TÉCNICO LISBOA

Damage Detection and Classification in Wind Turbine's Blades using Automated Visual Inspection

Tiago Miguel da Cunha Cesteiro

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. Alexandra Bento Moutinho

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Alexandra Bento Moutinho
Member of the Committee: Prof. Jacinto Carlos Marques Peixoto do Nascimento

July 2020

Acknowledgements

First of all, I want to express my gratitude to my family, especially to my parents, who supported me through thick and thin. They gave me the opportunity to study in the best possible faculty, to do Erasmus programme and, most important, to become an engineer.

My profound gratitude to my girlfriend, Constança Lino, not only for the motivation provided but also for managing time and effort to draw almost every scheme present in this work.

My sincere thanks to a long time friend, Gonçalo Pais, for the availability and patience to help me with the implementation of the deep learning solution. Without him this work would not be possible.

I am extremely grateful to all my friends from Colégio Militar, who have been by my side through all these years of studies and never stopped motivating me.

Finally, a huge thanks to my supervisor Prof. Alexandra Bento Moutinho, for the unconditional support and orientation. Her guidance was essential for the development of this work.

Abstract

With the rising of awareness regarding global warming, renewable energy sources have been increasingly the focus of the energy industry. According to [25], in 2017, wind power generated the second most electricity amount (7.7%), from all renewable energies (the first one was hydropower with 85.2%). Due to wind erosion and others factors, wind turbine blades are components highly affected during work. In that sense, periodic inspections should be performed to prolong their lifetime. Visual inspections are performed by trained inspectors, however an automatic toll can be implemented to cut down the inspection time and total cost, while also guaranteeing overall coherence in the damages identified. Knowing there are still no automatic solutions for that purpose, this work aims to explore tools for automatic visual damage detection and classification in blades photographies, which were collected from wind turbines farms. For the purpose, two approaches were considered. In a first attempt, an algorithm was developed, using only classic computer vision methods, along with point clustering techniques. Knowing the limitations of this approach, deep neural networks were explored as second option. The purpose of the first approach is to establishing a performance reference for the neural networks. In the end, the approaches were compared for the same set of images, confirming the limitations of the first approach, as well as the potential of deep neural networks to adapt to non-trivial tasks, achieving good generalization and robustness.

Keywords: wind turbines, automated visual inspections, damages detection and classification, computer vision, deep learning

Resumo

Com o aumento da consciencialização relativamente ao aquecimento global, as fontes de energias renováveis são cada vez mais o foco da indústria de energia . De acordo com [25], em 2017, a energia do vento foi responsável por gerar a segunda maior quantidade de electricidade (7.7%), de todas as energias renováveis (a que gerou mais foi a hídrica com 85.2%). Devido à erosão do vento e outros factores, as pás das turbinas eólicas são componentes altamente afectados durante o funcionamento das mesmas. Nesse sentido, inspeções periódicas devem ser efectuadas para prolongar a vida útil. Inspecções visuais são efectuadas por inspectores treinados, no entanto uma ferramenta automática pode ser implementada para reduzir o tempo de inspecção e o custo total, garantindo também uma coerência geral nos danos identificados. Sabendo que ainda não existem soluções automáticas para o efeito, este trabalho visa explorar ferramentas para detectar e classificar danos visuais em fotografias das pás, cujo foram recolhidas nas quintas de turbinas eólicas. Para o efeito foram consideradas duas abordagens. Numa primeira tentativa, foi desenvolvido um algoritmo, utilizando apenas métodos clássicos de visão computacional, juntamente com técnicas de agrupamento de pontos. Sabendo as limitações presentes nesta abordagem, foi explorada, como segunda solução, a utilização de *deep neural networks*. A primeira abordagem tem o propósito de estabelecer uma referência de desempenho para as redes neuronais. No fim, as abordagens foram comparadas para um mesmo conjunto de imagens, confirmando tanto as limitações da primeira abordagem, como o potencial das *deep neural networks* para se adaptarem a tarefas não triviais, atingindo uma boa generalização e robustez.

Palavras-chave: turbinas eólicas, inspeção visual automatica, deteção e classificação de danos, visão computacional, aprendizagem profunda

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wind power was one of the primordial energy sources to be used by humans. Since ancient times, various cultures and civilizations were able to convert wind energy into mechanical power. In the early history, over 5,000 years ago, the Egyptian civilization started using wind to sail ships across the Nile River. Later on, the firsts windmills were built, mostly to grind wheat, corn and other grains. Wind energy applications were reinvented and further used as years passed by, until they were replaced by steam and internal combustion engines, that utilize alternative energy sources, such as coil, fossil fuel, oil, natural gas and nuclear energy.

Only a few decades ago, the advance of technology and the increasing of awareness about global warming and environmental pollution have led the energy industry to rely, more and more, on renewable energy sources. Moreover, the costs reduction in this sector made it a reliable and competitive energy source. In this manner, wind turbines have become a more common technology for electricity generation. Figure illustrates the power generation capacity in Europe, for the past decade [54]. It is conclusive that wind power has been overtaking other power generation sources, being second in 2017.

1.1 Wind Turbines

With the rising of this industry, wind power generators went through some major innovations, like the optimization of blades design, that was addressed with improvement of material quality, structural design and aerodynamics, resulting in important gains towards the operational efficiency and noise reduction [27]. A typical modern wind turbin is shown in figure 1.2. On the bottom, a foundation (fig. 1.2-1) fixes the turbine into the ground, assuring overall stability, followed by a tower (fig. 1.2-2) designed not only to hold the weight of the upper body, but also to absorb the static loads induced by the wind power. On the top, a nacelle (fig. 1.2-3) aggregates all the machinery able to transform mechanical energy into

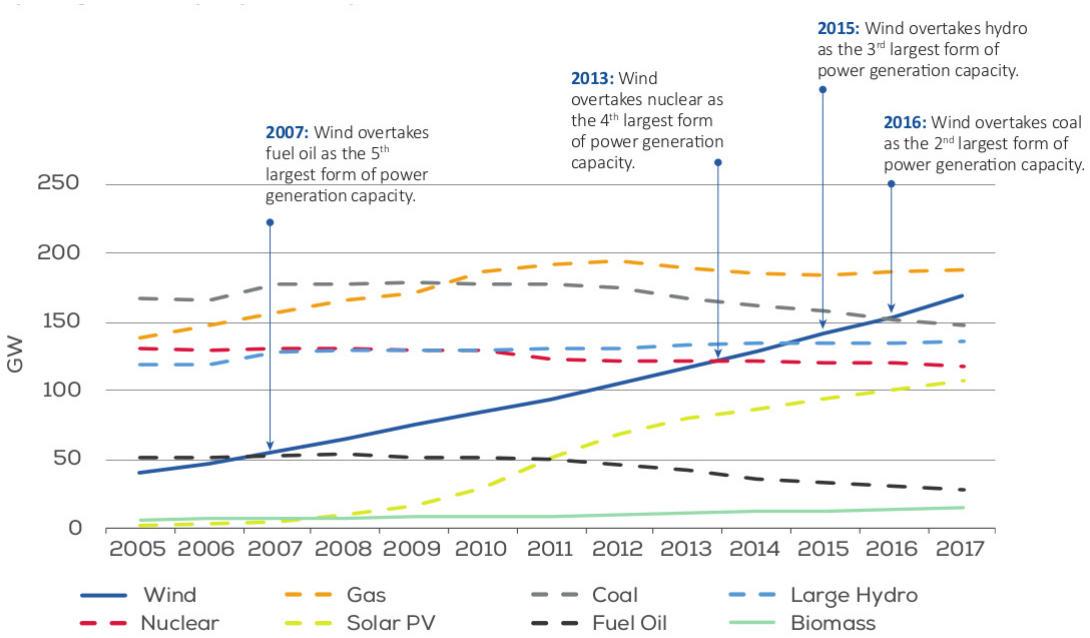


Figure 1.1: Total power generation capacity in the European Union 2005-2017 (source [54])

electrical one, for instance, generator, breaks and bearings. The hub (fig. 1.2-4) is the center piece of the rotor, where the rotor blades (fig. 1.2-5) are attached to. Based on the lift principle, the blades convert wind energy into mechanic one, which is then transformed into electrical energy by a generator present in the hub.

Due to the high exposure to wind loads, the rotor blades are excessively stressed parts, with high potential to develop damages. According to [55], blades costs can account for 15 to 20 % of the total turbines cost and have the most expensive and time consuming repair. Thus, blades are a crucial component to observe during design and operational time.

1.2 Rotor Blades

The rotor blade is a complex structure, composed mainly of glass-fiber, balsa wood (or plastic foam) and polymer adhesive [5]. Manufacturing a blade requires two halves, which are laminated separately, and a gel coat made entirely of epoxy or polyester, thickened with silicic acid, that is applied to protect it from UV degradation and water penetration [26]. The halves are assembled together around the spar using adhesive bounding.

A common blade cross-section is illustrated in Figure 1.3. It has a teardrop profile, similar to airplane

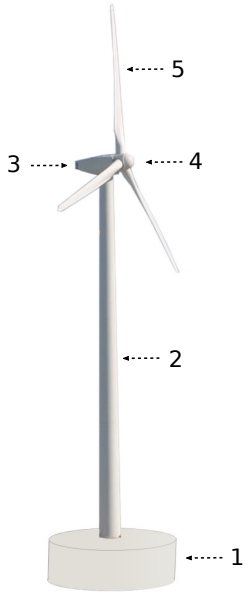


Figure 1.2: Wind turbine main components (adapted from [18]): 1 - Foundation, 2 - Tower, 3 - Nacelle, 4 - Hub , 5 - Rotor Blades

wings. The upper and lower shell (fig. 1.3-3 and fig. 1.3-4) shapes are responsible for the lift and drag principle, which causes the blades to spin. During work, the blade receives the wind flow from the leading edge (LE) (fig. 1.3-1) to the trailing edge (TE)(fig. 1.3-2). A main spar, constituted by the shear webs (fig. 1.3-5) and spar caps (fig. 1.3-6), is a key element to assure strength to the structure. As flapwise and edgewise bending moments are the origin of roughly 97% of the damages [51], it is essential to reinforce the blade against those. Figure 1.4 shows a representation of those two moments as well as the profile shape of the blades. While the edgewise bending moment can be prevented by adding composite sandwich laminates in the LE and TE, to prevent the flapwise bending moment it is necessary to include of spar caps to handle the longitudinal stresses. Finally, the spar webs carry the flapwise shear forces. All these elements increase the buckling resistance of the structure, assuring a longer lifetime.

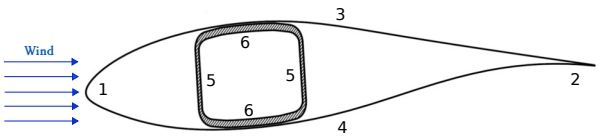


Figure 1.3: Cross-Section A-A of rotor blade with beam-like spar (source [36]): 1 - Leading edge, 2 - Trailing edge, 3 - Suction Side, 4 - Pressure Side, 5 - Shear Webs, 6 - Spar Caps

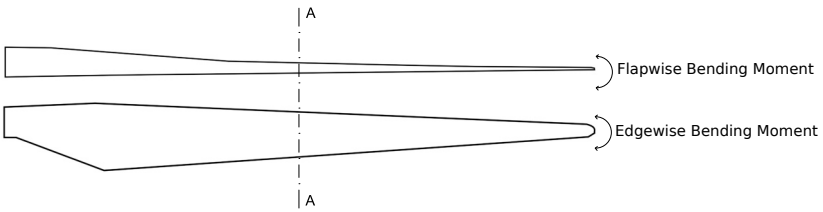


Figure 1.4: Representation of flapwise and edgewise bending moments

Even though the described design is carried to prevent rotor blades damages, with 90% of operational time [8], accumulation of cycling loads and therefore manifestation of fatigue is very likely to happen. This is why the expected lifetime of a wind turbine is around 20 years [27].

1.3 Rotor Blades Inspections

By now, it is clear that a good maintenance should be conducted by wind farm owners to extend, as far as possible, turbines lifetime. Any failure can cause significant repair costs, operational/production breaks and put human life at risk.

Damages usually start with small sizes and evolve over time, so an implementation of periodic inspections is essential to prevent unrepairable damages or highly priced repairs.

Nowadays, blade inspections have a range of different techniques, being all of them non-destructive testing. According to [5], this range contains visual inspections, infra-red thermography, ultrasonic and tap testing, digital radiography, acoustic emission, vibration analysis and microwave or terahertz techniques.

Visual inspections is the most commonly used technique, where an expert is attached to ropes and hanged at blades height for a closer look. Some disadvantages arise from this. For instance, the inspector must have a special training and be willing to put his/her life at risk. Also, inspectiong all the blades length is time consuming, even for trained experts, and rope access may be difficult in some cases, such as, in offshore wind farms.

Many companies already use drones to collect image data from rotor blades, so they can be inspected one-by-one by inspectors, using an online platform. This approach overcomes the usage of rope accessing and cuts down the average time of inspection per turbine, but it still requires manual inspection of the images, which can lead to inaccurate and time varying inspections. This is why modern industries search, every day, to automate as much as possible every process. Nevertheless, the automation of rotor blades inspections is still a newly research field and hence with a lot to explore.

1.4 Automated Visual inspections

Automated Visual inspections (AVI) are accomplished by means of computer vision processes. This field was firstly introduced in the 1970s [49] and focuses on methods to transfer human vision perception into machines. This perception comes manly from image processing methods, which enhances properties and Regions of Interest (RoIs) in images. By combining image processing with other tools, one can develop a computer vision algorithm for a specific AVI case.

Figure 1.5 displays a timeline with the main tools of computer vision, since 1970s. With the arrival of the new millennium, fields such as machine learning started to interact with computer vision, giving machines the ability to learn how to perform image processing, while also understanding the RoIs in images. This was a huge step towards artificial intelligence, since previous methods were less robust and highly dependent on the dataset variety. Without learning, computer vision methods require to be adjusted, by a human, to a set of images. Meaning, if the dataset presents high variety, the choice of right methods and parameters is not straightfoward.

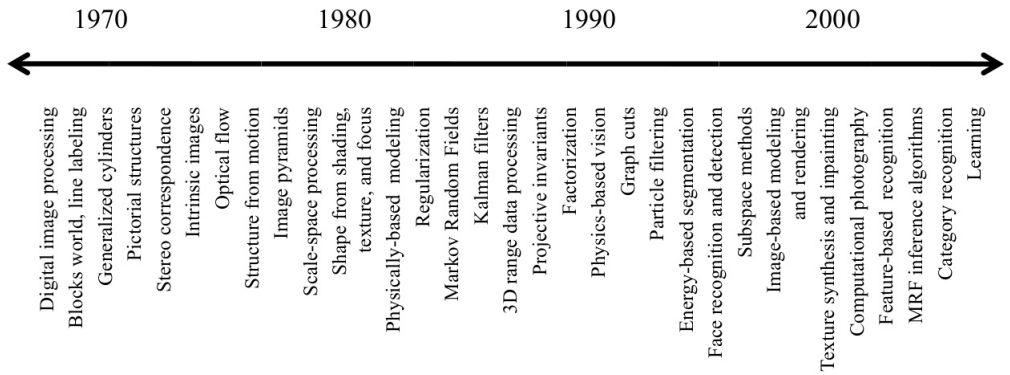


Figure 1.5: Computer Vision innovations timeline (source [49])

Nonetheless, AVI started to be used globally in the manufacturing sector in the beginning of 1980s [28]. By guaranteeing controlled image conditions, such as illuminance, scale, rotation and translation, computer vision methods showed to be reliable for real implementations. Since then, AVI expanded to many industries, such as, the automobile, packing, textile and others [28].

Concluding, AVI can be defined as the usage of computer vision and other additional tools to perform a human visual function, such as detecting damages in objects. Advantages of AVI over manual inspections can be numerous. It is cheaper and faster, more consistent, accurate and reliable. Also, it does not require human presence, which is an important factor in the case of inspecting turbine blades.

1.5 Problem Formulation

Having all this said, this work will investigate whether AVI is feasible to be implemented in blade inspections of wind turbines. Using a dataset of blades photographs containing several damages, as well a *.json* file with annotations of detections made by blade inspectors. The file included also the relative coordinates of bounding boxes, with a damage type identification and a description. Relevant comments from inspectors were included too.

In order to understand possible damages, the dataset was analyzed. With a first look at the inspections made, it was notice an obstacle. As the photos were evaluated by different inspectors, and because a universal labeling for damages does not exist, the given .json file had within 153 different labels in total, whereas many refer to the same damage type. In an attempt to unify all labels, a deep observation of the photos was performed, searching for labels with similar damage type. For instance, when the gel coat starts to degrade and presents small holes, some inspectors labeled as pitting, while others as peeling. Furthermore, some inspectors showed little specificity when classifying, or identified components of the blade only for record. After the aggregation of similar labels and exclusion of irrelevant labels, the remaining labels were the ones present in figure 1.6. Comparing these labels occurrences, it is notice that three labels show a considerable smaller amount of instances. The problem with lack of data in those labels is that results would not be conclusive. Thus the delamination, the oil stains and the blistering damage types were removed, as they are present in few images. Furthermore, as the data were analyzed, some of the bounding boxes showed to be imprecise or the damage type was incorrect, leading for the need to relabel all the dataset.

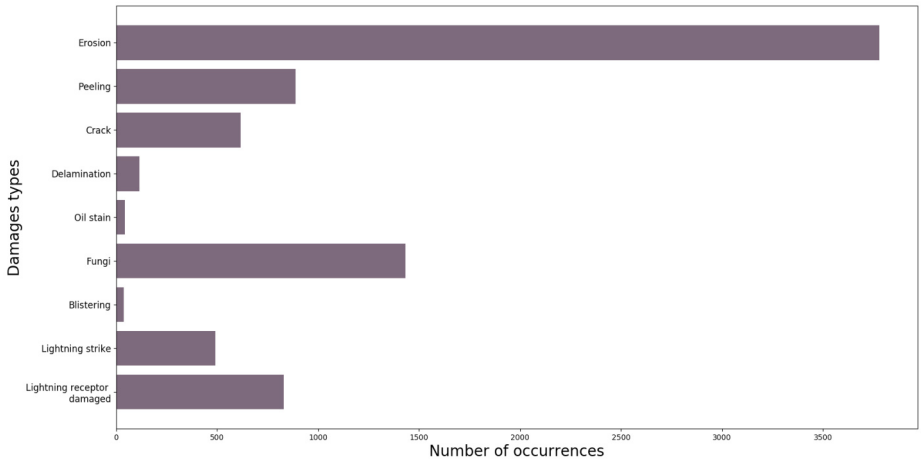


Figure 1.6: Number of instances for each damage label

The final dataset, with the correct and consistent labels is described in table 1.1. Figure 1.7 displays examples of the damages to be considered in this work, with the corresponding bounding boxes.

Damage type	**Number of Occurrences**	**Number of images**
Erosion	3948	2475
Peeling	1366	1021
Crack	890	507
Fungi	949	675
Lightning Strike	460	391
Lightning Receptor Damaged	744	743
Total	8357	5750

Table 1.1: Number of occurrences of each damage in revised dataset

(a) Erosion in the LE, leading to degradation of the gel coat

(b) Small peeling of the gel coat

(c) Longitudinal Crack

(d) Formation of fungus in the TE

(e) Lightning Strikes the receptor and a portion of the blade

(f) Receptor partially covered by top coat and with a crack on sealant around it

Figure 1.7: Examples of damages to identify

Each damage type was considered to have the following characteristics:

1. Erosion: appears only in the LE. Includes any damage provoked by wind erosion, like peeling, craters or pitting.

2. Peeling: appears in the pressure and suction side. Includes any size of peeling or pitting degradation of the gel coat.

3. Cracks: appears in any part of blade. Includes superficial crack or scratch of the gel coat.

4. Fungi: appears in any part of blade, typically closer to the LE. Includes the presence of fungus in the blade, which contains a high density pitting of the gel coat.

5. Lighting Strike (LS): appears in any part of the blade, typically on top of the lighting receptor. Includes any evidence of a lighting strike, such as burned areas.

6. Lighting Receptor Damaged (LRD): covers irregularities in the lighting receptor, for instance, if it is missing, covered by top coat or has a crack in the sealant around it.

1.6 Solutions Proposed

This work explores two different solutions. The first one gives use to classical computer vision methods. For the last decades, such methods have been the backbone of AVI, showing reliable results for many applications, yet presents some limitations. Developing this solution implies an individual description of each damage for feature extraction, which is hardworking and highly dependents on the images variety.

A second approach employs supervised deep learning. The relevant increase in computational power over the last years enabled deep learning to be used as alternative solution for many applications in the computer vision field. From image recognition and classification to pattern recognition, deep learning can outstand the human mind in many ways. The usage of deep neural networks is being progressively more accessible to everyone, as online frameworks provide libraries with complex network models architectures already built. Also, transfer learning allows models to be pre-trained in huge datasets, accomplishing optimization of the network parameters, which later on can be fine-tuned on a new dataset. This results in a cut down of computational time. However, the application of this solution requires large datasets with ground truth labels, which represents a time consuming effort. Moreover, the network training is a process that takes huge amount of time and computational power. Nonetheless, this solution is less dependent on images variety.

1.7 Objectives and Contributions

The main goal of this work is to explore automated visual inspection tools to detect and classify rotor blades damages. Giving the dataset provided, it was required to apply data preprocessing for overall coherence. As already referred in section 1.5, the new dataset was obtained by coupling similar labels, removing data containing few occurrences and relabeling of images damages.

There are still no solutions for this particular problem case, so two approaches were considered, tested and compared. The first approach corresponds to the development of an algorithm using known computer visions techniques to extract individually features of each damage and classify those based on regions properties, such as shapes or density. Also, it is included a segmentation module, where the blade is separated from its background, in an attempt to ease the detection and classification process. The second approach uses an existing deep neural network, which has been previously trained in a large dataset. By means of transfer learning the parameters are fine tuned for the new dataset.

1.8 Outline

This thesis presents a total of seven chapters, including the current one, which already addressed the motivation for this work, along with the problem formulation and proposed solutions, as well as the objectives and contributions regarding the implementation of such solutions.

Chapter 2 contains a theoretical review of computer vision techniques required to develop the first solution proposed.

Chapter 3 focuses on the theoretical foundations of the second solution. It starts with a brief introduction to artificial intelligence and deep learning concepts, followed by a choice of model to be implemented, which is then fully described through the remaining of the chapter.

Chapter 4 describes and divides the dataset used in the implementations. Also, the performance metrics are established here.

Chapter 5 presents the implementation of the proposed solutions, as well as the results obtained with the training dataset.

In chapter 6, both solutions are tested and compared for the same dataset.

Finally, chapter 7 finishes this document with conclusions and achievements concerning the objectives defined in the present chapter. Additionally, some suggestions for future work are also included.

Chapter 2

Computer Vision Background

The literature review shows that it is possible to detect damages in images using only classic computer vision techniques. For instance, in [56] a simple algorithm for cracks detection in high-speed steel bar in coil was developed, by only applying laplacian filters, to enhance the cracks, followed by binarization and a set of morphological operations. Furthermore, [47] accomplished good enhancing and detection of cracks on roads surface. Similar to [56], the authors in [47] applied histogram equalization for crack enhancement, along with a binarization and morphological operations. More complex algorithms, as in [10] and [37], exploited the detection of different damages in tiles. While both implement image preprocessing techniques, such as image enhancement and noise reduction, followed by edge detection and morphological operations, in [10] each damage is detected by a different set of operations. On the other hand, [37] proposed a sequence of operations which extracts all damages features from tiles and then classifies those based on binary analysis of blobs. Moreover, this paper showed a comparison of results between both algorithms, from which was concluded that the latest overcomes the first one in accuracy and processing speed.

However none of the previous cases presents such noisy background as the turbine blades images, because those cases were tested in more controlled environments. To overcome this issue, a preprocessing algorithm based on edges detection and hough transform to remove the background is proposed. Having the blade foreground defined, a combination of morphological operations, edges detection and thresholds are applied to get the RoIs. Finally, the RoIs are classified based on blob analysis, such as size, shape, density and others.

The remaining of this chapter focuses on the theoretical definition of the operations used to develop the Computer Vision Algorithm (CVA). Firstly, primordial concepts regarding image processing such as spatial filtering, binarization and morpholgical operations are briefly described over sections 2.1, 2.2 and 2.3, respectively. Then, sections 2.4 and 2.5 cover more complex techniques for edges, lines and circles

detection. Additionally , three clustering algorithms are introduced in section 2.6.

2.1 Spatial Filtering

Spatial filtering is a technique to enhance or remove certain features of an image, by convolving a filter through it. For instance, filtering techniques are applied for noise reduction, morphological operations and feature extraction. A filter, also known as kernel, is a matrix element, F, with dimensions $w \times h \times d$, being w the matrix width, h the height and d the number of channels of the input. This element is convolved across an image I, resulting in a new image I'. The output image is computed by summing the point-wise multiplication of the each element in the filter with one element of the input image I. This formulation is expressed as:

$$I'(x,y) = \sum_{i=1+m}^{m} \sum_{j=1+n}^{n} \sum_{k=1}^{d} I(x+i, y+j, k) \cdot F(i,j,k) \tag{2.1}$$

where $m = \left\lfloor \frac{w}{2} \right\rfloor$ and $n = \left\lfloor \frac{h}{2} \right\rfloor$. Figure 2.1 illustrates an example of aplying convolution to an input image I. The colored parts show the calculations happening to a single pixel, with coordinates $(x, y) = (3, 3)$.

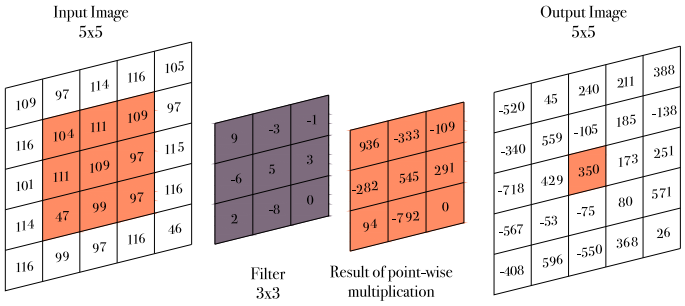


Figure 2.1: Linear image filter example: convolving a 3×3 filter across a 5×5 input image.

One observes that the input dimensions are preserved. This is possible due to a method called zero-padding, which consists of adding a margin of zero values around the image when convolving the filter. A visual demonstration of zero-padding is presented next, in figure 2.2. To the original image, represented in gray, is added a margin of zeros (white pixels), which allow the output image (orange pixels) to has the same dimensions as the original one.

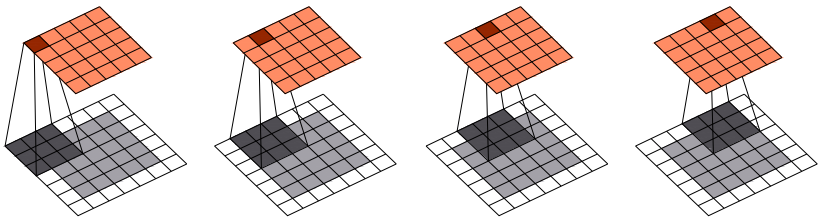


Figure 2.2: Zero-padding method

2.2 Binarization

Binarization of grayscale images is an important step towards segmentation and analysis of foreground objects. It segments grayscale pixels into black or white, considering a specific characteristic of the pixels. Many characteristics might be used depending on the purpose, however the most common is the pixels intensity. Applying a threshold to all pixels may be efficient, yet defining manually a threshold value that better splits foreground into background is not straightforward. Also when working with multiple images, the threshold value may differ from image to image. The best solution is to apply an automatic threshold criteria, such as Otsu or Adaptive threshold [35] [3].

2.2.1 Otsu Threshold

Otsu method finds a global threshold value that minimizes the intra-class variance of foreground and background. In an iteratively fashion, Otsu method computes different threshold values, followed by an analysis of the classes distribution. To visualize gray value image distributions, one may observe its histogram. An example of histogram is shown in figure 2.3. Each bin corresponds to a different gray value level and the histogram measures the occurrences of those levels in the image.

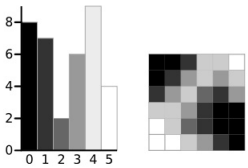


Figure 2.3: Histogram of a grayscale image (adapted from [19])

By increasing the threshold for each bin value, the image is divided into background for values bellow the threshold and into foreground for values above it. Having the classes established, the variance of each class is computed as:

$$\sigma^2 = \frac{\sum (v_i - \mu)^2 n_i}{\sum n_i} \tag{2.2}$$

where v_i and n_i are the value and number of pixels of level i, respectively. The μ represents the mean of pixels values inside that class. Following, the weight of each class, $w = \frac{\sum n_i}{N}$, is calculated, where N is the total number of pixels in the image. The intraclass variance is obtained from an weighted arithmetic mean of both classes variance, $\sigma^2_{global} = w_0 \cdot \sigma^2_0 + w_1 \cdot \sigma^2_1$.

Lastly, after iterating through each threshold value and obtaining the intraclass variances, the optimal threshold can be defined to be the minimum of intraclass variances.

2.2.2 Adaptive Threshold

Adaptive threshold technique computes a threshold for each pixel, based on the local mean intensity of the neighbourhood around it. For a given pixel, p, with a neighbourhood N, the local threshold is applied as:

$$p^* = \begin{cases} 0 & \text{if } p < \text{mean(N)} \cdot s \\ 1 & \text{otherwise} \end{cases} \tag{2.3}$$

where s is the sensitivity parameter in the range [0 1]. Higher values of sensitivity will threshold more pixels as foreground.

2.3 Morphological Operations

In computer vision morphological operations are mainly used to clean unwanted imperfections in binary images. Such operations depend only on a structuring element and on the shape of the foreground pixels. Structure elements are masks with various shapes and sizes built to convolve across all image pixels, performing a given morphological operation. Figure 2.4 shows an example of a Structure Element (SE) with the origin marked in orange. It is necessary to declare an origin of the mask, as it will dictate the position of the mask when passing through each pixel.

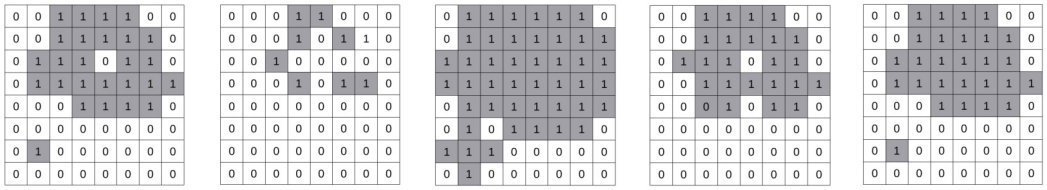
Figure 2.4: Structuring element with origin in central pixel

This work uses as baseline dilation and erosion morphological operations. As the names suggest, these operations dilate and erode the foreground pixels using a structuring element. This can be handy to connect or disconnect foreground objects, as well as removing small objects if used in the right sequence.

For this last case one requires an opening operation, where the image is eroded in such way that small objects disappear and then is dilated to resize all objects to their original shape. If the sequence order is inverted the operation is called closing.

When dealing with dilation, the structure element origin is passed through each target pixel location and the following condition yields: the target pixel is set to foreground if any of the structuring element coincides with a foreground pixel. Regarding the erosion, the conditions yields: the target pixel is set to background if any of the structuring element coincides with a foreground pixel, except in the case of all the structuring elements coinciding with foreground pixels, which the target pixel is left untouched.

To better understand these concepts let us consider the binary image represented in figure 2.5a. Using a structuring element as illustrated in figure 2.4, the output of each operation is the one showed in figure 2.5.



(a) Original binary image (b) Erosion (c) Dilation (d) Opening (e) Closing

Figure 2.5: Example of morphological operations

One observes how simple morphological operations can manipulate the foreground pixels for a desired purpose. For instance, using dilation to connect nearby objects, opening to remove small objects and closing to fill holes, as demonstrated in figure 2.5c, figure 2.5d and figure 2.5e, respectively.

2.4 Canny Edge Detection

Canny edge detection, introduced in [4], is a multi-step algorithm to detect edges in grayscale images. It starts by reducing the image noise with the application of a gaussian filter. Then, for each pixel the gradient magnitude is computed. Edges are most likely to contain strong gradient magnitudes. Following a Non-Maximum Suppression (NMS) is applied to thin out the edges. Finally, hysteresis is performed to threshold the gradients magnitudes.

2.4.1 Gaussian Filter

The gaussian filter, built from the bivariate gaussian distribution, is defined as:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}(\frac{x^2+y^2}{\sigma^2})} \tag{2.4}$$

where σ is the standard deviation.

An important characteristic of this distribution is the fact that 99 % of the distribution falls within 3σ, meaning that any point beyond that limit would contribute little for the convolution, while adding more operations when convolving. For this reason, the filter size is truncated to 3σ.

Since the gaussian distribution is continuous, the kernel values are approximations of it. For $\sigma = 1$, a typical 5×5 kernel has the values displayed in figure 2.6.

$$\frac{1}{273} \times \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 2.6: Approximation of gaussian kernel

2.4.2 Gradient Computation

In image processing it is good practice to use a central difference derivative, defined as:

$$\frac{df}{dx} = f(x+1) - f(x-1) \tag{2.5}$$

This equation is valid for discrete function values and can be translated to a linear filter, as [-1 0 1]. Instead of convolving a gaussian filter and then convolving derivatives filters for each direction, the gaussian gradient, $\nabla G = (\frac{\partial G}{\partial x}, \frac{\partial G}{\partial y})$, is computed and then convoluted through the image, which results in a decrease of processing time. Equations (2.6) and (2.7) show this associative principle:

$$g_x = \frac{\partial}{\partial x}(I \times G) = I \times \frac{\partial G}{\partial x} \tag{2.6}$$

$$g_y = \frac{\partial}{\partial y}(I \times G) = I \times \frac{\partial G}{\partial y} \tag{2.7}$$

The gaussian partial derivative is the output of convolving a derivative filter through a gaussian filter, such as the one in figure 2.6. After obtaining the partial derivatives, g_x and g_y, the gradient magnitude and direction are calculated for each pixel as:

$$|g| = \sqrt{(g_x)^2 + (g_y)^2} \tag{2.8}$$

$$\angle g = \arctan(\frac{g_y}{g_x}) \tag{2.9}$$

2.4.3 Non-maximum Suppression

For a given edge point, NMS investigates pixels in the gradient direction neighborhood and removes those who present smaller gradient magnitude than the current reference pixel. Figure 2.7 displays a typical method for thinning edges with NMS. Each black dot represents a pixel position, while gray ones are points between pixels.

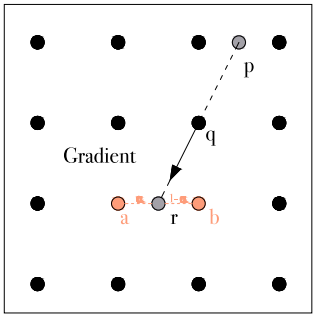


Figure 2.7: Non-maximum suppresion scheme (adapted from [12])

For a given pixel q, the gradient direction is propagated in both forward and backwards directions, obtaining the closest points in the gradient direction. As in the example, the resultant points, r and p, can happen to be intermediate point between pixels. In this situation the intermediate points gradient is extrapolated from a linear interpolation of the left and right pixels. Considering the point r, the interpolated gradient,g_r, is computed using the distance from point r to pixel a, α, as well as the gradients on the left and right pixels, g_a and g_b:

$$|g_r| = \alpha|g_b| + (1-\alpha)|g_a| \tag{2.10}$$

Having r and p defined, the pixel q is considered an edge if its magnitude is bigger than both r and p. In the case of being an edge, the next point to investigate will be obtained by the normal of the gradient direction. In this way it is ensured that the algorithm is iterating first through strong connected edges. For more details on this type of non-maximum suppression, recall to [12].

2.4.4 Hysteresis Threshold

In hysteresis, upper and lower thresholds are considered. From the thinned out gradient, any value below the lower threshold is set to 0, while values above the upper threshold are set to 1 and labeled as strong edges. Values that fall between both thresholds are weak edges and are set to 1 if they are

connected to a strong edge. Equation (2.11) summarizes this hysteresis threshold:

$$p(x,y) = \begin{cases} \textit{no edge} \implies p(x,y) = 0 & \text{if } \textit{lower threshold} < |g| \\ \textit{weak edge} & \text{if } \textit{lower threshold} < |g| < \textit{upper threshold} \\ \textit{strong edge} \implies p(x,y) = 1 & \text{if } |g| > \textit{upperthreshold} \end{cases} \tag{2.11}$$

$$\textit{weak edge} = \begin{cases} p(x,y) = 1 & \text{if } \textit{connected to strong edge} \\ p(x,y) = 0 & \text{otherwise} \end{cases}$$

The hysteresis threshold outputs the final edges from Canny algorithm in the shape of a binary image.

2.5 Hough Transform

Hough transform was firstly presented in [22], however the popular algorithm was only establish a few years later, in [9]. It is an algorithm designed to detect lines, circles and other curves, knowing their parametric equations. Detecting these shapes can be an important step when performing image segmentation, as in the case of this work, where lines and circles were detected using hough transform.

2.5.1 Lines in Hough Space

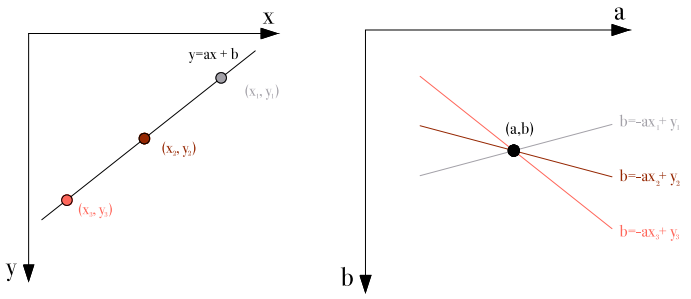
Before defining a line in parameter space, let us consider the following parametric equation for lines in image space:

$$y = a \cdot x + b \tag{2.12}$$

where (a, b) are parameters and (x, y) variables of the equation. To convert a line into parameter space, one should consider (x, y) as parameters and (a, b) as variables. Equation (2.12) can be rewritten to:

$$b = -x \cdot a + y \tag{2.13}$$

To acknowledge the advantages of parameter space let us consider the example provided in figure 2.8, where a line passing through 2 points is transformed to the parameter space. A noticeable property of hough transform is that lines become points in parameter space and vice versa. In fact a line in parameter space will represent all the possible lines passing through a point in image space, thus if two points belong to a line in image space, their representation in parameter space will intersect at a point (a, b), which corresponds to the parameters of the line. This last point is the foundation of hough transform. By observing the parameter space, points with high intersections density are more probable of representing true lines.



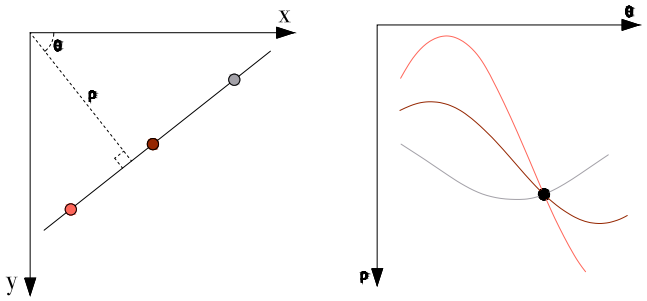
(a) Image space (b) Parameter space

Figure 2.8: Hough line transform using parametric equation

A problem from using (2.13) is vertical lines, which are not defined, since parameter a is infinite in those cases. A simple solution is to use instead the polar equation:

$$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta) \tag{2.14}$$

where (ρ, θ) are the variables, representing the distance to origin and the angle of the line, respectively. The principle applied will be the same, expect for the parameter space, where a point from the image space now represents a sinusoidal wave, instead of a line. This difference is displayed in figure 2.9.



(a) Image space (b) Parameter space

Figure 2.9: Hough line transform using polar equation

A simple way to find high intersection density points in parameter space is to use hough accumulator cells, as represented in figure 2.10. Cells are obtained by quantization of equally spaced increments, in both ρ and θ, meaning each cell represents a unique pair of (ρ, θ) values. The size of the increments is commonly referred as resolution of accumulator. Typically, for an image of size $M \times N$, resolution is

set to 1 in both ρ and θ, while constraining values to the domains: $\rho \in [-\sqrt{M^2+N^2}, \sqrt{M^2+N^2}]$ and $\theta \in [-90°, 90°]$.

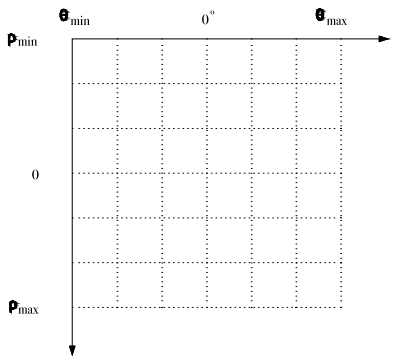


Figure 2.10: Hough accumulator cells

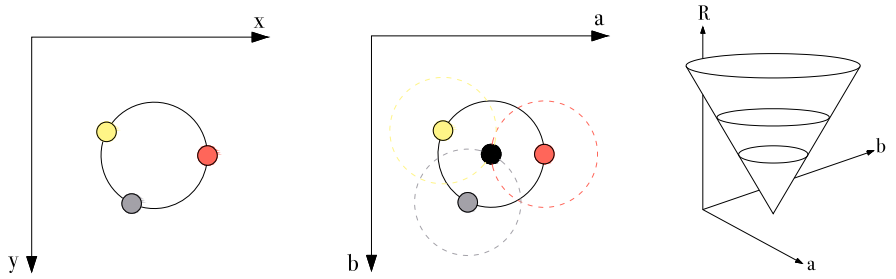
For a given point in image space, accumulator cells corresponding to all lines passing through that point are voted as line in a cumulative fashion, i.e., cells cumulate votes for all points. After performing this to every point in geometric space, one can evaluate the accumulator to find cells with higher values, which are more probable of representing lines.

2.5.2 Circles in Hough Space

For circles the Hough transform follows the same principle as lines, but with minor changes. A circle in image space can be described by the polar equations:

$$\begin{aligned} x &= a + R \cdot \cos(\theta) \\ y &= b + R \cdot \sin(\theta) \end{aligned} \tag{2.15}$$

where (a, b) are the circles center coordinates and R is the radius. Considering a constant radius, R, the transformation of points in image to parameter space creates circles now and points with higher accumulated votes are more probable of representing true circles center (a, b). This is demonstrated in figure 2.11, for a better perception.



(a) Image space

(b) Parameter space with constant radius

(c) Parameter space with varying radius

Figure 2.11: Hough circle transform using polar equation

Nevertheless, in real implementations the radius may be unknown, hence it is may not be a constant. To include a variant radius in the algorithm, it is necessary to extend the parameter space in the R direction. This means a point in image space will result in a conic surface, as illustrated in figure 2.11 c). In this manner, votes are accumulated for interceptions of conic surfaces.

2.6 Clustering

Clustering data algorithms are handy when no information about the data is known. With such algorithms, one can group data points into clusters based on a specific characteristics, such as similarity between points. For instance, a classic K-means clustering method [32] uses only the centroids distance as similarity, but other metrics may be used, as in the case of Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [11], which clusters points in high density neighborhoods. Additionally, more elaborated versions of K-means can segment images into similar regions based on distance and color of pixels, such as the Simple Linear Iterative Clustering (SLIC) algorithm [1]. Segmentation of images into regions plays an important role in finding and analyzing regions of interest.

2.6.1 SLIC

SLIC algorithm clusters images into approximately equally-sized regions based on the CIELAB color space and Euclidean distances similarity in (x, y) plane. Having this said, the cluster centers, $C_k = [l_k, a_k, b_k, x_k, y_k]$, where the first 3 variables are the values of CIELAB color space, are initialized by a grid of equally spaced intervals. The space between cluster centers, S, depends on the desired number of

clusters, K, and on the total number of pixels in the image, N:

$$S^2 = \frac{N}{K} \iff S = \sqrt{\frac{N}{K}} \tag{2.16}$$

where S^2 is the baseline area of clusters to initiate. Also on the initialization, the initial cluster centers are readjusted to the pixel with lowest gradient in a 3×3 window. This reduces the probability of the initial center landing on an edge, which can be a noisy pixel to start clustering.

As for the distance metric, D , SLIC uses a linear combination of CIELAB color space and euclidean distances, d_{lab} and d_{xy}, given as follow:

$$D = d_{lab} + \frac{m}{S} d_{xy} \tag{2.17}$$

where m is the compactness of clusters and regulates the emphasis that SLIC should give to the (x, y) plane distance when computing the distance metric. Both d_{lab} and d_{xy} are computed as euclidean distances:

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \tag{2.18}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{2.19}$$

Moreover, SLIC restricts distance computations for pixels inside a $2S \times 2S$ area around the cluster center. Comparing K-means with SLIC, both algorithms present an identical structure, yet SLIC shows a slightly higher complexity by the addition of color space distance. The SLIC algorithm is fully explainded in algorithm 1.

Algorithm 1 SLIC

```
Define number of clusters K
Initialize clusters centers C_k = [l_k, a_k, b_k, x_k, y_k] with k = 1, 2, ..., K
Adjust cluster centers to lowest gradient position in a 3 × 3 neighborhood
while Convergence not reached do
  for p in N-data points do
    Compute distance measure D (2.17)between p and all K cluster centers with d_xy ≤ 2S
    Assign p to nearest cluster center
  end for
  for k = 1 : K do
    Compute mean of points assigned to cluster k
    Update cluster C_k = mean(p_k)
  end for
end while
```

Chapter 3

Deep Learning Background

Artificial Intelligence was firstly introduced by John McCarthy in 1956 [33]. The idea of implementing intelligence into machines, making those capable to perform human tasks in environments with uncertain behavior, motivated many researchers to investigate deeper this field of research. Since then, many algorithms were developed to transform this idea into reality.

Nowadays, artificial intelligence has become a trend related with many areas of application. For instance, Figure 3.1 shows some of those areas. One should point out that these branches are not exclusive from each others. For example, in this solution computer vision is used, as well as deep and supervised learning.

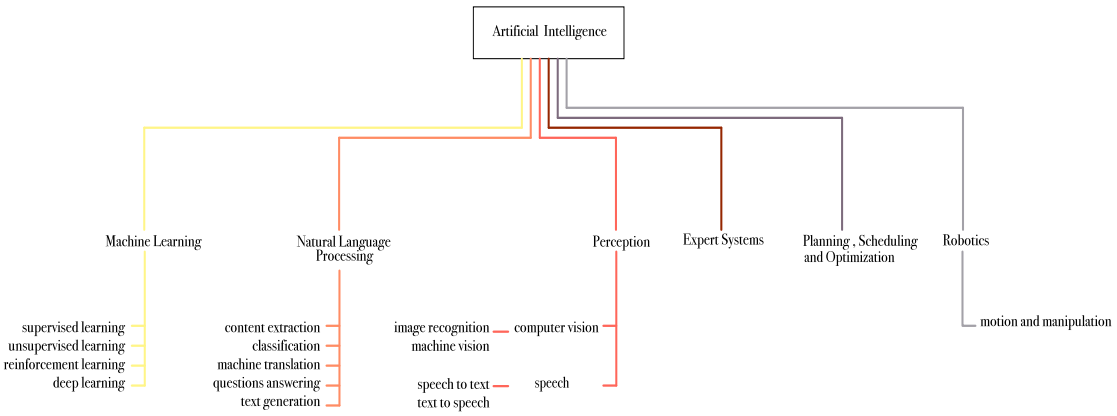


Figure 3.1: AI research areas

Machine learning is one of the master branches from artificial intelligence. It is based on the ability of human brain to learn from experience and retrieve acknowledge. The introduction of this concept gave birth to artificial neural networks, which enabled computers to complete complex tasks in unpredictable

environments.

Neural networks (NNs) use features, from the raw data, to obtain the mapping from a representation to the desired output. By means of representation learning, neural networks are capable of finding which features to be used during the learning process. Despite all the previous, simple neural networks have drawbacks, such as not taking into account the variations present in the data. As each feature may differ from case to case, this downside makes NNs impotent, when compared to more complex models, such as, Deep neural networks (DNNs). Within the Deep learning branch, DNNs use multilayers of simple linear models along with nonlinear activation functions to extract complex nonlinear levels of abstract representations [17]. This last topic is approached with deeper understanding in the remaining of this chapter. Firstly, in section 3.1, a brief history and concepts concerning DNNs is targeted, along with an overview of the current DNNs models and a choice of the model to use in this work. Secondly, section 3.2 covers Convolutional Neural Networks (CNNs) with a deeper look at the theoretical foundations. This section is relevant to understand the basics behind DNNs. Lastly, getting more related with this work objectives, section 3.3 approaches, with detail, the architecture of the network chosen to be applied in this work.

3.1 Deep Neural Networks

Based in human brain neurons interaction, NNs are composed of several nodes, called neurons, interconnected between each others to learn together the representation from an input to a desired output. This process can be accomplished by manners of supervised, unsupervised or reinforcement learning.

The first known implementation of such model was a single layer network with a threshold activation function, developed by McCulloch and Pitts, in 1943 [34]. Some years later, in 1957, Rosenblatt introduced his work on perceptrons [40], giving neurons the capacity to learn. Nevertheless, even with perceptrons, a single-layer network could only perform limited tasks. The usage of multilayer feed-forward networks only became feasible when the backpropagation algorithm was developed by Werbos in 1974 [53] and redeveloped by Rumelhart, Hinton and Williams in 1986 [41]. It provided a new learning rule for weights adjustment, which reflected on error minimization.

Relying on these concepts, various NN architectures were designed to accomplish different results. For instance, Fukushima was able to train a unsupervised multilayer network to recognize features in small images, invariant to position and small distortions in shape [13]. Carrying Fukushimas work, LeCun introduced the term CNN, by designing LeNet-5, composed of convolution, pooling[1], fully-connected and

[1] In the original paper this layer is referred as sub-sampling layer, however the name most commonly used is pooling layer

softmax layers [29]. Nowadays, DNNs are used worldwide along with computer vision and have achieved large depths of layers, resulting in state-of-art performances.

The rising of DNNS in computer vision field was more noticeable since the launch of Imagenet Large Scale Visual Recognition Challenge (ILSVRC), in 2010. Back then, the challenge focused on image classification, by putting together a competition between teams to obtain the best accuracy possible in a test dataset. Others variants are now present in these competitions, like object detection and segmentation.

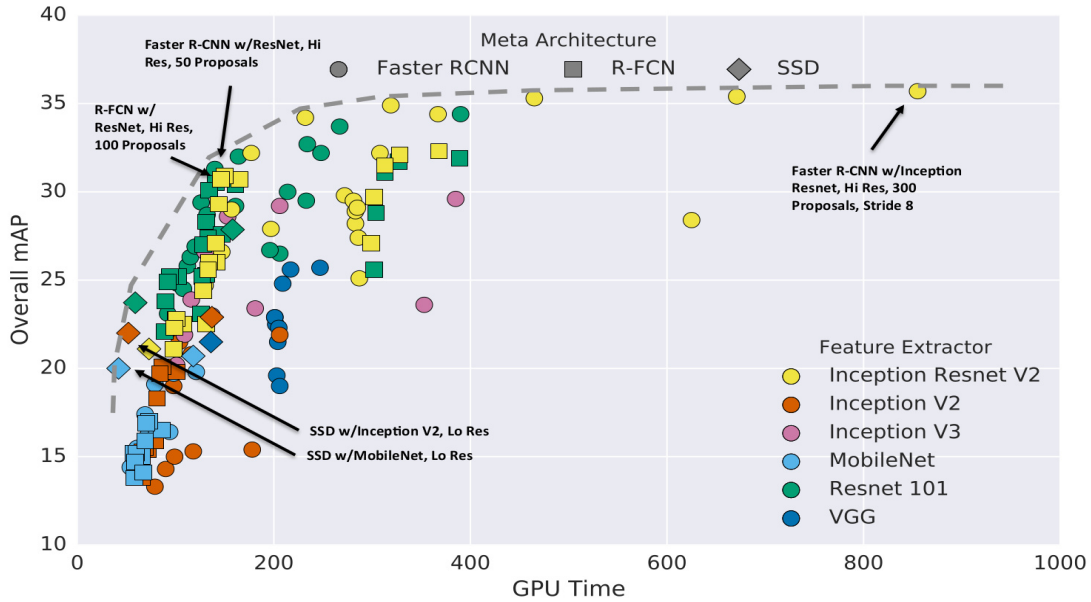
As this innovative competition got prestige, and other similar competitions appeared, an urge to accomplish DNNs with better performance motivated many machine learning researchers to design DNNs with peculiar architectures, such as Single Shot multibox Detector(SSD) [31], You Only Look Once (YOLO) [38], R-FCN [7], R-CNN [15], Fast R-CNN [14] and Faster R-CNN [39]. Having this diversity, choosing the model for a certain goal becomes less straightfoward, as each advantage and disadvantage should be taken into account. To evaluate each of the presented networks and choose a proper one, figure 3.2 illustrates a comparison between these networks regarding different parameters. The evaluation metric mAP, is the most used in object detectors and expresses within both precision and recall of the model. This metric is fully explained in section 4.2.

When fast processing time is required, for instance, in real time detections, networks with an architecture like SSD and YOLO show good performances. However, the accuracy of these two is overcome by slower models, like Faster R-CNN (figure 3.2a). Moreover, this Faster R-CNN has better performance when small objects must be detected (figure 3.2b) and can receive input images of any size. As the main goal of this work is accuracy, either in big, medium and small objects, while processing time has less relevance, the adopted model was a Faster R-CNN.

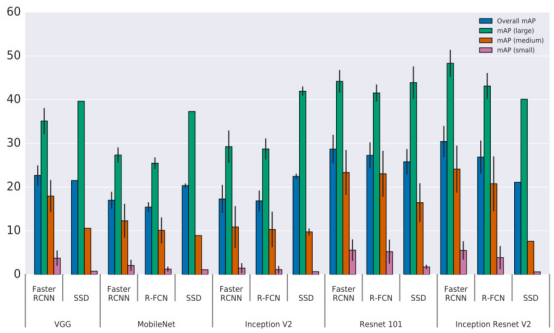
Concerning the feature extractor implemented in Faster R-CNN, ResNet-101 and Inception ResNet v2 show close results in overall mAP and accuracy (figure 3.2c), but the second one is much more computational demanding than the first (figure 3.2a). This last point was the main factor to choose ResNet-101 feature extractor over the Inception ResNet v2.

Finally, regarding DNNs learning process, the lack of training data can jeopardize the final results, as the sub-optimal solution is not reached. In those cases, transfer learning is a simple solution to be considered. Because this work dataset is not big enough[2], Faster R-CNN is trained using transfer learning, a method further described in section 3.2.4.

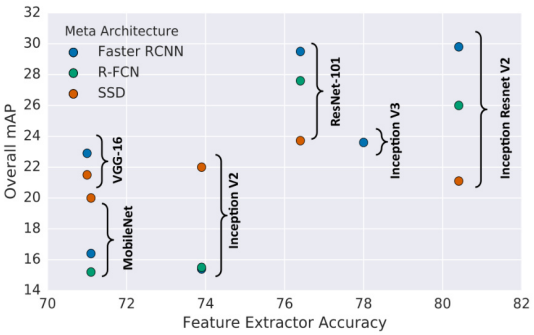
[2]For reference, the networks in figure 3.2 were trained with a dataset 35× and 180× larger, regarding the number of images and of object instances, respectively.



(a)



(b)



(c)

Figure 3.2: Three comparisons between networks with different architectures (source [23]): (a) - Comparing the trade-off between GPU processing speed and mAP, (b) - Comparing the mAP across three scales of objects: small, medium and large, (c) - Comparing the feature extractors accuracy, when implemented in different networks. Also the overall network mAP is shown for comparison

3.2 Convolutional Neural Networks Architecture

Convolutional neural networks are the foundations of DNNs, such as the one used in this work, Faster R-CNN. Thus, it becomes necessary to better explore the mathematics and algorithms behind those.

A typical CNN is composed of three layers: convolutional, pooling and fully-connected layer. Considering each of the previous layers has a different method and purpose, this section was divided into three subsections: one per layer.

3.2.1 Convolution Layer

Convolutional layers are responsible for generating abstract representations of an input, to be fed into the fully-connected layer, latter on. These representations, usually referred as feature/activation maps, are obtained by convolving a stack of linear filters on the input image. This process was already explained in section 2.1.

While training a CNN, weights will be adjusted to minimize the output error. In regard to the convolutional layer, these weights are present in the kernel. Once the training starts, each parameter will suffer modifications to obtain the best representation of the training data. Furthermore, a kernel is also defined by fixed parameters, called hyperparameters. The choice of these plays an important role in the results obtained. These include the number of filters n, the filter size w and h, the activation function of the layer and the stride s. While the previous parameters can assume many values and algorithms, the most commons are, respectively, $n \in \{32, 64, 128\}$, $w = h \in \{1, 3, 5, 11\}$, rectified linear unit (ReLU) activation and s = 1 [50].

The depth of a convolutional layer is dictated by the number of filters. If one increases the depth, the model becomes more complex, as the amount of weights and feature maps rises. To better observe the depth, figure 3.3 is represented next.

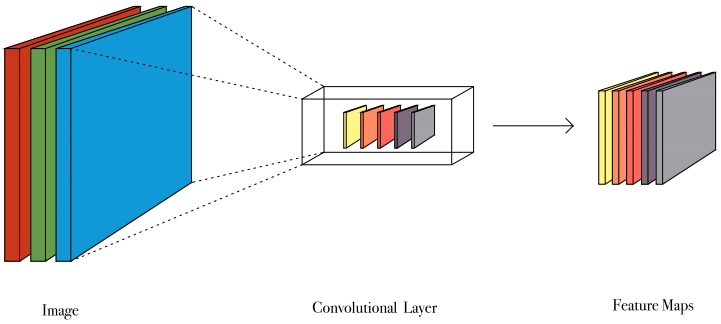


Figure 3.3: Convolutional layer with depth 5

An input image with depth of 3 is convoluted by a stack of 5 filters, resulting in 5 distinctive feature maps. The data depth may change, and will in most cases. As referred earlier, the most common number of filters used are $n \in \{32, 64, 128\}$, which implies an increase in depth when applied to RGB images, with depth 3.

It is common practice to share weights across the spatial regions of a depth. Because the goal is for each kernel to extract a different feature, this sharing is convenient. Additionally, it results in a massive reduction in the parameters produced by the convolutional layer.

3.2.2 Pooling Layer

Pooling layers are used to extract the most relevant features in the activation map. This application reduces the dimensionality of the representation and, consequently, the number of parameters needed. Furthermore, its application results in a local translation and minor changes invariance. Pooling can be implemented with different methods, but typically max pooling is chosen in many networks architectures.

Just like convolutional layers, pooling requires a kernel, with size and stride of choice. When the kernel is slid across the activation maps, max pooling will extract the features with highest score. Figure 3.4 shows an example of a 2×2 kernel applied to an activation map of size 6×4, with stride $s = 2$. The result is a 3×2 feature map, thus data were reduced to $\frac{1}{s^2}$ of the original activation map.

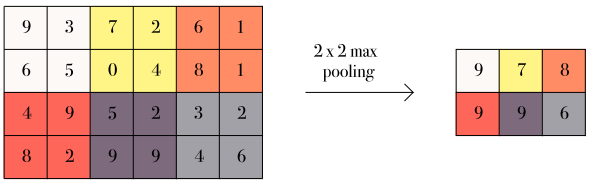


Figure 3.4: Max pooling: example adapted from [50]

3.2.3 Fully-Connected Layer

Fully-connected layers are based in multi-layer perceptron models. Each layer is constituted by independent neurons, i.e, there is no connections within a layer. In a given layer, each neuron receives input from all neurons of the previous layer and its output is feed to each neuron of the next layer. Figure 3.5 illustrates a generic architecture of the fully-connected layers, within a CNN. Three types of layers are represented: input, hidden and output layers. Connections between the first two layers are associated with a weight w_{ij} and from the last two with a weight W_{ij}. Also a bias is added to hidden and output layers.

A neurons input, z_j, is called logit. It results from the dot product between the weight of the connection, w_{ij} and the output of the previous layer x, plus a bias value, b:

$$z_j(x) = \sum_i w_{ij} x_i + b_j = w^T \cdot x + b \tag{3.1}$$

A neurons output, known as activation, is computed using the logit, z_j, and an activation function, f:

$$g(x) = f(z(x)) \tag{3.2}$$

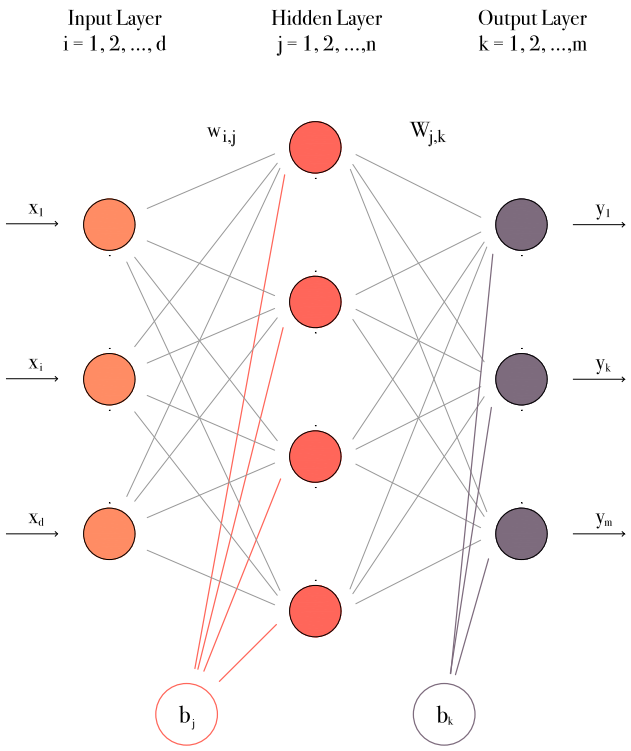


Figure 3.5: Generic representation of fully-connected layers

Merging (3.1) and (3.2), the input of the output layer is obtained:

$$z(x) = W^T \cdot g(x) + b \tag{3.3}$$

Last, analogously to (3.2), the output of the output layer is related to the input, z_k, and an activation function, σ:

$$y(x) = \sigma(z(x)) = \sigma(W^T \cdot g(x) + b) \tag{3.4}$$

The overall purpose of the activation functions in (3.2) and (3.4) is to introduce nonlinearities into the network. By converting the input of a neuron into its output, this function dictates how much neurons are fired when receiving a given input. A more detailed explanation regarding activation functions is presented in section 3.3.3.

3.2.4 Transfer Learning

Transfer learning is a method that uses the knowledge acquired from a certain task to train the network in a new similar task. Transposing to CNNs, the transfer learning is accomplished as follows: firstly, a

network is trained from scratch in a large source dataset, obtaining a robust model. This dataset must be large and diverse to guarantee a model with good generalization. Secondly, having the pre-trained model, the parameters are reused to initialize the new network. It is common practice to retrain all the layers of the model, but one can also freeze some layers. Figure 3.6 shows the transfer learning process with freezing of the DNN layers. In this scheme, only the output layers are retrained.

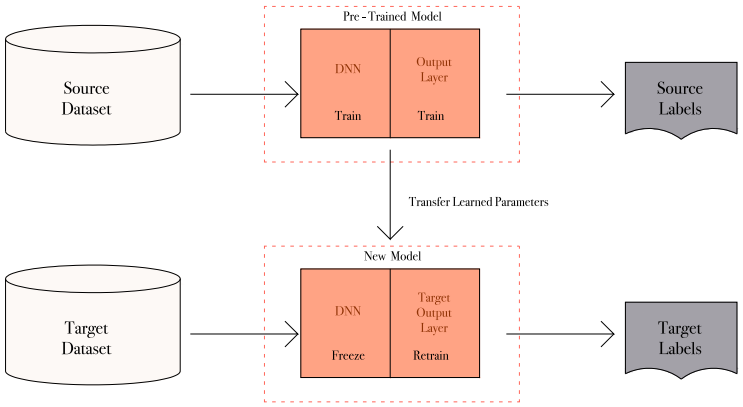


Figure 3.6: Transfer learning scheme

This approach is widely used on CNNs, as it proved to improve the network accuracy and accelerate the training time, when compared to the same network trained from scratch. Such improvement is reported in papers like [42], where the authors used pre-trained models, which had obtained state-of-art results on the ILSVRC, to address a classification problem. The pre-trained model was trained on the ImageNet dataset, while the new network was trained on a different dataset, composed of art images. ImageNet dataset is a large dataset made available for deep learning research. Considering that the ImageNet dataset differs considerably from the art dataset, one can conclude that pre-trained models which achieved good generalization on large datasets can be retrained in new different datasets, achieving better results that networks trained from scratch on those same new datasets.

3.3 Faster R-CNN

Faster R-CNN architecture brings together two main modules. The first one, noted as Regional Proposal Network (RPN), consists of a deep fully-connected network that proposes RoIs with various scales and aspect ratios. The second one, known as Fast R-CNN, with a DNN architecture, used as classifier and box regressor for the regional proposals. Both were developed by the authors of Faster R-CNN. To merge these two modules, a sharing of convolutional feature maps is implemented. In this

way, Faster R-CNN learns to generate regional proposals and to classify those, using the same feature maps. Such methodology makes possible to train jointly both networks and, consequently, reduce the training time, as well as the numbers of parameters used. Faster R-CNN architecture is displayed in figure 3.7.

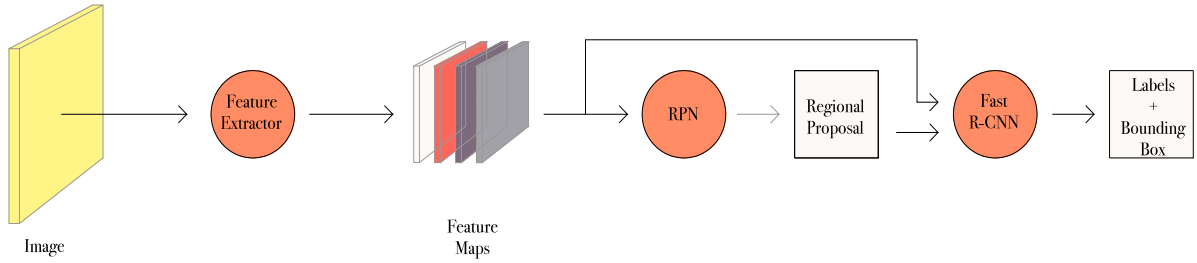


Figure 3.7: Faster R-CNN, as a unified network

The network receives as input an image, which is passed through a feature extractor[3], resulting in convolutional feature maps. These maps are then feed into the RPN and to Fast R-CNN modules. The first one uses the feature map to generate the regional proposals and the second to classify objects in each region proposed by the output of the RPN module.

3.3.1 Regional Proposal Network

As explained earlier, RPN receives, as input, a set of convolutional feature maps, generated by the last shared convolutional layer of the feature extractor, and outputs a set of boundary box proposals, with the respective scores. Figure 3.8 represents a detailed architecture of RPN, as it was firstly implemented in the original paper [39].

[3] As mention in section 3.1, in this work is implemented a Resnet-101.

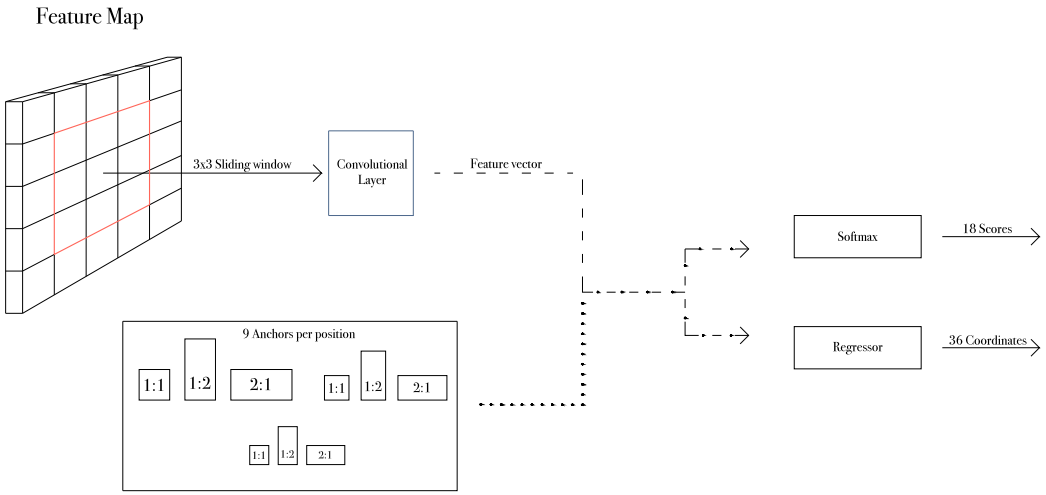


Figure 3.8: RPN architecture

This process starts by sliding a 3×3 spatial window through the feature map. At each window center location, 9 anchors with different aspect ratios and scales are generated. There are 3 scales with areas of 128^2, 256^2 and 512^2, and 3 aspect ratios of 1:1, 1:2 and 2:1. This approach was used to address multi-scale and translation-invariant regional proposals.

Then, for each one of the 9 anchors, a label, p^*, is computed as follows:

$$p^* = \begin{cases} 1 & \text{if } IoU > 0.7 \\ -1 & \text{if } IoU < 0.3 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where IoU is the highest value of intersection over union of the anchor with all the ground truth (GT) boxes, present in the same image, and it is computed as:

$$IoU = \frac{Anchor \cap GTBox}{Anchor \cup GTBox} \tag{3.6}$$

Equation (3.5) labels each anchor according to its IoU with the ground truth. Whenever this value is higher than 0.7, the label is positive, meaning there is an object inside the anchor. In some rare cases the previous condition may not happen for any anchor, so an extra one was created: if there are no anchors with IoU higher than 0.7, the positive label is given to the anchor with higher IoU. Furthermore, if the value is lower than 0.3, the label is negative, meaning there is no object inside the anchor. Lastly, anchors with values between 0.3 and 0.7 are excluded.

Afterwards, the 3×3 spatial feature window is mapped to a lower dimension feature[4] and is fed, along

[4]In this work is mapped to a 2048-d vector by mean of an intermediate convolutional layer. This dimension is coherent with the depth of the last convolution layer of the feature extractor.

with the anchors and their labels, p^*, to a fully-connected layer, which has two sibling output layers: a bounding box regression and a classification layer[5]. The first is responsible for predicting the coordinates of the bounding-box and the second for outputting the probability of the predicted box containing an object and of being background.

Furthermore, before feeding this anchor proposals to the Fast R-CNN module, some filtering is applied. A NMS is performed to reduce redundancy. Whenever two or more anchors have an IoU equal or higher than 0.7, the NMS excludes the anchors with lower classification score. This method reduces the amount of proposals generated by RPN, while not affecting the final detection accuracy. Lastly, during the training phase the cross-boundary anchors are ignored. The authors claimed those would jeopardize the training convergence. On the other hand, during testing, cross-boundary anchors are included, yet clipped to the image dimensions.

3.3.2 Fast R-CNN

Fast R-CNN was created as an improvement of R-CNN architecture [15]. When comparing to its baseline, Fast R-CNN not only showed gains in training and testing speed, but also in detection accuracy. Moreover, it introduced multi-task loss, making possible to train the network in a single stage. Figure 3.9 illustrates the Fast R-CNN architecture.

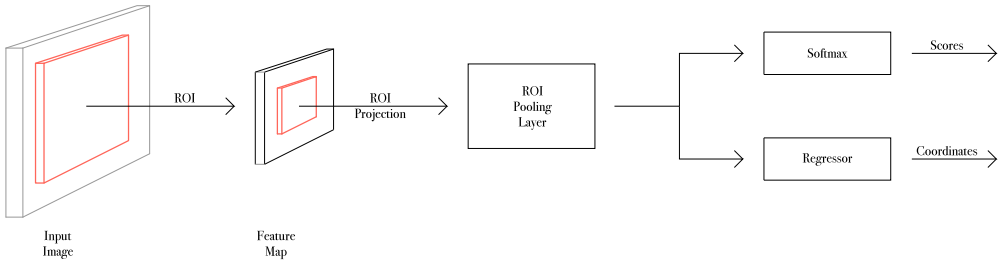


Figure 3.9: Fast R-CNN architecture

The network receives as input feature maps, resulted from the feature extractor, and a set of RoIs, proposed by an external method, and outputs a bounding box coordinates prediction and object classification.

Fast R-CNN starts by mapping each RoI to the feature map dimensions, reducing its size to a $w \times h$ window. Then, this window is used by a RoI pooling layer to extract a fixed-length feature vector. To do so, the RoI pooling layer starts by dividing each $w \times h$ window into a 7×7 cell grid. Afterwards,

[5]Regression and classification layers were not introduced in section 3.2 because they are not common to every CNN.

max pooling is applied to obtain the highest value inside each of the cells, which are then mapped to a feature vector. Lastly, the output of RoI pooling layer is fed to a fully-connected layer, identical to the one implemented in RPN module. As explained in section 3.3.1, two sibling output layers are responsible for predicting the bounding box coordinates and to classify each one of those.

3.3.3 Activation Functions

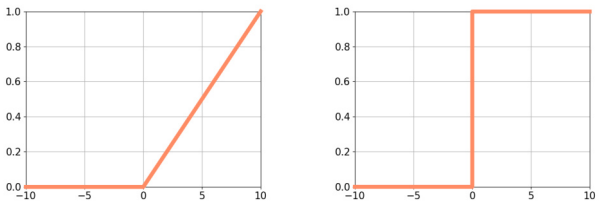
As already referred in section 3.2.3, neurons output depends on an activation function. This function has the purpose of giving the model non-linearity, which is a requirement when working with complex tasks. Despite the diversity of activation functions, all of them are differentiable. This happens because many optimization algorithms use a gradient descent with backpropagation, meaning the error will be propagated by mapping the gradients of each neuron, hence its activation function must be differentiable.

In section 3.2.3, different notations are used to address the activation functions of the hidden and output units, which in practical cases will differ. In fact, there will be 3 varieties of activation function: one for the hidden units, another for the classification units, and yet another one for regression units.

Hidden Layer

The most common function used in units from hidden layer is the ReLU, which is expressed as (see fig. 3.10a):

$$f(x) = max(0, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$



(a) (b)

Figure 3.10: Hidden units activation function: (a) - ReLU function, (b) - Derivative of ReLU function

One may note that the function is nonlinear due to the second condition in (3.7). However this condition brings a downside, as it can bring the activation of a neuron to zero and, therefore, stagnating in that point. This means the neuron will not be adjusted anymore and so it dies.

Nevertheless, this function is commonly used, as it is less computationally heavy when comparing to others, while also showing good approximations of the desired model.

Classification Layer

When working with classification problems, classification layers have an important role, since they are responsible for generating the predictions of the model. It can have many score function formulations, depending on the classification task. In this work a softmax formulation was used.

Softmax is used as activation function to obtain a normalized probability of being each class. It is formulated by the following equation:

$$p_k = \sigma(y)_k = \frac{e^{y_k}}{\sum_i^n e^{y_i}}, \qquad k = 1, 2, ..., n \tag{3.8}$$

where k is the index of each class, n the total number of classes, y the logits vector and p the vector with the probability of being each class, p_k, also known as confidence score. Having this last vector, the desired output of the classifier, $\widehat{y}$, is the class with higher probability:

$$\widehat{y} = \arg\max(p) \tag{3.9}$$

Box Regression Layer

Box regression layer is one of many regressions approaches to use on bounding box optimization. This algorithm receives three input vectors for each anchor: A, G and d. The first vector is composed of the center coordinates, the width and height of the proposed anchors, $A = (A_x, A_y, A_w, A_h)$. The second one, similar to the previous one, but regarding the ground truth box, $G = (G_x, G_y, G_w, G_h)$, and the third one is a vector, $d = (d_x, d_y, d_w, d_h)$, composed of logits, meaning each element is a linear function of the output from the last hidden layer, $g(x)$. Analogously to (3.3), this function is given as:

$$d(x) = W^T \cdot g(x) + b \tag{3.10}$$

To obtain the predicted ground truth box, P, a regression function formulation was used, as described in the following transformations:

$$\begin{aligned} P_x &= A_w \ d_x + A_x \\ P_y &= A_h \ d_y + A_y \\ P_w &= A_w \ e^{d_w} \\ P_h &= A_h \ e^{d_h} \end{aligned} \tag{3.11}$$

The first two transformations address scale-invariant translations, while the remaining two tackle log-space size changes. The linear function, d, is the regression target to learn. This is possible due to its vector of learnable parameters, W. It is important to note that because there are nine different ratio and scale anchors, there will be nine regressors. In this way, regressors do not share weights and each one is associated with a different aspect ratio and scale.

3.3.4 Feature extractor

When Faster R-CNN was first introduced, the feature extractor had either a Zeiler and Fergus (ZF) [57] or Simonyan and Zisserman (VGG-16) [44] network architecture. ZF and VGG-16 were picked considering they have 5 and 13 shareable convolutional layers, respectively. Those shareable layers are handy to address the goal of sharing convolutional feature maps across RPN and Fast R-CNN.

The previous networks have been proving over the years that depth is a crucial factor of the performance and more layers could mean an accuracy improvement. However, some problems, such as, vanishing/exploding gradients could jeopardize convergence from the beginning, as reported in [2] and [16]. After solving the previous with normalized initialization and intermediate normalization layers, another problem raised. With the depth increasing, accuracy would saturate and degrade quickly, according to [20] and [45]. A solution to overcome degradation was presented in [21], by introducing residual neural networks. Results showed that deeper networks could be trained, using residual mapping, and would achieve better performances, without any of the previous problems. The concept was based on the hypothesis that by optimizing a residual mapping, instead of an unreferenced mapping, the performance could have some improvements.

Being x the input of a set of layers and denoting the desired underlying mapping as $H(x)$, the residual mapping to fit, $F(x)$, is given by:

$$F(x) = H(x) - x \tag{3.12}$$

Arranging the previous equation, the desired underlying mapping becomes:

$$H(x) = F(x) + x \tag{3.13}$$

To better understand the concept, figure 3.11 shows a residual learning scheme. The implementation of (3.13) is accomplished with shortcut connections, responsible to output identity mapping, which is added to the output of stacked layers. Applying these connections does not increase the amount of parameters or computational complexity. Also, the network can be trained end-to-end, without further modifications, by SGD with backpropagation, as explained in sections 3.3.5.

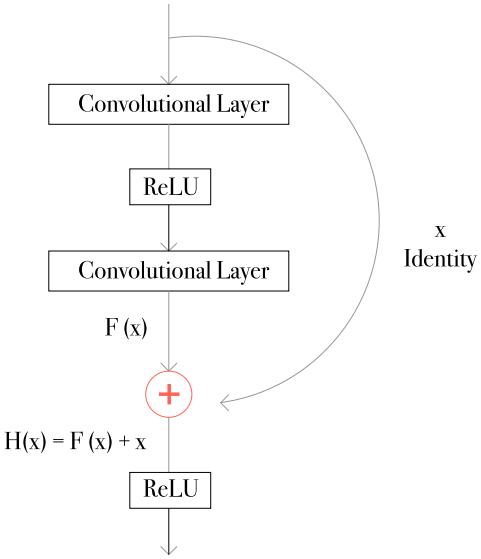


Figure 3.11: Residual learning scheme

Another important feature is the introduction of ReLU activation functions to produce a nonlinear network, as explained in section 3.3.3.

As mentioned in section 3.1, the chosen feature extractor was ResNet-101, which refers to a residual neural network with 101 layers. Despite its depth, this network has only 7.6 giga FLOPS [6], while a plain 16 layers network, such as, VGG-16 has 15.3 giga FLOPS.

Figure 3.12 illustrates the architecture of ResNet-101, as developed in the original paper [21]. It is important to note that when implemented in Faster R-CNN, the network does not has the last three layers. Because it is a feature extractor, the desired output corresponds to the feature maps that result from the last convolutional layer, as discussed in section 3.3.

Identity shortcut connections where placed in every block of 3 stacked layers. Each of these blocks has a bottleneck design. The first and third layers are 1×1 convolutions, responsible, respectively, for reducing and restoring dimensions. The 3×3 convolutional layer works as bottleneck with smaller input/output dimensions. This approach was firstly introduced in inception modules of GoogleNet [48] and its main advantage is the decrease of computational complexity, when compared with a non-bottleneck block.

[6] Floating-point operations per second (FLOPS) is a processor performance unit. In neural networks, FLOPS are used to indicate the amount of floating-point operations needed to be handle per second, i.e., a network with higher number of FLOPS will need a processor with, at least, the same amount of FLOPS.

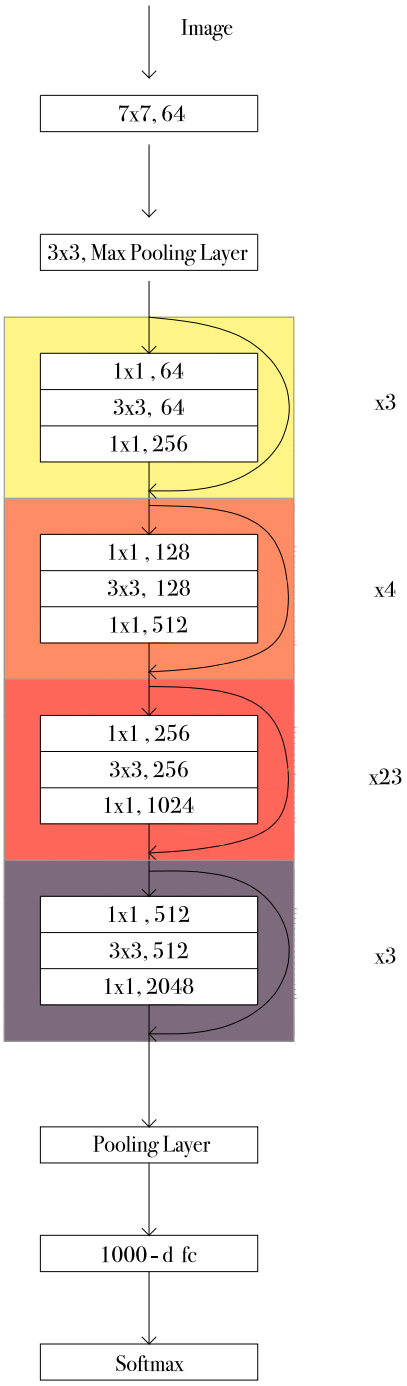


Figure 3.12: Resnet-101 architecture

3.3.5 Optimization

Optimization is a key process when training neural networks. Such algorithms are responsible for minimizing or maximizing an objective function, which depends on the learnable parameters, and translates the models error.

In neural networks, the previous function is referred to as loss function and the learnable parameters are the weights and bias described in section 3.2.3, which are updated at each iteration using gradient-based algorithms. As these parameters are connected in a cause-effect fashion, the algorithm is conjugated with backpropagation.

As networks got deeper and the datasets larger, the computational demand increased. The creation of mini-batches is a solution to decrease the computational demand. Instead of feeding the network with the full dataset, small batches of those are fed, i.e, at each iteration, a mini-batch is fed to the network and used to update the weights and bias, rather than only updating after passing the full dataset. This strategy achieves a decrease in memory required and a boost in processing time.

Gradient Descent

Gradient descent is a learning rule used to minimize the loss function by updating the models parameters in the opposite direction of the loss gradient. The goal is to take steps towards the direction of the downhill surface generated by the loss function, until a local minimum is reached. As neural networks are nonlinear models, the optimization becomes a non-convex problem, meaning there is no guarantee that the solution converges to a global minima.

Equation (3.14) represents this learning rule.

$$\theta_{t+1} = \theta_t - \eta \frac{\partial L}{\partial \theta_t} \tag{3.14}$$

The updated parameter, θ_{t+1}, will depend on the current parameter value, θ_t, as well as on the gradient of the loss function with respect to that parameter, $\frac{\partial L}{\partial \theta_t}$, and on the learning rate, η, responsible for controlling the size of the step taken between iterations.

The previous formulation is known either as vanilla or as batch gradient descent and follows the assumption that the gradient is computed using the entire training dataset. However, as already referred in this section, the usage of mini-batches is worth it, and was implemented in Faster R-CNN. In this work each mini-batch is constructed from a single image and contains RoIs within. In practical sense, the whole image is fed into Faster R-CNN, but only the mini-batch of RoIs is used to compute the loss function and apply gradient descent with backpropagation.

Another good practice in gradient descent is the usage of momentum. This method overcomes the problem present in [46], regarding navigation through surface curves with one dimension deepest than the others. Applying momentum boosts the convergence speed, by reducing the amount of iterations needed to reach the local minima.

Analogously to a ball accumulating momentum when rolling down a hill, the parameters update depends on a momentum accumulator, responsible for storing information from the previous gradients directions. In practice this means that if the gradient has the same direction as the accumulator, the update dimension of the parameters increases, otherwise it decreases. Adding the momentum accumulator, v, changes (3.14) into the following two:

$$v_{t+1} = v_t\alpha - \eta\frac{\partial L}{\partial \theta_t} \tag{3.15}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{3.16}$$

where α is the momentum term and determines how much of the previous momentum, v_t will accumulate in the new one, v_{t+1}.

Furthermore, a final modification is made in the update rule by adding regularization. This method avoids overfitting of the model, by penalizing the update made in parameters with high values. To compute this penalization value, a weight decay, λ, is multiplied by the current parameter, resulting in the amount to be decreased in the update rule. Bringing together this concept with (3.15), one can write the final learning rule:

$$v_{t+1} = v_t\alpha - \eta\frac{\partial L}{\partial \theta_t} - \lambda\eta\theta_t \tag{3.17}$$

Having establish the learning rule, it is important to notice the introduction of new hyperparameters, such as, the learning rate, batch size and momentum.

Loss Function

The loss function represents the error present in the output predictions, when compared to the ground truth, and because the output depends on the learnable parameters, the loss function does so, indirectly. Considering the diversity of existing loss functions formulations, choosing one to fit a model is an important step. Usually, loss functions fall into two main categories: classification and regression loss. However, Faster R-CNN uses a multi-task loss[7], where the previous two are coupled to result in a unified formulation. Before getting to this formulation it is important to establish the inputs of it.

As explained in sections 3.3.1 and 3.3.2, both RPN and Fast R-CNN modules have two sibling output layers, which outputs predictions to be computed in the loss function. The classification output

[7] This loss was firstly introduced along with Fast R-CNN, in [14]

predictions, p_k^P, are obtained by means of (3.8), as:

$$p_k^P = \frac{e^{y_k}}{\sum_i^n e^{y_i}}, \qquad k = 1, 2, ..., n \tag{3.18}$$

The box regression optimization is accomplished by means of normalized coordinates, benefiting from scale-invariant translations and log-space size changes, as explained in section 3.3.3. Arranging the transformations present in (3.11) to isolate the linear functions, which are the regression targets, one obtaines the normalization of the four coordinates. Denoting t^P and t^G as the normalized coordinates of the ground truth and predicted box, respectively, the following is obtained:

$$\begin{array}{ll} t_x^P = \frac{P_x - A_x}{A_w} & t_y^P = \frac{P_y - A_y}{A_h} \\ t_w^P = \log\left(\frac{P_w}{A_w}\right) & t_h^P = \log\left(\frac{P_h}{A_h}\right) \\ t_x^G = \frac{G_x - A_x}{A_w} & t_y^G = \frac{G_y - A_y}{A_h} \\ t_w^G = \log\left(\frac{G_w}{A_w}\right) & t_h^G = \log\left(\frac{G_h}{A_h}\right) \end{array} \tag{3.19}$$

Having the inputs, p and t, established, the multi-task loss, $L(p,t)$, is formulated as follows:

$$L(p,t) = \frac{1}{M} \underbrace{\sum_m^M L_{cls}(p_m^P, p_m^G)}_{(1)} + \lambda \frac{1}{R} \underbrace{\sum_m^M p_m^G L_{reg}(t_m^P, t_m^G)}_{(2)} \tag{3.20}$$

Multi-task loss is accomplished by summing the classification loss (3.20-1), with the box regression loss (3.20-2). The summation, $\sum_m^M$, in each term, tackles the usage of the RoIs mini-batches. The variable m is the index of the anchor inside a mini-batch of size M. Also, in the second term, the binary ground truth label of the contained object, p^G, is added to exclude this term in cases where the anchor does not contain the object. Furthermore, these terms are normalized by means of M and R, being R the total number of every possible anchor location and aspect-ratio combination. At last, a balancing parameter, λ, is introduced to control the weight of term in (3.20-2).

Regarding the classification loss formulation, L_{cls}, a log loss is accomplished by multiplying the ground truth label, p^G, with the negative common logarithm of the prediction made, p^P. Equation (3.21) puts together the previous formulation.

$$L_{cls}(p_m^P, p_m^G) = -\sum_n^N p_{m,n}^G \; log(p_{m,n}^P) \tag{3.21}$$

The summation is used to address multi-class problems, such as the case, by covering all the index, n, inside the total number of classes, N. It is important to notice that the ground truth label now refers to a certain class, i.e, it gives the binary ground truth label of an anchor containing a specific class.

Concerning the box regression loss, L_{reg}, the following formulation yields:

$$L_{reg}(t_m^P, t_m^G) = \sum_{j \in \{x,y,w,h\}} smooth_{L_1}(t_{m,j}^P - t_{m,j}^G) \tag{3.22}$$

where $smooth_{L_1}$ is a robust loss function given as:

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{3.23}$$

In an utopian case the loss would converge quickly and without oscillations to a global minima, but this is unlikely to happen, since the loss depends on many factors, such as, hyperparameters selection, weight initialization and regularization. Nevertheless, the loss function is a good evaluator of the models training progression.

Backpropagation

As explained earlier in this section, gradient descent updates the models learnable parameters in the opposite direction of the loss gradient, to minimize it. That is the purpose of the term $\frac{\partial L}{\partial \theta_t}$ in the learning rule established in (3.17). In simple words, it represents the sensitivity of the loss function, L, to small changes to parameter, θ. Yet, knowing how much each weight and bias will affect the loss function is not straightforward, as the network is connected in a cause-effect fashion. Backpropagation algorithm enables the network to compute the gradient term by means of chain rule.

The backpropagation algorithm is accomplished in two steps: a forward and a backwards iteration. The first step consists of running an input through the network to obtain predictions. This process was already described with mathematical formulations in earlier sections. Having the predictions and ground truth, the loss function is computed at the output and then its gradient is propagated backwards through the network. To better understand this concept, a mathematical formulation is explained next.

Let us consider the diagram shown in figure 3.13, as it represents a simple case of the last two layers of a network. One observes that the loss function, L, depends on many parameters connected in a cause-effect fashion.

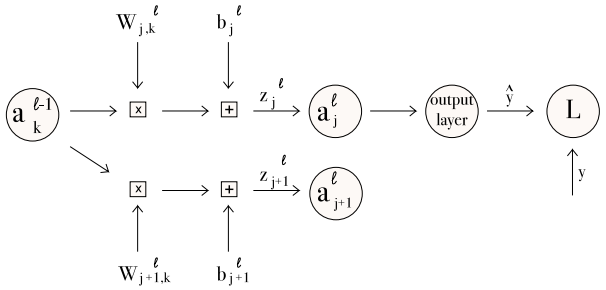


Figure 3.13: Block diagram of a neural network last layers

The index l refers to the last layer of the network and $l-1$ to the precedent layer. Also, neurons

lower index, $\{k, j, j+1\}$, represents the position of a neuron inside a layer. For now, let us consider only the upper branch, with neurons a_k^{l-1} and a_j^l, and neglect the previous low index.

As previously mentioned, the loss function depends on the ground truth, y, and on the activation from the neuron of the last layer, a^l. Furthermore, this activation depends on the weight, w^l, on the bias, b^l, and on the activation of the neuron from the previous layer, a^{l-1}. Recalling (3.4), the following is known:

$$z^l = w^l a^{l-1} + b^l \tag{3.24}$$

$$a^l = \sigma(z^l) \tag{3.25}$$

To understand how the weight affects the loss function, the term $\frac{\partial L}{\partial w^L}$ must be computed. This is accomplished by applying the chain rule as:

$$\frac{\partial L}{\partial w^l} = \frac{\partial L}{\partial a^l}\frac{\partial a^l}{\partial z^l}\frac{\partial z^l}{\partial w^l} = \frac{\partial L}{\partial a^l}\ \sigma'(z^l)\ a^{l-1} \tag{3.26}$$

The term $\frac{\partial L}{\partial a^l}$ represents the sensitivity of the neuron a^L to the loss function. Computing this therm is straightforward when applied to the output layer neurons. However in our case, the loss function is composed of two loss functions, meaning the total loss derivative will be the sum of two therms, $\frac{\partial L_{cls}}{\partial w^l}$ and $\frac{\partial L_{reg}}{\partial w^l}$. For simplicity reasons, these derivatives are provided in appendix B, as they are irrelevant to understand the backpropagation concept. To compute the derivative of the neuron, a^{l-1}, a similar approach of equation 3.26 is performed, using chain rule:

$$\frac{\partial L}{\partial a^{l-1}} = \sum_{j=0}^{n_l - 1}\frac{\partial L}{\partial a^l}\frac{\partial a^l}{\partial z^l}\frac{\partial z^l}{\partial a^{l-1}} = \frac{\partial L}{\partial a^l}\ \sigma'(z^l)\ w^l \tag{3.27}$$

The sum is added to the equation to cover the fact that neuron a^{l-1} will affect the loss function both through the neuron a_j^l as well as through the neuron a_{j+1}^l, i.e., all the neurons from layer l will be affected by neurons from previous layer, $l-1$.

Equations (3.26) and (3.27) are enough to propagate the gradient through the network, except for the RoI pooling layer. In that layer, the derivative of an input value, x_i, is given as follows:

$$\frac{\partial L}{\partial x_i}[i = i^*(r,j)] = \sum_r \sum_j \frac{\partial L}{\partial y_{r,j}} \tag{3.28}$$

being $y_{r,j}$, the j^{th} output from the RoI pooling layer that refers to the r^{th} RoI from a mini-batch. The term $i = i^*(r,j)$ reduces to zero the derivative of inputs that were not activated in the max pooling layer, meaning the gradient will only be propagated through maximum input values, x_{i^*}.

This finishes the backpropagation concept, as (3.26), (3.27) and (3.28) cover the needed tools to implement such algorithm.

3.3.6 Normalization

While normalization of neural networks components has many purposes, the main reason is to achieve a faster and more stable convergence. During the last chapter, some normalizations were already exposed. For instance, the classification layer is normalized by the softmax formulation, while the regression is normalized by the bounding box size, as in (3.19). Also, the classification and regression losses are normalized by the mini-batch size and total number of possible anchors positions, respectively. In addition, the ResNet-101 uses batch normalization, as in [24]. Moreover, all gradients are normalized by a global gradient norm, which gathers information from all gradients. In addition, this global gradient is clipped by a maximum threshold value, i.e., if the global gradient exceeds the threshold, its value is decreased to the threshold.

Chapter 4

Dataset and Performance Metrics

4.1 Dataset

The dataset was already establish and described in section 1.5, however for implementations purposes some preparation should be made first, as well as announcement of existing constraints. In this manner, this chapter covers the preparation of the dataset for implementation and establishes the performance metric used later in the analysis of results. The dataset is composed of images with a size of 5120×3415 pixels. Firstly, the dataset was split into three subsets: train(65%), validation(15%) and test(20%). Training and validation datasets will be used to develop the proposed solutions, while the test dataset, as the name suggest, will be left for evaluation and comparison of both solutions.

Regarding the CVA development, a set of 10 images was selected per damage from the training dataset, where erosion was not included, since it cannot be distinguished by strong features, rather, only by its spacial location in the blade. For instance, erosion was labeled in cases where peeling occurred in the LE, meaning the spatial context of the damage will strongly affect the label. Having this said and considering the limitations of the CVA algorithm, erosion was not considered for development and testing of such solution. Despite the previous, erosion will be used for the DLA implementation, as an attempt to verify the capability of neural networks to learn spatial context where objects are present.

Concerning the DLA development, the full train and validation datasets will be used for the learning process of Faster R-CNN. On one hand, the train dataset participates in the optimization of the network, meaning it will affect the learning process. On other hand, the validation dataset evaluates the network during the training, without interfering in the learning process.

Table 4.1 displays the division of damages into three subsets, train, validation and test. Notice there are two test sets, whereas the difference between both is the inclusion or not of erosion.

		Damages Occurrences						
	Images	Erosion	Peeling	Crack	Fungi	LS	LRD	Total
Training (65%)	3724	2526	875	570	607	296	477	5351
Validation (15%)	940	633	220	142	152	75	121	1343
Test CVA (20%)	633	-	271	178	190	89	146	874
Test DLA (20%)	1086	789	271	178	190	89	146	1663

Table 4.1: Dataset division for implementations

4.2 Object detector evaluation metrics

Before implementation, is crucial to establish the evaluation metrics. Many object detection challenges, such as, Common Objects in Context (COCO), PASCAL Visual Object Classes (VOC) and Imagenet challenge opted to use mean Average Precision (mAP) as evaluation metric. To compute it, one calculates the average precision for each of the classes and then averages those across all classes.

This metric differs from the classic accuracy, as it has the particularity of unifying both classification and regression tasks in a single accuracy metric. In order to understand mAP, the concepts of confusion matrix, recall, precision and average precision must be de defined a priori. One should emphasize that these concepts are computed for each one of the classes.

4.2.1 Confusion Matrix

This matrix is a performance measurement often found in classification problems. It is computed with the predictions and the ground truth boxes classifications, as well as the IoU of those. Lets consider the confusion matrix present in figure 4.1. When concerning object detectors, the true positive (TP), false positive (TP) and true negative (TN) are computed using the IoU, concept already explained in section 3.3.1. For instance, a predicted bounding box is considered true positive whenever its IoU with the ground truth box is bigger than a certain threshold and, simultaneously, its classification label is correct. In some particular cases, two or more predictions may have the conditions to be considered true positive, but because they overlap the same ground truth, only the prediction with the highest IoU is label as TP, while the remaining predictions are considered FP.

The choice of an IoU threshold takes an important role in the results analysis, yet there is no correct value to use. For instance, in PASCAL VOC challenge it is used a 0.5 IoU, while COCO challenge averages the results from the range [0.5:0.95] with a 0.05 step. However these values are strict, considering the networks should learn the best of bounding box adjustment. In this work it is considered a lower value of $IoU = \frac{1}{3}$ since the bounding box accuracy is not as relevant as detecting and classifying correctly the damage. Figure 4.1 shows also two examples of bounding boxes with the IoU value chosen.

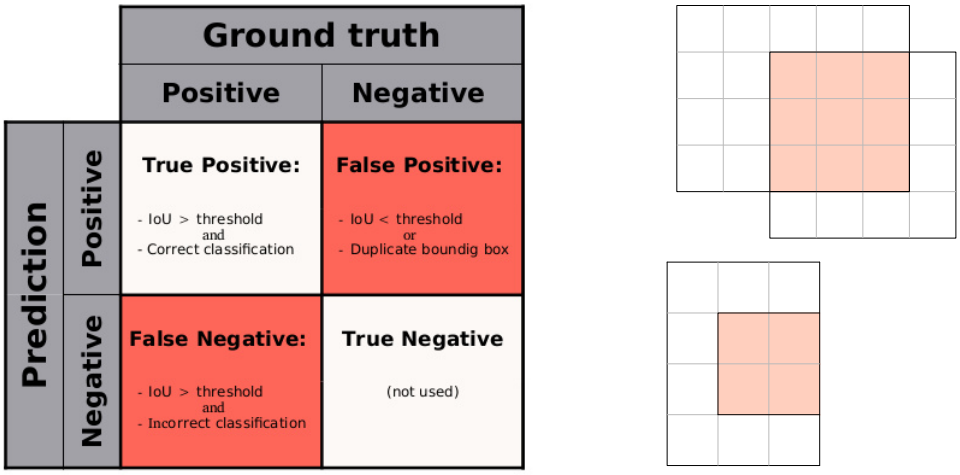


Figure 4.1: Confusion matrix and two examples of bounding boxes with $IoU = \frac{1}{3}$. The colored area represents the intersection of both areas.

4.2.2 Recall and Precision

Recall measures the ratio of true positives, which are the correct predictions, found in the total amount of the ground truth boxes, i.e, it shows the model capacity to find all the relevant objects. To compute it, the following formulation yields:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{Total\ Ground\ Truth} \tag{4.1}$$

Regarding the precision, it measures the ratio of true positives, found in the total amount of predicted boxes, i.e, it represents the model accuracy to classify the predicted boxes. Mathematically, precision is given as:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{Total\ Predicted} \tag{4.2}$$

A model is trained successfully whenever a good recall and precision is accomplished, yet, because theres a trade-off between both, the evaluation metric needs a precision-recall curve. An example of this curve is shown in figure 4.2. Each point is computed by varying the predictions confidence score threshold and computing the recall and precision for those threshold values. As referred before, there is a trade-off between recall and precision. For instance, if one desires to increase the recall, the confidence score threshold must decrease, which consequently results in a precision decrease. The same logic can be applied vive-versa. Having this in mind, figure 4.2 shows outliers in some intervals of the curve, where the recall level increases among with the precision. Because the score threshold suffers only small variations, these outliers can be interpolated to result in a smoother monotonically decreasing curve.

At each level of recall, r, the interpolated precision, p_{interp}, corresponds to the maximum precision value at a recall value higher than the current one, $\hat{r} > r$. This interpolation is expressed as:

$$p_{interp}(r) = max(p(\hat{r})) \tag{4.3}$$

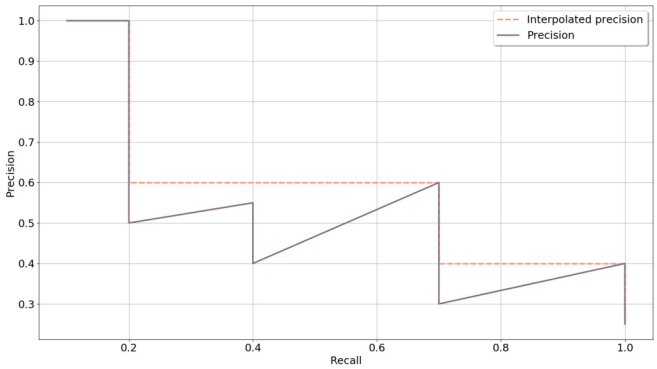


Figure 4.2: Interpolated precision-recall curve

4.2.3 Mean Average Precision

The recall-precision curve can be used to evaluate a network performance, however if one desires to compare between two curves, it may be not straightforward in cases where the curves intercept. For this reason, the average precision is calculated, obtaining a numerical comparison between curves.

The average precision is defined as the area bellow the interpolated recall-precision curve. In a ideal scenario, both recall and precision stay constant at the value 1, regardless the confidence score threshold. In this case the area bellow the curve is $1 \times 1 = 1$, meaning the average precision is 100 %. Yet, because in real cases the curve is not a perfect square, a formulation of the area must take place.

Observing figure 4.2, one concludes that the area can be calculated as the sum of a set of rectangular areas, which means the average precision, AP, is generalized to:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1}), \quad r = r_1, r_2, ..., r_n \tag{4.4}$$

where n is the total number of recall levels.

Finally, having the corresponding AP of each class, the mAP can be calculated as the average of AP across all K classes:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{4.5}$$

Overall, mAP is a good performance evaluator metric. Not only because it includes both precision and recall in itself, but also covers different levels of confidence score threshold, meaning it gives a great representation of the model robustness.

Chapter 5

Wind Turbine's Blades Damage Detection and Classification Algorithms

This chapter describes the implementation of the two solutions proposed. First, in section 5.1, the CVA solution is establish and analysed for a small dataset. Second, section 5.2 focus on experimental details concerning the deep learning approach, including the training process.

5.1 Computer Vision Algorithm

This section presents an algorithm for automatic damage detection, concerning the purposes of this work, using classic computer vision and clustering methods, already described in chapter 2, along with blob analysis and processing. This solution was implemented in *MATLAB* 2017a and contains the modules presented next:

- Main: being the head module, it manages all the detection processes . First, it performs image acquisition and then it handles the calling of the remaining modules. Figure 5.1 shows a scheme of this module.

- Segmentation: in this module, the blade is segmented from the background. It receives a RGB image and outputs the segmented grayscale image of the blade.

- Feature Extraction: in this module, relevant features of the blade are extracted from the segmented grayscale image. It outputs a set of binary images, containing features.

- Feature Classification: this module works the binary images for enhancement of RoIs. Such RoIs are then evaluated with blob analysis, in order to detect and classify the different damages.

- Non-Maximum Suppression: this module performs a NMS of bounding boxes, to remove redundant detections.

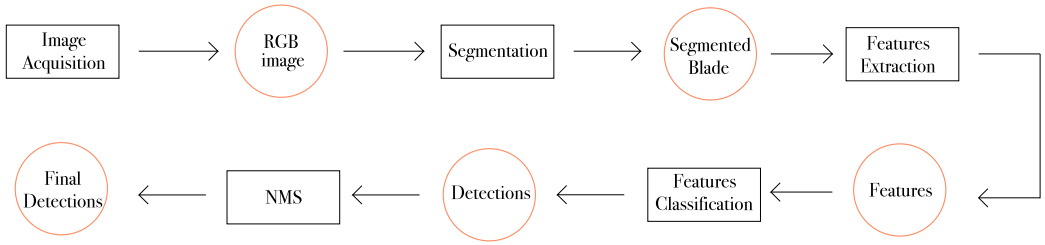


Figure 5.1: CVA: Main module

5.1.1 Segmentation

Segmentation of the blade is a crucial step to make sure that features from the background do not introduce error in the feature extraction module. For instance, the background may contain houses or even other turbines, which are rich in features and would jeopardize the algorithm performance. Moreover, without segmentation it would be much harder to establish a reliable method for each detection. Figure 5.2 shows a flowchart of the segmentation module, which is explained next.

This module starts by converting the RGB image into HSV, then by isolating the value component of pixels, a grayscale image is obtained. Working with grayscale is much faster, while also easier to extract features of interest. Moreover, most computer vision techniques are based in grayscale images, such as binarization or edge detection. An example of this conversion is presented in figure 5.3.

Following, a canny edge detection is performed to extract possible boundaries of the blade, which contains strong edges. Regarding the hysteresis threshold, it was found that values 0.2 and 0.5 for lower and upper thresholds, respectively, would achieve reasonable results. Additionally, the gaussian filter was constructed with a deviation of 3. Figure 5.4 a) shows the result of canny edge detection, when applied to figure 5.3 b).

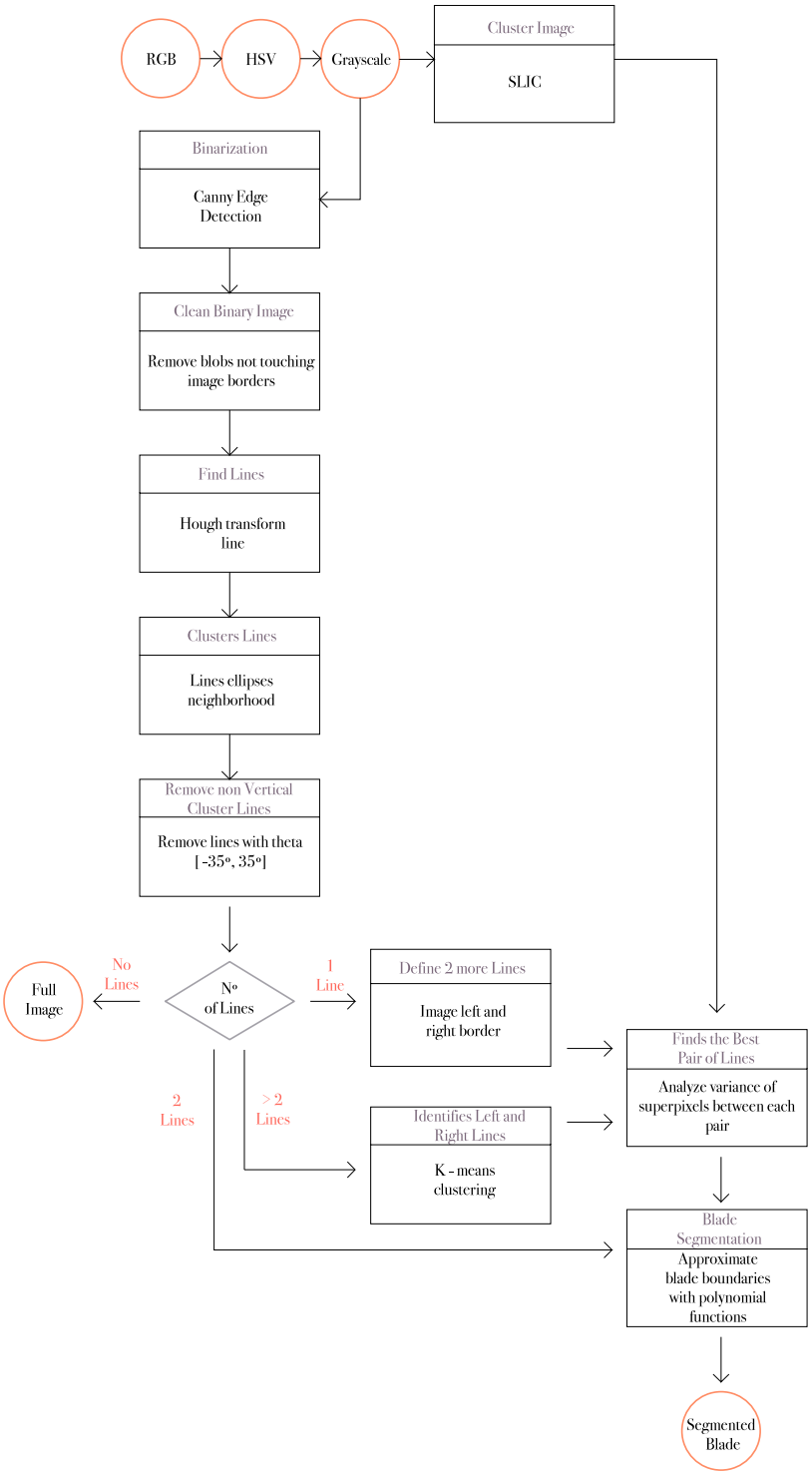
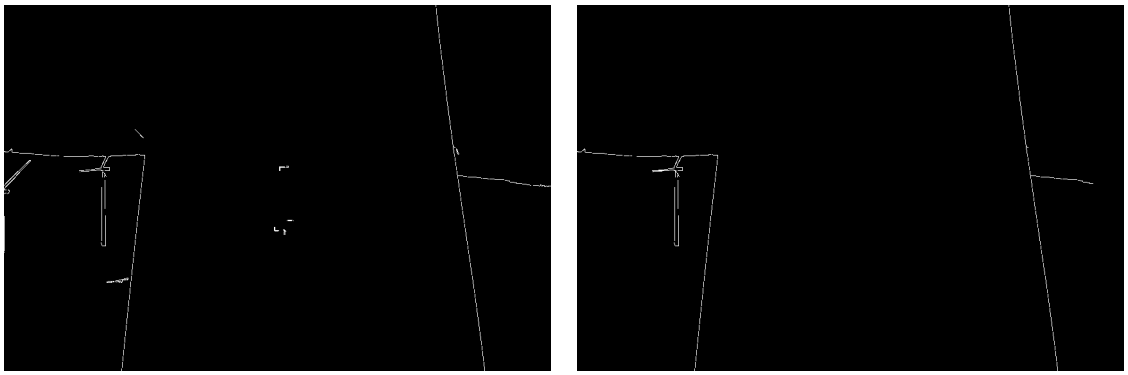


Figure 5.2: Segmentation module algorithm

(a) RGB image

(b) Resulting grayscale image

Figure 5.3: Convertion from RGB to grayscale image



(a) Edges

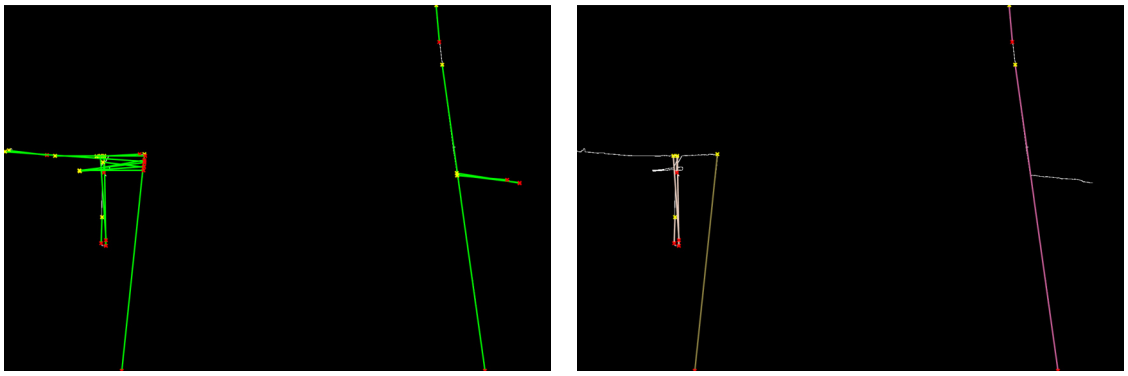
(b) Edges after removing blobs not connected to image border

Figure 5.4: Canny edge detection

With the edges detected (see fig. 5.4 a)), it is conclusive that the blade boundaries are most likely to represent blobs connected to the top or bottom borders of the image. To make sure this happens, a morphological closing was applied using a vertical line structuring element with length of 10. In this way, it will dilate and erode blobs in a vertical direction only, connecting edges close enough to the top or bottom of the image borders. All this done, all edges not connected to any border are excluded (see fig. 5.4 b))

The next step is finding lines among the edges detected. Blades boundaries are usually straight lines with vertical orientation, yet some variations may arise, depending on which section of the blade is shown in the image. For example, the blade tip may contain not so much vertical straight lines or even lines with a slight curvature. Nevertheless, Hough transform is a reliable technique to find lines defining the blade boundaries. Figure 5.5 a) shows how the Hough transform detects lines (in green) among the

edges. Yellow and red points represent the beginning and ending of the lines, respectively. Even though Hough lines achieves good results, it cannot cluster lines which are most probable to represent the same boundary. For this reason, an algorithm was developed to cluster lines based on ellipses. In a similar approach to points neighborhoods of DBSCAN, at each line a neighborhood with an elliptical shape was built. The major axis was set to be collinear and centered with the line in question, and the sizes of each axis are linearly dependent on the line length. In this work, the major and minor axis were set to $3.5\times$ and to $0.01\times$ of the length, respectively. For each line, if a start or ending point of another line falls within its ellipse, the lines angles are then compared. The angle difference cannot be larger than 10 degrees, otherwise lines are not considered to belong to a common cluster. After the clustering process, lines with angle ranging $[-35°, 35°]$ are considered not vertical and hence can be removed. An important property of clusters is that angles are transmissive between lines, i.e., all lines in a cluster are assumed to have the angle of the most vertical line. This is helpful in images containing the blade tip, since in this way, one will not remove relevant lines defining the boundary. Figure 5.5 b) is the result of clustering and removing non vertical lines. For instance, the two lines in pink color, on the right side, were considered to belong to the same cluster. One should note that white lines are the edges detected by the Canny edge detector and not lines from Hough transform.



(a) Lines

(b) Lines after clustering and exclusion of not vertical clusters.

Figure 5.5: Lines from Hough transform

At this point, where lines were already clustered and filtered out, there are four possible scenarios:

1. No cluster found: the blade is not segmented.
2. One cluster found: the algorithm considers the left and right borders of the image as two other clusters and will check which cluster-border pair best represents the blade.

3. Two clusters found: proceed to segmentation of blade.

4. More than two clusters found: use K-means clustering to split the starting and ending points of the Hough lines in left and right sides and find the pair of Hough lines clusters that better represent the blade. The possible pairs are always composed of a left and a right cluster. An example of the K-means clustering is presented in figure 5.6, where blue points were considered left side and green points right side. A cluster of Hough lines is labeled left or right side according to the majority of the starting and ending points within.

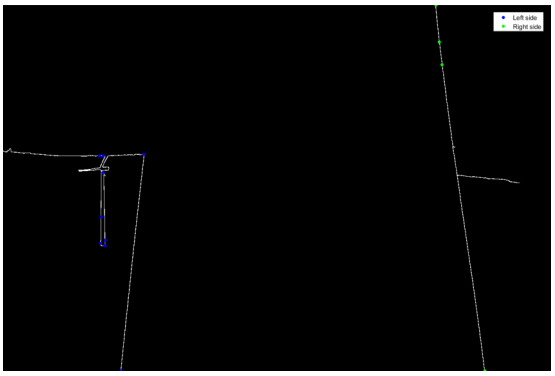
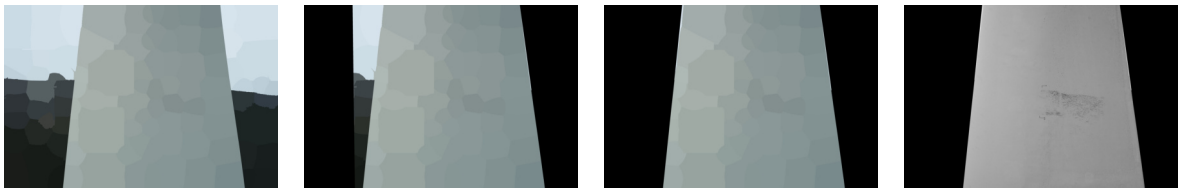


Figure 5.6: K-means clustering of starting and ending points of the lines

Finding which pairs better represent the blade is not so trivial as the dataset carries a large variety of images. A good solution was the application of the SLIC algorithm to cluster the image into similar color and distance regions, called superpixels, and then analyze the variance of RGB within each pair of clusters.

When performing the SLIC clustering, each superpixel color is obtained with a mean average of all pixels within, which means small objects are merged into the background, i.e., noise will be removed and a less complex representation of the image is obtained. In this way, analyzing the area within each pair of lines cluster will be less computational demanding, while also more accurate. To obtain the color variance, first, for each of the RGB channels, the variance, σ^2, was computed using the principle of intra-class variance of Otsu threshold, as described by (2.2). Then, the overall variance of colors is a simple summation of each color variance: $\sigma^2_{RGB} = \sigma^2_R + \sigma^2_G + \sigma^2_B$. The pair of cluster lines with lowest overall variance is most probable to contain a unique large object within. This was the criterion to choose the pair that best fits the blade boundaries. Having the left and right side cluster lines chosen, the algorithm performs an approximation of those with a one degree polynomial, except for clusters containing lines with high angle difference, where the approximation is made with a third degree polynomial. High angle

difference may imply the image refers to a blade tip and so a one degree polynomial is not good enough for approximation. Finally, the blade is segmented from the background using the computed approximations. Figure 5.7 shows the SLIC algorithm and color variance of the two possible pairs of clusters, previously decided with the k-means clustering. One observes that the segmentation presents less color variance when only the blade is considered foreground.



(a) Superpixels obtained from SLIC

(b) Superpixels for first pair (color variance = 2046)

(c) Superpixels for second pair (color variance = 420)

(d) Segmentation of blade using pair with lowest color variance

Figure 5.7: Variance of cluster pairs using SLIC

5.1.2 Feature Extraction

Having the blade segmented from the background, the next step is extracting strong features. The feature extraction module is displayed in figure 5.8 and will be explained with detail next. Before this module, the blade damages should be enhanced by increasing image contrast. The function *imadjust* applies saturation to pixels above the top 1% and to pixels bellow the bottom 1%, meaning brighter pixels will saturate to 1, while darker ones will saturate to 0.

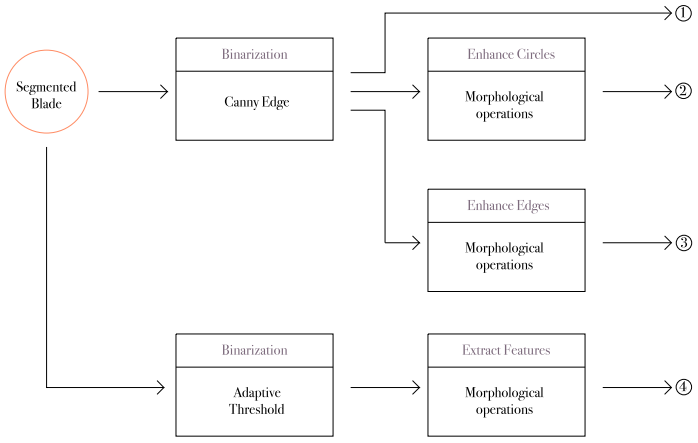
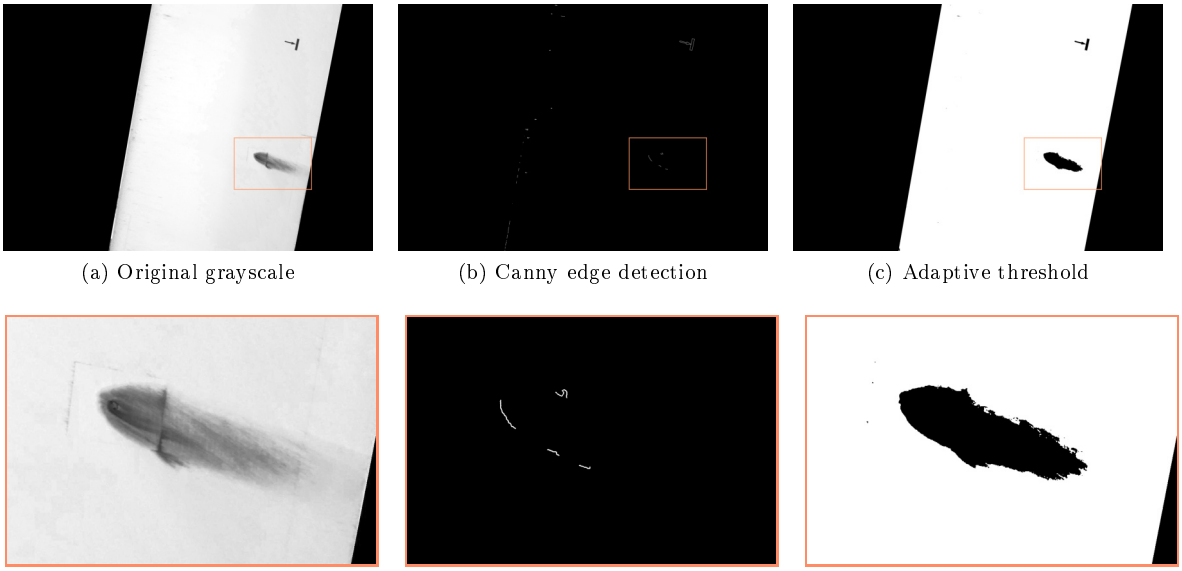


Figure 5.8: Feature Extraction module algorithm. The numbers in circles represent diferent outputs of the module, which are inputs of the classification modules

Binarization

As previously referred in section 2.2, binarization is a methodology of splitting foreground and background. Because damages can be described by strong edges or by local dark areas, two methods were applied for this purpose: Canny edge detection and adaptive threshold. On one hand, Canny edge will extract strong edges, which are present in most damages, such as peeling or crack. On the other hand, the adaptive threshold extracts locally darker areas. Using a local threshold, instead of a global one, like Otsu's threshold, extracts more possible damages. This happens because when defining a global threshold all damages will participate in its computation, and hence damages which are only locally darker will not be considered background, as desired. Figure 5.9 shows examples of both binarization methods. Orange bounding boxes show examples of the damages to identify. For better visualization, those boxes are also zoomed in.

Following, the binary images are manipulated with morphological operations, to enhance features corresponding to RoIs. Using the edges detected, the image is processed by two sets of morphological operations: one set to enhance existing circles in the image and another to enhance edges. Furthermore, the result of the adaptive threshold goes through another set of morphological operations, to extract the darkest areas for later processing.



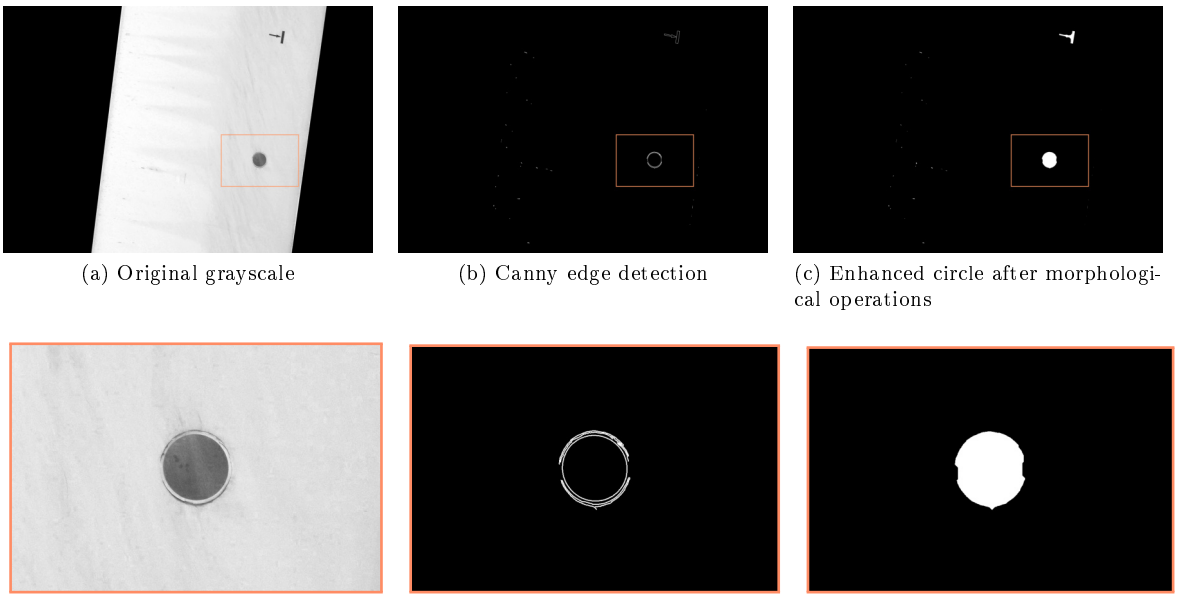
(a) Original grayscale (b) Canny edge detection (c) Adaptive threshold

(d) Zoom in grayscale (e) Zoom in Canny edge detection (f) Zoom in adaptive threshold

Figure 5.9: Binarization of LS case

Enhancing Circles

The goal of enhancing circles is so that in a latter stage, Hough circle algorithm can easily identify any existing circle, which is a crucial step towards checking for LRD. Knowing a circle only contains strong edges in its perimeter, applying a simple closing operation with a large disk structuring element will close the circle interior. Figure 5.10 shows an example of applying previous processes to enhance circles in an image containing a lightning receptor.



(a) Original grayscale (b) Canny edge detection (c) Enhanced circle after morphological operations

(d) Zoom in grayscale (e) Zoom in Canny edge detection (f) Zoom in enhanced circle

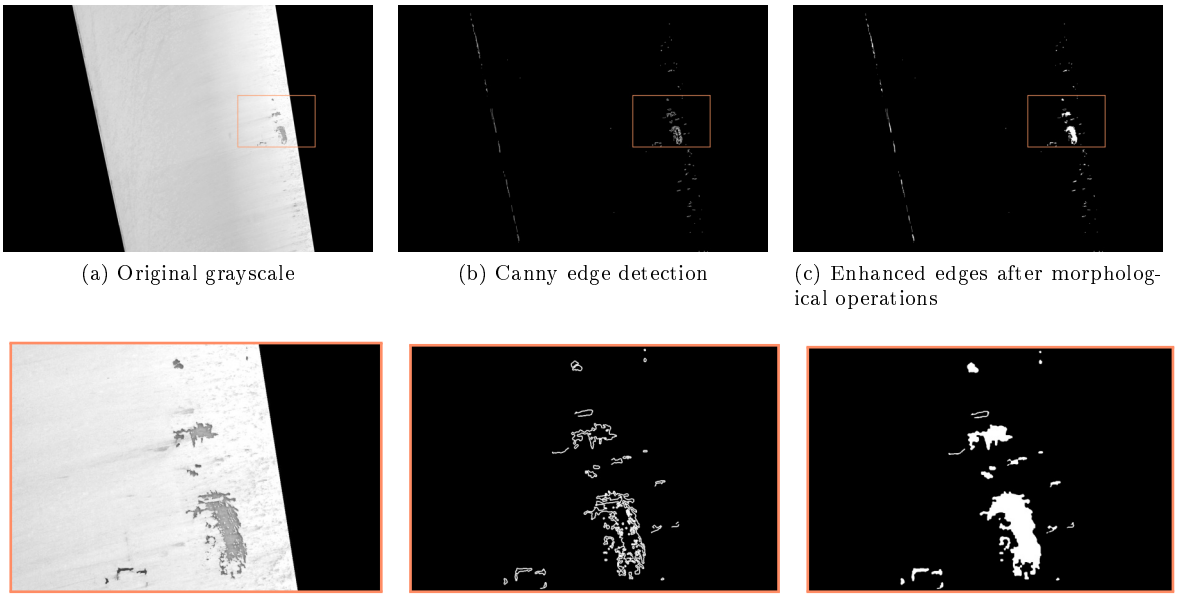
Figure 5.10: Enhancing circle in LRD case

Enhancing Edges

Similar to the previous case, other damages, like peeling or crack, contain strong edges in their boundaries and none in the interior area. The following set of operations has the purpose of closing any open boundary and then to fill closed areas:

1. Thickning: thickens edges, which will increase the accuracy of the bridge operation goal.
2. Bridge: connects edges close enough to each other
3. Fill: replaces to foreground any background pixel closed by foreground pixels.
4. Close: closing operation with a small structuring element will merge blobs close to each other, which are likely to belong to the same damage.

In figure 5.11 one observes an example of enhancing the edges of a large peeling damage.



(a) Original grayscale (b) Canny edge detection (c) Enhanced edges after morphological operations

(d) Zoom in grayscale (e) Zoom in Canny edge detection (f) Zoom in enhanced edges

Figure 5.11: Enhancing edges in large peeling case

Extract Dark Features

From the adapative threshold, most blade area is considered foreground (white), while damages represent background (black). To extract those damages from the image, one should get the mask containg the blade area and then subtract it with the result of the adaptive threshold. Figure 5.12 shows the application of this simple subtraction in the example of figure 5.9.

(a) Adaptive threshold (b) Mask from segmentation module (c) Extracted features

Figure 5.12: Extration of dark areas in LS case

5.1.3 Feature Classification

In order to classify damages it is necessary to establish which features best represent each damage type. Each damage can be identified by the features presented in table 5.1. Besides the ones already explained earlier, spatial density and eccentricity can also be used to describe a specific damage. For instance, fungi and small peeling are only distinguished from a density observation point. The eccentricity describes how close is a RoI shape to a line, where a perfect line eccentricity will be 1, while circular areas will have values closer to 0.

Damage type	Describer
Peeling	Strong edges, dark area and low spacial density
Fungi	Strong edges, dark area and high spacial density
Crack	Strong edges and high eccentricity
LS	Dark and large area
LRD	Circular shape with: less than 2 small circles inside, 1 large circle inside or strong edges in the surrounding area

Table 5.1: Damage describers for implementation of CVA

Fungi and Peeling

The classification of fungi and peeling was developed according to the flowchart presented in figure 5.13 and is explained next. As peeling and fungi present strong edges and locally dark area, to obtain such features one multiplies the result of the enhanced edges with the dark features extracted. Figures 5.14, 5.15 and 5.16 a) show examples of results from this multiplication, when applied to fungi, small and large peeling cases, respectively.

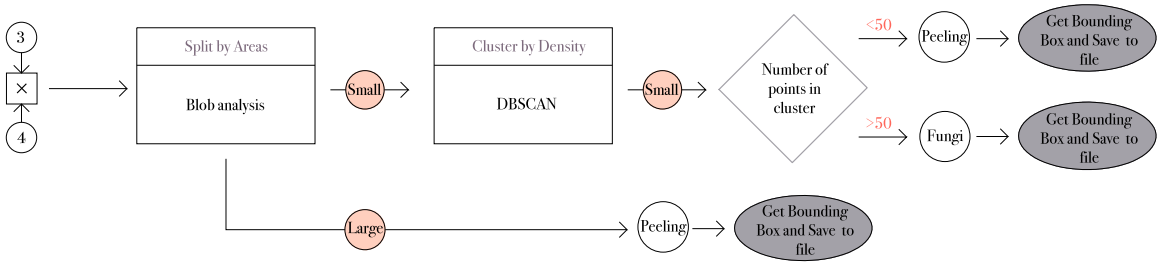
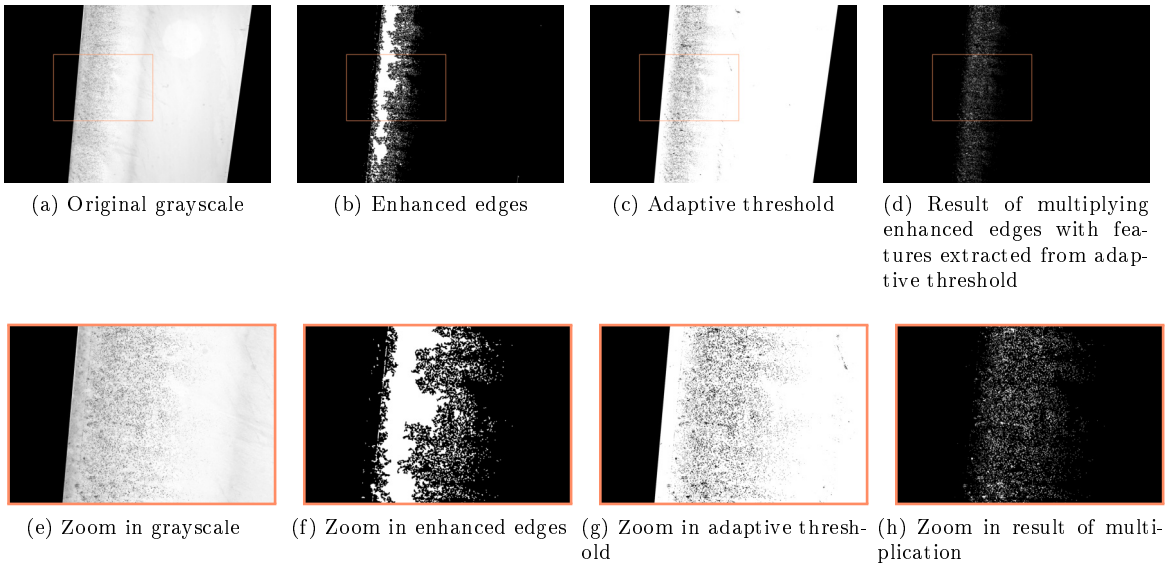


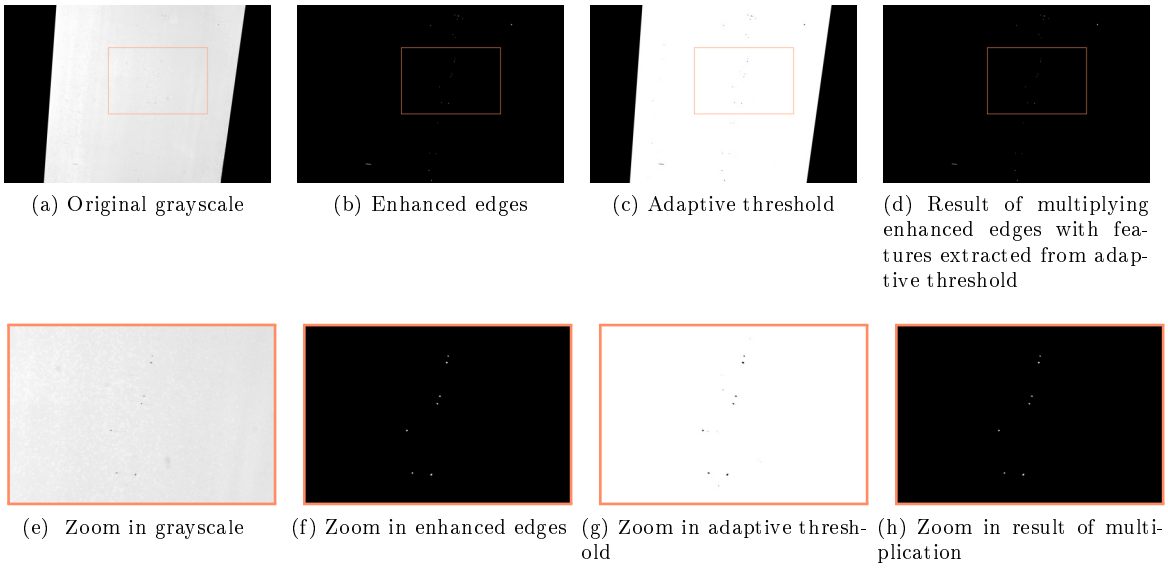
Figure 5.13: Fungi and small peeling classification module algorithm. The numbers 3 and 4 are the output of enhance edges and extract features sub-modules, respectively (see fig. 5.8). The circles in color refer to blobs filtered by small and large areas.



(a) Original grayscale (b) Enhanced edges (c) Adaptive threshold (d) Result of multiplying enhanced edges with features extracted from adaptive threshold

(e) Zoom in grayscale (f) Zoom in enhanced edges (g) Zoom in adaptive threshold (h) Zoom in result of multiplication

Figure 5.14: Extracting small edges in fungi case



(a) Original grayscale (b) Enhanced edges (c) Adaptive threshold (d) Result of multiplying enhanced edges with features extracted from adaptive threshold

(e) Zoom in grayscale (f) Zoom in enhanced edges (g) Zoom in adaptive threshold (h) Zoom in result of multiplication

Figure 5.15: Extracting small edges in small peeling case

Next, the blobs are split into small and large areas. This was manly applied so that small peeling and fungi can be distinguished from a density perspective. By doing so, large peeling will not be present in the DBSCAN process of fungi features, which could jeopardize the final detection and classification. Figure 5.16 b) shows how removing blobs with small areas produces a cleaner perception of the large peeling

present. When comparing DBSCAN results for small blobs, fungi will be characterized by clusters with high number of points, while small peeling will have low density clusters and many points considered noise. This can be confirmed in figure 5.17, where fungi presents only one cluster found and small peeling presents many low density clusters. The following parameters were chosen for these results:

1. Small areas (pixels): [5, 800].
2. Big areas (pixels): [800, inf].
3. DBSCAN for small areas: $\epsilon = 200, N_{min} = 2$.
4. DBSCAN for large areas: $\epsilon = 300, N_{min} = 2$.

(a) Result of multiplying enhanced edges with features extracted from adaptive threshold

(b) Result after removing small areas

Figure 5.16: Removing small areas in large peeling case (continuation of example present in fig. 5.11)

(a) DBSCAN in fungi case

(b) DBSCAN in small peeling case

Figure 5.17: DBSCAN in fungi and small peeling cases. Each color represents a cluster. When more than one cluster is found, red color represents points with no cluster.

Crack

The classification of cracks was developed according to the flowchart presented in figure 5.18 and is explained next. Cracks contain strong edges, yet not so much locally dark area, since its width is typically small, meaning cracks can be detected using only the result of the enhanced edges. First, areas smaller than 200 are excluded, since features with those dimensions are most likely to be peeling and fungi. Then, cracks can be identified with a blob analysis of eccentricities and density. When enhancing edges, fungi can be present, so a DBSCAN is performed to exclude clusters with high density. As mentioned earlier, a high eccentricity blob shows a shape close to a line, which describes cracks. For this purpose, cracks were considered blobs with eccentricity > 0.95. Figure 5.19 displays these steps applied to an example of cracks. It can be observed in 5.19 (b) that cracks features are enhanced, yet also some noise on the left side. Figure 5.19 (c) shows how DBSCAN can remove noise, by excluding high density clusters (red points do not belong to any cluster).

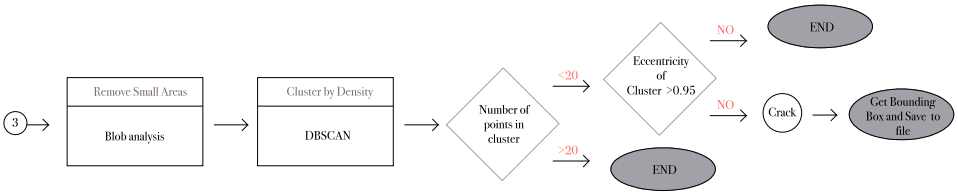
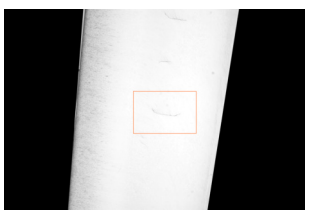
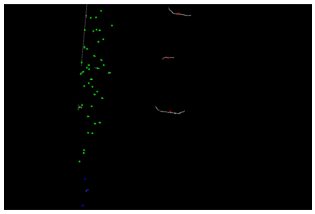


Figure 5.18: Crack classification module algorithm. The number 3 is the output of enhance edge sub-module (see fig. 5.8)



(a) Original grayscale

(b) Enhanced edges after removing small areas



(c) DBSCAN

(d) Zoom in original grayscale

(e) Zoom in enhanced edges

Figure 5.19: Enhanced edges in crack case

Lightning Strike

The classification of lightning strikes was developed according to the flowchart presented in figure 5.20 and is explained next. Lightning strikes have the peculiarity of containing large darker areas, however those areas are blurred and so will not show strong edges in their surrounding. Hence, analyzing large darker areas with no edges in the surrounding is enough to detect such damage. The subtraction block in figure 5.20, has the purpose of removing blobs with edges from the dark features extracted. Figure 5.21 shows how this subtraction removes any blob that was considered in the extraction of dark features and has strong edges. Moreover, to be considered lightning strike, blob areas must be larger than 5000 pixels.

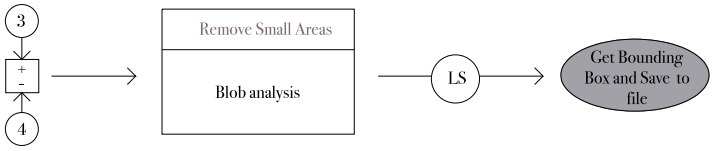
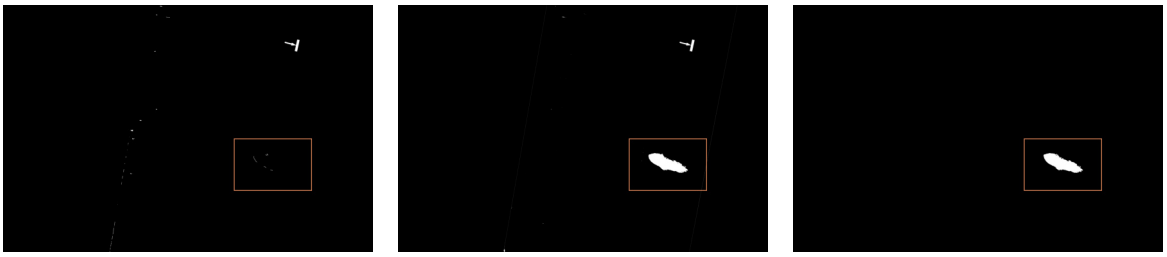


Figure 5.20: LS classification module algorithm. The numbers 3 and 4 are the output of enhance edges and extract features sub-modules, respectively (see fig. 5.8)



(a) Enhanced edges (b) Extracted features (c) Result of subtracting the enhanced edges from extracted features

Figure 5.21: Extract blobs with strong edges in LS case (continuation of the example present in fig. 5.12)

Lightning Receptor Damaged

Finally, the classification of lightning strikes was developed according to the flowcharts present in figures 5.22 and 5.23, which are explained next. To correctly identify LRD, it is necessary to proceed with a circle detection to the result of enhanced circles. This was carried out by the usage of Hough circles transform, with radius ranging in [20, 200] pixels. This range ensures some robustness to finding lightning receptors of any size. When applying Hough circle transform in *MATLAB*, using *findcircles* function, it is also required to specify a sensitivity parameter. Higher values of sensitivity will result in

less detections, and vice versa. A sensitivity of 0.85 is higher enough to detect lightning receptors, but not high enough to detect nonexistent circles.

If a circle is found, the algorithm will then evaluate whether the lightning receptor is damaged or not. Two types of LRD where already establish in section 1.5: lightning receptor covered or with peeling/crack in its surrounding.

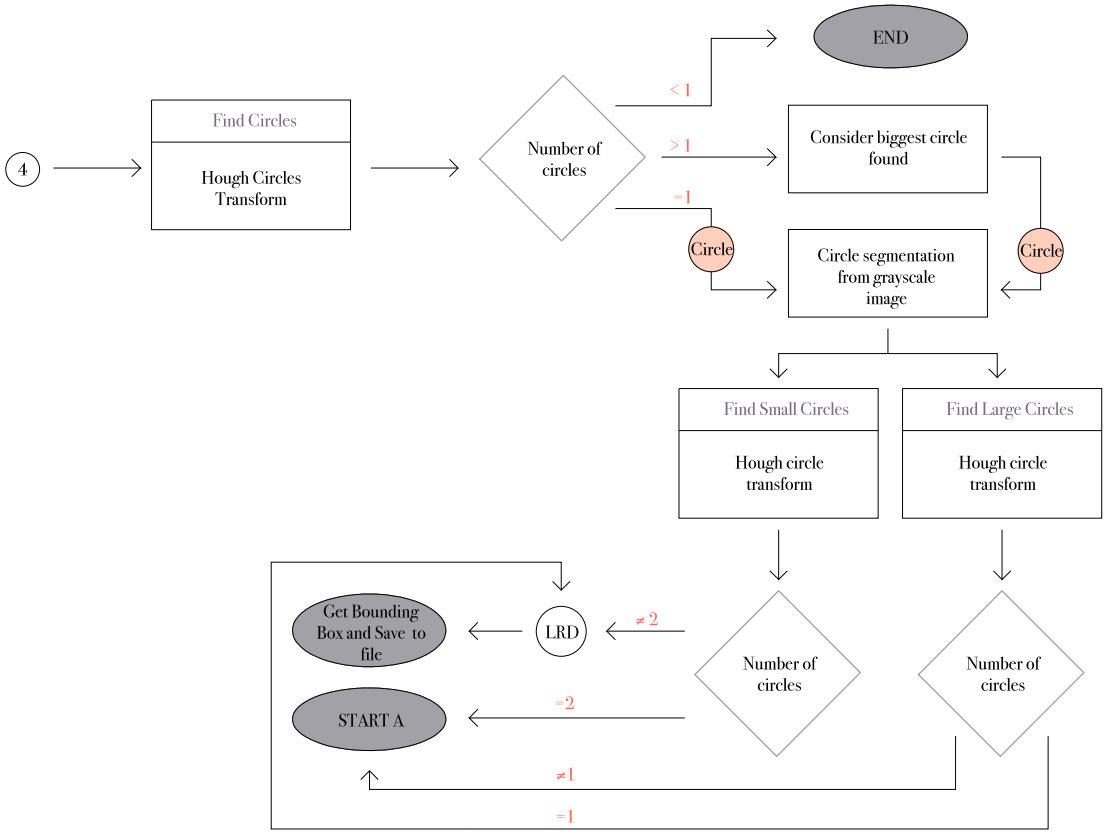


Figure 5.22: LRD classification module algorithm. The numbers 4 is the output of extract dark features sub-modules, respectively (see fig. 5.8). The "Start A" block is a sub-module (see fig. 5.23)

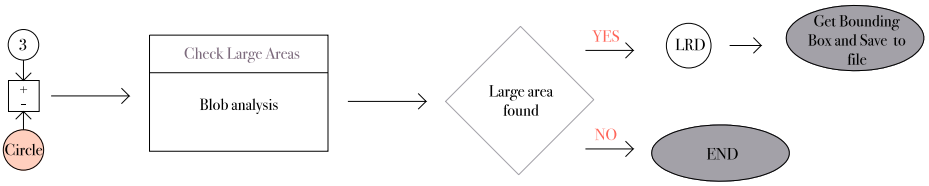
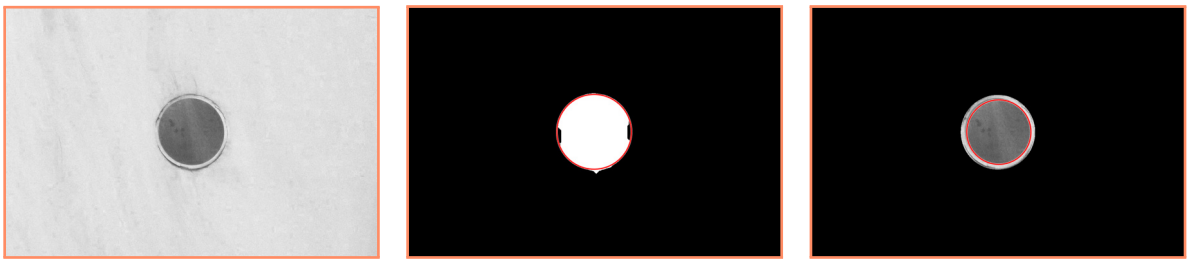


Figure 5.23: Start A sub-module from LRD classification module algortihm

To check whether it is covered or has a crack, the circle area is segmented from the grayscale image. For the covered case, a search for smaller circles is applied. If the algorithm does not find, exactly, 2 circles with similar radius, then the lightning receptor is probably covered and hence classified as LRD. Smaller circles were considered to have radius in range of 10% to 30% of the lightning receptor radius. Regarding the presence of cracks, the search of a large circle was made for a range of 70% to 90% of the lightning receptor radius. Whenever a large circle is found, the lightning receptor contains a crack in its surrounding. This strategy works because when enhancing circles in the feature extraction module, cracks are merged into the lightning receptor blob area and so it is possible to detect the true lightning receptor in its interior. Figure 5.24 shows an example of a lightning receptor covered and with a crack surrounding it, as well as the detection of the exterior and interior circles, using Hough transform.

Lastly, to investigate for peeling, the area corresponding to the lightning receptor found with the first circle search is removed from the enhanced circles binary image. The result will dictate the existence of peeling, by considering the remaining area of this subtraction. Any area larger than 800 pixels reveals peeling in the lightning receptor surrounding. Figure 5.25 shows an example of a lightning receptor with peeling surrounding it, as well as the detection of the exterior circle, small interior circles (meaning it is not covered) and of the peeling area.



(a) Original grayscale (b) Exterior circle detected (c) Interior large circle detected

Figure 5.24: Crack in LRD case

(a) Original grayscale (b) Exterior circle from enhanced circles detected (c) Interior small circles detected (d) Result of removing exterior circle from enhanced circles.

Figure 5.25: Peeling in LRD case

5.1.4 Non-Maximum Suppression

As the classification algorithm presents methods developed individually for each damage type, when coupling all together some lack of precision was notice , i.e., since each methodology did not take into consideration others damages, it is possible for those to mislead the algorithm for incorrect or redundant detections. In that sense, a non-maximum suppression was applied. The following conditions and consequent actions, were considered sufficient enough to improve the precision, while not ruining recall:

1. Equal label + IoU_{min}[1] $= 1 \implies$ smaller area is fully within the biggest one. Remove small area.

2. Equal label + $\frac{1}{3} < IoU_{min} < 1 \implies$ merge both bounding boxes to cover all the union area.

3. LS vs peeling/crack + $IoU_{min} > \frac{1}{3} \implies$ remove peeling/crack.

4. LRD vs peeling/crack +$IoU_{min} > \frac{1}{3} \implies$ remove peeling/crack.

5. Crack vs peeling + $IoU_{min} > \frac{1}{3} \implies$ remove peeling.

5.1.5 Results

In order to validate this solution, the CVA was ran for the training dataset already described in section 4. Table 5.2 shows the recall and precision of each damage when applying the algorithm described previously. Moreover, it is also displayed the results obtained without using NMS. It should be notice that this solution does not present confidence scores for the predictions and so the mAP metric described in section 4.2 can not be computed.

First, observing the recall values, the peeling presents the lowest value, which translates a poor detection of the ground truth cases. On the other hand, the remaining damages recall show that at least half of the ground truths were detected, except for the LS case, which detected correctly all the ground truth cases. The application of NMS improved slightly the precision of peeling, crack and LS damages, while maintaining the recall values. Nonetheless, the peeling precision is still pretty low, when compared to other damages, which means the algorithm is producing a large amount of false positives predictions. To understand these results and find the algorithm flaws, one should visually analyse the results.

[1] This is a variation of (3.6), where instead of considering the union area in the denominator, the smaller area between both bounding boxes is considered.

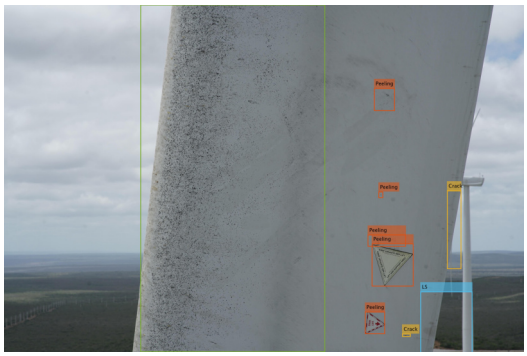
Damage type	*Recall*(%)		*Precision*(%)	
	Without NMS	NMS	Without NMS	NMS
Peeling	22.22	22.22	2.21	4
Crack	51.85	51.85	18.42	22.95
Fungi	56.25	56.25	45	45
LS	**100**	**100**	31.25	43.48
LRD	50	50	**62.50**	**62.50**

Table 5.2: Average precision for test dataset

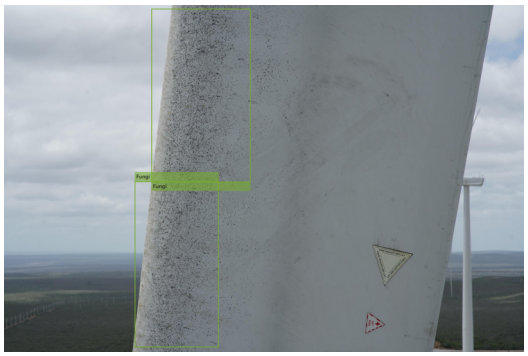
Figures 5.26, 5.27, 5.28, 5.29 and 5.30 show some examples of unsuccessful detections. Left side images display the predictions made by the algorithm, while the right side display the ground truth used for evaluation. Each damage type can be identified by its bounding box color. Looking at figure 5.26 and comparing the predictions made by the algorithm with the ground truth detections it is conclusive that many false positives are present, for different reasons. Despite that, there are some correct detections in these examples. For instant, the peeling detected in figure 5.29, cracks present in figures 5.28 and 5.30 and the LS in figure 5.27.

First, the segmentation module was not accurate, otherwise there would be no detections outside the blade area, which is the case of the crack (yellow) and LS (blue) damages. The same problem can be observed in the peeling prediction (orange) in figure 5.28. Second, the presence of stickers in the blade was not considered and so induces the algorithm to unwanted edges detections, which consequently affects the final result. Stickers usually produce false positives in the peeling class (fig. 5.26 and 5.27), but the same can happen for other classes, as the example of LRD (purple) in figure 5.28. Third, comparing the fungi bounding box (green) of the predictions with the ground truth, it is observed that even though the detection is correct, because the bounding box is not as tight fit adjusted to the damage area as the ground truth, it will be considered false positive. This was one of the main reasons to choose a low IoU threshold to compute the confusion matrix: the bounding box adjustment should not be an important factor to evaluate the results. Finally, the algorithm was able to detect true damages that are not present in the ground truth, e.g., the upper peeling damage detected. The exhaustive process of labelling the dataset generated errors, like missing some damages. Another example of this case is the crack prediction in figure 5.29, which is correct but is not present in the ground truth, so will be considered false positive.

Moreover, observing figure 5.30, one notices the algorithm tends to capture darker shadows as LS. Such result is logical since the algorithm considers any large darker area as LS. This is a case where illumination variance is a key factor for good results. The extracted features present rotation and translation invariance, but others factors such as illumination, scale and perspective may also affect the results. If illumination could be somehow controlled, the algorithm performance would increase.

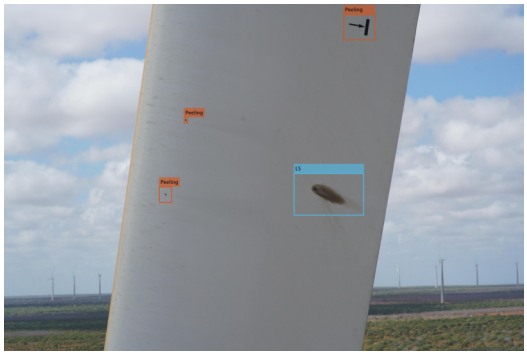


(a) CVApredictions



(b) Groundtruth

Figure 5.26: CVA incorrect predictions: example 1

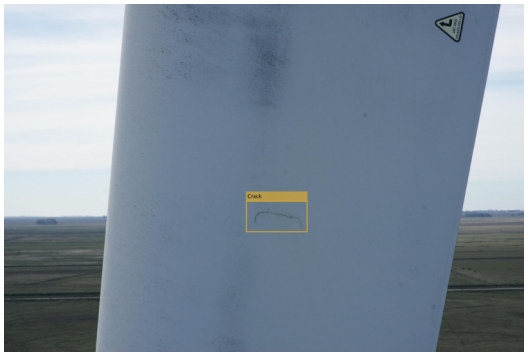


(a) CVApredictions

(b) Groundtruth

Figure 5.27: CVA incorrect predictions: example 2

(a) CVApredictions



(b) Groundtruth

Figure 5.28: CVA incorrect predictions: example 3

(a) CVApredictions

(b) Groundtruth

Figure 5.29: CVA incorrect predictions: example 4

(a) CVApredictions

(b) Groundtruth

Figure 5.30: CVA incorrect predictions: example 5

5.2 Deep Learning Algorithm

This section focuses on a practical implementation of Faster R-CNN, to detect damages in a dataset composed of wind turbine blades photographies. This solution was implemented in TensorFlow, using a high-level Application Programming Interface (API). For better understanding, a brief description of TensorFlow takes place in appendix C.

As transfer learning is applied, the next section describes the dataset used in the pre-training process.

5.2.1 Pre-Training Dataset

Common Objects in COntext (COCO) is a large-scale dataset, open for image recognition implementations. COCO dataset is used to tackle object detection and scene segmentation, meaning is appropriate

for Faster R-CNN training. Also, it was created with priority to non-iconic images, which have proved to provide a better generalization for the object detector [52]. Non-iconic images present objects in a natural context and from non-canonical perspectives. This results in images rich in context information, enabling the network to find correlations between objects and the environment where they are usually present. Moreover, non-iconic images tend to contain more object instances per image than iconic ones, which is visible when comparing benchmark datasets. In the original paper, [30], COCO presented 3.5 categories and 7.7 object instances per image, while both ImageNet and PASCAL VOC contained less than 2 categories and 3 object instances per image. Having this said, COCO dataset is a reliable dataset for applying transfer learning in this work. Nowadays, COCO contains more than 200 thousand labeled images, with roughly 1.5 million object instances, spread across 80 categories.

5.2.2 Hyperparameters Selection

Hyperparameters are parameters that can be adjusted by the user. Their values selection may influence significantly the network performance. In a time consuming process, a researcher should evaluate the sensitivity of a network to each of the different hyperparameters. Yet, because this work was restricted to the usage of only 1 GPU machine, this exhaustive process did not take place. Instead, the hyperparameters were reused from the pre-training process. Knowing which hyperparameters achieved state-of-art results in the COCO dataset, the chances of resulting in the new dataset are huge.

In the previous chapter, some hyperparameters were already established, while others were not. Table 5.3 summarizes all hyperparameters.

The new layers weights refer to the transfer learning process. As most of the learned parameters are obtained from a pre-training, the weights from the new layers are set to a Gaussian distribution. In Faster R-CNN these layers are the classification and regression layers from RPN and Fast R-CNN. All the other weights are reused from a pre-trained model.

Regarding the mini-batch used to compute the RPN loss, it is composed of a $1:1$ ratio between positive and negative RoIs[2], meaning there are 128 of each kind. Yet, because the amount of negatives is always superior to positives, whenever there is lack of positives to make the 128, negatives are padded to fill the gap. This asymmetry of negatives and positives is another reason to use mini-batches. If one used all RoIs from an image, the network would drive the gradient descent while relying mostly on negative RoIs, which is undesired.

The choice of the balancing parameter, λ, took into attention the normalizations applied to both classification and regression losses. Since the classification is normalized by 256 and the regression is

[2]The labeling process happens in RPN and is described in 3.5

normalized by roughly 2400, the value 10 would guarantee a good balance between both. Furthermore, the authors empirically proved that the network performance was insensitive to this parameter, for a large range of values. Lastly, the authors did not mention the reasons to choose a global gradient threshold of 10, but it may have been relied on experimental results.

Hyperparameters	
New layers weight initialization	zero-mean Gaussian distribution with standard deviation 0.01
ResNet-101 Convolutional Layers	as described in figures 3.11 and 3.12
ResNet-101 Max Pooling Layer	filter size(w × h): 2 × 2, stride: 2
RPN anchors	threshold for positive label: $IoU > 0.7$, threshold for negative label: $IoU < 0.3$, areas: (128^2, 256^2, 512^2) ratios: (1 : 1, 1 : 2, 2 : 1)
ROI Pooling Layer	filter size(w × h): $\frac{w_{ROI}}{7} \times \frac{h_{ROI}}{7}$, stride: $\frac{w_{ROI}}{7}$
Hidden Layer: activation function	ReLU
Classification Layer: activation function	Softmax
Regression Layer: activation function	Regression
Gradient Descent with Backpropagation	learning rate η: 0.0003, momentum α: 0.9, mini-batch size M: 256, balancing parameter λ: 10, global gradient norm threshold: 10

Table 5.3: Hyperparameters of Faster R-CNN, as in the original paper [39]

5.2.3 Input Resizing

Faster R-CNN has the peculiarity of working in datasets composed of images with different sizes. To do it, Faster R-CNN resizes images so that the shorter side has a maximum of 600 pixels, while maintaining, if possible, the aspect ratio. Also, the longest side is restricted to a maximum of 1024 pixels. An image of size 5120×3415 is resized to 900×600.

A problem can arise from this approach, since the dataset used has images roughly 28 times bigger than the resized ones. This means small objects will be resized, in such a way that information may be lost as the background blurs with the object.

5.2.4 Data augmentation

Data augmentation provides simple image pre-processing techniques to enlarge the dataset. This becomes extremely handy when dealing with small datasets, such as the one of this work. In that sense, during the training stage 3 operations for data augmentation were randomly applied: horizontal flip, brightness adjustment and contrast adjustment. The combination of these operations decreases the

chances of Faster R-CNN overfitting with the training dataset.

5.2.5 Results

The training process was set to a total of 300K steps, during which the validation dataset was evaluated at every 10K steps. This evaluation is important to understand whether the model is overfitting or not. Furthermore the loss is also a good measurement to evaluate the training process, as it represents the error of the model to the training dataset. Concerning the transfer learning process, no layers were frozen during training.

The loss was stored in memory every 100 steps and is displayed in figure 5.31. A smoothed loss was also computed, to better visualize the decrease in loss values. Even though the loss presents many peak fluctuations, the magnitude and frequency decrease as the training evolves. These fluctuations are associated with the batch size of 1 implemented in Faster R-CNN. This means the model computes the loss and updates the parameters after seing each image, causing a stochastic behaviour. The smoothed loss was calculated with a Savitzky - Golay filter [43]. This digital filter goes through each point and collects a window, with the previous point at the center of neighborhoods. Having this, the collection of points is fitted by a polynomial function. Then the center point is replaced by the value obtained by the polynomial function. To obtain the smoothed loss, a window width of 201 points and a third degree polynomial to fit each window it were chosen. These values were chosen with the concern of not loosing too much information.

Regarding figure 5.31, since the loss globally decreases, it is conclusive that, while training, the model learned how to adapt to the training dataset. Figure 5.32 gathers the mAP of the evaluations performed during training, for the range 50K to 300K steps.

It is important to check if the model reached overfitting, so one should analize the evaluation with the validation dataset. It was establish a minimum of 50K steps for training the model, meaning the first validation was only performed after that amount of training the network. In case the model overfitted with the training dataset, the validation mAP would decrease in a conclusive way, but that is not the case. Instead, it is observed small variations of the mAP. Nonetheless, the model seems to perform better with the validation dataset before overpassing 80K steps, which could mean the model is overfitting with the training dataset after those steps (since the loss continues to decrease). For this reason the model trained for 80K steps was selected for later testing.

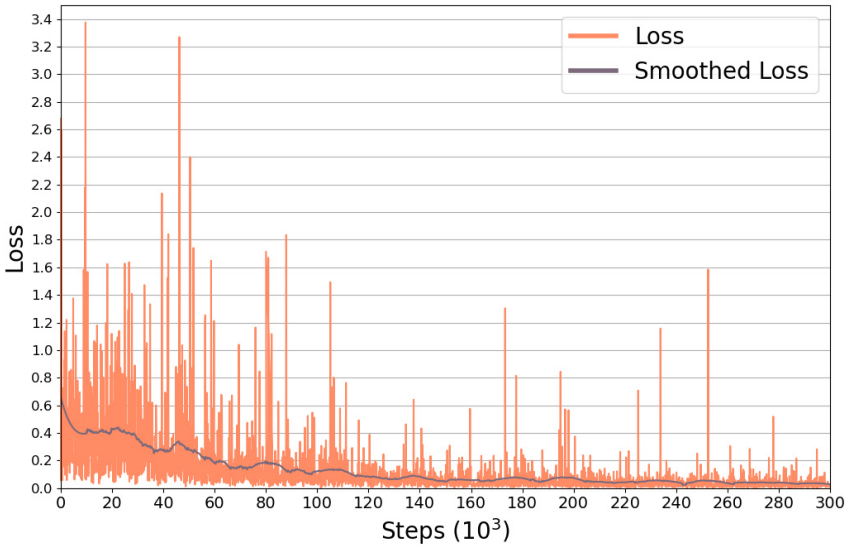


Figure 5.31: Loss value every 100 training steps

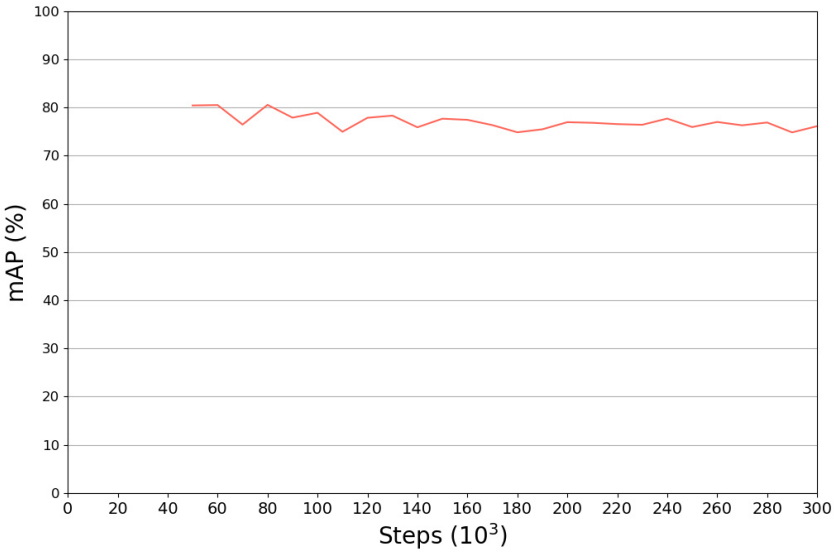


Figure 5.32: Evalutation of validation dataset every 10K steps

Chapter 6

Analysis and Comparison of Results

This chapter gathers the analysis and comparison of the results obtained with both proposed solutions, when applied to the same test dataset.

6.1 Testing Results CVA

The CVA solution was run for the test dataset defined in section 4. One should remind that images containing erosion were excluded, because this damage is identified for its spacial environment and not its features. The results obtained for the test dataset are presented in table 6.1. Comparing these results with the ones obtained with the training dataset (table 5.2), it is conclusive the algorithm performed worst for the test dataset. This could be explained knowing the algorithm was built on few images, relatively to the test dataset size. In that sense, even though the calibrated parameters showed feasible results during the development, the same can not be ensured for testing. The fungi algorithm seems to be the more robust, presenting the lowest performance drops. On the other hand, the LS algorithm, which obtained a 100% recall and 43.48% precision on the development stage, dropped significantly its performance on testing stage. From all the damages, LS presented the simplest algorithm, with few extracted featured for the classification module. In section 5.1.5 was already referenced that the LS algorithm was classifying darker shadows as a damage, which implies the features used are dependent on the image illumination. In fact, observing the example images of LS at the appendix D, it is noticed that illumination varies greatly from case to case. This can justify the fact that the parameters optimized in the development stage where not successful in this stage.

Concerning the peeling and crack damages, the bad results can be related to a lack of generalization in the algorithms. Peeling and cracks have the most irregular shapes and sizes, so the usage of classical computer vision techniques is not enough to ensure appropriate results. In an overall view, only the fungi

and LRD algorithms were successful, but these damages are also easily identified by strong and dissimilar features. The LRD class achieved relatively good results, yet it should be considered that all datasets presented few occurrences of Lightning receptors in good conditions. This means the results may not be conclusive, as the algorithm could be detecting any lightning receptor, instead of catching damaged ones.

Damage type	Recall(%)	Precision(%)
Peeling	15.13	2.66
Crack	26.14	8.30
Fungi	**44.74**	38.46
LS	23.60	6.86
LRD	36.99	**41.86**

Table 6.1: CVA recall for test dataset

6.2 Testing Results DLA

The model proposed in section 5.2.5 was evaluated using the test dataset described in section 4, and was set to output constantly 300 detections, by choosing those with higher confidence score. Table 6.2 presents the Average Precision per class, as explained in section 4.2.3, while table 6.3 shows the mAP for the test and validation datasets. Once again, the LRD stands out for having the highest result, but as explained in section 5.2.5, that could be related to a lack of non-damaged lightning receptors in the test dataset.

As explained in section 4, the class erosion was only evaluated for the deep learning solution, as an attempt to confirm whether the network can distinguish a class which depends on the natural context where it is inserted. With a look at the average precision per class, the erosion presents an Average Precision (AP) close to other classes, meaning the previous hypothesis is confirmed. Nevertheless, this class contains the most amount of occurrences, which could be beneficial in the training stage, achieving better results.

Regarding table 6.3, the performance of the testing stage was worst than the validation performed during training (see fig. 5.32). Even though the difference is not as large as for the CVA results, these performance drops could indicate a disparity between the testing dataset and the training/validation datasets used in the design stage.

Damage type	Average Precision(%)
Erosion	70.34
Peeling	62.92
Crack	72.57
Fungi	68.94
LS	66.99
LRD	**93.47**

Table 6.2: Average precision for test dataset, for the DLA

	Validation	Test
mAP(%)	**80.54**	72.54

Table 6.3: Comparison between mAP in validation and test datasets, for the DLA

6.3 Performance Comparison

The comparison of the solutions proposed cannot be done directly using mAP, described in section 4.2.3, since this metric requests the existence of confidence scores to compute the average precision area. As the CVA does not output a confidence score, it was established the following comparison: precision and recall are computed for CVA and checked whether the corresponding point would fall inside or outside the Faster R-CNN average precision area. In case it falls inside, the CVA is considered to have lower performance. Figure 6.1 demonstrates exactly this comparison.

The results conclude that the CVA solution is outperformed by Faster R-CNN. Both peeling, crack and LS detections from CVA show really low precision values, which are consequences of the limitations and invariances explained in section 6.1. Furthermore, the confusion matrix was constructed from the IoU of the detections and ground truths, which is sketchy for the CVA evaluation, since it measures the overlap of the bounding boxes. The process of fitting the bounding box to damages is not the focus of this work and so was not considered when developing the CVA solution. The classes fungi and LRD show slightly larger values of recall and precision, but are also outperformed by Faster R-CNN.

In addition, the CVA algorithm takes on average roughly 36 seconds to detect damages on each image, while Faster R-CNN only takes 5 seconds. There are two reasons to justify such difference. First, the CVA algorithm was programmed from scratch and small efforts were made to build the algorithm coding more efficient. Contrarily, Faster R-CNN was implemented from optimized coding libraries. Secondly, as explained in section 5.2.3, the images are resized to smaller sizes before being fed to Faster R-CNN. On the other hand, CVA uses the original images sizes. Nonetheless, detection time is just a comparison reference, since no goal was establish for that purpose.

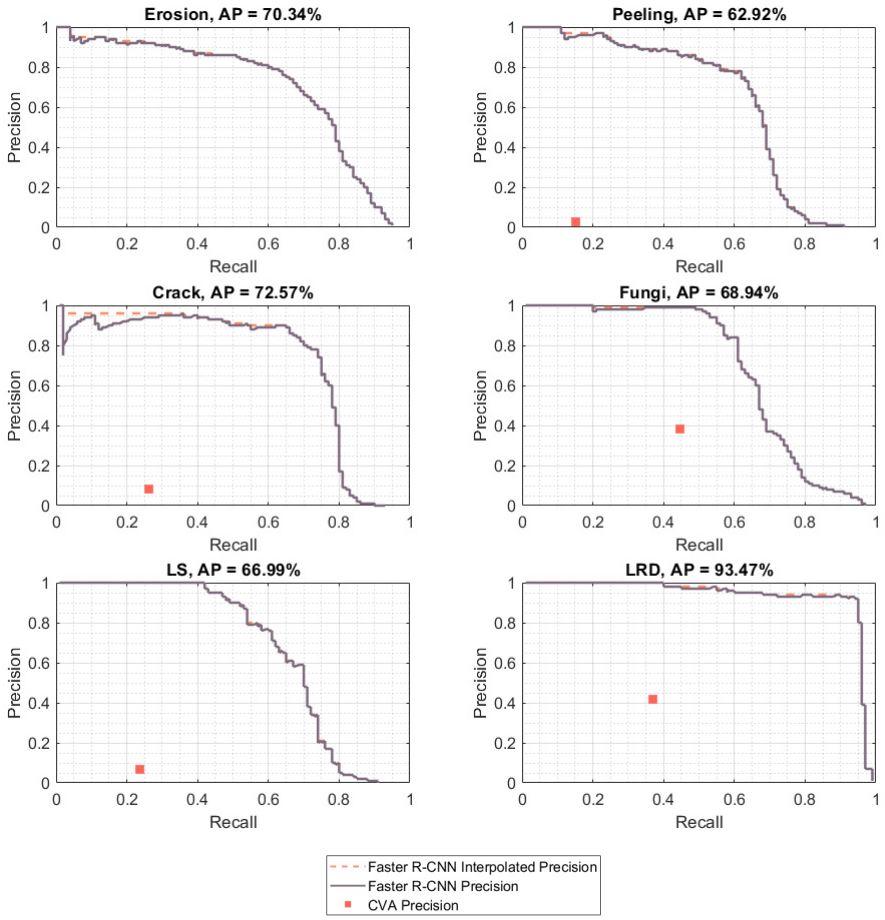


Figure 6.1: CVA and DLA performance comparison

	CVA	**Faster R-CNN**
Detection time(s)	35.59	**5.16**

Table 6.4: CVA and DLA detection time per image comparison

Chapter 7

Conclusions

Automatic visual inspection of wind turbines blades is not a trivial task. First, concerning the damage classification, there is still no standard labelling system, as it will depend on the inspector, country and training. Second, the photographs themselves contain external factors such as illumination, which can not be controlled and will influence the results. In section 5.1, limitations were found in the development of the first solution. Besides the external factors already referred, the manual calibration of parameters was a large workload and showed good results in a design stage, even though during testing (see section 6.3) its performance dropped. The fact that segmentation of the blade is a key factor to obtain reasonable results is considered a huge disadvantage when compared to the second solution.

In section 5.2 transfer learning was applied in a Faster R-CNN with a Resnet-101 feature extractor. The hyperparameters were reused from the pre-training process. In seek for better results, the hyperparameters sensitivity should have been evaluated, but that would demand a larger amount of computation time.

Since the loss converged to zero, its conclusive the model adapted to the training dataset, yet during the last $\frac{2}{3}$ of the training the validation mAP did not show much variation, meaning the network kept to learn the damages, while not overfitting with the training dataset.

The model trained until 80K steps was chosen for testing, since it had the highest mAP in the validation. The testing confirmed that erosion can be detected in the same manner as other damages, meaning the model learned the spatial context of damages. As the LRD class achieved a surprisingly high AP, it is presumed that this was caused by the lack of examples of lightning receptors in good conditions. Even though data augmentation was performed during training, the dataset should be larger to improve the results.

Both solutions were compared using precision and recall. For all classes, Faster R-CNN performance was much higher that for the CVA solution. A note should be made since the CVA did not cover the

best adjustment of bounding boxes to the object, which greatly influences the confusion matrix and so precision and recall. Using IoU as a threshold for the confusion matrix is a harsh metric for a problem were the bounding box adjustment is not a requirement to achieve the goals.

Regarding processing time, CVA is roughly $7\times$ slower than Faster R-CNN. This is explained by the not so good coding when developing the algorithm, as well for the fact that CVA uses the original size of the images (5120×3415), while Faster R-CNN resizes it to 900×600. Moreover, CVA contains the segmentation module, which takes half of the processing time. This module is not necessary in Faster R-CNN, which is another proof of the model learning the natural context of the damages.

Concluding, deep learning is a more feasible implementation for the problem presented, overcoming traditional computer vision techniques.

Regarding the deep learning solution, some improvements could be done to achieve better results. First the hyperparameters could be adjusted and better analysed for sensitivity. For instance, the anchors ratios and areas could be adapted for the sizes and shapes of the damages to identify. Second, the model would achieve higher mAP if the training dataset was larger and the training set to more steps. Also, more examples of lightning receptors in good conditions could be added. Third, the training dataset was unbalanced between damages occurrences, which could be handle using methods for balancing data.

Lastly, the deep learning solution could be extended to real-time detections, using lighter and faster networks.

Bibliography

[1] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P. and Süsstrunk, S. [2012], 'Slic superpixels compared to state-of-the-art superpixel methods', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **34**(11), 2274–2282. 21

[2] Bengio, Y., Simard, P. and Frasconi, P. [1994], 'Learning long-term dependencies with gradient descent is difficult', *IEEE Transactions on Neural Networks* **5**(2), 157–166. 36

[3] Bradley, D. and Roth, G. [2007], 'Adaptive thresholding using the integral image', *Journal of graphics tools* **12**(2), 13–21. 13

[4] Canny, J. [1986], 'A computational approach to edge detection', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 679–698. 15

[5] Chady, T., Sikora, R., Lopato, P., Psuj, G., Szymanik, B., Balasubramaniam, K. and Rajagopal, P. [2016], 'Wind turbine blades inspection techniques', *Organ* **5**, 16. 2, 4

[6] contributors, W. [2020], 'Dbscan — wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=DBSCAN&oldid=944848601`. 88

[7] Dai, J., Li, Y., He, K. and Sun, J. [2016], R-fcn: Object detection via region-based fully convolutional networks, *in* 'Advances in Neural Information Processing Systems', pp. 379–387. 25

[8] Drewry, M. A. and Georgiou, G. [2007], 'A review of ndt techniques for wind turbines', *Insight-Non-Destructive Testing and Condition Monitoring* **49**(3), 137–141. 4

[9] Duda, R. O. and Hart, P. E. [1971], Use of the hough transformation to detect lines and curves in pictures, Technical report, Sri International Menlo Park Ca Artificial Intelligence Center. 18

[10] Elbehiery, H., Hefnawy, A. and Elewa, M. [2005], 'Surface defects detection for ceramic tiles using image processing and morphological techniques'. 11

[11] Ester, M., Kriegel, H.-P., Sander, J., Xu, X. et al. [1996], A density-based algorithm for discovering clusters in large spatial databases with noise., *in* 'Kdd', Vol. 96, pp. 226–231. 21

[12] Forsyth, D. A. and Ponce, J. [2002], *Computer vision: a modern approach*, Prentice Hall Professional Technical Reference. 17

[13] Fukushima, K. and Miyake, S. [1982], Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, *in* 'Competition and Cooperation in Neural Nets', Springer, pp. 267–285. 24

[14] Girshick, R. [2015], Fast r-cnn, *in* 'Proceedings of the IEEE International Conference on Computer Vision', pp. 1440–1448. 25, 40

[15] Girshick, R., Donahue, J., Darrell, T. and Malik, J. [2014], Rich feature hierarchies for accurate object detection and semantic segmentation, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 580–587. 25, 33

[16] Glorot, X. and Bengio, Y. [2010], Understanding the difficulty of training deep feedforward neural networks, *in* 'Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics', pp. 249–256. 36

[17] Goodfellow, I., Bengio, Y. and Courville, A. [2016], *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`. 24

[18] Gordon, P. [2020], 'Smart energy international - vestas wins turbine order for russia's largest wind farm', `https://www.smart-energy.com/renewable-energy/vestas-wins-turbine-order-for-russias-largest-wind-farm/`. 3

[19] Greensted, A. [2010], 'The lab book: Otsu thresholding', `http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html`. 13

[20] He, K. and Sun, J. [2015], Convolutional neural networks at constrained time cost, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 5353–5360. 36

[21] He, K., Zhang, X., Ren, S. and Sun, J. [2016], Deep residual learning for image recognition, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 770–778. 36, 37

[22] Hough, P. V. [1962], 'Method and means for recognizing complex patterns'. US Patent 3,069,654. 18

[23] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. et al. [n.d.], 'Speed/accuracy trade-offs for modern convolutional object detectors (2016)', *arXiv preprint arXiv:1611.10012* . 26

[24] Ioffe, S. and Szegedy, C. [2015], 'Batch normalization: Accelerating deep network training by reducing internal covariate shift', *arXiv preprint arXiv:1502.03167* . 44

[25] IRENA [2020], 'Renewable energy statistics 2019 - electricity generation', `https://www.irena.org/Statistics/View-Data-by-Topic/Capacity-and-Generation/Statistics-Time-Series`. v, vii

[26] Jüngert, A. [2008], 'Damage detection in wind turbine blades using two different acoustic techniques', *The NDT Database & Journal (NDT)* . 2

[27] Karhunen, M. [2017], Statutory of safety equipment in Wind Turbines - The development of the service product and its implementation, PhD thesis, Tampere University of Technology. 1, 4

[28] Ke-ming, Z. and Hua-bing, W. [2012], Automated visual inspection and its application on automated inspection, *in* 'Proceedings of the 1st International Conference on Mechanical Engineering and Material Science', Atlantis Press. 5

[29] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. [1998], 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324. 25

[30] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C. L. [2014], Microsoft coco: Common objects in context, *in* 'European Conference on Computer Vision', Springer, pp. 740–755. 70

[31] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. and Berg, A. C. [2016], Ssd: Single shot multibox detector, *in* 'European Conference on Computer Vision', Springer, pp. 21–37. 25

[32] MacQueen, J. et al. [1967], Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability', Vol. 1, Oakland, CA, USA, pp. 281–297. 21

[33] McCarthy, J., Minsky, M., Rochester, N. and Shannon, C. [1955], 'A proposal for the dartmouth summer conference on artificial intelligence', *Conference Announcement* . 23

[34] McCulloch, W. S. and Pitts, W. [1943], 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics* **5**(4), 115–133. 24

[35] Otsu, N. [1979], 'A threshold selection method from gray-level histograms', *IEEE Transactions on Systems, Man, and Cybernetics* **9**(1), 62–66. 13

[36] Quispitupa, A., Vestergaard, B. and Sieradzan, T. [2013], 'Certification of wind turbine blades–the dnv procedure'. 3

[37] Rahaman, G., Hossain, M. et al. [2009], 'Automatic defect detection and classification technique from image: a special case using ceramic tiles', *arXiv preprint arXiv:0906.3770* . 11

[38] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. [2016], You only look once: unified, real-time object detection, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 779–788. 25

[39] Ren, S., He, K., Girshick, R. and Sun, J. [2015], Faster r-cnn: Towards real-time object detection with region proposal networks, *in* 'Advances in Neural Information Processing Systems', pp. 91–99. 25, 31, 71

[40] Rosenblatt, F. [1958], 'The perceptron: a probabilistic model for information storage and organization in the brain.', *Psychological review* **65**(6), 386. 24

[41] Rumelhart, D. E. and McClelland, J. L. [1986], 'Parallel distributed processing: explorations in the microstructure of cognition. volume 1. foundations'. 24

[42] Sabatelli, M., Kestemont, M., Daelemans, W. and Geurts, P. [2018], Deep transfer learning for art classification problems, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 0–0. 30

[43] Savitzky, A. and Golay, M. J. [1964], 'Smoothing and differentiation of data by simplified least squares procedures.', *Analytical chemistry* **36**(8), 1627–1639. 72

[44] Simonyan, K. and Zisserman, A. [2014], 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* . 36

[45] Srivastava, R. K., Greff, K. and Schmidhuber, J. [2015], 'Highway networks', *arXiv preprint arXiv:1505.00387* . 36

[46] Sutton, R. [1986], Two problems with back propagation and other steepest descent learning procedures for networks, *in* 'Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986', pp. 823–832. 40

[47] Sy, N., Avila, M., Begot, S. and Bardet, J.-C. [2008], Detection of defects in road surface by a vision system, *in* 'MELECON 2008-The 14th IEEE Mediterranean Electrotechnical Conference', IEEE, pp. 847–851. 11

[48] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. [2015], Going deeper with convolutions, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 1–9. 37

[49] Szeliski, R. [2010], *Computer vision: algorithms and applications*, Springer Science & Business Media. 5

[50] Thoma, M. [2017], 'Analysis and optimization of convolutional neural network architectures', *arXiv preprint arXiv:1707.09725* . 27, 28

[51] Thomsen, O. T. [2009], 'Sandwich materials for wind turbine blades—present and future', *Journal of Sandwich Structures & Materials* **11**(1), 7–26. 3

[52] Torralba, A., Efros, A. A. et al. [2011], Unbiased look at dataset bias., *in* 'CVPR', Vol. 2, Citeseer, p. 7. 70

[53] Werbos, P. [1974], 'Beyond regression:" new tools for prediction and analysis in the behavioral sciences', *Ph. D. dissertation, Harvard University* . 24

[54] WindEurope Business Intelligence, Daniel Fraile, A. M. [2017], Wind in power 2017 - annual combined onshore and offshore wind energy statistics, Report, Wind Europe. 1, 2

[55] Yun, D. and Lim, H. [2013], 'Study of the damage monitoring system on wind turbine blades'. 2

[56] Yun, J. P., Park, Y., Seo, B., Kim, S. W., Choi, S. H., Park, C. H., Bae, H. M. and Hwang, H. W. [2006], Development of real-time defect detection algorithm for high-speed steel bar in coil (bic), *in* '2006 SICE-ICASE International Joint Conference', IEEE, pp. 2495–2498. 11

[57] Zeiler, M. D. and Fergus, R. [2014], Visualizing and understanding convolutional networks, *in* 'European Conference on Computer Vision', Springer, pp. 818–833. 36

Appendix A

Clustering Algorithms

A.1 K-means

K-means clustering is a powerful unsupervised algorithm, able to cluster N-data points into K clusters. A simple case of K-means clustering is the centroid clustering. For this particular case, clusters are initialized at predefined positions $C_k = [x_k, y_k]$, and are iteratively changed in the direction of decreasing a distance metric. A commonly used distance metric is the euclidean squared distance between cluster centers and data points as:

$$D = (x_k - x_i)^2 + (y_k - y_i)^2 \tag{A.1}$$

A downside of K-means is the dependency on the initialization of cluster centers, as well as the required specification of number of clusters, K. For a deeper understand of how K-means iteratively clusters data, appendix 2 describes all crucial steps.

Algorithm 2 K-means clustering

```
Define number of clusters K
Define clusters centers C_k = [x_k, y_k] with k = 1, 2, ..., K
while Convergence not reached do
  for p in N-data points do
    Compute distance measure D (equation A.1) between p and all K clusters
    Assign p to nearest cluster
  end for
  for k = 1 : K do
    Compute mean of points assigned to cluster k
    Update cluster C_k = mean(p_k)
  end for
end while
```

A.2 DBSCAN

As referred earlier, DBSCAN clusters data points in high and low density areas. By inspecting the neighborhood of each point, it efficiently clusters dense areas, while also removing outliers. In a 2 dimensional dataset, the neighborhood is typically a circle defined by a radius, ϵ. Points are considered core of a dense cluster whenever its neighborhood contains a minimum of points, N_{min}. Lets consider the example scheme presented in figure A.1, with $N_{min} = 4$. Orange points are considered core, since its neighborhoods present a minimum of 4 points. Yellow points are called borders because they do not have a minimum of 4 points, yet, because are caught in the neighborhood of core points, are associated with the cluster. The purple point has low density, so it represents noise.

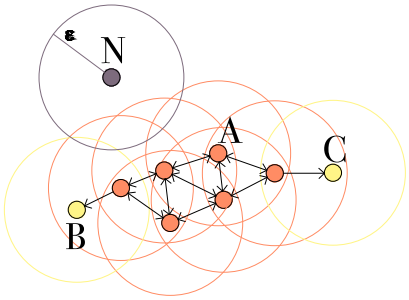


Figure A.1: DBSCAN scheme (adapted from [6]

Algorithm 3 DBSCAN

```
Define radius of circle neighborhood ϵ
Define minimum number of points inside circle N_min
while Not evaluated every point do
  Randomly choose point to evaluate P
  if P is already evaluated then
    Continue
  else
    Label P as evaluated
    Compute number of points in neighborhood N_ϵ
    if N_ϵ < N_min then
      P is noise
    else
      while Cluster C not stop expanding do
        P and neighborhood are core of C
        Evaluate points inside cores neighborhoods of C
      end while
    end if
  end if
end while
```

Appendix B

Backpropagation Formulations

$$\frac{\partial L_{cls}}{\partial w^l} = -\sum_{n}^{N} \frac{\partial L_{cls}}{\partial \hat{p}_n} \frac{\partial \hat{p}_n}{\partial a^l} \frac{\partial a^l}{\partial z^l} \frac{\partial z^L}{\partial w^l} = (\hat{p}_n - p_n) a^{l-1} \sigma'(z^l) \tag{B.1}$$

$$\frac{\partial L_{reg}}{\partial w^l} = \frac{\partial L_{reg}}{\partial \hat{t}} \frac{\partial \hat{t}}{\partial P} \frac{\partial P}{\partial a^l} \frac{\partial a^l}{\partial z^l} \frac{\partial z^l}{\partial w^l} \tag{B.2}$$

$$\frac{\partial L_{reg}}{\partial \hat{t}} = \sum_{j \in \{x,y,w,h\}} \frac{\partial smooth(\hat{t}_j - t_j)}{\partial \hat{t}_j} = \begin{cases} \hat{t}_j - t_j & \text{if } |\hat{t}_j - t_j| < 1 \\ \frac{|\hat{t}_j - t_j|}{\hat{t}_j - t_j} & \text{otherwise} \end{cases} \tag{B.3}$$

$$\frac{\partial \hat{t}}{\partial P} = \left\{ \frac{\partial \hat{t}_x}{\partial P_x} = \frac{1}{A_w}, \quad \frac{\partial \hat{t}_y}{\partial P_y} = \frac{1}{A_h}, \quad \frac{\partial \hat{t}_w}{\partial P_w} = \frac{1}{P_w}, \quad \frac{\partial \hat{t}_h}{\partial P_h} = \frac{1}{P_h} \right\} \tag{B.4}$$

$$\frac{\partial P}{\partial a^l} = \left\{ \frac{\partial P_x}{\partial a_x^l} = A_w, \quad \frac{\partial P_y}{\partial a_y^l} = A_h, \quad \frac{\partial P_w}{\partial a_w^l} = A_w e^{a_w^l}, \quad \frac{\partial P_h}{\partial a_h^l} = A_h e^{a_h^l} \right\} \tag{B.5}$$

Appendix C

Tensorflow

TensorFlow is an open source library that ease the implementation of machine learning models. It was firstly launched by Google, in 2017, and nowadays is used worldwide.

TensorFlow works with multi-dimensional arrays, commonly known as tensors, that, as the name suggest, are passed end-to-end through a flowchart, known as computational graph. Each graph gathers a set of numerical operations to be applied to the input tensor. The usage of graphs allows to find out non-dependent operations, which means multiple computations can be executed in parallel and workload can be distributed across several workers (GPUs, CPUs or TPUs) and/or machines. Furthermore, graphs are language-independent, allowing to save and restore models in different code languages, always preserving the computations. This property adds flexibility and portability to TensorFlow. Moreover, another advantage of Tensorflow is the kit tools provided, which enables a developer to construct models at any level of abstraction. To better understand this levels, let us consider the TensorFlow toolkit hierarchy present in figure C.1.

All 3 levels of abstraction are accomplished via Application Programming Interface (API), responsible for establishing the connection between the developer and the TensorFlow core. At high-level API, the developer has the freedom to choose a canned estimator, either from Estimators or Keras model libraries. Canned estimators are pre-built models, meaning the developer cannot change its architecture, hence this level presents the lowest flexibility. If one desires to built a custom model, mid-level API allows to do it in a simple way. By providing libraries with methods to built common components from neural networks, developers can customize easily a network layers configuration, dataset preprocessing and input queuing, or even the loss function and the metrics to be used. Following, at low-level API, the developer can use a preference code interface to make changes in the computational graph. Even though the variety of interfaces, the most used is Python, as it shows to be implemented more efficiently. Furthermore, and looking deeper into the hierarchy, all languages are wrapped into a main code language, C++. At this

level, the list of numerical operations are implemented in a computational graph. Although this languages give flexibility for the developer, they represent a low-level API, hence are rarely modify. Additionally, an execution engine uses kernels to translate and execute the graphs created in upper levels in different machine types.

To sum up, TensorFlow is a great library to use when implementing deep learning, providing the necessary tools to do it without spending large work hours coding.

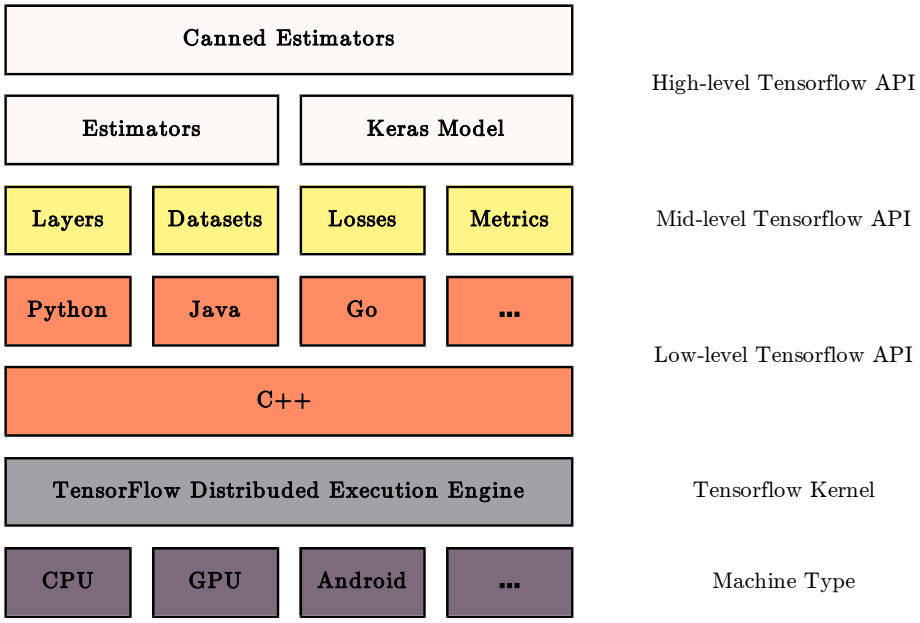


Figure C.1: TensorFlow toolkit hierarchy

Appendix D

Damages Examples

D.1 Erosion

Figure D.1: Examples of Erosion

D.2 Peeling

Figure D.2: Examples of peeling

D.3 Crack

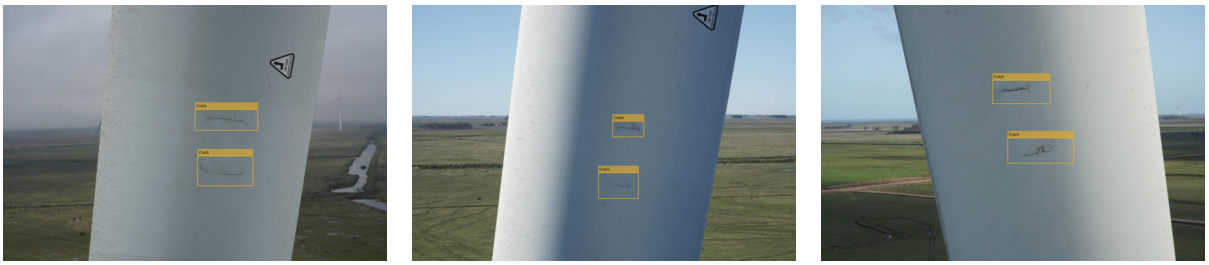


Figure D.3: Examples of cracks

D.4 Fungi

Figure D.4: Examples of fungi

D.5 Lightning Strike

Figure D.5: Examples of LS

D.6 Lightning Receptor Damaged

Figure D.6: Examples of LRD