large economic aspect of industrial structure maintenance, since the most common practice is to deploy a person or group of people to the field in order to assess the condition of the structure, this also raises concerns about the safety of those involved in the inspection task, as these people may need to climb the structures to evaluate their conditions and often make a second climb when it is determined that some kind of intervention is needed to fix the corroded or rusted area.

In recent years, with the development and usage of aerial vehicles, such as drones, it is possible to reduce the amount of personnel that is put in hazardous conditions for performing the inspection of infrastructures. Nonetheless, the amount of work that specialists must undertake in order to detect dangerous levels of rust in industrial structures could still be further reduced. Even though the usage of drones reduces the amount of time necessary to access the state of a structure, there are still a large amount of captured images that is. Therefore, these workers must still go through large amount of irrelevant images of the structure captured by the drone to identify structure areas that need maintenance interventions.

1.2 Objectives

This thesis analyses several methods of detection of rust in images. The goal is to develop a system that is capable of making an automatic detection of rust in the imaged structure, as a way to reduce the amount of images that an expert needs to go through when assessing the state of industrial structures. The images are taken from different distances and on structures with varied background, which increases the difficulty of detecting which regions are structure and which regions are rust.

1.3 Contribution

This thesis compares different methodologies for rust detection that include color analysis, color and texture analysis and deep learning approaches. This comparison led to the testing of two different approaches:

- Block-based labeling, using a convolutional neural network; a similar methodology has already shown promising results detecting rust and thesis will expand this approach to detecting structure as well, from this the amount of windows classified as "rust" and "structure" can be counted, making it possible to analyse how much of the structure contains rust.
- Pixel based labeling; this approach uses a SegNet network in order to generate a mask of the original images, where each class ("rust", "structure" and "background") is classified by a different color. From this mask it is possible to then calculate how much rust there is in the structure, by counting the amount of pixels for each class.

By implementing these two methodologies, this thesis hopes to compare the results of these two different approaches. Although the block-based approach has already been used in this context, it was not used to detect structure and the SegNet approach, to our knowledge, was never applied to this type of problem.

1.4 Organization of the Document

Chapter 2 reviews the available literature, briefly discussing the methods that have been developed to detect corrosion in images; it also presents the concept of deep neural networks, with emphasis on CNNs and provides some detail about how CNNs work. Chapter 3 describes the implementation of the first proposed solution, which is based on the usage of a CNN, analysing the input images using a block-based approach. Chapter 4, explains the implementation of the second solution developed, based on the SegNet architecture; the differences between this solution and the block-based approach are also described. Chapter 5 describes the experimental work, reporting the results obtained with each of the developed approaches. The obtained results are discussed and conclusions are drawn. Chapter 6 concludes this Master thesis, making a summary of the work done and discussing directions for future work.

2 Detecting Corrosion

Contents

This chapter will take a look at what corrosion is and what kinds of techniques are used to detect corrosion on industrial structures, as well as discussing which existing methodologies are best suited to address our problem.

2.1 Corrosion

Before trying to detect corrosion, one must understand what corrosion is and more importantly what kind of corrosion should to be detected.

Corrosion can be defined as the degradation of materials' properties due to interaction with the environment. Even though several approaches are available for controlling corrosion, such as the application of protective coatings, addition of chemical elements to the environment and treatment of the surface of the metal to increase its resistance to corrosion, it remains a fact that the corrosion of most metals is inevitable [10].

As indicated in the previous chapter, the kind of corrosion that will be tackled in this thesis is rust (the corrosion of iron). Rust is the common name for iron oxide, Fe_2O_3. This chemical compound occurs from the combination of iron with oxygen, a combination so common that pure iron is rarely found in nature. An example of the kind of corrosion that should be detected is given in Figure 2.1.

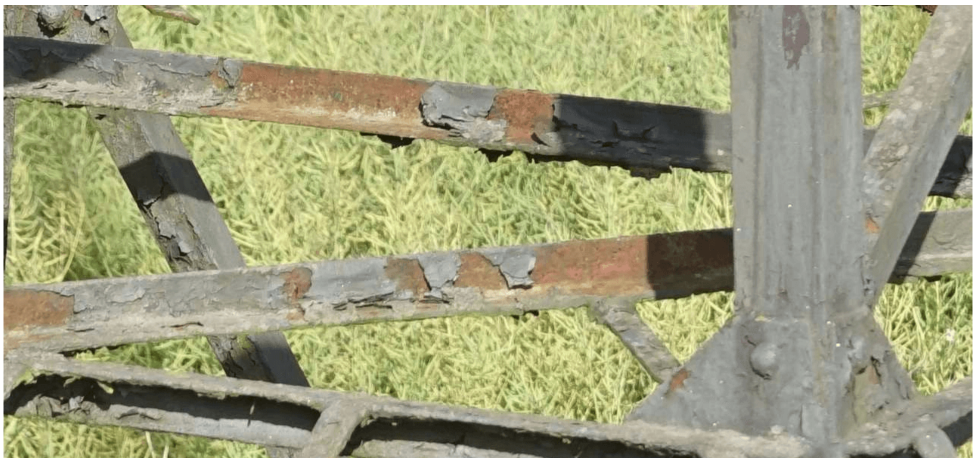


Figure 2.1: Corrosion of on an electrical tower (image provided by Axians).

The corrosion of materials can occur through a direct chemical reaction of a metal with its environment or due to the flow of electricity in a electrochemical reaction in an aggressive medium [6]. These corrosion environments consist of four types: atmospheric exposure, which can occur in rural, marine (for example near the sea) and industrial environments; aqueous exposure, which can occur where there is direct contact with natural or industrial waters; underground exposure, which can occur in underground environments, such as installations for piping or on the underside of structures; industrial environments, under the influence of organic or inorganic synthesis and other processing environments under the influence of liquids or gases [6].

The main effect of corrosion is that, over time, it can cause structural flaws, making the structures more fragile which in turn can lead to the malfunctioning or even the fall of electrical towers. Structural flaws due to corrosion are characterized differently depending if they affect the surface or the interior of the structure, Table 2.1 depicts some common methods used in nondestructive structure evaluation for the detection of rust.

Evaluation Methods	
Surface Rust	Interior Rust
Visual	Magnetic field
Liquid Penetrant	Radiography
Eddy current	Energy dispersive X-rays
Magnetic particle	Microwave
Ultrasonics	Ultrasonics
Acoustic emission	Acoustic emission

Table 2.1: Table with examples of non-destructive rust detection and characterization methods (adapted from [6]).

This work will focus on exploring several methods based on visual inspection, which is one of the most unintrusive methods and is commonly employed as the first step in evaluating a rusted area. Even though visual inspection does not give the most accurate depiction of the real condition of the structure, a majority of the time a visual inspection is done as a preliminary analysis of the structure condition.

There are several aspects that visually indicate the presence of corrosion, for example, in the case of rust, it can easily be seen that rusting leaves a clear red coloration on steel and creates coarser texture when compared to non-corroded areas. Looking at these characteristics, it is possible to see how methodologies based on color, texture and shape could be used for detecting rust.

2.2 Image-based Rust Detection Methodologies

There have been several studies for detection and classification of objects, people and animals in images [8, 11–15], however the number of these studies that focused on image-based corrosion detection is limited. Often image-based rust detection systems rely on some sort of color analysis, a combination of color and texture analysis, possibly resorting to deep learning approaches [7, 8, 16, 17]. Other systems use techniques based on shape detection, such as using a stereo-adapter on a regular camera to measure the 3D shape of the corroded area to detect subsurface corrosion [18], or a system focusing on the pitting corrosion of aluminum, depending on the optical contrast of the corroded regions when compared to their surroundings and using edge detection to compute each closed and corroded region [19], these methods have some problems that make them inapplicable for this thesis. In the first case, subsurface corrosion can be identified by changes to the surface shape [16], this method needs, as mentioned previously, a camera with a specific adaptor, which is not ideal, given that the goal is to use footage captured by drones. As for the second approach, pitting is not a dominant feature of most types of regular corrosion, being a more common occurrence in aluminium corrosion, in particular rust's dominate features tend to be discoloration and texture degradation [8, 17], as such this technique is very specific and not useful for rust detection.

2.2.1 Color Analysis

What first jumps to mind when thinking about the physical characteristics of rust and corrosion in is the discolouration. Having said this it is no wonder that many of the first works on corrosion detection were based on color analysis. These techniques showed some promising results, but had some underlying flaws.

A fairly straightforward approach was done by Petricca *et al.* [7], with the goal of comparing the results of more standard computer vision techniques with deep learning approaches in the detection of steel corrosion. In order to develop the classic computer vision technique, a series of image preprocessing methods were applied focusing on the red component of the rust, this was done using the OpenCV library. In order to detect rust, the images suffered a series of filtering and a change to the HSV color space. After some testing a range of intervals for rust in the HSV color space was defined, and later the image was segmented, to create a black and white mask, where the white pixels would correspond to the positions in the image where rust was detected. If the amount of white pixels in an image is superior to a certain percentage, then it is determined that the image is labelled as containing rust. The largest problem this method faces is how easily it can be mistaken. Petricca *et al.* even mentions that, since the system is only looking for red pixels, if an image of a red apple passed through, it would be classified as rust. In figure 2.2,example of the data obtained from the using the HSV color space can be seen.

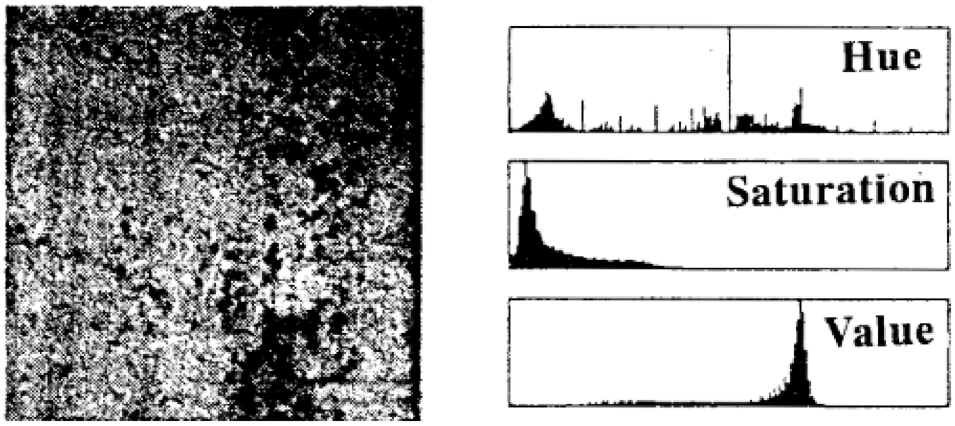


Figure 2.2: Results of image processing in HSV color space (adapted from [1]).

From these results some failures of the color analysis method can be seen, since it can misclassify an apple as rust it is quite possible that objects such as red leafs could also be misclassified, which immensely hinders its practicality. Yet one could argue that this approach was fairly simple and prone to error, as such further exploration on the works published in this field should be done.

In his paper, Furuta [1], developed a rust detection system that extracts the deteriorated area, determining the degree of deterioration of the image. For this the HSV color space of the image is analysed and information of each of the components (hue, saturation and value) is extracted. In order to extract meaningful data from the HSV, individual thresholds are needed: the hue threshold was defined from red (0) to yellow (42), according to Furuta; the saturation was difficult to narrow down, so it was assumed that hue and value were enough to extract the rust area; the value of the threshold had to be defined for each image using a neural network, since this threshold varies considerably from image to image.

Another work was published by Lee [2], utilizing a statistical analysis of the RGB color space. In this method images from a dataset were split into two sub sets, defective and non-defective and converted to scatter plots in the RGB color space as seen in Figure 2.3, from which several statistical data or variables were obtained. These variables were later analysed to select relevant ones to classify defective and non-defective images. From this analysis three variables were obtained: mean for the red component, difference for the green component and difference for the blue component. Using the three variables, multivariate discriminant models were created. The discriminant model used showed to be able to separate the defective and non-defective images in the validation phase.

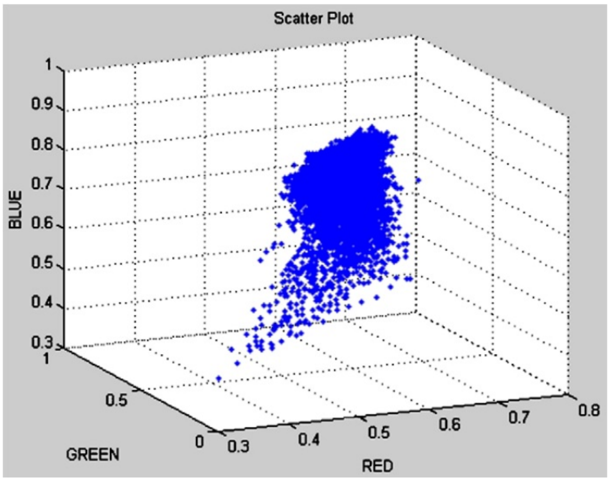


Figure 2.3: Image conversion to scatter plot in the RGB color space (taken from [2])

Even though they showed some success, color based systems have been shown to perform worse than systems based on a combination of texture and color, as illustrated in Medeiro's *et al.* paper [20], as discussed in the following subsection.

2.2.2 Color and Texture Analysis

The approaches that are based on color and texture analysis comprise the majority of image based corrosion detection techniques and these can be further divided into statistical and filter-based approaches [17].

Statistical Approaches

A great number of the statistical based approaches can be seen utilizing gray level co-occurrence matrices (GLCMs), which give information on how often different combinations of pixel gray levels occur, mainly to extract statistical texture features [21]. These works typically use these GLCMs to obtain texture information and statistical analysis on HSV or hue, saturation, intensity (HSI) color spaces, similar to what is done in color analysis.

Taking as an example Medeiro's *et al.* [20] paper, in it GLCMs are used to obtain several texture features or attributes, from which four were utilized: contrast, correlation, energy, and homogeneity. Then the HSI color space was used to obtain color features. From these it was defined that: the hue (H) for a corroded surface lies between yellow and red wavelengths; the saturation (S) of a corroded area tend to be higher than other areas; metallic surfaces are usually gray or white, tending towards a whiter wavelength, which in turn leads to high intensity (I). By using a combination of the subsets of color and texture features, it was verified a better performance when compared to the results obtained from each individual subset.

Choi [22] created a system that was also based on obtaining color features through the HSI color space and GLCMs for obtaining texture features. In this approach, principal component analysis and varimax techniques were applied to the HSI color space in order to optimize the set of attributes, while eliminating the insignificant ones and minimize even further the attributes. For obtaining texture features, a method of GLCMs based on the azimuth difference of points on the surface was used.

Some downsides of using GLCMs, as pointed out by Xie [23], include the challenge of reducing the number of gray levels to keep the co-occurrence matrix from being too big of a computational burden, finding a number of entries for the matrix that maintains statistical reliability and optimizing the displacement vector.

Filter Based Approaches

Filter based approaches, as the name implies, are based on applying filter banks to the image and computing the energy of the response to those filters [23]. In corrosion detection, the most relevant and widespread techniques that utilize filter banks rely on the wavelet transform [17]. This is due to the wavelet transform having the capability to provide understanding in both the spatial and frequency domain of an image [17, 23]

Ghanta [24] develop a system for rust defect detection on steel bridges based on using Haar wavelet. This method involves separating the training data into either pure rust or non-rust, using the Haar wavelet to transform each of the RGB components of the images into the wavelet domain. Then entropy and energy were calculated for each of the four sub-bands (LL (low-low), HL (high-low), LH (low-high) and HH (high-high)) of the three components of the RGB color space (R,G and B), wielding a total of 24 features. The average luminace of the image is then calculated, resulting in a new feature, thus creating a final vector of 25 features. Principal component analysis is used to reduce the dimensionality of this feature vector from 25 to 5. Since the rust and non-rust classes are separable, the least mean squared method is used for creating the classifier. After all this is done the detection of rust on an image is done by obtaining an $8 \times 8 \times 3$ region of an image and the process described previously is calculates a feature vector of dimension 5. Then, using the designed classifier based on the least mean squared method, the system classifies that region as rust or non-rust based on the feature vector calculated. Testing proved this approach to be effective at detecting rust, however rust regions that are smaller than an 8×8 region will not be detected as rust, however this is very a small window and high resolution images are becoming increasingly more common.

Jahanshahi [17] expanded the research done on corrosion detection using wavelets by applying depth perception and different color spaces in order to improve reliability and performance. A depth index that represents the actual size of a pixel of a given image was defined in order to obtain depth perception, this was calculated using a model for a pinhole camera where the following values are needed: working distance; camera focal length; camera sensor size and camera sensor resolution. From these values, only the working distance cannot be obtained with existing information from either the image itself or the camera manufacturer, and therefore has to be measured or estimated. This causes a bit of a problem when different cameras are used and when measuring the distance of the camera to the subject is not possible. Another improvement was the utilization of different color spaces. Through testing it was found that the best color space with best results in texture analysis is the chroma blue-difference/chroma red-difference (CbCr). The classification of the regions as corroded or non-corroded is done through a neural network, where the inputs are the features computed based on the wavelet filter bank. This method created results better than any of the previous techniques, being, in essence, an improvement on the previous wavelet-based approaches.

2.2.3 Deep Learning Approaches

Rust detection is a niche field when compared to other types of object detection, meaning that the amount of works that focus on deep learning based approaches for rust detection is quite small. Nonetheless, there have been some studies realized in this area the show the effectiveness of these kinds of approaches. Later, a more detailed information about deep learning will be given, explaining the fundamental concepts that make up these methodologies. As such this section will focus on the results and overall methodologies that use deep learning fro at the moment.

As mentioned previously, Petricca *et al.* performed comparisons between color and deep learning based approaches. The deep learning method used is based on using a pretrained model of the AlexNet [25], this network showed how promising convolutional neural networks could be for image classification when it won the ImageNet [26] challenge in 2012. Nonetheless, this is a fairly old architecture and as been surpassed in said challenge by numerous others. By fine tuning, re-training the already trained network with new images, the network is able to learn high level features and not lose time leaning low level ones. As it can be seen, in Table 2.2, the deep learning approach in almost ten percent more effective than the color based one when comparing the total accuracy.

	Color	Deep Learning
False Positive	27/63	14/63
Partial accuracy for "non-rust"	57%	78%
False Negative	4/37	8/37
Partial Accuracy for "rust"	89%	78%
Total Accuracy	**69%**	**78%**

Table 2.2: Color and deep learning method comparison (adapted from [7]).

A method that has shown to surpass the previous state of the art developed by Jahanshahi, is the one developed by Atha [8]. This method is based on the use of a pretrained CNN that showed good performance in the ImageNet challenge, the VGG [27] networks, and fine tuning it to better detect corrosion. Detecting corrosion on the image was done applying said pretrained network to a portion of an image and classifying that portion as corroded or not. The portion of the image is obtained through the use of a sliding window, where the size is predefined by the user. In this paper it was also tested the use of several color spaces, since it was already shown in previous works (such as Jahanshahi's [17]), as well as different architectures of CNNs and sizes for the sliding window. From this it was concluded that, from all the tested parameters (network, window size and color space), the most robust combination of parameters was: a sliding window of 128×128 pixels, on either the RGB or luminace, chroma: blue, chroma: red (YCbCr) color spaces and VGG16 (VGG network with 16 hidden layers). However VGG16 proved to be the more computationally heavy network, taking a long time to train when compared to others. The comparison of the results obtained utilizing VGG16 with the wavelet and neural network approach developed by Jahanshahi can be seen in Table 2.3. In this Table the overall superiority of using a CNN can be seen, as it is able to beat the previous state of the art in recall (the amount of regions correctly classified as corrosion divided by the sum of regions correctly classified as corrosion and regions wrongly classified as non-corrosion), precision (the amount of regions correctly classified as corrosion divided by all regions classified as corrosion) and F1 score (the harmonic mean of recall and precision).

Algorithm	Color Space	Recall (%)	Precision (%)	F1 Score (%)
VGG16	RGB	98.31	98.64	98.47
Wavelet NN	RGB	93.01	93.65	93.32
Wavelet NN	YCbCr	85.41	89.85	87.53
Wavelet NN	CbCr	75.85	84.87	80.03

Table 2.3: Deep learning performance compared to the performance of wavelet and neural network for corrosion detection (adapted from [8]).

Taking into account the results of this research, it can be seen that methodologies based on deep learning approaches have proved highly effective and were even able to surpass previous state of the art approaches. Given this, the methodology of this thesis will be based on a deep learning approach.

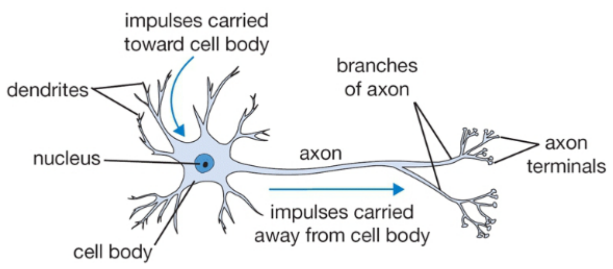
2.3 Deep Learning Overview

The solution to be developed on this thesis will be based on deep learning methods,therefore, this sections discusses deep learning basic concepts, its history and how it works.

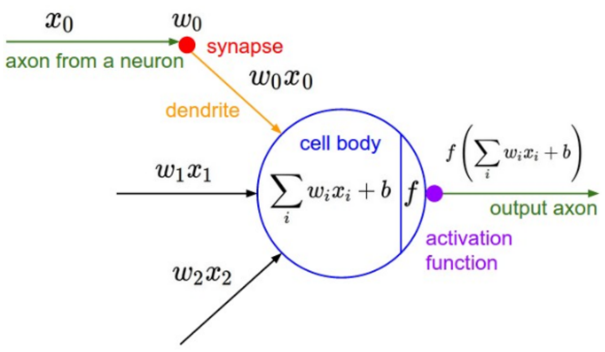
Deep learning is a field of artificial intelligence (AI) that has seen its interest rise in recent years. Despite the current interest and heavy usage of neural networks (NNs), the research in this field is by no means new, being that the first steps towards an artificial neural networks (ANNs) came in 1943, when Warren McCulloch and Walter Pitts wrote a paper on how the neurons might work [28], introducing a computational model for NNs. It would however take years for NNs to reach the levels that they are at now, due to the computational complexity of these methods. Nonetheless as more powerful Graphics Processing Units and large amount of labeled open source datasets became available (such as ImageNet [8] and Microsoft COCO [29]), made NNs the success that they are today.

2.3.1 Artificial Neural Networks

Artificial Neural Networks are, as mentioned before, mathematical models that were inspired in the biology of the human brain, trying to mimic the way biological neural networks process information. ANNs are constituted by a combination of neurons. A neuron (also known as a node), as represented in Figure 2.4(a) [3], may be connected to several other neurons thus creating a network. The input of each neuron (x_i) has a weight associated to it (w_{ij}), it is this weight that determines the "importance" of that input relative to the rest. The idea is that these weights are learnable and control the strength of influence that one neuron has on another [3]. These weighted inputs are then summed together and that this phase a bias (b) will also be summed, this bias component is essentially a constant that enhances the flexibility of the neuron, in Figure 2.4(b) it is possible to see the mathematical representation of this


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

(a) Biological model of a neuron



(b) Mathematical model of a neuron

Figure 2.4: Comparison of Biological and Mathematical Neuron models (adapted from [3])

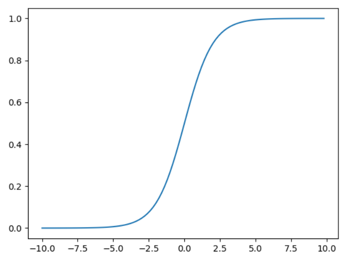
neuron.

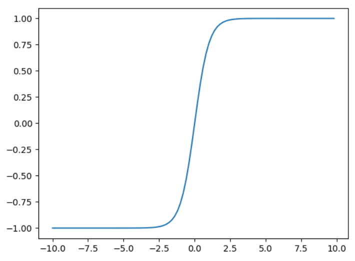
The activity of the neuron is given by the expression seen in Figure 2.4(b), where f is a non-linear function that determines the output of the neuron, this function is called an activation function. Every activation function takes a single value as an input and performs a certain fixed mathematical operation on it, where the most commonly used activation functions are the Sigmoid, the hyperbolic tangent (*tanh*) and the rectified linear unit (ReLU), the graph for these functions can be seen in Figure 2.5.

Presently the most used activation function is the ReLU. It was found that this function greatly increases the rate of convergence of the stochastic gradient descent when compared to the sigmoid/tanh function (as much as by a factor of 6 in some cases [25]), also it involves less expensive computational operations, since the ReLU can be implemented by simply thresholding a matrix of activations at zero. However, because of this zero threshold, it is possible for the ReLU function to "die". For example, if a large gradient passes through a neuron, it could cause the weights to update in a way that the neuron will never update on any datapoint again. This is usually the case when the learning rate is set too high, and a proper setting of the learning rate should make this a less frequent problem [3].

2.3.2 Neural Network Architecture

Neural networks are collections of neurons that are connected in an acyclic graph [3]. This means that the output of some neurons in the network are the inputs of other neurons without any loops, as seen in Figure 2.6. A network is usually comprised of organized layers of neurons, where each neuron of a layer is connected to another neuron of the next layer. The most common type of layers for regular NN is the fully connected layer, in which neurons of adjacent layers are fully pairwise connected and no neuron of the same layer is connected to each other [3]. An example of this topology can be seen in Figure 2.6.An ANN is usually made from an input layer, one or more hidden layers and an output layer.


(a) Sigmoid activation function.


(b) tanh activation function

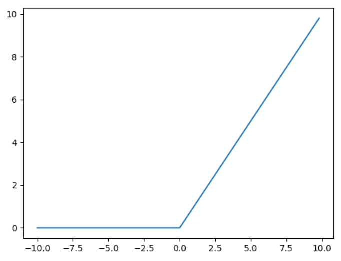

(c) ReLU activation function.

Figure 2.5: Graphs of some of the most used activation functions.

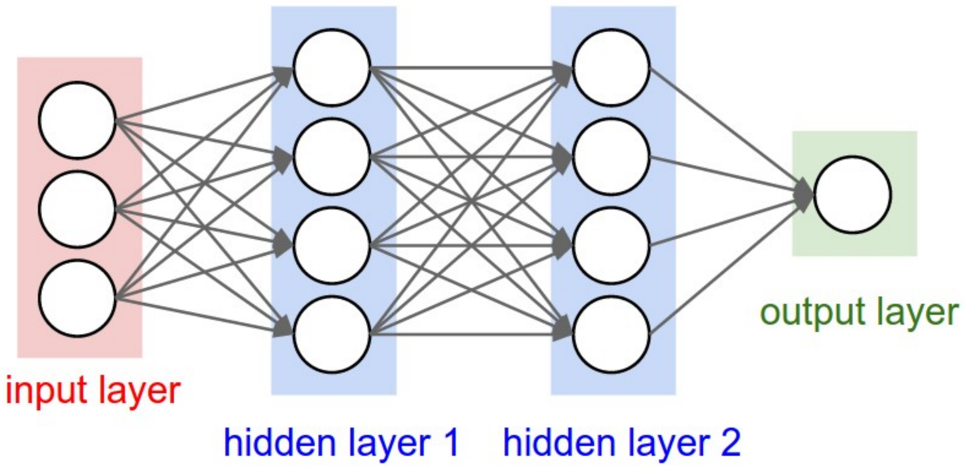


Figure 2.6: Example of a 3-layer fully connected neural network (adapted from [3]).

The input layer receives data from the external environment and passes it down to the first hidden layer, the hidden layers then are then responsible for extracting patterns from the analyzed data.

It is in these hidden layers that most of the computation of the network is and each hidden layer is connected to another hidden layer or, if it is the last hidden layer, to the output layer. The amount of hidden layers is one of the design patterns to consider when creating a NN [30], there can be as few as zero hidden layers, making a network a simple direct mapping of the input nodes to the output nodes. In contrast modern convolutional networks are made up of more than 20 layers (this is what gives the name *deep learning*) [3].

The final layer of a neural network is the output layer, which does not commonly have an activation function since the output values of this layer are usually taken to represent the class scores (in the case of classification) or some kind of real-valued target (in the case of regression) [3].

The size of a neural network can be defined in one of two ways: the number of neurons can be used or, more commonly, the number of parameters. Taking the network represented in Figure 2.6, it is possible to see that the number of neurons is given by the sum of the number hidden layer neurons (four in each hidden layer) and output neurons (just one in this case), which gives a total of 9 neurons. The number of learnable parameters is given by the sum of all the weights (one for each connection/line in the Figure) and all the biases (9 since the input layer as no bias), in this case the number of learnable parameters would be $(3 \times 4) + (4 \times 4) + (4 \times 1)) = 32$ weights and $(4 + 4 + 1) = 9$ bias, this means 41 parameters.

2.3.3 Loss Function

This section will tackle two important concepts that are used in the training process of NN and that are closely related to one another.

The loss or cost function is an integral part of ANN, since it is used to measure the disparity in the predicted output $\hat{y}$ and the expected output, usually called label in classification cases, y. Broadly speaking, the loss function indicates the performance of the network for a certain task, where one of the most used ones is the mean squared error (MSE), shown in equation 2.1.

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.1}$$

In this case *N* is the number of neurons in the output layer. Computing this loss function will result in a certain real number, this number gives the performance of the NN, where the lower the number the better the performance of the network in the given task. The objective is, for a certain training data, minimizing the cost function by tweaking the learnable parameters of the network.

Another very utilized loss function is categorical cross entropy, seen in 2.2.

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{j=1}^{N} \sum_{i=1}^{C} (y_{ij} \times log(\hat{y}_{ij})) \tag{2.2}$$

In this formula, *N* is the total number of observations (or samples), *C* is the amount of existing classes, y_{ij} is the indicator function of observation *i* belonging to class *j* and $\hat{y_{ij}}$ is the probability outputted by the network of the observation *i* belonging to class *j*. When the number of classes is more than two, the output of the network will be a vector of size *C*, where each element of the vector is the probability of the input of the network being the respective class, for example $C = [0.1, 0.8, 0.1]$, would mean the probability of the input being $C[1]$ is 80%. When the number of classes is two, the network simply outputs a probability $\hat{y_i}$, defining the second class probability as $1 - \hat{y_i}$, this is a special case defined as binary cross entropy, which can be obtained from 2.2, resulting in 2.3.

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{j=1}^{N}(y_i \times log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i)) \tag{2.3}$$

2.3.4 Back-propagation

Back-propagation is an algorithm that is used to minimize the aforementioned loss function $L(y, \hat{y})$ by recursively altering the weight coefficients based on on the gradient search technique [31].

The algorithm calculates the gradient of the loss function with respect to the weights. So, by using back-propagation, it is possible to understand how much each weight (w_{ij}) affects the final result of the loss function. From this method the weights can be updated for each training equation 2.4, that is how each weight should be updated.

$$w_{ij}^{n} = w_{ij}^{n-1} - \eta\frac{\partial L(y, \hat{y})}{\partial w_{ij}^{n-1}} \tag{2.4}$$

2.3.5 Training a Neural Network

At a high level of abstraction, a neural network has two major modes, learning and prediction. In the prediction process, the NN, given a certain input, generates an output that is determined by the learnable parameters of the network that should be determined previously in the learning process.

In the learning process, in which the NN receives a combination of inputs and desired outputs, the goal is (in the case of classification) to obtain the value of the weights. These values can be obtained using the back-propagation method discussed previously. For a certain number of training examples, called training data, the network will first do a forward pass of the data, with the weights randomly initialized, then apply the back-propagation algorithm to obtain a new value for the weights. This cycle is repeated until a certain stopping criterion is met (such as a certain value of the loss function or a certain number of epochs).

2.3.6 Convolutional Neural Networks

Using a fully connected neural network would prove to be impractical for image classification. Just taking a simple 32x32x3 image (32 wide, 32 height, 3 for the RGB color channels), the total amount of weights in a single fully connected layer would be $32 \times 32 \times 3 = 3072$. Considering that a NN would probably have a lot more layers and how a 32x32x3 is quite a small image, it is possible to see how this would become a problem for deeper networks and bigger images, since the number of weights to be learned would be very large, which would lead to a slow learning process as well as introduce overfitting due to the large amount of learnable parameters.

In order to circumvent this problem, convolutional layers were introduced, hence the name CNN. These layers replace a large number of connections in the network by utilizing convolutional filters [32] (also called feature or image maps). It is simpler to learn a set of filters, than a matrix with sometimes millions of entries [32]. The result of each convolution is then passed through a non-linear function like in a standard NN (commonly a ReLU). After this the obtained matrix is called an activation map. Several of these layers are applied in a CNN and in between these layers there may also be what are called pooling layers. What these pooling layers do is basically downsample the size of the activation maps.

With that said, apart from this process, CNN do not vary that much from normal NN. They are still comprised of an input layer, several hidden layers and an output layer, having neurons with learnable weights and biases. The hidden layers of a CNN have less parameters and are deeper than in conventional NN.

For image classification, the average CNN structure is comprised of convolution layers (already passed through a ReLU layer), pooling layers and fully connected layers, as shown in Figure 2.7. The first convolutional layer will detect simpler features (such as lines and curves), and the output activation maps will become the input of the next convolutional layer and so forth. At the end of the convolution + pooling layers phase, as seen in Figure 2.7, the resulting activation map will be able to recognize ears, paws, faces, etc.

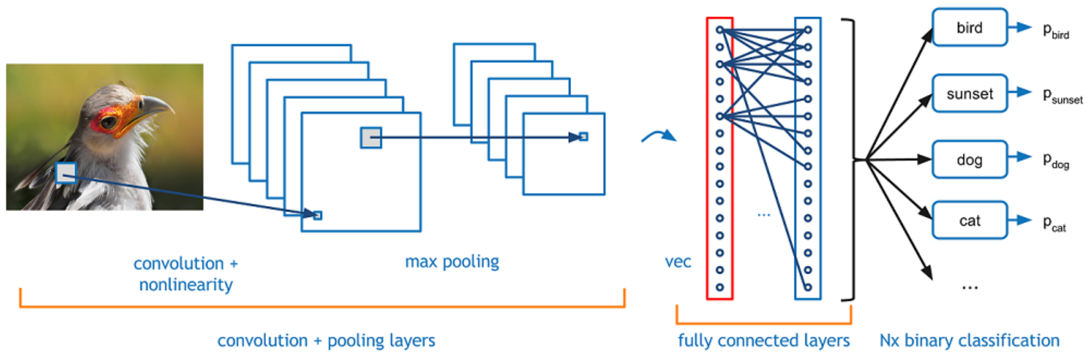


Figure 2.7: Structure of a standard CNN (adapted from [4]).

2.3.7 Convolutional Layer

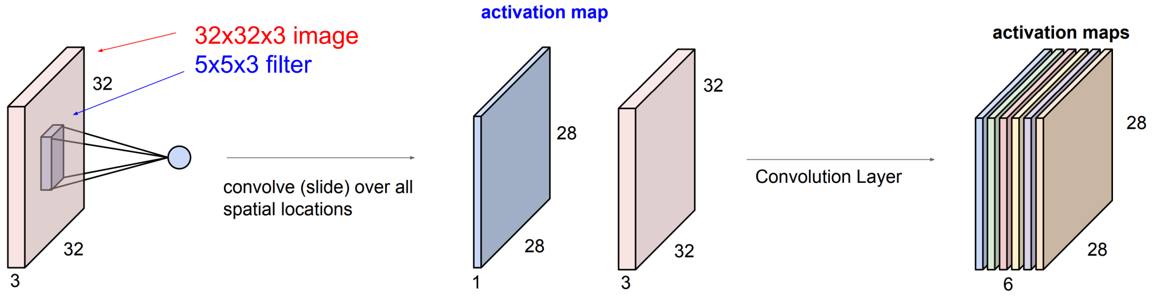
The convolutional layer is the main component of a CNN. It is comprised of a set of K filters that, although smaller in height and width, extend the full depth of the input matrix (for example if the input image is in the RGB color space it will have a depth of 3, this means it will have to apply the filter matrix for each of the color components, a gray-scale image would only have a depth of 1). During the forward pass, each filter slides (more precisely, convolves) across the width and height of the input volume computing the dot product of the filter and the image region at each position [3](the amount of pixels that are slid each time is called a stride). This process produces a 2-dimensional matrix called an activation map that gives the response of that filter at that region, as seen in Figure 2.8(a). Where

W_1xH_1xD_1 are, respectively, the input width, height and depth of the convolution layer and W_2xH_2xD_2 are the width, height and depth of the output activation map. While F is the size of the filter (a filter is given by FxFxD_1), S is the stride of the filter, and P is the amount of zero-padding, which is placing pixels with zero value around the edges of the image, this is sometimes done in order to maintain the original size of the input.

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1$$
$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1 \tag{2.5}$$
$$D_2 = K$$

Given these equations it can be seen that a convolutional layer will have four hyperparameters, the number of filters (K) that will be used, size of the filters (F), the stride (S) and the amount of zero padding (P).

The activation map or layer is comprised of neurons, which are obtained from the aforementioned convolutions of a specific region. Each neuron in an activation layer only "looks" at a certain region of the input image, which is a hyperparameter called the receptive field (this hyperparameter is equivalent to the filter size). Note that, even tough each neuron produced by a given filter is only connected to a certain region of the image, there may be several neurons (produced by different filters) "looking" at the same region. These neurons look for different kinds of features in the same section of the image. After the convolution of all filters a "new image" is obtained, that is made up of all the activation maps produced, as is represented in Figure 2.8(b).



(a) Convolution of the first filter. **(b)** Convolution layer with six filters.

Figure 2.8: Application of a convolution layer with six filters and resulting activation maps (adapted from [4]).

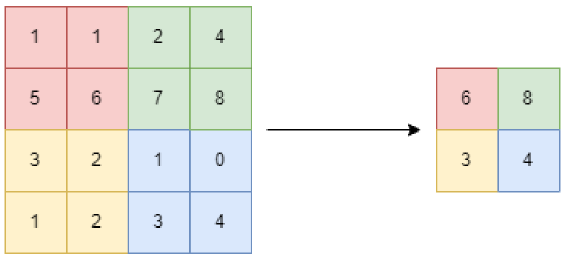
Convolutional Neural Networks use something called parameter sharing in order to control the amount of parameters. Taking a real world example, taken from [3], imagine an output of a convolutional layer with volume of 55x55x96, such a convolutional layer would have $55 \times 55 \times 96 = 290,400$ neurons. Assuming the filters have dimensions 11x11x3, each neuron would be connected to an 11x11x3 region of the input, which would be mean 363 weights plus 1 bias, meaning that, in total, there would be

$364 \times 290,400 = 105,705,600$ parameters in just a single layer, clearly a very large number.

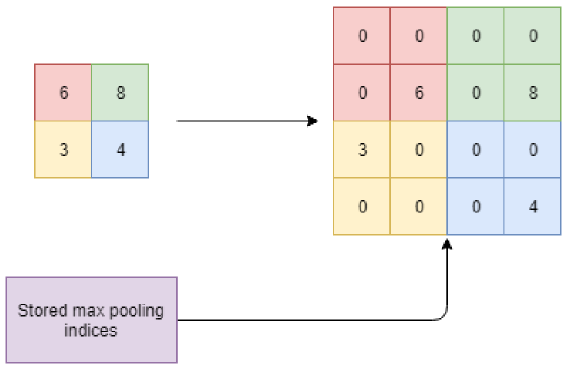
Parameter sharing can drastically reduce the number of parameters of convolutional layer, it does this assuming a useful feature at a position (x,y) will probably also be useful at another position (x2, y2). Another way of visualizing this is, defining a 2-dimensional slice of depth as a depth slice, the neurons of each depth slice will be constrained to have the same weights and bias. This means that in the previous example, there would be 96 depth slices and each would have dimensions connected to 11x11x3 regions of the input, which would give a total of $96 \times (11 \times 11 \times 3) = 34,848$ weights plus 96 bias, meaning a total of 34,944 parameters [3].

2.3.8 Pooling Layers

As mention before, the job of a pooling layer is to downsample the activation maps. This results in a progressive reduction of the spatial size of representation leading to a decrease in the amount of parameters and computation in the network [3].



(a) Max pooling with pooling extent of 2 and stride 2 (adapted from [3]).



(b) Example of upsampling using max pooling indicies

Figure 2.9: Example of max pooling and respecting upsampling

A pooling layer receives an input volume of size W_1xH_1xD_1, requiring as hyperparameters its spatial extent/size (F) and stride (S), and outputs a volume of size W_2xH_2xD_2, these values can be calculated with the equations in 2.6.

$$\begin{aligned} W_2 &= \frac{(W_1 - F)}{S} + 1 \\ H_2 &= \frac{(H_1 - F)}{S} + 1 \\ D_2 &= D_1 \end{aligned} \tag{2.6}$$

2.3.9 Fully Connected Layer

After the final convolution layer, the output activation map of said layer will be the input of the fully connected layer. This layer takes as an input a certain volume that represents the activation maps of high level features, and outputs a vector the size of the number of possible classes (N). The activation function of this layer is usually a softmax function, that estimates the probability of each class label of the N classes, these probabilities can be calculated by the following formula

$$S(y_i) = \frac{exp(y_i)}{\sum_{j=1}^{N} exp(y_j)} \tag{2.7}$$

From this it is possible to obtain the output probability $S(y_i)$ (this will be a vector of size N), where y_i is the is the i-*th* position of the output vector of fully connected layer.

2.4 Image Segmentation

In computer vision, image segmentation is the act of separating the image into several segments, where each segment represents a different component of the image.

In recent years image segmentation as become increasingly more popular, being used in applications such as self driving cars and security cameras. There are two different types of segmentation, semantic and instance segmentation.

2.4.1 Instance Segmentation

Instance segmentation is a process where an image is segmented while identifying each element of a certain class as different object. For example, in a an image with two people, instance segmentation would segment each person as *person1* and *person2*. Although this type of segmentation may be useful if it is desired to detect different instances of a certain class and to verify information such as the number of people in a certain place, if the objective is only to detect if a certain class is present in an image or how much of the image is occupied by a class, this approach adds a layer of complexity, detecting different instances, without being necessary.

2.4.2 Semantic Segmentation

Semantic segmentation consists in segmenting the images into different classes without separating the elements of the same class, for example, in an image with two people, each person is classified simply as person, and not as *person1* and *person2*.

This methodology is usually applied when it is not necessary do detect how many instances of a certain class exist. Some methods developed for semantic segmentation include, SegNet [15], composed of an encoder and a decoder, fully convolutional network or FCN [33], composed of only convolutional and pooling layers and terminating on a upsampling layer, and the U-Net [34], composed by an encoder and decoder just like the SegNet architecture, changing the way the information from the encoder is passed to the decoder. This thesis will experiment with the SegNet approach, as such the functioning of this methodology will be explained in further detail.

SegNet

SegNet is an architecture developed for semantic pixel labelling [15]. Meaning that it, given an image as an input, it outputs a segmentation mask, corresponding to the input image divided into different segments, where each of these segments represents a class (represented by a different label in the segmentation mask). The architecture of SegNet is based on an encoder-decoder network, as illustrated in Figure 2.10, where the encoder is composed by convolutions and max pooling, while the decoder is composed of upsampling and convolutions, followed by using a softmax classifier, that assigns a class label to each pixel. During the max pooling process, the original indices or locations are stored and are then used during the upsampling process, as illustrated in Figure 2.9(b).

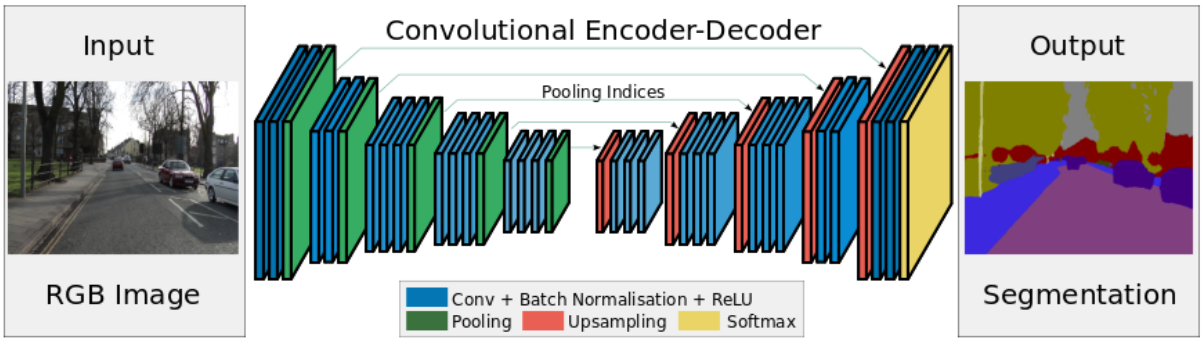


Figure 2.10: SegNet architecture (taken from [5])

It is possible to take a standard CNN architecture that as already shown good results in image classification and modify it in order to obtain a SegNet model based on that architecture. This can be done by removing the fully connected layer and replacing it by a corresponding decoder network.

Just like a normal CNN, the SegNet architecture is trained resorting to labeled data, however this labeled data is different than the one used in classification problems, where the goal is typically to assign a classification label to the entire image. Since the objective of the SegNet is to give a prediction for each pixel, the training images should be accompanied by a "ground truth" segmentation mask, where all pixels are classified as belonging to a certain class.

2.5 Deep Learning Advantages and Disadvantages

Deep learning based approaches are advantageous due to the speed at which they can classify an image (or segment it in the case of the SegNet approach), this is because the feed forwarding of a CNN is quite fast. It also has no need for intricate feature crafting, since, given a capable dataset, a convolutional neural network will, through back propagation, learn by itself relevant features and how these are connected to each class.

However, these positive points are also met with some disadvantages. Firstly, even though the feed forward is fast, the back propagation for CNN based methodologies, is quite slow, which means that training on a large size dataset will take a long time. Another problem is that, in order to avoid overfitting the network, it is important to have a dataset that is well constructed. This means a large and generic dataset is necessary in order to create a robust system. Finally, even though learning features and their connection to each class on their own is an advantage, this makes it hard to analyse and justify the choices of a CNN when classifying an image as a certain class.

3

Block-based Architecture

Contents

This chapter presents the architecture of the developed block-based system. It also explains the development of each component of the system.

3.1 System Architecture

The objective of this thesis is to construct a tool capable of reducing the amount of images that experts have to look at in order determine the corrosion state of a structure. For this, it is necessary to detect the presence of rust in an imaged structure. However, not all images that contain rust provide useful information for experts, since some of them only show a small area of rust. As such this solution proposes a methodology that takes into account the relation between the areas of rust and structure found in an image.

As discussed in the previous chapter, deep learning based approaches, namely CNN approaches, have already shown to have good performance for detecting rust, being able to achieve results that are better than other methodologies [7, 8]. Even though they have their disadvantages, given the capability of classifying an image into one of several classes (provided all desired classes were present in the training dataset) without having to craft complex features for each class, the utility and performance that CNNs have, is enough for justifying their use.

Since convolutional neural networks are capable of classifying images as one of several different classes, it is also possible to use this approach for detecting the existence of structure in an image. With the capability of detecting both rust and structure areas in an image, it is possible to determine the percentage of the structure that is corroded in a given image.

The problem of rust and structure detection will be tackled by using two different approaches. The first approach is based on classifying several blocks of the image resorting to a standard CNN to detect rust and structure in an image dividing the imagine into blocks and classifying each block. The second is based on doing a pixel based classification resorting to a SegNet architecture to segment the image into three possible colors, which represent, "rust" , "structure" and "background".

Figure 3.1, shows a high level overview of the proposed system architecture based on utilizing a block-based approach to classify segments of the image, which is comprised of four main components: pre-processing, feature extraction, classification and result analysis. The original input image goes through a pre-processing step in order to make it usable by the feature extractor, that is based on a CNN approach. This feature extraction obtains information on the image features, which in turn is used in order to classify the input of the CNN as "rust" or "structure". Afterwards the results obtained from the classification step are analysed to make a final classification.

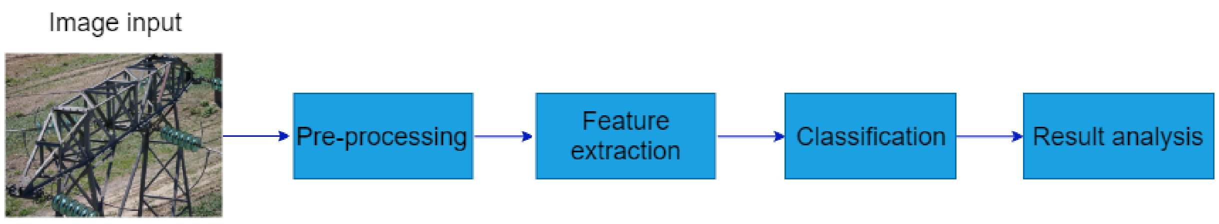
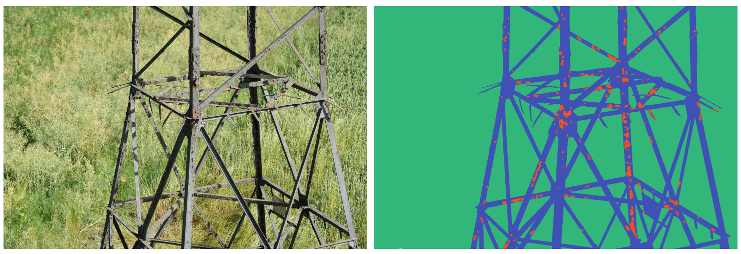


Figure 3.1: Architecture of the developed black-based system

3.2 Pre-processing

To understand the options made for the development of a rust detection solution, it is important to know the characteristics of the images that are expected to be available as input. The available dataset was provided by Axians and is comprised of 97 labeled images. The images in the dataset are of different resolutions, ranging from *6000x4500* to *1024x768* pixels. This variety is derived from different cameras being used to capture the footage, this is something to take into account, as the input image resolution cannot be predicted and the developed methodology must take this into account. The data is comprised of the original image captured, illustrated in Figure 3.2(a), and the ground truth information or labeled data, which was annotated by hand. The labeled data is made up of hand annotated masks, as depicted in Figure 3.2(b), which contain three types of labels: green for "background", blue for "structure" and red for "rust".

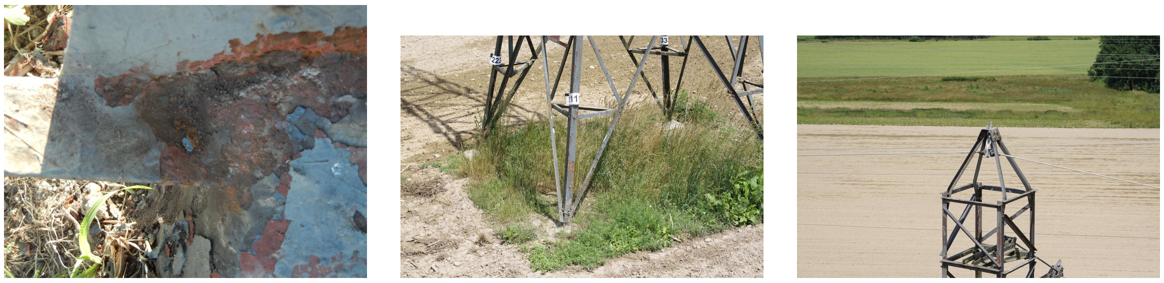


(a) Original image **(b)** Labeled image

Figure 3.2: Illustration of original and labeled image (provided by Axians).

There are more differences to the images than their resolution. Another important point is that the distance to the structure is not constant between images and it can change in a very dramatic way, as illustrated in Figure 3.2(a) and Figure 3.3. This difference in distance may cause some problems, since it makes the already small data set not very uniform.

As discussed previously, the annotation of this dataset is done by manually painting masks that represent each of the classes, which implies that the labeling is pixel based. Since it is desired to craft a CNN based methodology, the training data must undergo some kind of processing before training the network.



(a) Image taken close to the structure

(b) Image taken at a medium distance of the structure

(c) Image taken far from the structure

Figure 3.3: Image sample of the dataset illustrating the different distances to the structure when acquiring images (provided by Axians).

3.2.1 Dataset processing

A standard CNN classifies each image as belonging to one single class. Therefore, the pixel-based mask depicted in Figure 3.2(b), cannot directly be used to train the CNN, since the three classes can be present in the same labeled image (rust, structure and background). A way of approaching this problem is by dividing the original images into several blocks or windows - an example of this is shown in Figure 3.6, and classifying each block into one of the three possible classes: "rust", "structure" or "background". This labeling is done according to the contents of the selected block/window. For example, if a window is mainly composed of red pixels, it would mean that the corresponding window should be labeled as "rust". With this approach a new, block-based ground-truth dataset can be created, to be used to train a CNN capable of classifying each window as "rust", "structure" or "background".

Sliding window

A sliding window, consists of a window of a predetermined size (*64x64* for example), that iterates through an image, sliding a certain amount of pixels at a time, as seen in Figure 3.4.

The larger the stride, the faster the sliding window will iterate through an image, however, having a stride that is too large may also lead to the loss of information. An easy to understand example is facial detection, with a large stride it may occur that incomplete parts of a face will be observed in four different windows, making it impossible to detect a face in any of them, while if a small stride was adopted would lead to more windows being analyzed, but one of them would contain the entire face, allowing to obtain the desired results.

Just like the window size, fine-tuning the stride is important in order to find the optimal relation between detection performance and the speed of the algorithm.

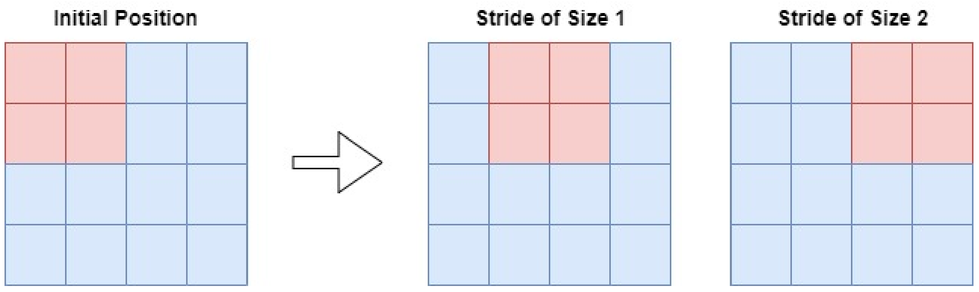


Figure 3.4: Example of sliding window with different strides

Parameter Definition for Dataset Processing

There are important decisions that need to be considered when developing this window based approach:

- The window size; this will determine what is at the input of the CNN, a smaller window will result in less detail being given to the CNN.

- Determining how a window should be labeled as "rust", "structure" or "background" when processing the training set; the labelling of the dataset used for training is one of the most important components when dealing with convolutional neural networks. Meaning that the way these are constructed will greatly affect the way a CNN performs.

Window size

Looking at the window size, a few preexisting CNNs already have input sizes defined, which is usually defined by the kind of dataset for which they were developed. For example the VGG network [27], trained on the ImageNet dataset, has an original input shape of *224x224*.

From this starting point it can be can seen how different window sizes fit with the original images, this comparison is observable in Figure 3.5, where it can be concluded that a window size of *224x224* will be too large and, in some cases, capture to much of the image (in this particular example it could capture several beams of the structure). The same could be said about a *112x112* window, although it would not capture several beams, it could capture several rust spots, potentially leading to a misleading classification result. Taking this into account, the selected window size should be of *64x64* pixels or smaller. Something to consider is that a smaller window means less information for the network to make a classification.

For processing the dataset, it was chosen a stride that would lead to no image overlap, which makes the processing faster.

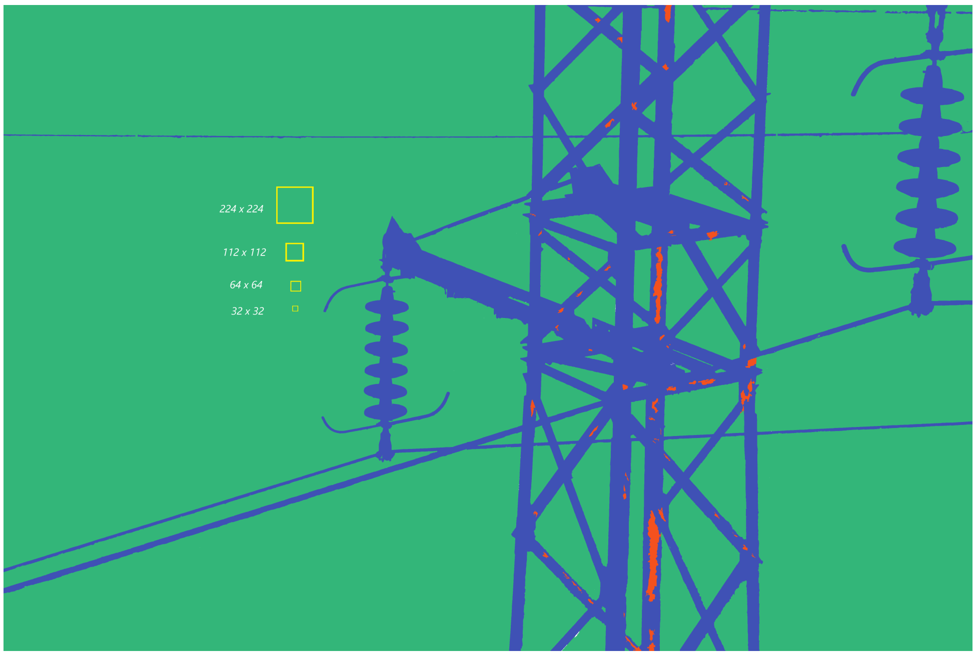


Figure 3.5: Example of different window sizes overlaid on the segmentation mask of a sample from an image (data provided by Axians).

Rust and structure labeling when using a window based approach

The main issue to take into account when generating the window based ground truth dataset from the available data, is how the rust and structure labeling is defined. The minimum amount of "rust" and "structure" that must be present on each window, so that the window receives that ground truth label will define how the network perceives the concepts of "rust" and "structure". Atha [8] trained a CNN where all images of the class "rust" were 100% rust. However, different definitions of what is considered "rust" and "structure", will affect how a network will detect these classes.

In order to verify which is the best way of defining the "rust" and "structure" classes, several sets with different labeling will be made, each with a different combination of "structure" and "rust" percentage.

Several combinations of "rust" and "structure" percentage are initially tested when considering *64x64* windows. The results from this experiment will then be used to choose which of the combinations produce the best results. For example, if windows of size *64x64* where both "structure" and "rust" are defined as at least 90% of the obtained window, were the only cases that showed promising results, then only these two cases would be considered relevant for the remaining experiments.

3.2.2 Data Balancing

After labeling the image windows into the three distinct classes, it is noticeable that the number of "background" and "structure" classes extremely out number the amount of images belonging to the

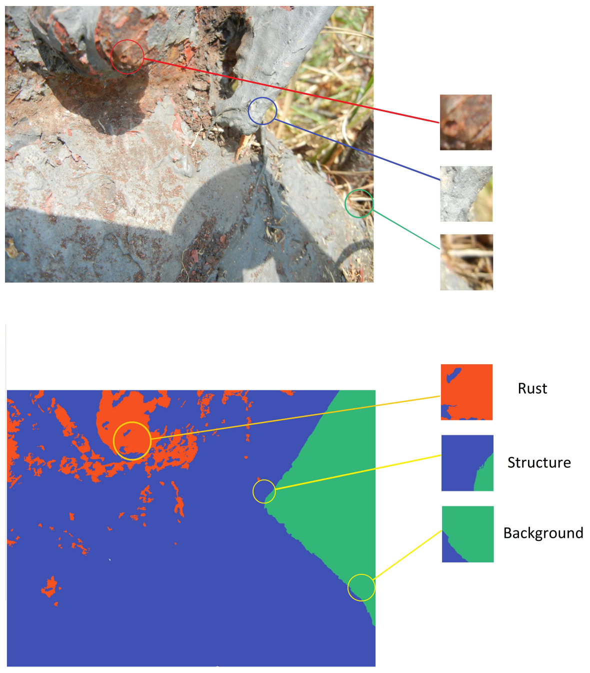


Figure 3.6: Example of a window from an image (original images provided by Axians).

"rust" class. This poses a problem, as the goal is to be able to classify correctly rust. But, if this class is so underrepresented, it will most likely cause the system to be more biased towards learning the "structure" and "background" classes, increasing the accuracy for those classes, and decreasing the accuracy for the "rust" class.

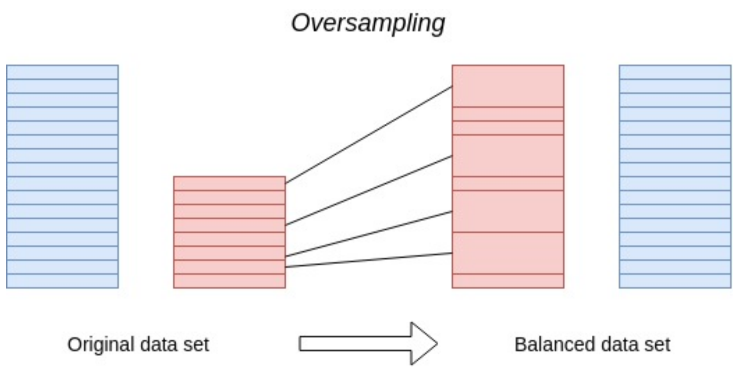


Figure 3.7: Example of oversampling.

There are two commonly used methods to balance datasets: weight balancing; and over and under sampling. Weight balancing, as the name implies, consists of attributing weights to the classes, which will reduce their impact when calculating the loss of the network, this is a good approach to take when a class is more important than others. For example, when developing an application to buy a house with classes "buy" and "not buy", the class "buy" is more important than the "not buy, since there will always be more houses to buy. Oversampling consists in increasing the amount of data of the less represented classes in the dataset. This is done by taking the existing data and replicating it, thus increasing the amount of samples, an example can be seen in Figure 3.7.

Undersampling does the opposite of oversampling, taking the over represented classes and reducing the amount of samples until the amount of samples in all classes is the same, as seen in Figure 3.8.

In this thesis the performance of all classes is equally important and it is not desirable to misclassify any classes, because this will affect the calculation of the ratio between "rust" and "structure" classes for the entire image. Since the amount of data is quite limited and contains several identical elements, oversampling might lead to a less generic model, which in turn could increase overfitting. Because of this, it was chosen to apply undersampling to the data.

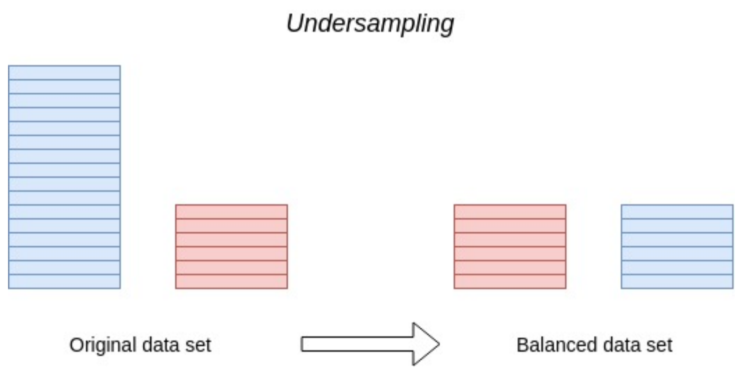


Figure 3.8: Example of undersampling.

3.3 Feature Extraction

This thesis proposes the utilization of a convolution neural network as a means of extracting the features of an image. This approach, has discussed previously, as already shown good results in detecting rust in images [8]. Since CNNs learn the important features of the classes they want to learn by themselves during training, it is possible to extent these methodologies from a binary classification of "rust" and "non rust" to a categorical classification of "rust", "structure" and "background".

3.3.1 Developing the CNN model

There are several ways of implementing a CNN and several frameworks available in order facilitate the development of CNN based methodologies. Some of the existing frameworks include: Tensorflow [35], Pytorch [36] and Keras [37]. For this thesis, it was chosen to utilize Keras. Due to it being a higher-level API built on top of lower level libraries (such as Tensorflow), the learning curve necessary to start implementing and testing massive models of deep learning is much more reduced when compared to other frameworks. However this comes at a cost of it being less efficient than other frameworks and lacking some useful functionalities such as debugging (present in Pytorch).

Creating a CNN from scratch with Keras is a relatively straightforward task, however there are other ways of implementing a CNN. A very common and successful way, is utilizing a CNN that was created and that obtained a good result in classification challenges, such as ImageNet [26]. It is also possible to employ the use of the features learned on these large and generic datasets on different classification problems. Taking into account the amount of data available for this thesis, as well as its repetitive nature, utilizing a base model that was pretrained on a large dataset will help reduce overfitting.

Keras has a decent amount of popular and successful CNN architectures available that are easy to implement. Of these available architectures, the VGG [27] and MobileNet [38] were experimented with.

The VGG16, as mentioned previously as already shown good results in this kind of application [8], hence the choice of implementing it. Meanwhile the MobileNet architecture is well known for being lightweight, which allows it to be used in mobile devices that have difficulty running more complex networks. This characteristic of the MobileNet is important since, if it shows good performance, it can be directly implemented in the software of a drone to automatically detect rust.

These networks, have been built in order to classify several different classes that are of no interest for this thesiss application (the VGG network for example has an output of 1000 classes, none of which are "rust" or "structure"). So, before training these networks it is necessary to make alterations in order for them to be able do classify the desired classes. In Keras, this can be done by removing the last fully connected layer of the network and replacing it by a new fully connected layer that has a final layer with three outputs("rust", "structure" and "background"). After doing this the networks are ready to be trained on the data Axians dataset.

3.3.2 Training the CNN

Something to take into consideration, is that the data that these networks are going to be trained may not be sufficiently generic. Even though the generated dataset may have thousands of images, all of them were obtained from a small selection of 97 available images. This means that training a network from end-to-end on this dataset may result in considerable overfitting. However, the weights that these networks learned from being trained on ImageNet can be loaded into the CNN using Keras, resulting in pretrained networks, that have feature information from being trained on large datasets. Even though these networks can be utilized without any further training, they would have no relevant information regarding the kind of data that is desired to classify. As such the pretrained CNNs models should be fine tuned on the dataset provided by Axians.

Fine-tuning is a transfer learning method that can be utilized when the amount of data available to train a CNN is small and may not be sufficient to train the network end-to-end without overfitting. This method involves "freezing" most of the convolutional blocks, except the last one, since this last convolutional block will learn the high level features necessary for classifying "rust", "structure" and "background". This means that the weights of the "frozen" blocks will not be updated during back propagation, which keeps the low level features the network learned by being trained on a larger dataset, while learning new higher level features. This not only reduces overfitting when working with small datasets, but also makes the training faster, since the only the last convolutional block will have its weights updated by the back propagation algorithm.

Even though fine-tuning a pretrained model helps reduce overfitting, there is still something that can be done in order to alleviate it, which is augmenting the original dataset. Data augmentation is applying some sequence of image transformation to the dataset in order to generate new images that are, essentially just transformations of the already existing data. These transformations can include an horizontal flip, a vertical flip, a 45^o tilt on a image, zoom in, zoom out, between many other options. These transformations help the model learn to classify images with several orientations. For this method the data generated from the original dataset was augmented resorting to the *ImageDataGenerator* class, which allows augmenting the data by declaring which type of augmentation is desired. For this approach, the following augmentations were implemented:

- $rotation\_range = 60$; random rotations from 0 to 60 degrees.
- $width\_shift\_range = 0.5$; random shifts in width in the range of [-0.5; 0.5].
- $height\_shift\_range = 0.5$; random shifts in height in the range of [-0.5 and 0.5].
- $shear\_range = 0.4$; applies a random shearing transformation.
- $zoom\_range = 0.5$; random zooms in the range of [-0.5; 0.5]
- $horizontal\_flip = True$; random horizontal flips.
- $vertical\_flip = True$; random vertical flips.

Hyperparameter Optimization

There are several hyperparameters that have to be chosen and fine-tuned in order to optimize the training of the CNN, these parameters include:

- **batch size**: the batch size is the amount of samples (images in this case) that are passed through to the network at one time. The values of a batch can range from 1 to the number of samples in the dataset. Training the model in larger batches increases the training speed of the network, however it also may lead to a worse generalization by the model. Another thing to take into account is that, if the batch size is to large, there is a possibility that the machine where the network is being trained cannot handle the memory load of working with that large amount of images.

- **optimizer**: the optimizer is a tool that helps to speed up the time necessary to find the optimum minimum of the loss function. There are several optimizers that can be applied, however, the adaptive momentum estimation (ADAM) optimizer [39], has become popular in recent years and has been utilized in several projects due to its convergence speed combined with good performance. The ADAM optimizer has four parameters to be tuned as shown in Equation 3.1. Of these four parameters, the one that varies more is the learning rate, η. The other three do not very much in value, β_1 is usually set to 0.9, β_2 is set at 0.999 and ϵ is at very small values, usually something like 10^{-8}.
- **number of epochs**: an epoch is an iteration through all of the training data. As such the number of epochs is the number of times the training data is iterated through. If this value is to small, the model will underfit, and too large of a value will make the model overfit.

$$
\begin{aligned}
m^{n+1} &= \frac{\beta_1 m^n + (1-\beta_1)\frac{\partial L(y,\hat{y})}{\partial \omega_{ij}^n}}{1-\beta_1} \\
s^{n+1} &= \frac{\beta_2 s^n + (1-\beta_2)\frac{\partial L^2(y,\hat{y})}{\partial \omega_{ij}^n}}{1-\beta_2} \\
\omega^{n+1} &= \omega^n - \eta \frac{m^n}{\sqrt{s^n+\epsilon}}\omega
\end{aligned}
\tag{3.1}
$$

3.4 Classification

In this thesis, the classification of the window was done with a fully connected neural network. As discussed previously, this is a standard procedure when utilizing a CNN. However there is the possibility of utilizing other classifiers that take advantage of the features extracted through the convolutional and pooling layers. The choice of class is dictated by a softmax activation function that, given a vector of scores, outputs a vector that represents the probability distributions of a list of potential outcomes. A practical example of the softmax activation function can be seen in Figure 3.9.

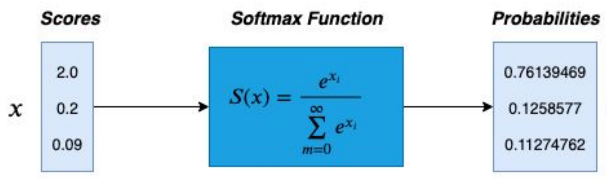


Figure 3.9: Example of softmax function application.

After obtaining the probability vector, the class of the window is defined based on class with the largest value in the probability vector. This in turn increments one of two variables, one that records the number of windows classified as "rust" and another does the same for the windows classified as "structure", which will be designated as *rust_count* and *structure_count* respectively.

3.5 Metrics Adopted for Result Analysis

3.5.1 Ratio Calculation

The objective of this thesis is to filter the amount of images that experts have to look through, so that they can make decisions about infrastructure maintenance planning. As such only images that have a relevant amount of "rust" should be shown and images that only show "background" or "structure" but little or no "rust" should not be shown. This, however, poses the following challenges:

- How much rust should be detected in an image for it to be considered relevant?
- How to obtain the amount of rust existing in a given image?

Regarding the first challenge, if rust is detected in a very small section of the image (only a *64x64* square is classified as rust in a *6000x4000* image, for example) it does not make sense to consider it relevant. This means that a threshold must be set in order to define whether the amount of rust present in a certain image is significant or not.

As for the second challenge, the ratio chosen for this evaluation is given by

$$\frac{number\ of\ rust\ windows}{(number\ of\ rust\ windows) + (number\ of\ structure\ windows)} \tag{3.2}$$

This can be done by applying the formula defined in 3.2. Having this ratio value it is possible to compare it to a threshold previously set for this ratio. For example, it could be defined that an image that shows a ratio of less than X% rust present in structure has no need to be examined by an expert, that an image with a ratio above X% needs to be looked at by an expert.

3.5.2 Intersection over Union

Since the available dataset contains no information regarding how much rust in a structure can be considered too much, a cutoff ratio of 5% was defined for this thesis, meaning that it should be looked at by an expert if more than 5% of the structure contains rust. Of course this ratio can be adjusted depending on the particular type of structure being analyzed and the maintenance criteria defined.

$$IoU = \frac{prediction \cap groundtruth}{prediction \cup groundtruth} \tag{3.3}$$

Another metric that will be used to evaluate the performance of the intersection over union (IoU), that shows how much two images coincide, this can be seen in Equation 3.3. In order to use this in the block-based approach, each time a block is classified as "rust", "structure" or "background", that block is colored with a pre-determined color for each class. Then the intersection over union between the original ground-truth mask and the predicted mask is calculated.

3.5.3 Precision, Recall and F1-Score

The precision, recall and F1-score [40] are metrics that are commonly used to measure the performance of machine learning models [8, 17]. These metrics will be calculated for each relevant class, "rust" and "structure", which can be done resorting to the confusion matrix represented in Table 3.1.

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	true positive (TP)	false negative (FN)
	Class = 0	false positive (FP)	true negative (TN)

Table 3.1: Confusion matrix example for binary classification (where class = 1 represents a class like "rust" or "structure" and class = 0 represents "not rust" or "not structure")

The true positives and true negatives obtained from this matrix are the correctly classified samples, while the false negatives and false positives are the samples that were wrongly classified. The end goal is o maximize the TP and the TN and minimize the FP and FN.

Precision

Precision represents the ratio between the correctly predicted positive predictions and the total amount of positive predictions made. This value can be calculated with the following formula

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.4}$$

Recall

Recall is the ration between the correctly predicted positive predictions and the total amount of truly positive observations. This value can be calculated by the following formula

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.5}$$

F1-Score

This metric is the harmonic mean of precision and recall, which means this metric takes into account both the false positives and false negatives into account. This metric should be used when the importance of false positives and false negatives is not balanced. In this case, not classifying a rusted area is more important than classifying a non rusted area as having rust, since failing to detect some rusted areas may lead to important structural flaws being missed. The f1-score can be calculated by the following formula

$$F1 - Score = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{3.6}$$

4

Pixel-based Architecture

Contents

This presents overview of the pixel-based architecture developed, using the SegNet network to obtain a segmentation mask of the input image that identifies structure and rust pixels. The implementation of each component of the system is detailed.

4.1 System Architecture

Detecting rust using a pixel-based architecture relies on the SegNet for obtaining a segmentation mask. The high level block diagram corresponding to this system is illustrated on Figure 4.1, which is very similar to the system architecture of the block-based architecture described previously. However there are some key differences to be taken into account. The pre-processing done is different, and the feature extraction and classification are now the addressed by the SegNet block. The analysis of results should also be different in this case, since the output of the SegNet block is a pixel-based segmentation mask, while the classification output in Figure 3.1 is the amount of windows classified as "rust" and "structure".

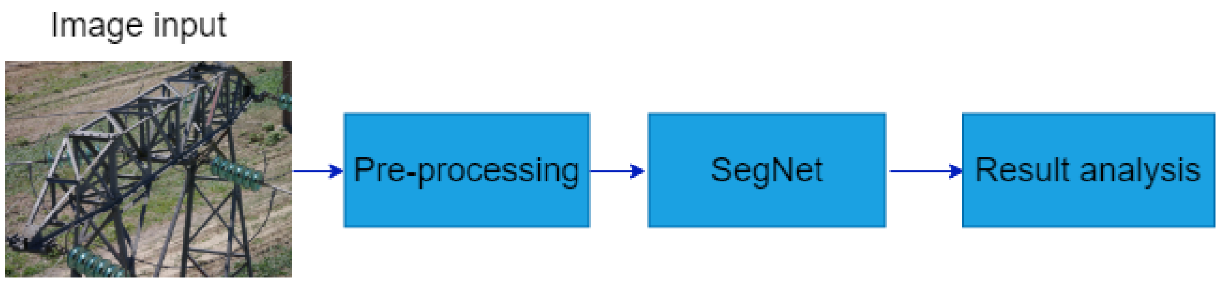


Figure 4.1: Architecture of the developed system based on a SegNet model

4.2 Pre-Processing

The input of a SegNet, like most CNNs, is a static image. As discussed previously, the dataset available is composed by images of several different sizes, meaning that, a pre-processing operation, notably to resize the input according to the size expected by the SegNet, may be required.

The size to which the image is resized is an important step, since then larger the image, the larger the amount of memory necessary to train and run the network. However, the larger the image the more detail the network has to work with, which can lead to a better segmentation, and therefore to a more accurate classification.

4.3 SegNet

It is desired to generate a pixel based segmentation mask, where each class ("rust," "structure" and "background") is labeled with a different color. Using this mask, is is possible to see how much of the structure has rust. For this it was chosen to use the SegNet architecture, since it has already shown to obtain good results in several applications [41]. The instance segmentation techniques were not used in this case because there was no need for identifying each instance of rust, since what is desired to obtained is the total amount of rust on the structure. These techniques could be used if it was desired to classify the degree of corrosion of each rust instance, however the available data set does not contain information for this kind of classification.

Just like implementing a regular CNN, there are several ways of implementing a SegNet model. This can be done in Keras by either creating an encoder and the respective decoder or even by taking an already existing CNN architecture, like VGG [27], take off the fully connected layer, and create the corresponding decoder for this network. However there are some open source libraries that simplifies the implementation of a SegNet architecture and in this thesis, a library called *keras_segmentation* [42] was used, that contains several implementations of SegNet architectures, including an architecture based on VGG16.

4.3.1 Training the SegNet

Training of the SegNet is very similar to the training of a CNN, however, since the SegNet present in the used library was not pretrained on a large data, it needs to be trained from scratch. As such the SegNet was trained on a ninety-three images, while the remaining four were set aside for testing. This means that some sort of overfit is likely, even if data augmentation is done.

4.3.2 Hyperparameters

Implementing a SegNet model resorting to this library is very straightforward and it has the possibility of choosing hyperparameters, with the caveat that the amount of hyperparameters that can be tuned being more reduced when compared to implementing this architecture from scratch. The hyperparameters that were tuned are the number of epochs, the type of optimizer used and the batch size.

- The number of epochs in this case, will most likely have to be be lower when compared to the value used in the block-based approach. Since for this approach the original dataset will not be divided into several blocks, this means the amount of existing images for training and validation will be much less.

- The optimizer used for this approach will also be the ADAM optimizer, although the SegNet has a decoder and does not end on a fully connected layer, the back propagation algorithm works the same way.

- The batch size will also have to be smaller. This is because the input image will be larger, which means a larger amount of memory will be used.

4.4 Metrics Adopted for Result Analysis

The way the result analysis is performed for this approach is essentially the same as the way it works for the sliding window approach. The same type of metrics will be used to evaluate the results of the SegNet, precision, recall, f1_score, rust-structure ratio and intersection over union, however, since the output from the SegNet approach is a segmentation of the original image, in order to obtain the ratio of rust in the structure, it is first necessary to calculate how many pixels were defined as "rust" and "structure".

These values can then be used to calculate the ratio defined in Equation 3.2, utilizing pixels instead of the number of windows. As mentioned previously, this is the ratio that will determine if the image should or should be shown to an expert. The value of the intersection over union was also calculated in order to be able to gauge how much of the generated segmentation coincides with the ground-truth.

5

Results

Contents

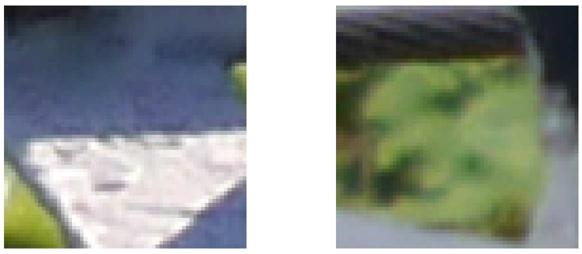
This chapter will go through the development of both architectures, the sliding window and CNN approach and the SegNet approach. It also shows the experiments done for each of these approaches as well as the results obtained.

5.1 Block-based Dataset Pre-Processing

The dataset pre-processing required to perform a block-based rust detection approach, involves obtaining new datasets composed of windows obtained from the original datasets.

Generating a block-based ground-truth dataset

Processing the dataset starts by defining two vectors. These vectors each contain the percentages of "rust" and "structure" that will be used to generate the block-based ground-truth training datasets. Since it is not possible to verify every single combination of "rust" and "structure", in this thesis the vectors utilized will be $sp = [40, 60, 80, 90]$ and $rp = [25, 50, 75, 90]$, where *sp* and *rp* are the structure and rust percentage, respectively. For the structure vector, the starting value of 40% was chosen due to the structure being a large focus of the image. In theory most of the images taken should contain a large portion of the structure, as such detecting structure when only 20% of it is on the window may not be very relevant. As for rust, having a base percentage of at least 25% rust makes more sense given that, in this case, the amount of rust on the image is quite smaller when compared with the amount of structure and this means that making it easier for rust to be detected may be an advantage.


(a) Sample of dataset with at least 90% structure

(b) Sample of dataset with at least 40% structure

Figure 5.1: Comparison of datasets generated with a *64x64* window and different percentages of structure pixels.

Figure 5.1 shows two image blocks labeled as structure if at least 90% of the pixels were labeled as structure and if that percentage was at least 40%. It should be noted that not all images in the dataset generated with at least 40% rust are like Figure 5.1(b), however it is important to understand that situations such as this may occur and can lead to bad detection during practice.

The choice of the best combination of these two parameters was done by training the VGG16 network [27] on each of these datasets and then comparing the results of applying this model on test images taken from the dataset.

5.2 Model Development

The base architecture trained was the VGG16. This network has already shown to have good performance at detecting rust in images in other works [8], making it a good starting point to establish which "rust" and "structure" percentage obtain better results.

5.2.1 Parameter Definition

In order to train the VGG16 network, as mentioned previously, some hyperparameters must be defined.

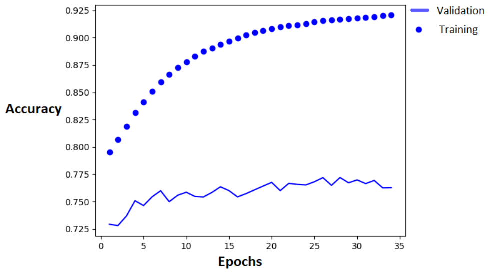
- Starting with the batch size, since overfitting is already a concern due to the nature of the dataset, the batch size value should not be too large. As such, it was initially set to 60 and, after conducting some tests, it was found that a batch size of 20 obtained the best results.
- For the optimizer, it was chosen the ADAM optimizer. The learning rate in Keras is predefined as 10^{-3}, which was the starting point of the training for this thesis. By tuning the values of the learning rate, it was defined a learning rate of 10^{-4} the value that seems to achieve the best equilibrium between the lowest loss and faster training time.
- The number of epochs performed during each training session varies for each dataset. This is because an early stopping metric was defined, that would stop the training 20 epochs after no improvement in the validation loss occurred. As such the epochs of the training were set to 60, which was the amount where overfitting was starting to become noticeable.

These parameters were the same for all the datasets with different "rust" and "structure" percentages, initial testing did not show any significant improvement by changing the learning rate or the optimizer. The number of epochs also started showing signs of overfitting around 60 epochs.

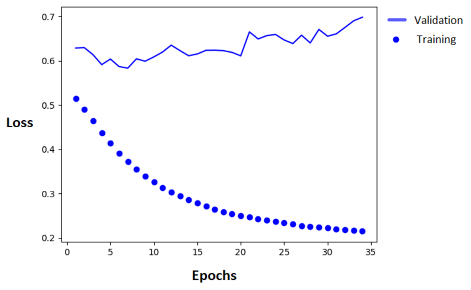
5.2.2 Accuracy and loss of the VGG16 models

After defining the values of the hyperparameters, it was obtained the accuracy and loss graphs of the different models during training, which were then saved in order to be tested. Comparing Figures 5.2 and 5.3, it is possible to see how, despite the original dataset being the same, a change to the definition of what the network considered "rust" and "structure" can have an impact during training.

However, the results from the training of the network are mainly for understanding how the network is learning and if it is capable of generalizing, if the validation loss and accuracy show low accuracy values it can be assumed that the generated datasets have some inherent problems that do not allow the model to correctly distinguish between "rust", "structure" and "background".



(a) Training and validation accuracy

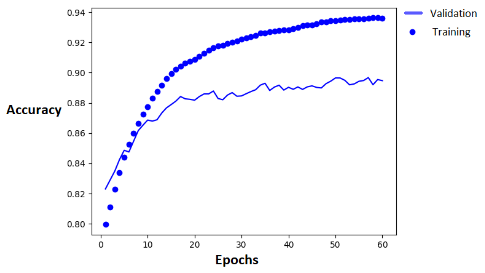


(b) Training and validation loss

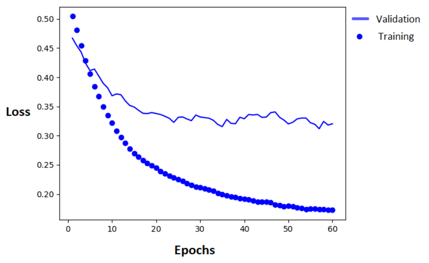
Figure 5.2: Model trained on 80% "structure percentage and 75% "rust" percentage

In Figure 5.2 it possible to see how the training accuracy tends towards 100% while no improvement occurs on the validation accuracy, which stagnates at about $0,75$. Meanwhile, the training loss decreases at a rapid rate while the validation loss tends to increase. These two signs indicate that the model is not generalizing well and is overfitting.

Looking at the results from training on a different generated dataset, Figure 5.3, it can be seen that the validation accuracy steadily increases until it reaches about 89% accuracy, while the validation loss stagnates at about $0,325$. Although the validation loss does not increase, as seen in Figure 5.2(b), it also does not keep decreasing like the training loss, this means that, even though the training loss reaches a value close to $0,15$, when faced with unknown images the performance of the model will be closer to the validation values, which represent unknown data.



(a) Training and validation accuracy



(b) Training and validation loss

Figure 5.3: Model trained on 90% "structure percentage and 90% "rust" percentage

5.3 Block-based Segmentation using VGG16

Utilizing the block-based approach, a total of four images from the original dataset were used in order to analyse how the model behaved on a real world setting, although this is a small amount, using more could also lead to a model that is poorly trained. These images can be seen in Figure 5.4 and were not used during training in order to avoid any bias in the features the network learned.

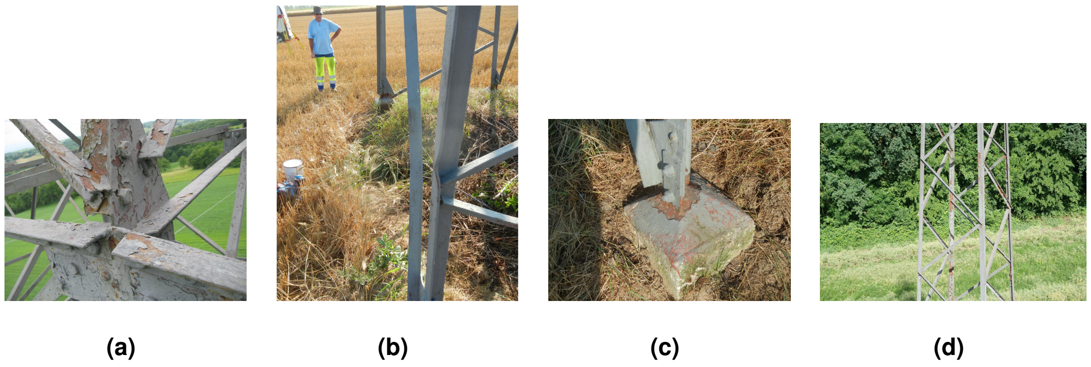


Figure 5.4: Images used to verify the performance of the models (provided by Axians)

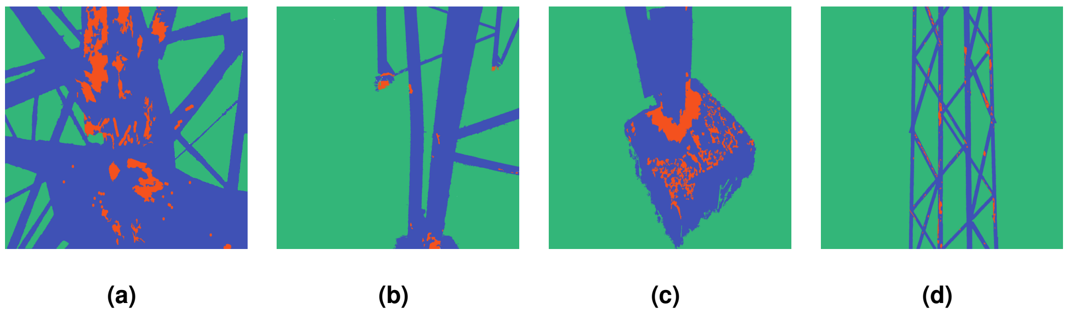


Figure 5.5: Hand labeled masks of the images used for testing (provided by Axians)

5.3.1 Sliding window parameters

The sliding window also has some parameters that need to be set before starting to segment the image. Namely the window size and stride. At the start a window of *64x64*, since this is the kind of data that the VGG network was trained on, with the stride of 64 was utilized to segment the image with the previously obtained models. A sample of the results obtained with this configuration, with two different models, can be seen in Figure 5.6, comparing them to the original mask on Figure 5.6(a).

Having obtained the generated masks, it is possible to quantify how much of the image was correctly segmented by comparing the intersection over union between the original and generated mask. With this

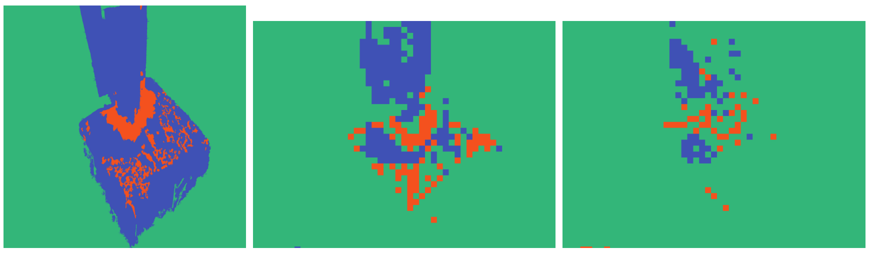

(a) Hand labeled mask (provided by Axians) **(b)** Model trained with 90 "rust" and 90 "structure" **(c)** Model trained with 25% "rust" and 40% "structure"

Figure 5.6: Comparison of models with same window and stride, but trained on different datasets (provided by Axians)

metric it is possible to see how much of the generated image was correctly segmented when compared to the ground truth. The results of the intersection over union obtained for each of the images for a window with size *64x64* with stride 64 can be be seen in Table 5.1.

		Rust (%)			
Image Analysed	Structure (%)	25	50	75	90
Figure 5.4(a)	40	25.82%	39.26%	40.09%	35.78%
	60	31.46%	42.46%	38.86%	43.14%
	80	28.29%	20.57%	18.56%	28.78%
	90	36.36%	34.52%	34.05%	45.39%
Figure 5.4(b)	40	67.07%	67.61%	68.42%	69.99%
	60	69.44%	66.53%	68.40%	75.20%
	80	66.28%	66.53%	64.48%	63.53%
	90	67.12%	69.26%	71.86%	68.02%
Figure 5.4(c)	40	60.49%	63.42%	65.16%	59.73%
	60	64.79%	59.66%	64.88%	73.98%
	80	63.59%	58.77%	56.79%	64.66%
	90	70.22%	68.74%	71.39%	71.90%
Figure 5.4(d)	40	79.93%	82.44%	84.39%	83.64%
	60	78.73%	79.53%	80.38%	83.75%
	80	78.61%	77.39%	77.39%	77.06%
	90	80.35%	79.23%	82.22%	80.72%

Table 5.1: Intersection over Union values for a *64x64* with stride *64*

Since it is important to reduce the influence of other parameters, another test was done resorting to a sliding window with stride 32, meaning that there is some overlap between windows, which can change the results obtained by each of the models. The values obtained while applying a window with stride 32 are shown in Table 5.3.

In order to analyse which of the models has the best performance, the averages of each "rust" and "structure" percentages were taken. Looking at Table 5.2, it is possible to see that, between using a stride of 32 or 64, the average of obtained on each o these is quite similar. It is also noticeable that the percentages that obtained the best average in both cases were the "rust" percentage at 90% and the "structure" percentage at 90% as well. However, the average performance with the "structure" percentage at 60% was close to the 90% dataset, being less than 1% apart from each other.

Taking this information into account, more testes were conducted with different window sizes and other CNN architectures, using a "rust" percentage of 90% and a "structure" percentage of 90% and 60%.

	Rust (%)				Structure (%)			
Average intersection over union	25	50	75	90	40	60	80	90
Stride 64	60.53%	60.99%	61.71%	64.08%	62.08%	63.83%	56.96%	64.46%
Stride 32	60.26%	60.74%	61.42%	63.66%	61.87%	63.19%	56.93%	64.09%

Table 5.2: Average of the intersection over union for each value of "rust" and "structure" percentage for sliding window with stride 64 and stride 32

		Rust (%)			
Image Analysed	Structure (%)	25	50	75	90
Figure 5.4(a)	40	25.52%	38.92%	38.73%	33.57%
	60	33.17%	41.56%	38.18%	38.491%
	80	28.37%	20.15%	18.12%	28.35%
	90	36.09%	33.17%	32.71%	43.15%
Figure 5.4(b)	40	66.87%	68.92%	66.94%	71.77%
	60	68.83%	66.09%	68.14%	76.15%
	80	66.08%	67.03%	64.67%	63.62%
	90	67.17%	68.88%	72.20%	68.34%
Figure 5.4(c)	40	59.76%	62.58%	64.66%	59.46%
	60	63.19%	58.80%	64.35%	73.22%
	80	63.19%	58.76%	56.37%	65.42%
	90	69.92%	67.73%	70.74%	71.70%
Figure 5.4(d)	40	80.44%	82.52%	86.11%	83.16%
	60	78.85%	79.75%	81.06%	84.28%
	80	78.85%	77.39%	77.41%	77.04%
	90	80.71%	79.61%	82.39%	80.87%

Table 5.3: Intersection over Union values for a *64x64* with stride *32*

5.3.2 Testing different window sizes and strides

In order to choose a window size and stride, it is necessary to take into account two factors. One of them is the performance of the CNN while segmenting the test images (results of the intersection over union) and the second is the speed at which the system segments the image. If the performance gain of one configuration does not justify the speed decrease, this configuration can be discard. It is important to evaluate the speed of the approach, since slow performing system will occupy resources for too long, removing the practicality of the system. For example, a performance gain of 0.1% is not justified if the system takes ten minutes more to obtain a result.

In Table 5.4, it is possible to see, on average, how long the system took to segment the image. The system took a lot longer to segment each image, even though no performance increase was seen, as shown in Tables 5.1 and 5.3.

	Average detection time (seconds)	
Image Analysed	Stride 64	Stride 32
Figure 5.4(a)	25.445	82.7
Figure 5.4(b)	99.05	355.64
Figure 5.4(c)	17.06	57.09
Figure 5.4(d)	69.31	246.87

Table 5.4: Average detection time of the system using the VGG16 architecture on a *64x64* window

Testing with a *32x32* windows shows that, reducing the window does not always provide an improvement in the intersection over union results. It can also be concluded, by looking at Tables 5.5 and 5.2, that decreasing the stride of the sliding window does not yield a better performance, while providing a significant slow down of execution time, as seen in Tables 5.4 and 5.5.

	Average intersection over union		Average detection time (seconds)	
Image Analysed	Stride 32	Stride 16	Stride 32	Stride 16
Figure 5.4(a)	46.88%	46.94%	82.42	309.95
Figure 5.4(b)	67.07%	67.63%	356.69	1317.68
Figure 5.4(c)	62.67%	62.65%	56.85	230.89
Figure 5.4(d)	84.25%	84.13%	247.42	959.88

Table 5.5: Average intersection over union and average detection time of segmentation done with a 32x32 window

From the obtained results, it can be seen that having a smaller window size and stride does not give much of a performance gain of the system. This means using *64x64* window with a stride of 64 is the best option in terms of the ratio of performance and execution time.

In Table 5.6 it can be seen the comparison between the values of the amount of rust present in the ground truth mask and the values of the amount of rust that the VGG16 for each image. It seems that this approach is detecting more rust than it is supposed to detect, since for all but one of the test images, the amount of rust detected was superior to the ground truth, with two cases, Figure 5.4(b) and Figure 5.4(d), the difference between the prediction and the ground truth was quite significant.

Image Analysed	Ground Truth	VGG16
Figure 5.4(a)	9.55%	13.48%
Figure 5.4(b)	2.29%	13.83%
Figure 5.4(c)	15.15%	34.62%
Figure 5.4(d)	6.06%	1.52%

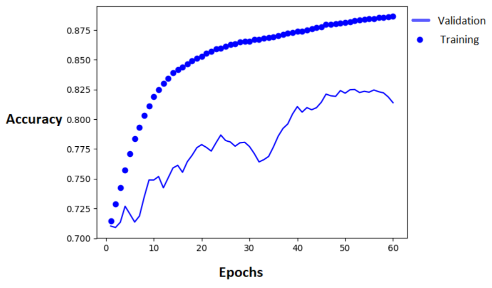
Table 5.6: Percentage of structure that contains rust calculated for the VGG16 architecture

5.4 Block-based tests using the MobileNet Architecture

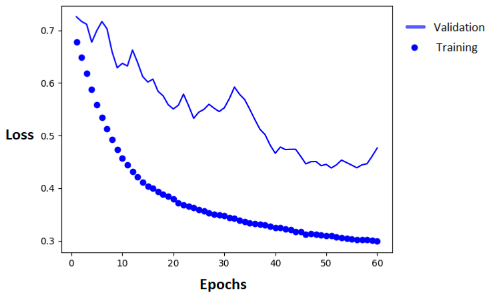
MobileNet is an architecture that was developed in order to be implemented in less powerful devices such as smartphones. This can be beneficial if it is desired to use the drone to acquire the images and also process them using a simpler processor, with less resources, but with less energy consumption.

The MobileNet architecture, being a less computationally intensive network is expected to have weaker performance when compared to the VGG16. Through testing it was found that the hyperparameters that showed the best results were very similar to the ones used with the VGG16 architecture. The following values were used: number of epochs, 60; a batch size of 20 and the ADAM optimizer with a learning rate of 10^{-5}, which was the only one that changed compared to the VGG16 implementation. This network was trained on two datasets, both of them with a "rust percentage" of 90% and with "structure percentage" of 60% and 90%. Since both of these models had similar performance in the both the VGG16 approach and in the MobileNet, the results of the intersection over union and of the execution time is the average of the values obtained resorting to both datasets.

The results during training are, as seen on Figure 5.7, inferior when compared to the ones obtained by the VGG16. Both the validation accuracy and loss of the model, although they do increase and decrease respectively, they do so in a very inconsistent manner, which shows signs of bad generalization.



(a) Training and validation accuracy



(b) Training and validation loss

Figure 5.7: Accuracy and loss of the MobileNet, trained on "rust" and "structure" percentage of 90% with *64x64* window

Some examples of the results obtained by segmenting the test images using the MobileNet network can be seen in 5.8, while the results of the intersection over union can be seen in Table 5.7. From these results it is possible to see, in Figure 5.8(b), that the model is giving more importance to the "structure" class. However in Figure 5.8(a), a better segmentation of the structure can be seen, which is reflected in the intersection over union value obtained.

Something to note in Figure 5.8(a) is that, despite having a relatively decent intersection over union, no rust was detected, when in reality, there is quite a bit rust in the image, as seen on Figure 5.6(a). In

Image Analysed	Average intersection over union	Average execution time (seconds)
Figure 5.4(a)	31.70%	40.91
Figure 5.4(b)	76.25%	76.89
Figure 5.4(c)	63.37%	165.88
Figure 5.4(d)	25.22%	234.81

Table 5.7: Average intersection over union of the MobileNet model, using a 64x64 window with stride 64

both examples present in Figure 5.8, the model has a very large difficulty detecting rust.

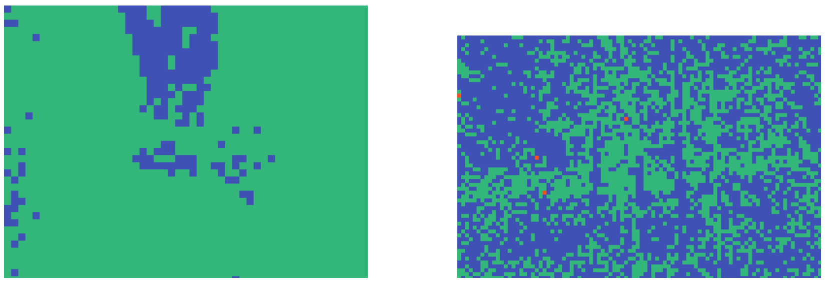

(a) Segmentation of 5.4(c) **(b)** Segmentation of 5.4(d)

Figure 5.8: Segmentation using MobileNet with a *64x64* window and stride 64

In Table 5.8 it is possible to see the results for the obtained rust percentage existing in the structure for each of the test images. From these results it is clear that the calculated ratio is very far from the ground truth. From this it can be concluded that the model based on the MobileNet architecture could not obtain good results to be used as a rust detection approach, this could be because the MobileNet, being a smaller and overall less powerful network, even though it was pre-trained on a large dataset, it could not learn features that were as robust as the ones learned by the VGG16 network.

Image Analysed	Ground Truth	MobileNet
Figure 5.4(a)	9.55%	0%
Figure 5.4(b)	2.29%	0%
Figure 5.4(c)	15.15%	3.53%
Figure 5.4(d)	6.06%	1.12%

Table 5.8: Percentage of structure that contains rust calculated for the MobileNet architecture

5.5 Pixel-based approach Results

The SegNet architecture used, as mentioned previously, was implemented resorting to the $keras\_segmentation$ [42] open source library, which contains an already implemented model of a SegNet based on the VGG16 architecture. In order to implement the model utilizing this library, it is necessary to define the input size of the model, since the segmentation image at the output of the SegNet model will have the same size as the input image.

5.5.1 Data Pre-processing

For this approach, the input requires some sort of resize in order for it to be able to be used in the SegNet, this resize should be done to . However, the resize of the image may lead to loss of information that influences the final result. As such the original data was resized to two different sizes *608x608*, which was the default input size of the implemented SegNet architecture, and *1024x1024*, which is the smallest width of the images in the original dataset. Since the dataset consists of images with different aspect ratios, the width and height of the resize were defined to be the same.

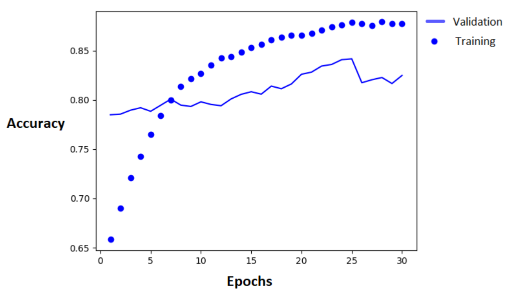
5.5.2 Training the SegNet model

Training of a SegNet is very similar to the training of a regular CNN. In order to obtain good results and avoid overfitting as much as possible, tuning of the hyperparameters is an extremely important part of training. As such, the following hyperparameters were tuned according to observations done during training.

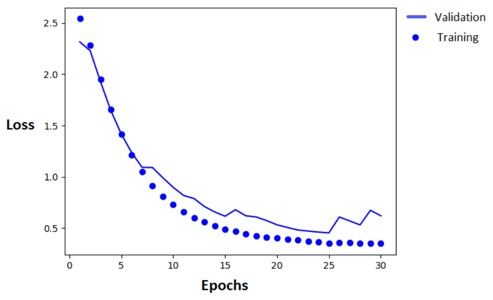
- **batch size**; for this approach the batch size used was 2, this value is much smaller than the one used in the sliding window approach. This is due to the larger size of images being used, which consume more memory individually.
- **number of epochs**; after some testing, the number of epochs chosen to train the model was 30. Since the amount of available data is not very large, even after performing augmentation, it was verified that the model started overfitting if trained for longer.
- **optimizer**; the optimizer used was the ADAM optimizer. However, due to the limitations of the library used, the learning rate of the optimizer was not tuned in this case, and was left at the default of 10^{-3}.

5.5.3 Training and Testing Results of the SegNet Model

The accuracy and loss curves for the SegNet model trained with an input of *1024x1024* can be seen in Figure 5.9. It is possible to see that the validation accuracy and loss are somewhat irregular, which to be expected since the VGG16 is a deep neural network that was trained end-to-end on a small dataset. Even so the model starts to stagnate at around 80% accuracy with a loss of 0.5, it should be noted that, in the final five epochs, the loss tends to start increasing, which is a sign that the model could be starting to overfit the training data.



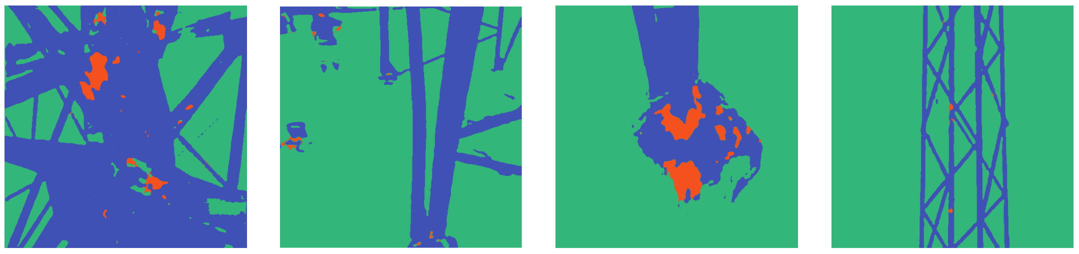
(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 5.9: Accuracy and loss for SegNet model trained with images of size *1024x1024*

Figure 5.10 shows the segmentation's obtained utilizing this model, while Table 5.9 show the results of the intersection over union with the original mask, as well as the execution time of this methodology. Looking at these results, it is possible to see that the values of the intersection over union are quite good, given that all of them are above 75%, however, in particular in Figure 5.10(b) the shortcomings of the training dataset can be seen. Since the training and validation dataset have few examples of people in the background, the models decides that the closest thing to the clothes is the class "structure", while determining that the face and hands are more similar to "rust".



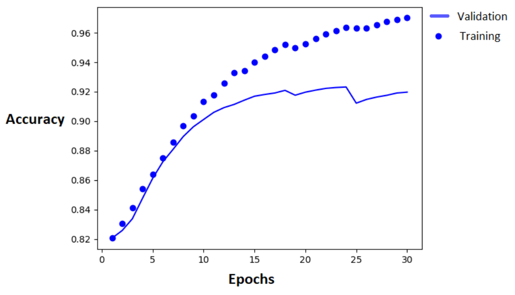
(a) **(b)** **(c)** **(d)**

Figure 5.10: Segmentation obtained from using a SegNet methodology with an input size of *1024x1024*

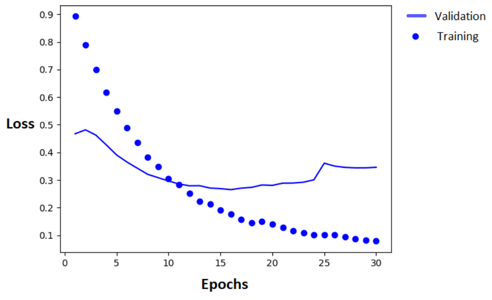
Image Analysed	Intersection over Union	Execution Time (seconds)
Figure 5.4(a)	76.54%	2.73
Figure 5.4(b)	82.00%	0.37
Figure 5.4(c)	76.89%	0.43
Figure 5.4(d)	83.69%	0.36

Table 5.9: Intersection over Union and execution time of SegNet segmentation for an input of *1024x1024*

In Figure 5.11, it is possible to see the accuracy and loss graphs for the SegNet trained with input images of *608x608*. The validation accuracy of this model starts stabilizing at around 91% while the loss drops to 0.3 and then, similarly to the model trained with *1024x1024* images, starts increasing in the last five epochs. However, the training accuracy of this model does not seem to stagnate and keeps improving, tending to a training accuracy of 100%, this is another sign that the model might be overfitting to the training data.



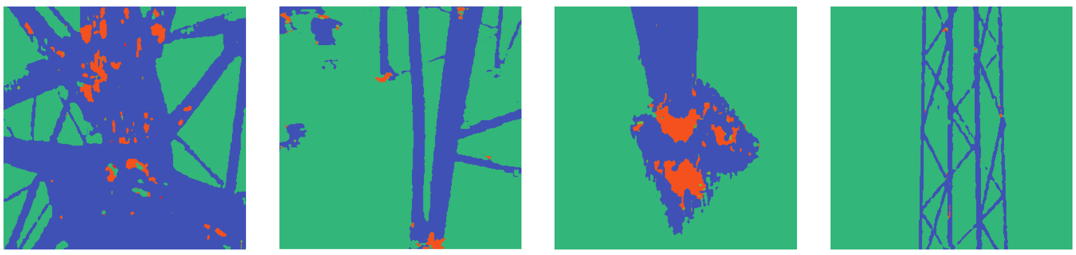
(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 5.11: Accuracy and loss for SegNet model trained with images of size *608x608*

The results of the segmentation obtained from using this model can be seen in Figure 5.12 and, just like the results obtained for the model trained with *1024x1024* images, the network incorrectly detects people as "structure" and "rust". The intersection over union of the original and generated masks can be seen in Figure 5.10. These results are very similar to the ones obtained through the model trained on images with size *1024x1024*, however all results expect the one obtained for Figure 5.4(c) were slightly inferior. Even though using a smaller image provided a faster execution time with only 1-2% worse performance, the execution times are so small in both cases that it does not justify the drop in performance.



(a) **(b)** **(c)** **(d)**

Figure 5.12: Segmentation obtained from using a SegNet methodology with an input size of *608x608*

Image Analysed	Intersection over Union	Execution Time (seconds)
Figure 5.4(a)	74.18%	1.84
Figure 5.4(b)	81.92%	0.10
Figure 5.4(c)	79.17%	0.10
Figure 5.4(d)	81.36%	0.10

Table 5.10: Intersection over Union and execution time of SegNet segmentation for an input of *608x608*

5.5.4 Ratio Calculation

In Table 5.11, the values of the ratio of rusted structure can be seen for the original mask, the resized original masks and the predictions. It is to be noted that, a resize in the image will lead to an error in the percentage of structure that is corroded.

Some values obtained by the SegNet architecture are closer to the ground truth ratios than the resized ground truth image ratios. However this is due to the models not being 100% accurate, which is not ideal. It is also important to note that the resized ground truth in both cases showed a reduction in the value of the rust-structure ratio, which is a disadvantage of using this approach.

Image Analysed	Ground Truth	Resized *1024x1024*	Resized *608x608*	*1024x1024*	*608x608*
Figure 5.4(a)	9.55%	8.10%	5.84%	3%	4.45%
Figure 5.4(b)	2.29%	1.17%	0.75%	0.84%	2.81%
Figure 5.4(c)	15.15%	10.21%	8.51%	15.28%	15.73%
Figure 5.4(d)	6.06%	1.6%	0.67%	0.52%	0.62%

Table 5.11: Percentage of structure that contains rust calculated for the SegNet approach.

5.6 Comparison of Pixel-based and Block-based Approaches on a Larger Test Set

After obtaining preliminary results for both methodologies, the two of them were compared resorting to the intersection over union of each class, "rust" and "structure" as well as the precision, recall and f1-score. For this, nine test sets were generated, each with ten random test images from the original data set. Each approach was then trained on the remaining images of the original data set and tested with each test set, with each one of these nine pairs of training and test data being designated as a " fold". The average results of the 10-fold cross validation tests will be reported.

Obtaining a larger amount of results in this way will give a better representation on how each approach is detecting each class as well as how they perform in different situations, notably when images are captured at different distances or when a large amount of background is present in the image.
Table 5.12, presents the average block-based result for the intersection over union, precision, recall and f1-score, obtained by testing and training the model on each of "fold". Similar results for the pixel-based

approach can be seen of Table 5.13.

	Intersection over Union	Precision	Recall	F1-Score
Rust	7.17%	16.77%	10.46%	11.36%
Structure	20.80%	74.64%	22.96%	31.91%

Table 5.12: Mean of the evaluation metrics for the block-based approach

	Intersection over Union	Precision	Recall	F1-Score
Rust	13.67%	25.87%	23.88%	21.26%
Structure	68.37%	79.89%	83.48%	78.62%

Table 5.13: Mean of the evaluation metrics for the pixel-based approach

These results show that, with this data set, the block-based approach shows a weaker performance than the pixel-based approach. Something that is noticeable is that the "rust" class is the one that shows the worse performance in both cases. This can be the due to limitations of the original data set, but, in the pixel-based approach case, the image re-scaling could have affected these results.

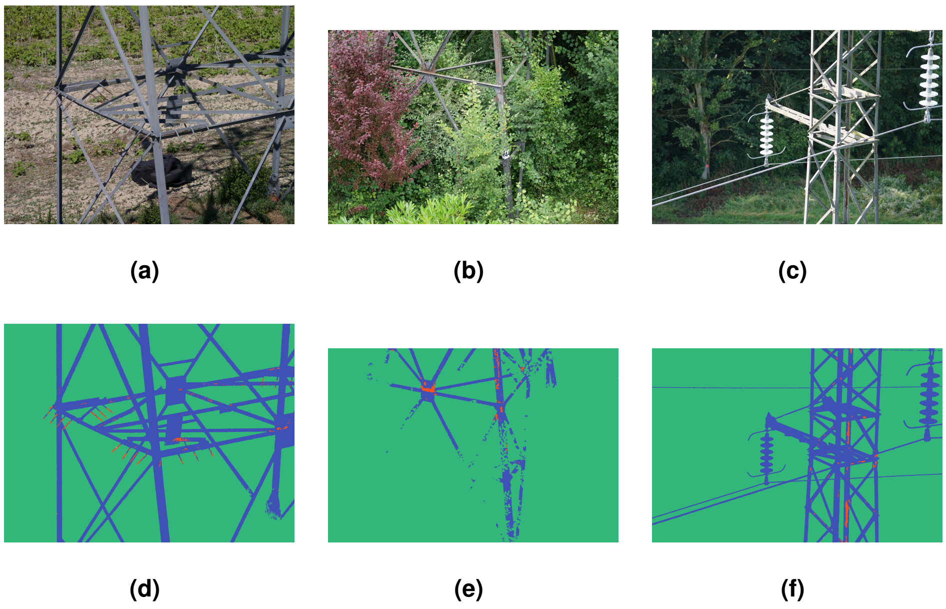


Figure 5.13: Original and ground truth of some images that showed lower performance that the average using the pixel-based approach. a)-Original test image 1; b)-Original test image 2; c)-Original test image 3; d)-Ground truth of test image 1; e)-Ground truth of test image 2 ; f)-Ground truth of test image 3

In order to verify the effects of this scaling, a few images that showed lower than average performance were analysed and divided into several smaller, *1024x1024* pixels, images in order to be analysed again by the pixel-based approach. The original image and the ground truth of these images can be seen in Figure 5.13.

In Figure 5.14, it is possible to see the results obtained by the pixel-based approach by resizing the original image to a size of *1024x1024* pixels, while in Figure 5.15, it is possible to see the results of applying the same methodology to several blocks of the image.

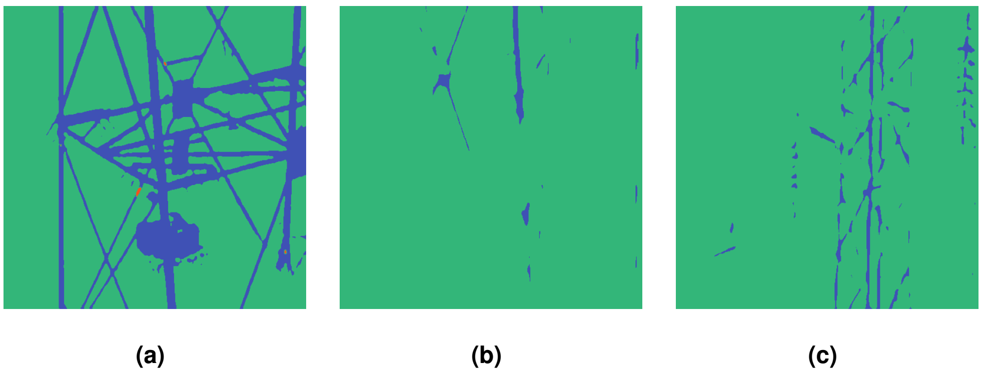


Figure 5.14: Segmentation masks obtained by resizing the images and using the pixel-based approach. a)-Resized segmentation of test image 1; b)-Resized segmentation of test image 2; c)-Resized segmentation of test image 3

Comparing the segmentation of these figures, it is possible to see that, although some parts of the image have become better segmented, like how in Figure 5.15(b) the structure more well defined when compared to Figure 5.14(b). However, it is possible to see that the background in the images in Figure 5.15 contains much more misclassified areas, this might be due to the network having less information from the total image to work with or due to the fact that the lack of background only images in the original data set.

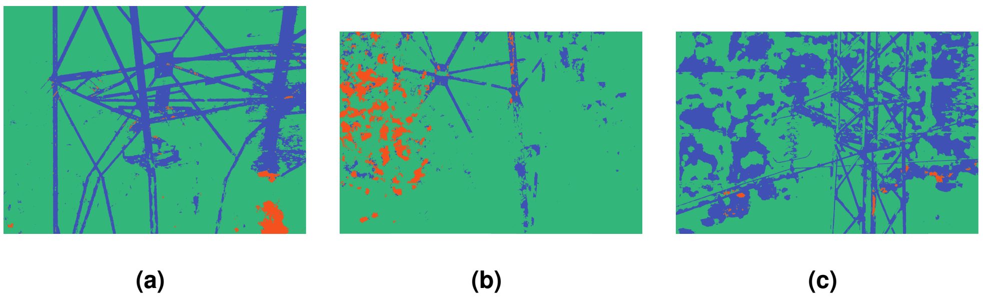


Figure 5.15: Segmentation masks obtained by analysing sections of the images and using the pixel-based approach. a)-Area segmentation of test image 1; b)-Area segmentation of test image 2; c)-Area segmentation of test image 3

In Table 5.14 and Table 5.15, it can be seen the metric results obtained for the "rust" and "structure" classes, respectively. From these results it it possible to see that, even though there are some minor improvements in the "rust" detection, the "structure" detection results are considerably lower, which is most likely due to problems in classifying the more detailed background regions, as this type of detail was not present in the training set images.

Even thought dividing the image into several areas and analysing each one resulted in a higher amount of false positives, which is more noticeable in the misclassified "rust" in Figure 5.15(b) and the misclassified "structure" in Figure 5.15(c), it should be noted that there are areas that are much better classified, in particular in Figure 5.15(b). The results obtained by applying the metrics do not show a good performance, however, by simply observing the image, it is possible to see how much the structure segmentation as improved, being that in Figure 5.14(b) it is almost non-existent.

Input		Intersection over Union	Precision	Recall	F1-Score
Image Area	Figure 5.13(a)	0.03 %	0.04%	0.09%	0.05%
	Figure 5.13(b)	0.17%	0.18%	0.29%	0.34%
	Figure 5.13(c)	0.27%	0.56%	0.51%	0.53%
Full Image	Figure 5.13(a)	0.18%	0.25%	0.64%	0.36%
	Figure 5.13(b)	0	0	0	0
	Figure 5.13(c)	0	0	0	0

Table 5.14: Metrics results for the "rust" class for the full image input and image area input

Input		Intersection over Union	Precision	Recall	F1-Score
Image Area	Figure 5.13(a)	17.025%	31.43%	27.08	29.09%
	Figure 5.13(b)	17.75%	35.81%	26.02%	30.14%
	Figure 5.13(c)	15.29%	19.99%	39.35%	26.52%
Full Image	Figure 5.13(a)	72.51%	73.78%	97.69%	84.07%
	Figure 5.13(b)	22.89%	62.97%	26.45%	37.25%
	Figure 5.13(c)	25.29%	84.67%	26.49%	40.36%

Table 5.15: Metrics results for the "structure" class for the full image input and image area input

5.7 Performance at Different Distances

In order to verify if the systems perform differently for each of the distances in the data set, more testes where done, training each data set on several folds of each distance, where 10% of the images from different distances were used for testing and the rest were used for training. This was repeated until all images taken at that distance were used for testing. The results from these tests are presented in Table 5.16 and Table 5.17.

Distance	Class	Intersection over Union	Precision	Recall	F1-Score
Far	Rust	2.6%	22.98%	3.01%	5.02%
	Structure	11.19%	36.52%	25.99%	19.58%
Medium	Rust	3.80%	17.98%	4.02%	6.01%
	Structure	15.60%	79.30%	17.79%	26.14%
Close	Rust	25.63%	55.32%	35.13%	38.87%
	Structure	35.44%	82.52%	36.91%	51.34%

Table 5.16: Performance of the block-based approach on the different distances of images taken

Distance	Class	Intersection over Union	Precision	Recall	F1-Score
Far	Rust	22.50%	63.05%	25.63%	35.22%
	Structure	66.66%	85.48%	75.79%	79.38%
Medium	Rust	19.00%	48.56%	21.98%	29.82%
	Structure	74.12%	89.94%	78.20%	83.53%
Close	Rust	54.65%	71.44%	77.44%	69.34%
	Structure	81.11%	97.89%	82.57%	89.26%

Table 5.17: Performance of the pixel-based approach on the different distances of images taken

From these results, it is possible to see that, in both approaches, that the closer to the structure the image was taken, the better the performance. In particular, the pixel-based approach has a dramatic increase in the performance obtained, being above 50% for rust and above 80% for structure.

5.8 Discussion

From the obtained results, it can be concluded that there are aspects that can still be improved. Although the intersection over union metric results seem to have good results, the percentage of structure that is rusted still has a significant amount of error in some cases.

Analysing the results obtained from utilizing the block-based approach, it can be seen that this approach was not very effective. Not only are the results obtained below what was expected but the execution time of this approach is also slower than expected, taking anywhere from 25 seconds for lower resolution images, to 99 seconds for higher resolution images, while utilizing a window with size *64x64* pixels with no overlap. The performance can possibly be improved by using a larger and more generic dataset, since the available dataset was quite small and had a large amount of similar images, which could have led the model to overfit for the dominant class of images, while not performing sufficiently well for the most different image samples.

Comparing the results obtained from using the VGG16 and the MobileNet architectures, it can be seen that the MobileNet performs much worse than the VGG16. As it stands, a MobileNet based architecture is unusable due to the low intersection over union score. However, it is also noticeable that the network overfits during training, which means that using a larger and more generic dataset may improve results and make this architecture usable. Also using other training approaches, such as distillation, following a student-teacher learning model, might lead to increased performance.

The results obtained from pixel-based approach, utilizing the SegNet, are superior to the ones obtained using the block-based approach. Not only does the segmentation obtained from the SegNet obtain better results, but it also has a much faster execution time. The major problem of this approach is that having to resize the image causes a natural drop in the amount of rust existing in structure, caused by a reduction in the amount of information derived from the image resizing.

Although the results obtained from using the pixel-based approach on different areas of the image showed a performance reduction in the "structure" class and very small improvements in the results for the "rust" class, it can be seen that there are cases, where it can be observed that the segmentation of the structure is much better that when segmenting the entire image, at the cost of the background being often misclassified.

The size and uniformity problems of the data set can justify the lack of performance that these methods shown, as such, using a more well constructed data set to train the models can increase their performance.

6

Conclusion

Contents

6.1 Conclusions

The goal for this thesis was to build a system with the goal of being able to detect the amount of rust existing in an image and use that to determine if it needs human inspection. For this two deep learning systems were used, an approach based utilizing a traditional CNN with a sliding window to detect areas with "rust" and "structure" and another approached based on utilizing a SegNet to generate a semantic segmentation of the image and using that image to see how many pixels were classified as "rust" and how many pixels were classified as "structure".

The idea of using a sliding window approach came from the results obtained in the work done by Atha [8]. This approach was put into practice with the goal of detecting regions on images that contained rust and implemented a VGG16 architecture, which obtained results with high accuracy for rust detection. Given these high accuracy results, it was chosen to experiment with this approach and create a system that could detect both "rust" and "structure".

The results obtained from applying the sliding window approach were worse than expected, which could have been due to the construction of the data set used in order to train the networks. There are also cases where the approach presents higher than desired execution times, which could be a problem if there is a large quantity of images to evaluate. It was concluded that, with these results, the sliding window system was not capable of detecting rust on structure in a reliable manner.

After obtaining results that were less optimal than the ones expected, it was proposed the implementation of a SegNet approach that could directly output an image segmentation from the original image, which could in turn be used to determine how much of the pixels of an image were classified as "rust" and how many where classified as "structure". When testing this approach it was found that the obtained segmentation was much smoother than a segmentation obtained using a sliding window approach. More importantly, this approach proved to show higher accuracy and faster execution time on the same limited data set. However it does suffer from the need to resize an image before segmenting it, which may affect the ratio between the amount of "rust" and "structure", making it less accurate.

This thesis shows that it is possible to build a system bases on semantic segmentation that is capable of successfully detect the amount of rust existing in a structure. Although there are some problems involving the change in image size affecting the amount of rust present in a structure, the system was still capable of producing an acceptable segmentation of the the image, given the limited data available, and showed promising results determining the amount of rust present in a structure. Utilizing a SegNet in order to detect the presence of rust in structure seems to be a promising way of approaching rust detection in industrial structures in the future.

6.2 Future Work

This thesis demonstrates that utilizing semantic segmentation approaches is an effective way of detecting the presence of rust and structure in an image. The SegNet model based on the VGG16 showed showed better performance than the sliding window approach with a limited data set even though it was not pre-trained on any kind of larger data set. Training this model on a large segmentation data set like CityScapes [43] could help improve its performance by later fine-tuning the model with the Axians data set. Segmenting areas of the image in order to avoid losing too much detail of the structure seems like it would worthwhile investigating with a network pre-trained on a larger data set and it would also be interesting to apply some sort of background subtraction before applying this approach to different areas of large images.

An interesting proposition for future projects is to experiment with different kinds of segmentation models, approaches like U-Net [34] and fully convolutional networks [33] are other semantic segmentation methods that have been used and have also performed well in other applications, namely the U-Net architecture as shown impressive results in medical applications [44].

Techniques based on instance segmentation could also be interesting experiments, since it could allow the development of a system that detects distinct rust areas. This makes it possible to develop an approach that analysis each section of the image, or one that presents all rusted areas detected in a structure for evaluation or even that classifies each rust region as a certain degree of damage. However making a solution like this would need a new data set, composed of different rust degrees. Architectures like Mask-RCNN [45] are very effective at instance segmentation and are even possible to be used on very high definition videos to detect objects in real time.

Bibliography

[1] H. Furuta, T. Deguchi, and M. Kushida, “Neural network analysis of structural damage due to corrosion,” Tech. Rep., 2002.

[2] S. Lee, L. M. Chang, and M. Skibniewski, “Automated recognition of surface defects using digital color image processing,” *Automation in Construction*, vol. 15, no. 4, pp. 540–549, 7 2006. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0926580505000981

[3] K. Simonyan, A. Zisserman, S. Ruder, S. J. Pan, Q. Yang, OpenAI, I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, A. Courville, D. Frossard, D.-A. Clevert, T. Unterthiner, S. Hochreiter, A. Cook, V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, J. Yosinski, J. Clune, Y. Bengio, H. Lipson, V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, and Stanford University, “CS231n Convolutional Neural Networks for Visual Recognition,” *arXiv preprint arXiv:1511.07289*, vol. 22, no. 10, pp. 1345–1359, 2016. [Online]. Available: http://cs231n.github.io/

[4] F.-F. Li, J. Johnson, and S. Yeung, “Lecture 5: Convolutional Neural Networks,” 2017. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture5.pdf

[5] R. C. Alex Kendall, Vijay Badrinarayanan. Segnet. [Online]. Available: http://www-cs-faculty.stanford.edu/~uno/abcde.html

[6] M. E. V. S. SASTRI, EDWARD GHALI, *Corrosion Prevention and Protection Practical Solutions*. Wiley, 2007.

[7] 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,” *Computer Science & Information Technology (CS & IT)*, pp. 91–99, 2016. [Online]. Available: http://www.airccj.org/CSCP/vol6/csit65308.pdf

[8] 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.

[9] G. Koch, J. Varney, N. Thompson, O. Moghissi, M. Gould, and J. Payer, “NACE 2016 Impact Study,” Tech. Rep., 2016. [Online]. Available: http://impact.nace.org/documents/Nace-International-Report.pdf

[10] P. T. Gilbert, “What is Corrosion?” *British Corrosion Journal*, vol. 13, no. 4, pp. 153–153, 2012. [Online]. Available: https://www.electrochem.org/dl/interface/spr/spr06/spr06_p24-26.pdf

[11] P. Viola and M. Jones, “Managing work role performance: Challenges for twenty-first century organizations and their employees.” *Rapid Object Detection using a Boosted Cascade of Simple Features*, pp. 511–518, 2001.

[12] T. Surasak, I. Takahiro, C. H. Cheng, C. E. Wang, and P. Y. Sheng, “Histogram of oriented gradients for human detection in video,” *Proceedings of 2018 5th International Conference on Business and Industrial Research: Smart Technology for Next Generation of Information, Engineering, Business and Social Science, ICBIR 2018*, pp. 172–176, 2018.

[13] P. Viola, M. Jones, and M. Energy, “Robust Real-Time Face Detection Intro to Face Detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Region-Based Convolutional Networks for Accurate Object Detection and Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016.

[15] V. Badrinarayanan, A. Handa, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling,” 2015. [Online]. Available: http://arxiv.org/abs/1505.07293

[16] M. R. Jahanshahi, J. S. Kelly, S. F. Masri, and G. S. Sukhatme, “A survey and evaluation of promising approaches for automatic image-based defect detection of bridge structures,” *Structure and Infrastructure Engineering*, vol. 5, no. 6, pp. 455–486, 2009.

[17] M. R. Jahanshahi and S. F. Masri, “Parametric Performance Evaluation of Wavelet-Based Corrosion Detection Algorithms for Condition Assessment of Civil Infrastructure Systems,” *Journal of Computing in Civil Engineering*, vol. 27, no. 4, pp. 345–357, 2012.

[18] T. K. Hanji, T. and K. Kitagawa, “3-D shape measurement of corroded surface by using digital stereography,” 2003, structural Health Monitoring and Intelligent Infrastructure: Proc., 1st Int. Conf.

on Structural Health Monitoring and Intelligent Infrastructure, Swets & Zeitlinger B.V., Lisse, The Netherlands, Vol. 1, 699–704.

[19] V. Pakrashi, F. Schoefs, J. B. Memet, and A. O'Connor, "ROC dependent event isolation method for image processing based assessment of corroded harbour structures," *Structure and Infrastructure Engineering*, vol. 6, no. 3, pp. 365–378, 2010.

[20] F. N. Medeiros, G. L. Ramalho, M. P. Bento, and L. C. Medeiros, "On the evaluation of texture and color features for nondestructive corrosion detection," *Eurasip Journal on Advances in Signal Processing*, vol. 2010, 2010. [Online]. Available: https://link.springer.com/content/pdf/10.1155%2F2010%2F817473.pdf

[21] R. M. Haralick, I. Dinstein, and K. Shanmugam, "Textural Features for Image Classification," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, 11 1973. [Online]. Available: http://ieeexplore.ieee.org/document/4309314/

[22] K. Y. Choi and S. S. Kim, "Morphological analysis and classification of types of surface corrosion damage by digital image processing," *Corrosion Science*, vol. 47, no. 1, pp. 1–15, 1 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0010938X0400126X

[23] X. Xie, "A Review of Recent Advances in Surface Defect Detection using Texture analysis Techniques," *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, vol. 7, no. 3, p. 1, 2008.

[24] S. Ghanta, T. Karp, and S. Lee, "Wavelet domain detection of rust in steel bridge images," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 1033–1036, 2011.

[25] G. Antonellis, A. G. Gavras, M. Panagiotou, B. L. Kutter, G. Guerrini, A. C. Sander, and P. J. Fox, "ImageNet Classification with Deep Convolutional Neural Networks," *ImageNet Classification with Deep Convolutional Neural Networks*, p. 9, 2012. [Online]. Available: http://code.google.com/p/cuda-convnet/

[26] L. Fei-Fei, J. Deng, and K. Li, "ImageNet: Constructing a large-scale image database," *Journal of Vision*, vol. 9, no. 8, pp. 1037–1037, 2010.

[27] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[28] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 12 1943. [Online]. Available: http://link.springer.com/10.1007/BF02478259

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," no. June, pp. 740–755, 2014.

[30] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 3 1996. [Online]. Available: http://ieeexplore.ieee.org/document/485891/

[31] H. Leung and S. Haykin, "The Complex Backpropagation Algorithm," *IEEE Transactions on Signal Processing*, vol. 39, no. 9, pp. 2101–2104, 1991. [Online]. Available: http://ieeexplore.ieee.org/document/134446/

[32] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. S. Kruthiventi, and R. V. Babu, "A Taxonomy of Deep Convolutional Neural Nets for Computer Vision," *Frontiers in Robotics and AI*, vol. 2, p. 36, 1 2016. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/frobt.2015.00036/abstract

[33] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," Tech. Rep. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

[34] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9351. Springer, Cham, 2015, pp. 234–241. [Online]. Available: http://link.springer.com/10.1007/978-3-319-24574-4_28

[35] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[36] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[37] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520. [Online]. Available: https://arxiv.org/pdf/1801.04381.pdf

[39] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” 12 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[40] M. Hossin and N. Sulaiman, “A Review on Evaluation Metrics For Data Classification Evaluations,” *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, pp. 1–11, 2015.

[41] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, “A survey on deep learning techniques for image and video semantic segmentation,” *Applied Soft Computing Journal*, vol. 70, pp. 41–65, 2018. [Online]. Available: https://doi.org/10.1016/j.asoc.2018.05.018

[42] D. Gupta, “image-segmentation-keras,” https://github.com/divamgupta/image-segmentation-keras.git, 2017.

[43] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 3213–3223, 2016.

[44] S. E. A. Raza, L. Cheung, D. Epstein, S. Pelengaris, M. Khan, and N. M. R. B, “MIMONet : Gland Segmentation Using Neural Network,” *Computer Methods and Programs in Biomedicine*, vol. 1, no. d, pp. 698–706, 2017.

[45] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask R-CNN,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2980–2988, 2017.