UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Study of aircraft damage detection with AI

Document:

Report

Author:

Laura Gonzalez Huguet

Director:

Oriol Lordan Gonzalez

Degree:

Màster Universitari en Enginyeria Aeronàutica

Call:

Spring, year 2023.

Abstract

Durante los últimos años la Inteligencia Artificial ha ido ganando protagonismo en nuestras vidas, estando cada vez más presente en nuestro día a día. Asimismo las industrias están adoptando este tipo de tecnologías con el objetivo de aumentar la automatización y optimizar procesos y, en consecuencia, costes.

El presente estudio pretende acercar estos avances al mundo del mantenimiento aeronáutico analizando la viabilidad de desarrollar un algoritmo capaz de determinar si las imágenes de aviones que se le muestran contienen daños estructurales o no. Además, el objetivo es conseguir que pueda diferenciar entre los tipos de daños más comunes en una flota de una aerolínea comercial. Así pues, se quieren sentar las bases para que las inspecciones rutinarias puedan pasar de durar múltiples horas a tan solo unos minutos.

Para ello se emplean técnicas de Machine Learning y se desarrolla un código en R. En concreto se usan las librerías TensorFlow y Keras, y RStudio como IDE. Por otro lado, se estudian diferentes arquitecturas, con el foco en las redes neuronales convolucionales (CNN), y se realizan múltiples pruebas con ellas para ver cuales alcanzan unas mayores prestaciones y porque.

El resultado de este proyecto es un modelo capaz de distinguir entre estructura sana y dañada, pero no entre tipos de daños. Esto se debe principalmente al hecho de que a mayor número de categorías entre las que clasificar, mayor debe ser el número de imágenes con el que se entrena la IA. En este caso en particular se cuenta con 1268 fotografías, que incluyen tanto distintos tipos de defectos como estructura no dañada.

Una vez definido el modelo final, se desarrolla una aplicación sencilla para que la interacción con el usuario sea lo más fluida posible, **AirScan**. Así se dan los primeros pasos hacia posteriores estadios de desarrollo y testeo en los que el sistema sería probado en un entorno de trabajo real con imágenes totalmente nuevas.

Abstract

During the last few years Artificial Intelligence has gained the public's attention by being more present in our daily lives. Likewise, industries are taking on this kind of technology with the aim of increasing automation and optimizing processes, ultimately reducing costs.

The present study intends to bring these new systems closer to the world of aircraft maintenance, and therefore studies the feasibility of developing an algorithm that can successfully determine whether the images of aircraft given as input show damages or not. Furthermore, the idea is that it should also be capable of differentiating between the most common damage types within the fleet of a commercial airline. As a result, the foundations will be set so that routine aircraft inspections can be cut down from multiple hours to several minutes.

For this purpose, Machine Learning techniques will be employed and a code in R will be developed. To be more specific, it will be carried out by means of the TensorFlow and Keras libraries, and RStudio as the IDE. On the other hand, different architectures and ways of stacking layers will be studied, with the focus set on Convolutional Neural Networks (CNN). Several tests will be performed with them in order to see which ones can achieve a better performance and why.

The outcome of this project is a model capable of distinguishing between healthy and damaged structures, but not between damage types. This is mainly due to the fact that the higher the number of classes among which the classification has to be done, the more images are required for training the AI. In this case, the database comprises 1268 pictures, which include different types of defects as well as undamaged structure.

Once the final model is defined, a simple app, **AirScan**, is developed to enhance the user interaction. Therefore the first steps are taken for further development and testing stages, in which the system would be tested in a real work environment with completely new images.

Index

List of figures

List of tables

Abbreviations list

AI: Artificial Intelligence

AMM: Aircraft Maintenance Manual

ATA: Air Transport Association

CNN: Convolutional Neural Network

EFB: Electronic Flight Bag

ML: Machine Learning

MRO: Maintenance, Repair and Ovehaul

SRM: Structural Repair Manual

Chapter 1

Introduction

1.1 Objective

The aim of the present study is to assess the feasibility of developing an Artificial Intelligence (AI) capable of detecting aircraft damages. For that purpose, research in several fields will be performed including Machine Learning and aircraft maintenance processes. For an easier approach, the problem is going to be broken down into two main parts: image collection and code development. Once a suitable solution is reached, the model will be exported to a simple user-friendly interface so that it can be easily tested with new pictures.

To achieve the first objective, image gathering, extensive research has to be done in order to create an adequate database for the model's training, validation and test. Additionally, all pictures have to be correctly processed and labeled, and the ultimate goal is that the AI is capable of distinguishing not only whether there is a damage or not, but it can also tell which kind of damage it's looking at.

As part of the preliminary research, the market is going to be checked to see if airlines or maintenance providers are already implementing or studying any kind of technology like the one under assessment in this project. This will help to better understand if there would be a real interest in systems of such kind, and also if the actual implementation would be feasible when the time is right.

The second step is based on an analysis of the different Machine Learning techniques. All the basics are studied as foundations for the model that is being developed in this project, so as to build up an algorithm that can successfully perform the intended task. The goal is to find out which are the most used architectures and techniques for computer vision problems, and adapt them for the problem in hand by understanding how changes in the code affect the overall output.

Since the resulting code is developed in R, knowledge will have to be acquired in that field as well, with the focus in machine learning libraries. In that regard, existing codes from books, papers and other sources will be analysed to understand what works best for which tasks and why.

All in all, the aim of this project is to prove whether classifying images of aircraft damages is possible or not,

and if so, to develop a model and deploy it in a small-scale app. To achieve that, a database of pictures will be built and a model will be developed with the most suitable Machine Learning techniques.

1.2 Scope

In order to achieve the final solution, the following deliverables are required:

- A data base with multiple images for each of the considered classes, namely dent, burn mark, scratch, other and no damage. There should be at least a few hundreds of pictures for each class. Note that only the outer structure of the aircraft is considered, that is outer fuselage, wings, vertical and horizontal stabilizer, engine cowls and the nose radome.
- A code developed in R for image pre-processing and model training.
- A user interface that contains the final weights of the model. The user will provide pictures and the app will output its predictions. The actual implementation of the model in a production environment falls out of the scope of the project.
- All the algorithms will be run by means of a laptop with an Intel Core I7 processor. Therefore, the computation time and power will be restricted to what this laptop can provide.

1.3 Requirements

The requirements that the project should meet in order to be considered successful, are the ones listed below.

- **RQ 1:** A detailed study shall be performed regarding AI and the different Machine Learning techniques available.
- **RQ 2:** Research in aircraft maintenance procedures shall be done in order to understand the possible interest of the industry in a technology like the one developed in this project
- **RQ 3:** A database of aircraft pictures shall be built. It should contain both images of damaged and undamaged structure and enough pictures to train a model.
- **RQ 4:** The final model should at least differentiate damaged and healthy structure. If possible, it should also distinguish damage types.
- **RQ 5:** An interface shall be developed for users to interact with the model. The interface should be intuitive and easy to handle.
- **RQ 6:** All code will be written in R with RStudio as the IDE.
- **RQ 7:** The code shall be efficient and avoid redundancy to optimise the computational cost.
- **RQ 8:** All code will be run with a laptop and thus the computational power needed shall remain within its capabilities.

1.4 Justification

Even though the aerospace industry, especially commercial aviation, is known for being slow in the implementation of new technologies, mainly due to certification requirements, in the recent years the interest in the field of automation has increased notably. From smart hangars to substituting paper by tablets for technicians [1], the sector is moving towards innovation.

As a result of that, a new concept has emerged: **MRO 4.0**. Breaking down this idea, MRO stands for Maintenance, Repair and Overhaul, and it designates the companies and organisations that work on aircraft maintenance and fall under Part 145 of Regulation (EU) No 1321/2014 [2]. On the other hand, the 4.0 is a reference to the industry 4.0, or what is also known as the fourth industrial revolution. This term encompasses all the recent breakthroughs in terms of robotics, Artificial Intelligence, 3D printing and autonomous vehicles among many others.

For example, the transition to a fully digital and paperless environment has already begun and its closer to its total fulfillment than ever. That's why pilots from Vueling, Iberia, British Airways and many other airlines use the Electronic Flight Bag (EFB), a tablet-like device with many of the manuals, navigational charts and other documents that used to be carried on board in paper format. The EFB has helped reduce costs not only by cutting down the amount of paper required, but also by reducing the weight on board to the equivalent of one person. If all flights around a year are considered, this leads to a noticeable decrease in fuel usage, one of the main costs of an airline.

When applied to aircraft maintenance, the idea of industry 4.0, and thus MRO 4.0, includes data-based technologies as well as augmented and virtual reality to name a few. The ultimate objective of such innovations is to optimise all processes involved in making sure that an aircraft is airworthy and thus can fly safely. This will finally result in a decrease of the amount of time that aircraft spend on ground and an increase in cost savings.

As an example of that, Airbus' Aircraft Maintenance Manual (AMM) considers now drones as an alternative method of inspection after a lightning strike. After such a meteorological event, the standard procedure is for technicians to inspect the entire airplane in order to find burn marks and perform a temporary or permanent repair on them. For an A320 this task can take several hours and requires the use of lifting platforms, but the magnitude of the job quickly scalates for aircraft of the size of an A350 or even an A380.

Following this line of work, airlines like EasyJet, KLM or LATAM have their own team of engineers dedicated to study the feasibility of having an in-house drone inspection service, and eventually to the deployment of this kind of technologies.

Coming back to this study, its objectives are aligned with this new tendency in the sense that it aims at further automation of processes. If step one would be to have a drone or camera system capable of taking pictures of those parts of the aircraft that are out of the engineer's reach, this project aims at taking a further step and letting the computer be the one to check the images and determine whether there's a damage or not.

The fact that a drone can perform an inspection certainly saves time and prevents risks, as technicians don't

need to be working in heights for this task. Nevertheless, it can be further improved if the system itself can take pictures of the entire aircraft, process them in a short amount of time, and tell the technicians and engineers the exact location, amount and type of damages.

Another advantage of the implementation of such an AI would be the reduction of human error, the enhancement of damage reporting and the possibility for the engineers to have real-time information from their office.

What's more, if the system is capable of distinguishing damage types, a further improvement would be to link each one to their corresponding chapter in the Structural Repair Manual (SRM) and giving as an output the reference to a possible repair.

Chapter 2

Background and state of the art

Although Machine Learning techniques aren't new, as the earliest models are dating back to the 1950s, over the last few years this kind of technology has undergone a significant development and has widely attracted the public's attention. Day by day, they are more present in our daily lives in the form of facial recognition when tagging someone in social media, or in search algorithms, chatbots and even navigation apps.

The following sections are a brief revision of the evolution of ML together with a study of the state of the art and the degree of implementation of these systems for aircraft damage detection. This will serve as a starting point for the present study, because the intention is to understand which architectures have been used through the years for Machine Learning models, and image processing in particular.

Additionally, several techniques to perform aircraft inspections will be outlined to understand the position in which aircraft maintenance stands nowadays, and which path has it followed to get there.

2.1 The evolution of Machine Learning throughout the years

As stated previously, some early forms of ML like probabilistic modeling go back to the 1950s. Early versions of neural networks are also from the 1950s, although it is during the 1980s when they really take off with the use of back-propagation algorithms.

The back-propagation algorithm is a gradient-based optimizer which computes the loss of a model as the mismatch between the predictions and the true values, and updates the weights of said model by aiming at reducing the loss. It's the central algorithm in deep learning and the foundation of the improvements that came later on.

In the 1990s kernel methods replaced the neural networks as a new approach to machine learning. Within this set of classification algorithms, the best known is the Support Vector Machine (SVM), although later on this method proved to be difficult to implement with larger data sets, and therefore harder to scale. Additionally, it was shown that they didn't perform well on image classification problems.

During the early 2000s, other ML models like decision trees, random forests or gradient-boosting machines

were the most popular, although a few years later researchers began to obtain promising results with neural networks, which at the time were forgotten by most of the community. Ever since 2012 deep convolutional neural networks have become the most used architectures for computer vision problems along with back-propagation [3].

The main cause of back-propagation and CNN's comeback can be found in the technological progress that led to a higher computational power and from there to deeper neural networks that, with a higher number of layers, are capable of learning more complex parameters. All in all, from computer vision tasks to natural language processing, they have replaced other types of ML thanks to their better performance and extreme scalability.

2.2 Technologies for aircraft damage detection

Every time an aircraft undergoes a potentially harmful situation it has to be thoroughly inspected to detect any damages that may have occurred. These situations range from a hail storm to a lightning strike, considering as well bird strikes or even on-ground collisions with other equipment or vehicles.

While in some cases the harm is very clear and enclosed, like for example when the passenger stairs collide with the fuselage, some other times the post-occurrence inspections need to cover a larger portion of the airplane.

It's in those latter cases when technicians may spend a large amount of hours to complete the examination, which leads to aircraft being stopped for days and thus costing the airline a lot of money. The main reason behind this fact is that they perform the inspections by themselves with the only help of a lifting platform, which also needs to be relocated several times during the process.

Another example of long lasting inspections are those performed when an aircraft enters the hangar for a heavy check, which is when they are stopped for several days or weeks to perform various maintenance tasks. The technicians cover the entire aircraft in order to find new defects and repair them before it goes back into service, which again is a costly task in terms of time.

Nevertheless, although most MROs perform inspections without any additional technology, there are several on-going projects that aim at optimising this process. For instance, Sanz et al. back in 2010 proposed a robotic arm with ultrasound sensors at its end, to perform crack inspections on an aircraft's fuselage [4]. Following this line of work, Wang et al. worked on a platform that could be placed directly on the aircraft with the same purpose [5]. Nevertheless, note that both these projects aimed at performing deeper inspections than the ones covered by this study.

Coming back to visual inspections, and also when focusing on a more commercial aspect rather than academic, several major airlines have been exploring new ways of carrying them out. Since 2022, LATAM has been employing drones for this task at its MRO facilities in Brazil, which has helped them reduce the time spent down to 40 minutes, when it used to take 8 hours and two technicians [6]. In particular, the drone they have been testing allegedly can take around 2000 detailed pictures of the fuselage, which are later analysed.

In conclusion, although there is a rising interest in technologies and systems that can enhance aircraft inspections, their use hasn't been widely extended yet. Most of the airlines that have the resources are still at early stages of testing and, while it's very likely that ML will have more presence in aircraft maintenance processes, it's still soon to see them implemented to their full potential.

Chapter 3

Methodology

The problem that this study is facing, to develop an algorithm that can discern images of damaged and healthy aircraft, can be broken down into two main parts: building a database and developing the model.

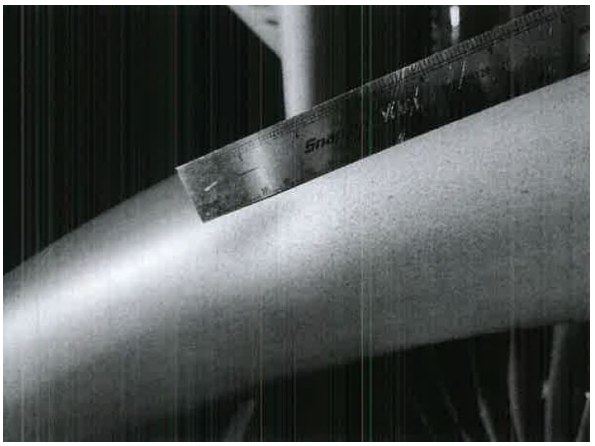
This chapter provides a detailed explanation of the processes carried out to perform these two tasks, along with all the necessary theoretical concepts that have been learned along the way.

3.1 Image gathering and processing

The first, and probably most arduous work, is to search and collect as many images as possible of damaged and undamaged aircraft. The resulting ensemble will be the database upon which the model is going to be trained, validated and tested. Note that the final amount of images might determine the overall success, as not having enough of them might mean that the problem can't be solved. As a rule of thumb, a few hundreds of images for each class are needed, although the more categories there are, the higher amount of images per each are required.

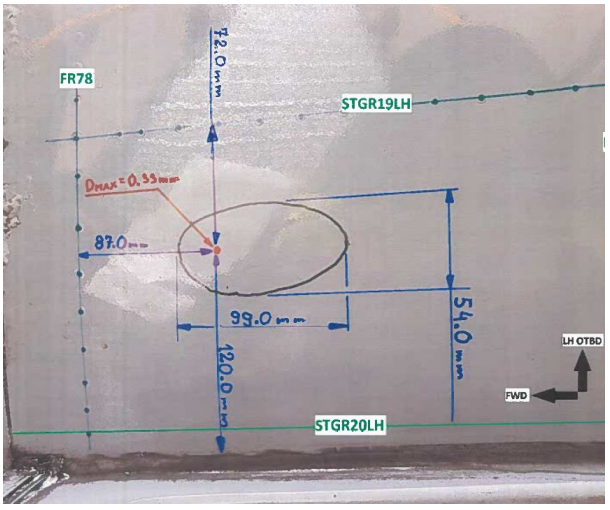
The pictures that make up the final dataset have been carefully selected from damage reports of a fleet of approximately 125 aircraft composed by Airbus' A320 family (A319, A320 and A321). Since the model needs to learn from them, they have to be clear and representative of each class, which is why a notable amount of them have been discarded. Among the problems encountered while searching for suitable pictures, the following have been the most remarkable:

- **Poor quality.** Sometimes the pictures received are blurry, in black and white or have been printed out and then scanned again, which makes it hard to get a good understanding of the situation. For instance, in the image shown in figure 3.1 the human eye can tell that there's a dent on the lip skin, but these kinds of pictures aren't the best for training a model, as they can be misleading. Additionally, the code developed for this project considers images in colour, and thus black and white images are discarded to avoid possible issues.



Figures 3.1: Example of a discarded picture due to poor quality

- **Not representative of reality.** A clear example of this statement is shown in figure 3.2. In it one can see a dent that has already been assessed, and has all kinds of annotations and measurements. While this is useful for an engineer to compare the damage with the maintenance manuals, an AI might learn to identify a dent by the circle around it, not by its actual shape or shadow, and thus it should be avoided.



Figures 3.2: Example of a discarded picture due to lack of representativeness

Before deciding which damage categories should be considered, the starting point of this process was a brief research meant to determine the most common damages discovered on this fleet. Table 3.1 shows the 5 types of damage found to be more usual, as percentages of the overall number of damages.

Table 3.1: Top 5 most common damages

Type of damage	%
Dent	20.68
Burn Mark	17.84
Crack	13.98
Corrosion	9.76
Scratch	6.78

Note that, although the percentage of dents is higher than burn marks, in the final data set there are more images of the latter. That is because many of the pictures of dents were like the one in figure 3.2, and therefore discarded.

Another matter to take into consideration is that cracks and corrosion usually affect the internal structure rather than the outer fuselage. For this reason, they fall out of the scope of the project and haven't been considered. When encountered, pictures of these kinds of damages on the outer structure have been labeled as "other".

While the images were being chosen, they were also being labeled according to the category (or class) they belong to, namely dents, scratches, burn marks, other damages and undamaged structure. In total there are 5 classes for the model to distinguish between, although note that only 3 of them designate actual types of damage. That's because they are the most common and the amount of pictures of them is higher. Nevertheless, there's an additional category for other types which still allows to detect any other damage outside of these main 3 categories. Lastly, there's a class called "no damage" intentionally defined so that the model can also learn to identify aircraft in good condition.

All in all, the final data base consists of 1268 images distributed among each category as shown in table 3.2.

Table 3.2: Final data set segregated by category

Type of damage	**Number of pictures**
Burn Mark	404
Scratch	156
No damage	315
Other	253
Dent	140

3.1.1 Image pre-processing and data augmentation techniques

Once all the pictures are collected and correctly labeled, they have to be pre-processed before they can be fed to the neural network. At this stage they all have different sizes and are in different formats, so it needs to be ensured that all the input matrices share the same characteristics and are homogeneous.

For that purpose, several functions from the Keras library will be implemented. First they have to be decoded and transformed from the usual image format (.jpg, .png) to RGB values. Then, they have to be resized so that all of them have the same dimensions, 180x180 in this case. Lastly the matrices will be normalized by dividing them by 255, which will make them take values between 0 and 1. As a result, since the numbers are smaller, the computations become faster and easier to handle. Once all the described is accomplished, the resulting RGB matrices of pixels are going to be the model's input.

Taking into consideration that the more images there are, the more likely will be the model to perform better, the possibility of enlarging the database has to be considered. Furthermore, knowing that the available number of pictures for each one of the five classes defined is rather low, several techniques are going to be studied in order to try to enhance the overall performance.

In particular, these kinds of methods are known as data augmentation. It basically consists of artificially expanding a database by manipulating the images. What follows is a list of the techniques applied for this project.

- **Rotation:** Pictures are rotated a random angle within a given interval. To be more specific, this project considers a 10%, and therefore a range between -36º and +36º.
- **Flip:** Images are flipped both horizontally and vertically.
- **Zoom:** Similarly to rotation, a random zoom within a defined range is applied. Here a 0.2 factor is considered, which translates into values between -20% and + 20%.

Although it will be later explained with further detail, note that data augmentation is only applied during the training stage. That's because these techniques don't really provide new information, but rather a new standpoint. When employed correctly they can help the model learn new patterns, but too much of them may lead to redundancy of data and lack of generalisation.

3.2 Developing a model for image classification

Once the data set is prepared, the second part of the problem needs to be approached. Research shows that deep convolutional neural networks are the models that can achieve the best results in computer vision problems. In fact, ever since 2012 this type of architectures have won the ImageNet Large Scale Visual Recognition Challenge, which is a well-known competition that consists on classifying colour images into 1000 different categories after training with 1.4 million images [7].

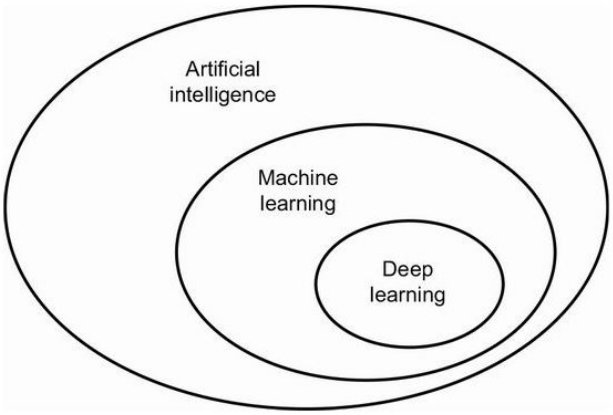
In the following sections, the fundamentals of deep convolutional neural networks are analysed in order to understand how they work and which are the features that make them perform better than other forms of

ML. Additionally, research is done on the existing tools that can be used to develop the code for the model in R.

3.2.1 Understanding Neural Networks and Deep Learning

So far it has been established that the most promising model to solve the problem under study in this project are Deep Convolutional Neural Networks, but what does that really mean?

On one hand, Deep Learning is a sub-field of Machine Learning, just like ML is a field within AI. The main objective of this discipline is to generate rules that allow computers to "learn" when they are given a set of data and its associated expected output.



Figures 3.3: Relationship between AI, ML and DL. Source: [3]

The concept of "deep" mainly stands for the idea that these models are made of successive layers. In fact, the amount of layers within a model is known as its depth. As for the term Neural Network, it refers to the overall assembly of layers.

The idea of creating an architecture by stacking several layers is based on the fact that each of them learns different information from the provided data. This is done by means of transforming the input into representations that at every step differ more from the original, but that contain more meaningful information for the model.

The way that a layer is capable of retaining the knowledge is through its weights, which are essentially a set of parameters. The weights are updated at every iteration in order to find the combination that is able to perform better, which is why learning in this context means to find the best set of weights.

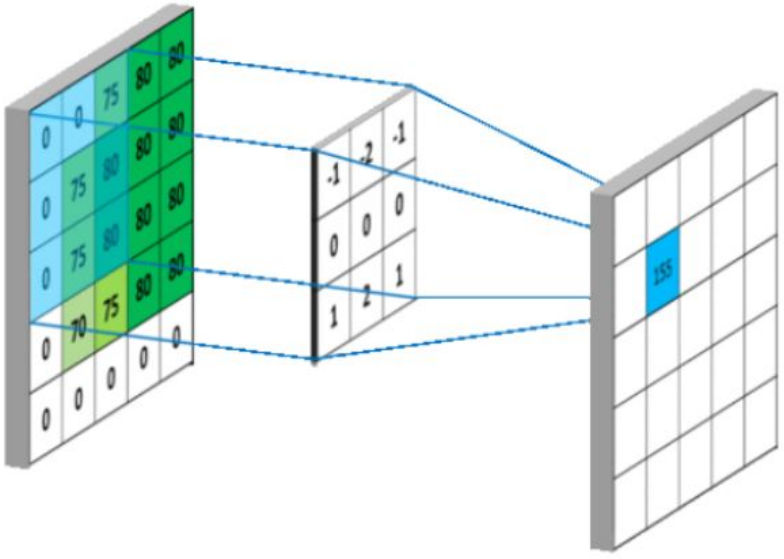
In order for the model to know how well it is performing, or whether a set of weights is working properly or not, it needs to be able to evaluate how far its predictions are from the expected outputs. For that purpose the loss function is defined. It's also known as the cost function or the objective function, and its mission is to compute a score that evaluates how different are the model's outputs from the truth. As it will be shown later on, different loss values may have different meanings depending on the loss function that is chosen.

Once the loss is computed, its value is used to adjust the weights through an optimizer, so that the loss is minimized. This is implemented through the back-propagation algorithm, which is the core of deep learning.

The process of setting weights, evaluating the loss and optimizing, is repeated as much as needed depending on the problem to be solved, the amount of data available and the architecture of the model [3].

3.2.2 Understanding Convolutional Neural Networks

Mathematically, a convolution is an operation of two functions that result into a third one that represents how the shape of one is modified by the other. When applied to matrices, a convolution consists of sliding a small matrix, called kernel or filter, over a bigger one, which is the matrix of pixels from the input data, and performing the summation of the element-wise multiplication, which will be stored in the output matrix.



Figures 3.4: Visual representation of matrix convolution. Source: [8]

Convolutions are performed over tensors of rank 3, which are called feature maps. They have 2 spatial axes, corresponding to height and width, and a depth axis, also called the channels axis. In this case, since the images considered are in colour (RGB), the depth axis is of size 3, standing for red, green and blue.

In figure 3.4 note that the size of the output matrix will be smaller than that of the input matrix. That's because the kernel can't be centered in the outer rows and columns. In order to address this issue, a technique called padding can be implemented. It consists of adding an extra row and column of zeros at each side, so that the center of the kernel can go over the entire input image.

Convnets have two main characteristics that make them more useful than other models for computer vision problems. The first one is that once they learn a pattern they can recognize it anywhere within a picture. Therefore they need less training samples and have more generalization power, as otherwise they would need to see the same patterns in all possible locations during training.

The other important feature is that they can learn spatial hierarchies of patterns. The first layers will learn small local patterns like edges, while deeper layers will learn more complex patterns made of those learned in earlier stages.

3.2.3 Building the model: an overview of the different layers

Although the backbone of convolutional networks are the convolutional layers, there are many other types of layers that, combined, make image classification tasks possible for a computer. Depending on how they are stacked and how many of each there are within a model, many different architectures can be developed. Note that there are many possibilities and that each of them will be suitable for different problems, mainly subject to their complexity and the amount of data available for training.

The following list is a compilation of the most commonly used layers, which are also the ones that will be implemented in the models developed for this project.

- **Input layer:** Its main purpose is to instantiate a Keras tensor, and thus define the shape that the input matrices will have [9].
- **Convolutional layer:** It's the one in charge of performing the convolution operation as explained before. To do so, it takes several arguments: the number of filters, the size of the kernel and the activation function among others [10].
- **Pooling layer:** Pooling is a technique that merges pixel regions after the convolution is done, and gives as a result a smaller matrix. In particular max pooling is considered here, which means that the output matrix from a convolutional layer is shrunk to half its size by taking groups of pixels and obtaining their maximum value, which is then stored in the new matrix. By doing this, the number of coefficients to process is decreased and the subsequent convolution layers are able to look at larger portions of the initial picture, as only the features with more presence remain [11].
- **Dropout layer:** This layer implements dropout, which is one of the most common techniques to avoid overfitting (check subsection 3.2.4). It takes a number between 0 and 1 as input, which is the fraction of nodes that will be randomly set to zero. The usual values for this ratio range between 0.2 and 0.5. Above 0.5 the loss of information is too aggressive for the network to perform well [12].
- **Flatten layer:** It converts matrices into one-dimensional arrays. This layer is usually implemented towards the end of a model with the objective of obtaining as output a vector with the probabilities for an image to belong to each class [13].
- **Dense layer:** Also known as a fully-connected layer, it is usually the last layer within the architecture. Mathematically, it performs a matrix-vector multiplication [14].
- **Custom layers:** This category encompasses all data transformations that aren't directly called from Keras. In particular, for this project only one custom layer has been implemented for data augmentation, which features the techniques explained in section 3.1.1.

Among all the arguments that a layer may take, one of the most relevant is the activation. It helps to define the transformations that will shape the input into the output and assists the network in learning complex patterns. Their importance mainly lies in their ability of adding non-linearities to the CNN. Although there are a few different activations, this project will consider two: *ReLu* and *Softmax*.

ReLu stands for Rectified Linear Unit and it is probably the most used in CNNs. In fact, all convolutional layers implemented in this project will have a *ReLu* activation. Mathematically it can be expressed as follows:

$$f(x) = max(0, x) \tag{3.1}$$

Therefore, all negative values will be zero and only the positive ones will remain. Among its advantages there is the fact that it's very easy to calculate and thus the computing times won't increase because of it.

The second activation, *Softmax*, is used in classification models. In particular, it is used only in their final layer, which is the one that produces the probabilities of being in each class. It takes values between 0 and 1, which correspond to the percentage of chances of an image belonging to each one of the classes considered.

3.2.4 The loss function and its relation to overfitting

One of the most important things to define when developing a ML model is the mechanism that will be established to determine whether it has reached its optimal performance or not. As stated previously, this is done by means of the loss function. Choosing the most appropriate one is key for the success of the model, as it will try to minimize the loss by all means possible and, if not chosen well, the model may end up doing things that don't fully correlate with the objectives of the task.

In the case of image classification, since it has been widely studied, there are a handful of loss functions that have proven to work properly [3]. In particular, for multi-class classification problems categorical cross-entropy is the one chosen.

Equation 3.2 shows the mathematical implementation of categorical cross-entropy. In it, t_i is the real score for each class i. It's a binary parameter that has a value of 1 when the image belongs to the class i, and 0 when it doesn't. As for s_i it's the probability of belonging to each class i, computed by the model.

$$CE = -\sum_{i} t_i \cdot ln(s_i) \tag{3.2}$$

For example, when an image A belongs to a certain class in i, t_i will be 1. If the model is able to make a prediction that says that there is a 80% of probability of image A belonging to said class, s_i will be 0.8 and the resulting loss score will be computed as follows:

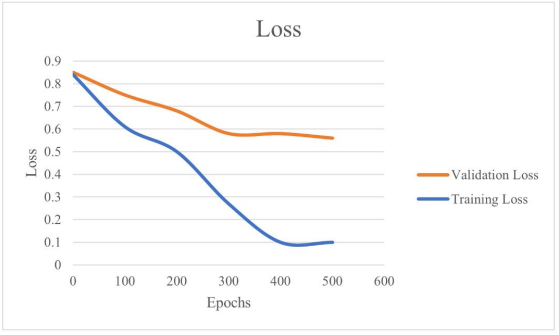
$$CE = -\sum_{i} t_i \cdot ln(s_i) = -1 \cdot ln(0.8) = 0.223 \tag{3.3}$$

On the other hand, if the model's prediction for the same image and class was a 10% of probability, the resulting loss would be:

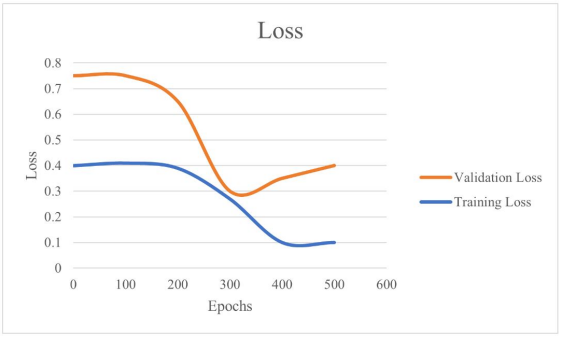
$$CE = -\sum_i t_i \cdot ln(s_i) = -1 \cdot ln(0.1) = 2.302 \tag{3.4}$$

In order to evaluate the model, it's a common practice to establish 1 as a threshold to say that the model isn't performing well and there's something wrong with it. That's because a loss score of 1 would correspond to a probability of approximately 35% which could be translated as the model failing to provide accurate predictions two thirds of the times.

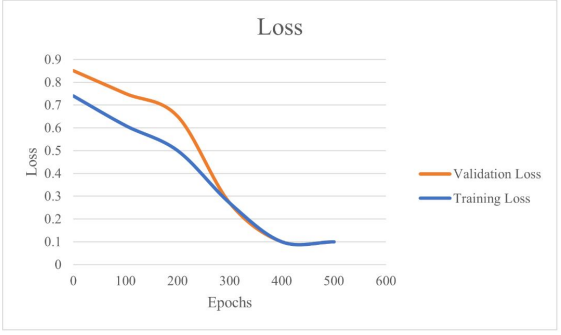
Besides assessing the model's performance, the loss value can be also used to determine whether more iterations (or epochs) are needed for training or not. Figure 3.5 shows 3 different scenarios according to the evolution of the loss score.



(a) Underfitting



(b) Overfitting



(c) Optimal fitting

Figures 3.5: Evolution of the loss value with iterations. Source: [15]

Underfitting as shown in figure 3.5a happens when the validation loss (orange) is much higher than the training loss (blue) and they have never crossed paths, which means that the CNN can't accurately model the input data. This issue could be fixed with more training, by increasing the complexity of the model (i.e. adding more layers) or reducing the dropout rate if implemented.

Another problem would be overfitting, which happens when the validation loss gradually decreases but then increases again. It usually means that the model can't generalize on new data, and it starts to perform too well only on the training data. That means that it isn't able to recognise general patterns but rather specific

images within the training set. Overfitting can be delayed in terms of epochs with a larger dataset, less epochs during training or more regularisation techniques such as dropout.

If the lowest loss value achieved is considered to yield a good enough performance, the training of the model could be stopped earlier to save the set of weights in that iteration. For that, accuracy would also need to be checked. Otherwise, it can be enhanced by adding more images to the database, by adding dropout or by reducing the complexity of the architecture.

3.2.5 Training a neural network

So far the terms training and validation have appeared several times, but what do they actually imply?

Once the architecture of a model and its parameters have been defined, the network has to be trained so that it can make predictions afterwards. In other words, it has to learn which patterns correspond to which category and it has to evaluate itself in order to understand if it's going in the right direction or not.

The process of finding a set of weights that leads to an appropriate performance according to the guidelines established, consists mainly of 3 stages: training, validation and test. As a consequence, the data set has to be divided into 3 smaller batches. Although there is no rule to determine the portion of images that should go into each batch, the highest amount should be for training as it's the central task. In this case, the original set of 1268 pictures has been divided so that 50% is for training, 20% for validation and 30% for test. Note that, while training and validation are closely related, testing happens at a later stage.

At every epoch the model learns from the training set, and at the end the accuracy and loss are evaluated on the validation set. The weights are updated with the goal of minimising the validation loss, because eventually the model will fit with high precision the training data and thus it will achieve very low loss scores during training.

Once all iterations are done, the model is evaluated on the test set. That's the closest case to a real-life scenario because the model is seeing this data for the first time. The metrics obtained here will be similar to what would be obtained in a production stage, which is why they are the most significant ones.

3.2.6 Keras and TensorFlow

So far the theoretical foundations that will enable the development of the model that will perform the damage classification have been set. Nevertheless, one pending topic remains, and that is the translation of all this theory into actual code.

All the code developed for this project will be written in R, an open-source programming language very popular among data scientists, with data analysis and visualization as its cornerstone. In particular, the TensorFlow and Keras libraries will be implemented. As for the IDE, RStudio is the one chosen.

TensorFlow is a library capable of performing multiple machine learning tasks, while Keras is a high-level neural network library that runs on top of TensorFlow. Although it is possible to work directly with TensorFlow, the use of Keras as an interface has been preferred because it is said to be easier to use and

also because it is focused on neural networks. Note that there are other frameworks available but these two libraries are the most used and well-known, and therefore the amount of documentation is higher.

Chapter 4

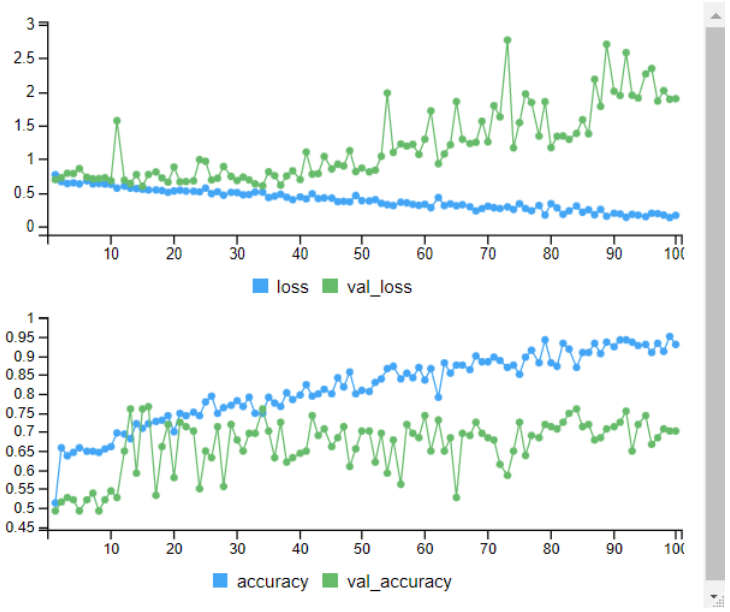
Initial approach and alternative solutions

Before developing an architecture for the model, it was considered to be a good option to take an existing model that already can successfully classify images, so as to get a sense of how well an AI would be able perform with the amount of images gathered. Nowadays there are many well known problems like the ImageNet competition mentioned earlier [7], or the MNIST database of hand-written numbers [16], but these data sets are significantly larger than the one built for this project. Therefore, the chosen model has been obtained from [3], and it shows a good performance on the "cats vs. dogs" classification problem, one of the firsts approaches to ML for new learners. To be more specific, it's able to achieve loss values of approximately 0.45 for validation, with an accuracy ranging between 80% and 85% and it shows overfitting after 60 epochs.

The very first attempt with this model to classify images into 5 categories, led to diverging loss values of the order of thousands already from early iterations. In order to address that, two different approaches are considered.

- **Classify into 2 categories instead of 5.** Since it is known that this model can perform well in binary classification problems, the perspective of the complete issue is changed. Instead of distinguishing types of damages, it will be asked to discern between damaged and undamaged aircraft. In that way, there will be more images available of the "damaged" class, as it will be a combination of all damage kinds considered before. This is still valuable information, as the engineers will know where to look for damages and won't have to inspect the entire airplane by themselves.
- **Adding data augmentation techniques.** Data augmentation consists of artificially adding more images to the training data set by modifying the currently existing. For example, several images are zoomed in, rotated and horizontally or vertically flipped, and introduced only to the training batch as new images.

Figure 4.1 shows the results obtained when classifying within 2 classes, applying data augmentation and training for 100 epochs. Although it can be seen that the model overfits already after a low number of iterations, the lowest value of loss achieved is around 0.7, which is a promising result.



Figures 4.1: First test, 2 classes with data augmentation

As a result of these early tests, the approach taken for this study shifts from considering 5 classes into considering also 2. As a result, all trials performed in the following chapters are carried out twice, once considering classification into 5 classes and then into 2 classes. Another consequence is that the training data set will always be enlarged by applying data augmentation techniques as described in section 3.1.1.

As a side note, it's important to highlight the fact that the database of pictures will be split into 3 sets as described in section 3.2.5: one for training (50%), another for validation (20%) and the last one for test (30%). Notice that data augmentation will be applied only to the training data set, as if applied to the validation or test ones it may tamper with the loss score used to assess the model's performance.

Chapter 5

Aircraft damage classification with convolutional networks

Taking as a starting point all the theory detailed in Chapter 3, the aim of the following sections is to define the most suitable architecture for the problem in hand. For that purpose, several combinations of layers are tested in order to see how they behave and how changing them affects the overall performance of the system.

In particular, the 4 following architectures are considered:

- **Model 1:** A linear classifier consisting of dense layers. Since it has been established that convolutional neural networks are the ones working best for image classification, this model is only tested as an experiment to check how different the results are. Although it has been proven to be useful in simple classification problems it's likely that it won't succeed with aircraft damage classification, but it should be an opportunity to learn how dense layers work.
- **Model 2:** A convolutional layer with relu activation followed by a dense layer. This model is the simplest convolutional neural network possible. The intention is to check if lowering to the minimum the amount of layers has a significant impact on the computation time and the performance of the system.
- **Model 3:** A convolutional layer with relu activation followed by a max pooling layer and a dense layer. In order to give more depth to the network, this structure is repeated twice to shape the architecture. Several variations of this model are also implemented so as to evaluate the effect that techniques like padding or dropout have on the overall performance.
- **Model 4:** This last model is the deepest one considered, and it is made of several repetitions of the structure [convolutional layer → convolutional layer → max pooling layer] followed by two dense layers. The idea is that with more layers the model should be able to learn more complex features in order to improve its performance, especially regarding the classification into 5 classes.

Before diving into the results, a few considerations need to be explained. For one, all test are performed for 100 epochs. In that way, one will be able to see the progression of the accuracy and loss scores, and will be able to determine whether the model overfits or underfits the data. After training, all tests are going to be performed with the weights from the best iteration in terms of validation loss. For that purpose, the algorithm will be configured so that it only saves the weights if they show an improvement. Additionally, only for comparison purposes, the accuracy and loss values from this best iteration in terms of validation will be stored.

When it comes to the training configuration, all models follow the same standard. The loss function is categorical crossentropy, and the optimizer is *RMSprop*, a gradient-based optimisation technique well-known for its use in neural networks.

Once all the models are tested, a final model is going to be assembled. It may be one of the existing models or a mix of them depending on the results obtained.

5.1 Model 1: Linear classifier

Model 1 is the simplest model considered, a linear classifier. As shown in figure 5.1, it mainly consists of two dense layers with a dropout layer in between.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(5, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.1: Model 1 architecture

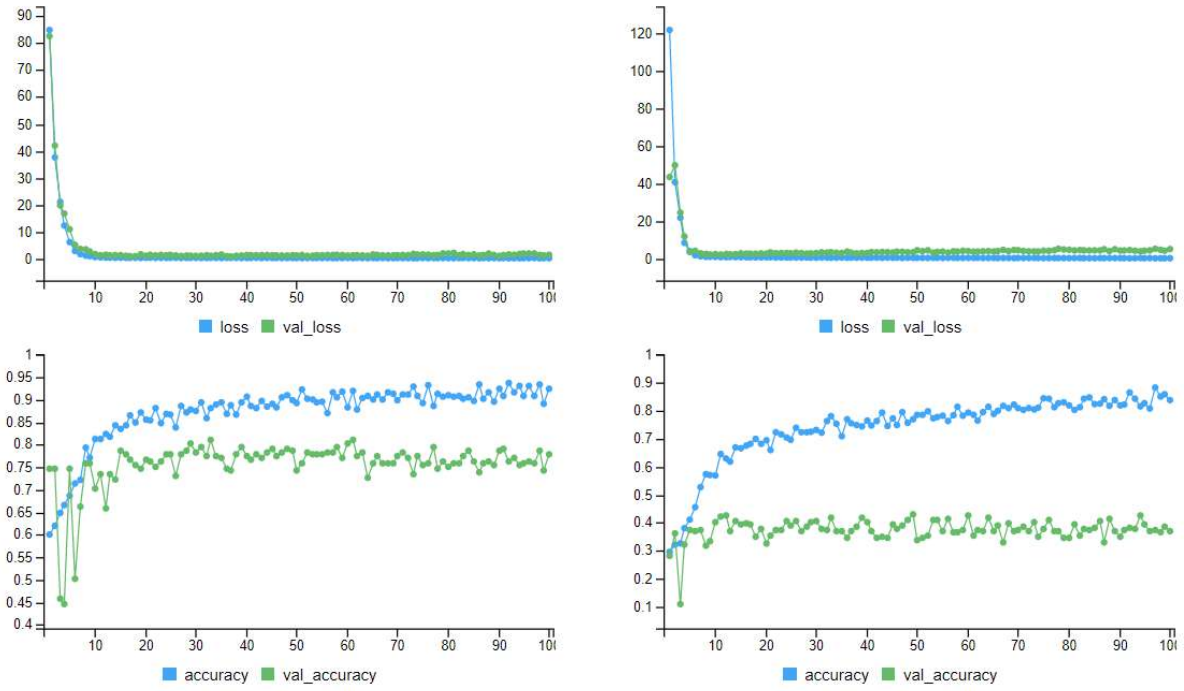
The first one is the input layer, which defines the size of the expected input matrices as established during the image pre-processing. Then the data augmentation techniques are applied and the values for each pixel are normalised as described in section 3.1.

What follows is a dense layer with 128 filters and *relu* activation. Note that there is no explicit rule to establish the number of filters. Since many of the codes analysed from Kaggle competitions and the examples in [3] have dense layers with 128, this has been the amount chosen for this first attempt.

This first dense layer is followed by a flatten layer, which will transform the data into a one-dimensional array. The next one is a dropout layer with a rate of 0.5 to randomly set half of the coefficients to zero and therefore try to delay overfitting as much as possible. Lastly, there's another dense layer with five filters and *softmax* activation. Here the number of filters has a purpose, because it has to match the number of classes considered for the classification. Therefore, it will be either 5 or 2 depending on whether the model is trying

to identify types of damages or only whether there's damage. This last layer will originate the final output vector of probabilities.

Figure 5.10 shows the evolution of accuracy and loss during training (blue) and validation (green) for both cases considered, classification into 2 and 5 classes.



(a) 2 classes

(b) 5 classes

Figures 5.2: Model 1, accuracy and loss evolution during training and validation

Although the high loss at the beginning alters the scale of the plots, it can be seen that the model overfits the training data within the 100 epochs considered. Therefore there's no need for further training and the model's best performance is achieved within this amount of iterations.

Note that, as could have been expected from the early attempt in Chapter 4, the validation accuracy is much higher for the 2-class classification problem than for the 5-class one (almost twice).

Table 5.1 shows the most significant results in order of importance. The test metrics are considered to be the most relevant because they show the behaviour of the model after training when facing new data, which would be the closest to a real-world situation. Then come the validation results for the best iteration, which shows the optimal performance of the model. If this was to be the chosen model, it would be retrained for the amount of epochs corresponding to these values as further training only leads to higher overfitting and less generalisation.

The last two scores, the ones from validation and training on epoch 100, aren't as important. On one

hand, training metrics aren't an accurate reflection of how the model would behave on new data because of overfitting, meaning that eventually it learns how to fit the training data accurately but can't generalise. On the other hand, validation at epoch 100 isn't the optimal but only a reference to understand the overall behaviour of the model.

Table 5.1: Accuracy and loss values achieved with Model 1

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.6684	1.0761	0.3395	2.2716
Validation (best)	0.743	0.8677	0.4217	2.3757
Validation	0.7791	1.4496	0.3695	5.2656
Training	0.9249	0.1759	0.8387	0.4293

From the values shown in table 5.1 one can conclude that a linear classifier isn't good enough for the problem in hand. That's because in both cases the loss during test is above 1, and thus more suitable models should be able to improve these scores. While it's true that maybe some parameters could be adjusted to try to enhance the performance, since research already shows convnets to be more adequate, it has been considered that spending more calculation time on this model is not worthy.

5.2 Model 2: A simple convolutional network

The second model studied is a simple implementation of a convolutional layer followed by a dense layer.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu") %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(5, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.3: Model 2 architecture

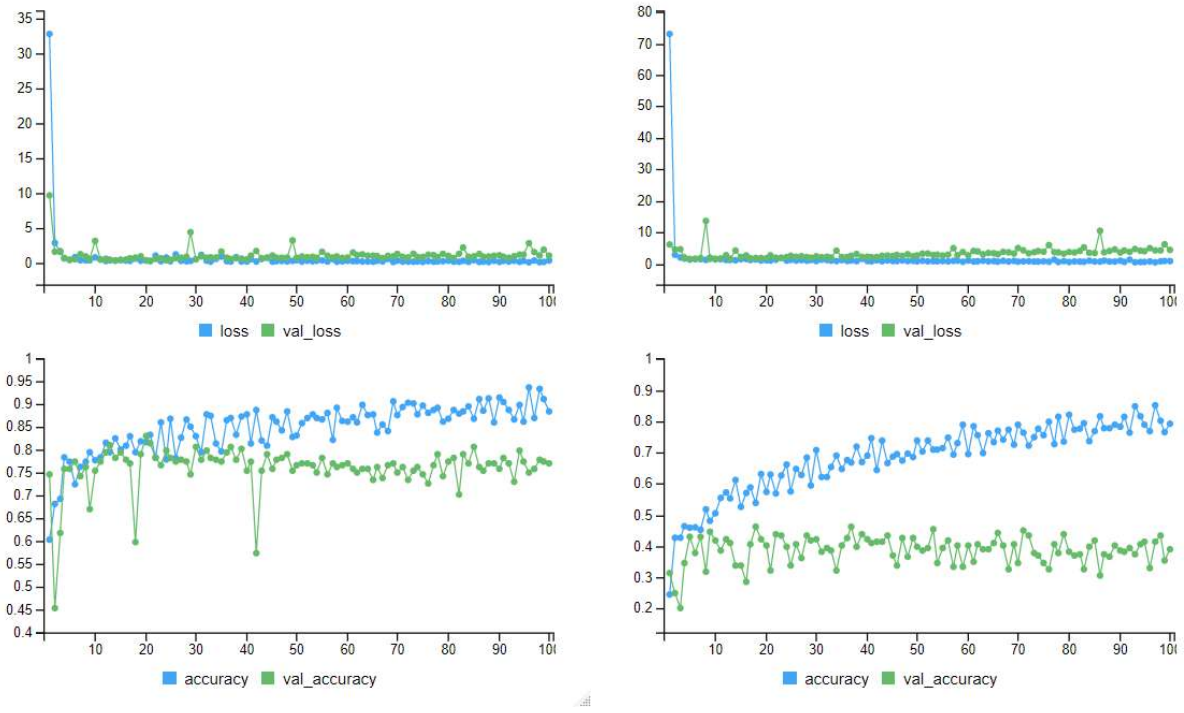
The input, data augmentation and rescaling layers are the same as for model 1. Then there's the convolutional layer with 32 filters and a kernel size of 3, meaning that the convolution over the input matrix is performed with a 3x3 matrix. What follows is a dense layer with 128 filters and *relu* activation, a flatten layer, dropout and the last dense layer with 5 or 2 filters and *softmax* activation.

Note that the last 4 layers of this model match the structure of the linear classifier previously seen. That's because, although the convolutional layer is in charge of extracting more meaningful and relevant information

(or features) from the input data, the model still needs a classifier in order to provide the required output, a vector of probabilities. That is why, from now on, all the models will have a very similar structure on the deepest layers.

Although it was previously stated that there are no written rules to establish the number of filters within a layer, there are exceptions to this statement. When considering the entire model, the number of filters should be consistently increasing with each layer because that enables the learning of more complex patterns.

Figure 5.4 shows the metrics of model 2 when trained during 100 epochs. In spite of the scale issues, it can be seen that validation and training loss drift apart for both cases, which could be an indication of overfitting. And again, accuracy for classification into 2 classes is significantly higher than for 5 classes.



(a) 2 classes

(b) 5 classes

Figures 5.4: Model 2, accuracy and loss evolution during training and validation

When looking at the scores in table 5.2, one can see meaningful improvements when comparing it to model 1. Even though there's still room for refinement, the 2-class case shows a loss value of 0.6 during test, and when considering the best iteration, the value descends to 0.39. On the downside, the 5-class case still shows very high loss scores, although they are almost half of those from the linear classifier.

Table 5.2: Accuracy and loss values achieved with Model 2

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.7427	0.6019	0.4164	1.5373
Validation (best)	0.8153	0.3903	0.4096	1.5813
Validation	0.7711	1.1300	0.3896	4.419
Training	0.8850	0.4257	0.7923	0.8546

5.3 Model 3: The combination of convolutions with the pooling operation

The last 2 models are more complex convnet architectures. In particular this one has two convolutional layers combined with max pooling layers. Furthermore, taking advantage of the fact that this model considers different parameters that can be adjusted, several variations are studied in order to assess their influence on the overall performance.

As a starting point, the baseline architecture for model 3 looks as shown in figure 5.5. The initial and final parts are the same as in the previous two models: first the input layer with the defined dimensions followed by data augmentation and value normalisation, and at the end the classifier with the dense layer as its core.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(2, activation = "softmax")
model <- keras_model(inputs, outputs)
```

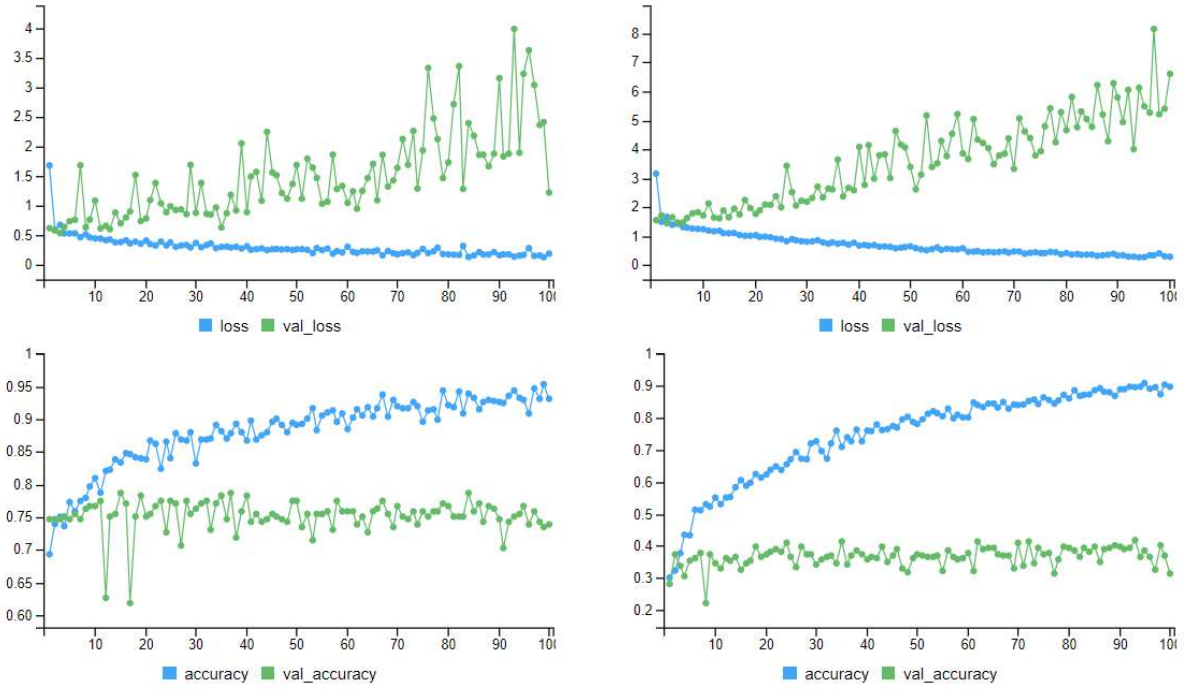
Figures 5.5: Model 3 architecture

The middle part of the model consists of the combination of a convolutional layer and a max pooling layer repeated twice. The first convolution has 32 filters of size 3 and *relu* activation, while the second has 64 filters with the same dimensions and activation. In the max pooling layer, the pool size defined determines the dimensions of the resulting matrix, which in this case will have half the size of its input matrix.

The evolution of the loss and accuracy for this model is shown in figure 5.6. The first that stands out from these plots is that the loss value has been significantly reduced. In particular, for the first iterations, it has been reduced by a factor of 10 for both cases, which could show that this model is more suitable for the

problem in hand.

Another consideration is that, once again, the validation and training loss distance from each other with the iterations, showing overfitting even from early stages of training, which is something that didn't happen with model 2 as it appeared later. Therefore, only with a first glance one can already tell that there are advantages and downsides on both architectures, meaning that an optimal solution will be found by balancing them out.



(a) 2 classes

(b) 5 classes

Figures 5.6: Model 3, accuracy and loss evolution during training and validation

Note that, although validation accuracy seems to remain rather stable when compared to the other models, the fact that the validation loss is lower could be an indication of a better performance which should lead to better predictions when facing new data.

Table 5.3 shows the most significant accuracy and loss values. Although the test values for the 2-class case improve with respect to the previous models, the metrics for the 5-class case remain without significant changes.

Table 5.3: Accuracy and loss values achieved with Model 3

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.7613	0.5593	0.2785	1.5244
Validation (best)	0.751	0.6024	0.3534	1.1469
Validation	0.7390	1.2236	0.3133	6.6079
Training	0.9313	0.1907	0.8978	0.2902

5.3.1 Variations

Given the fact that model 3 is more complex than the previous ones and contains more adjustable parameters, the following tests are carried out in order to see the effect that changing said parameters has in the global behaviour of the model. The aim is to get a better insight and understanding of what techniques like pooling and dropout, can accomplish when tuned well for a particular problem.

Variation 1: Study of the effect of padding

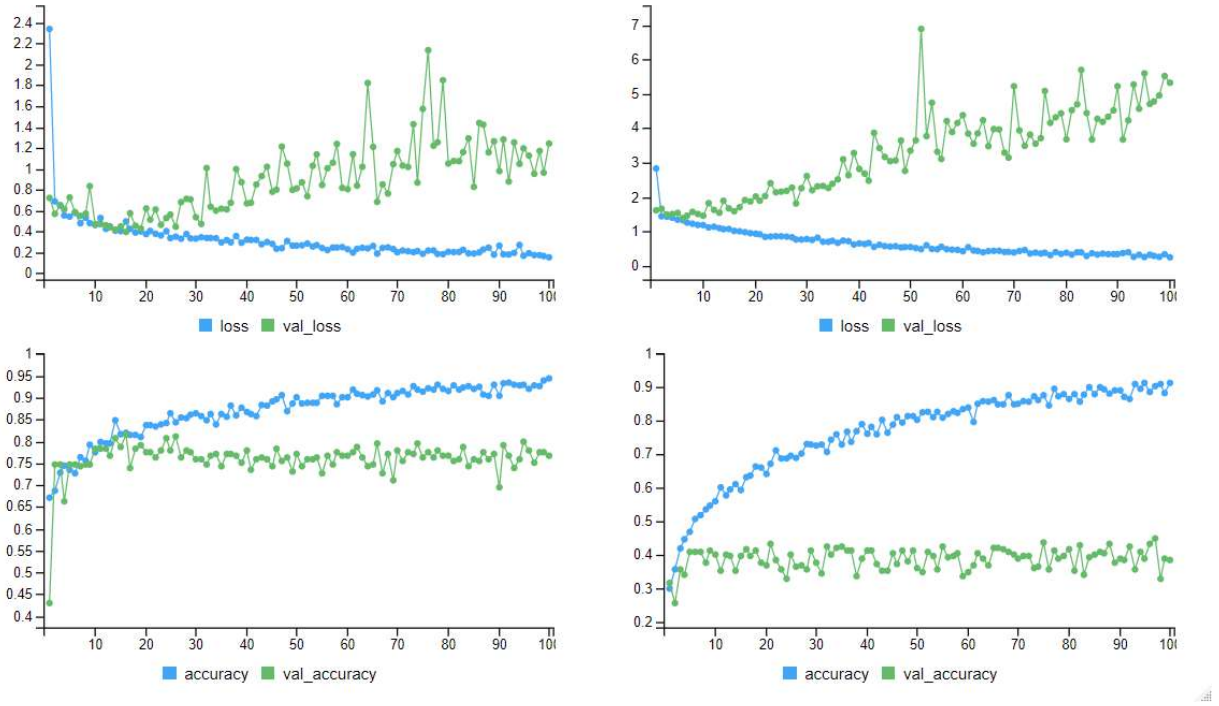
Padding consists of adding an extra row and column of zeros at each side of the input matrix to avoid losing the information at the edges when performing the convolution (refer to Chapter 3). The original version of model 3 doesn't consider it, so now it will be added when setting up the convolutional layers.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu", padding = "same") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu", padding = "same") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(5, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.7: Model 3 architecture - Variation 1

Convolutional layers take padding as an argument, and it can have two different values: *valid* or *same*. The first means no padding while the second one adds the additional rows and columns of zeros. When not specified, the default value is *valid*.

When comparing the evolution of the metrics shown in figure 5.8 to the ones obtained without padding, it can be noticed that overfitting appears later, especially in the 2-class case. In particular, it now appears around epoch 30 while it used to come up after only 5 iterations. As a consequence, the model is capable of learning more complex patterns and features, which translates into a minimum validation loss value lower than for the original version of model 3.



(a) 2 classes

(b) 5 classes

Figures 5.8: Model 3 variation 1, accuracy and loss evolution during training and validation

When it comes to the 5-class case, one can appreciate that the improvement is not as remarkable. Nevertheless the loss values achieved are slightly lower than before, which can already be considered a step in the right direction.

Table 5.4 is a compilation of the relevant metrics for this variation. All the values during test for both cases show improvement, although the loss of the 5-class problem is still too high to provide an acceptable performance.

Table 5.4: Accuracy and loss values achieved with Model 3 variation 1

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.7798	0.4932	0.3316	1.4723
Validation (best)	0.8072	0.4169	0.4096	1.3872
Validation	0.7671	1.2412	0.3855	5.3231
Training	0.9441	0.1517	0.9137	0.2498

All in all, padding has proven to be beneficial for the classification problem considered in this study. Initially it hadn't been added to the convolutional layers because in the example models seen its use wasn't spread.

Actually, in some way it makes sense that padding is helpful, as it prevents information on the edges of the image from being lost. Moreover, in this case in particular, since all pictures are close-up to damages, relevant information may be located at the borders.

Variation 2: Study of the effect of dropout

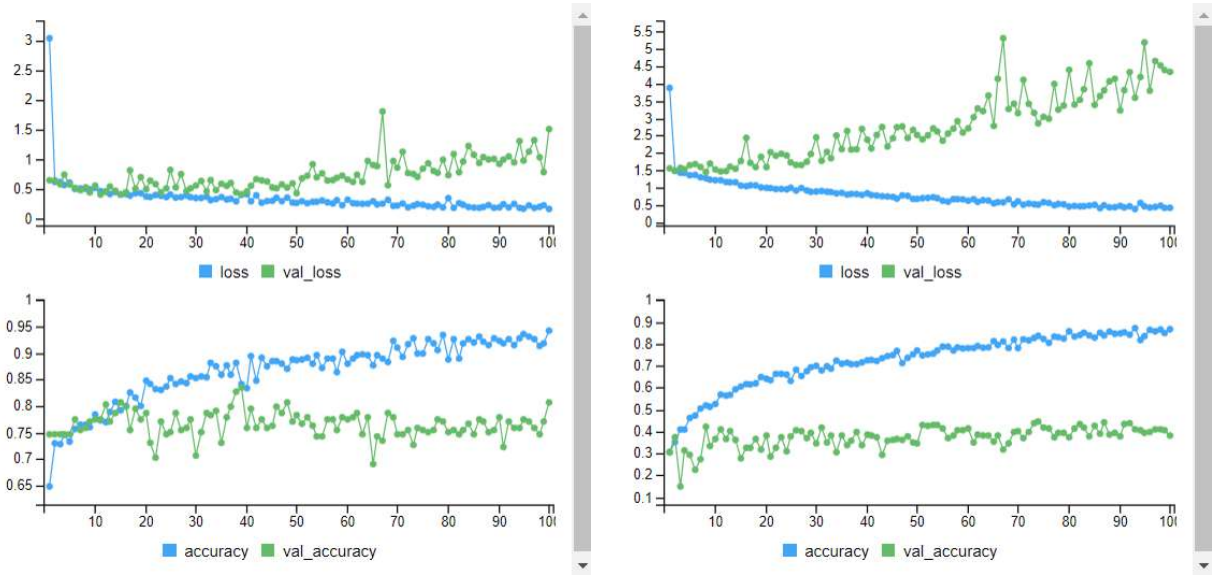
The next technique under study is dropout, more specifically, how changes on the dropout rates may affect the solution. Already in chapter 3 was established that dropout rates should range between 0.2 and 0.5, which is why the values considered will remain within this interval. To be more specific, two tests will be performed, one with dropout ratios on the lower side of the spectrum and another one with higher values.

While model 3 has only 1 dropout layer right before the classifier, several dropout layers will now be added throughout the code. In particular, they will be located after each block of convolutional layer plus max pooling.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%

  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(0.25)%>%

  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(0.25)%>%

  layer_dense(128, activation = "relu") %>%
  layer_flatten()%>%
  layer_dropout(0.4) %>%
  layer_dense(5, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.9: Model 3 architecture - Variation 2A

The code snippet shown in figure 5.9 shows three dropout layers with rates 0.25, 0.25 and 0.4 successively. Note that the 0.4 is kept as it has been implemented like this for the classifier in all the tests performed so far.



(a) 2 classes

(b) 5 classes

Figures 5.10: Model 3 variation 2A, accuracy and loss evolution during training and validation

Just like it happened with pooling, the addition of more dropout layers helps prevent overfitting. In the case of 2-class classification this phenomenon doesn't become obvious until almost 40 epochs, which also leads to lower global loss values.

Nevertheless, when it comes to the 5-class case, overfitting still appears at early stages of training. The main enhancement can be seen on the overall loss scores, as the maximum values achieved for validation are almost half of what they used to be. In spite of that, they are still well above 1, which is why maybe the problem isn't so much on the model itself but on the amount of data available.

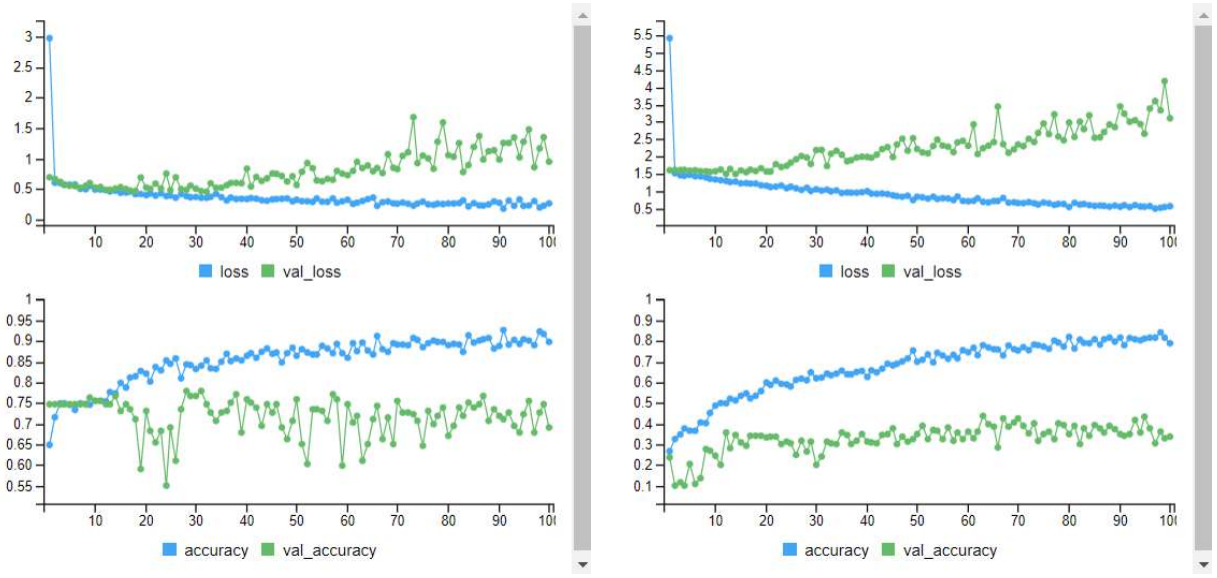
When looking at the metrics shown in table 5.5, one can see that the test values for the 2-class problem remain within the expected values, while for the 5-class case they slightly improve, although not enough.

The most promising results are for best case validation with 2-classes, where accuracy spikes and loss is maintained within reasonably good values. Nevertheless, for the 5-class scenario the loss actually increases although the accuracy improves.

Table 5.5: Accuracy and loss values achieved with Model 3 variation 2A

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.6976	0.5582	0.316	1.4702
Validation (best)	0.8313	0.4056	0.4378	1.4166
Validation	0.7912	1.2142	0.4458	3.7028
Training	0.9169	0.2787	0.8546	0.4793

Before moving on to the next variation, the dropout values for all the layers are risen to the maximum recommended, 0.5. The behaviour of the model can be seen in figure 5.11.



(a) 2 classes

(b) 5 classes

Figures 5.11: Model 3 variation 2B, accuracy and loss evolution during training and validation

Once again, overfitting is delayed, although with some nuances. For the 5-class case one can clearly see on the plot that the loss for training and validation diverge at later iterations than with lower dropout rates, but for the 2-class case the difference is not so clear. One could say that overfitting appears after approximately the same amount of iterations.

When it comes to the relevant scores, one can actually see that the test metrics improve for the 2-class case but downgrade a bit for the 5-class case. What's more, when it comes to the best validation iteration, all values worsen for both cases.

Table 5.6: Accuracy and loss values achieved with Model 3 variation 2B

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.7082	0.6245	0.2785	1.5296
Validation (best)	0.7751	0.4349	0.3815	1.4709
Validation	0.7912	1.0448	0.3735	3.6187
Training	0.9265	0.1952	0.8243	0.5457

To sum up, dropout helps to delay overfitting and thus should be beneficial for the global performance of the model. Nevertheless, there isn't a direct correlation between the actual dropout rates and the behaviour of the system, which may be caused by the random component inherent to this technique or its interference and interrelation with the other layers. What's more, too much dropout may be harmful, as it ultimately consists of setting to zero several coefficients, therefore inevitably losing information.

Variation 3

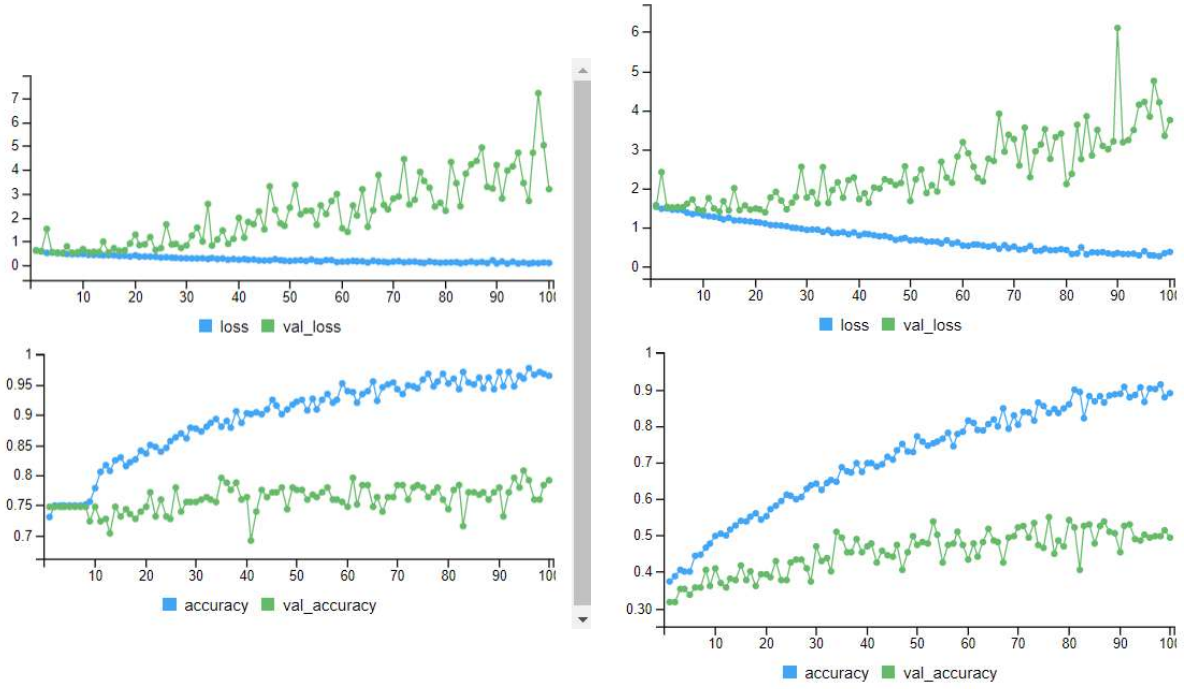
The final variation of model 3 consists of adding more convolutional layers to the architecture. The idea is based on the fact that the model with 2 convolutional layers has worked properly for the 2-class problem. Therefore having a total of 5 convolutional layers may improve the performance for the 5-class problem. According to the theory, deeper networks are able to learn more complex patterns, which is why maybe the one proposed here can learn the subtleties that help distinguish between damage types.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 128, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 256, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 256, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(2, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.12: Model 3 architecture - Variation 3

As indicated, the main changes in this architecture are the added convolutional and pooling layers. Note that the number of filters increases with the depth, but only up to a maximum of 256. That is because, the more filters there are, the higher amount of weights (or coefficients) have to be computed and stored, which makes the program stall due to lack of memory.

Note that padding hasn't been added here, although proved beneficial, because the objective is to evaluate only the effects of adding convolution layers by comparing the results to those of the baseline model.



(a) 2 classes

(b) 5 classes

Figures 5.13: Model 3 variation 3, accuracy and loss evolution during training and validation

When comparing figure 5.13 to the original plots from model 3 for the 2-class problem, one can see that overfitting appears later, which should allow the model to learn more complex patterns. Nevertheless, the improvement doesn't seem to be as remarkable as for the first variation, which considered padding.

For the 5-class classification case overfitting also commences after a higher number of epochs, although the real impact on the metrics is minimal. That's because the test loss barely decreases, and the best validation loss actually increases. This fact could be evidence for what could already be suspected from the tests performed so far: there isn't enough data for the model to learn how to successfully and accurately distinguish damage types. Therefore, it's very likely that no matter for how long the model is trained or how deep it is, the performance won't be significantly improved unless more images can be attained.

When it comes to the actual values of loss and accuracy during test, the improvement is almost negligible. Note that it's a mere coincidence that the value of test accuracy for 2-class classification are the same here and in the original version of model 3. On the other hand, since overfitting appears later, the best validation values show a slight improvement in terms of loss for the 2-class case.

Table 5.7: Accuracy and loss values achieved with Model 3 variation 3

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.7613	0.5673	0.3369	1.5257
Validation (best)	0.7470	0.5224	0.4297	1.3835
Validation	0.7912	3.1891	0.4940	3.7467
Training	0.9649	0.0986	0.8898	0.3700

5.4 Model 4: Deep convolutional neural network

The last architecture studied for this project consists of stacking two convolutional layers followed by a max pooling layer, and repeating 3 times this structure. The objective is to increase the depth of the network to check if it would be able to deal with the 5-class classification problem.

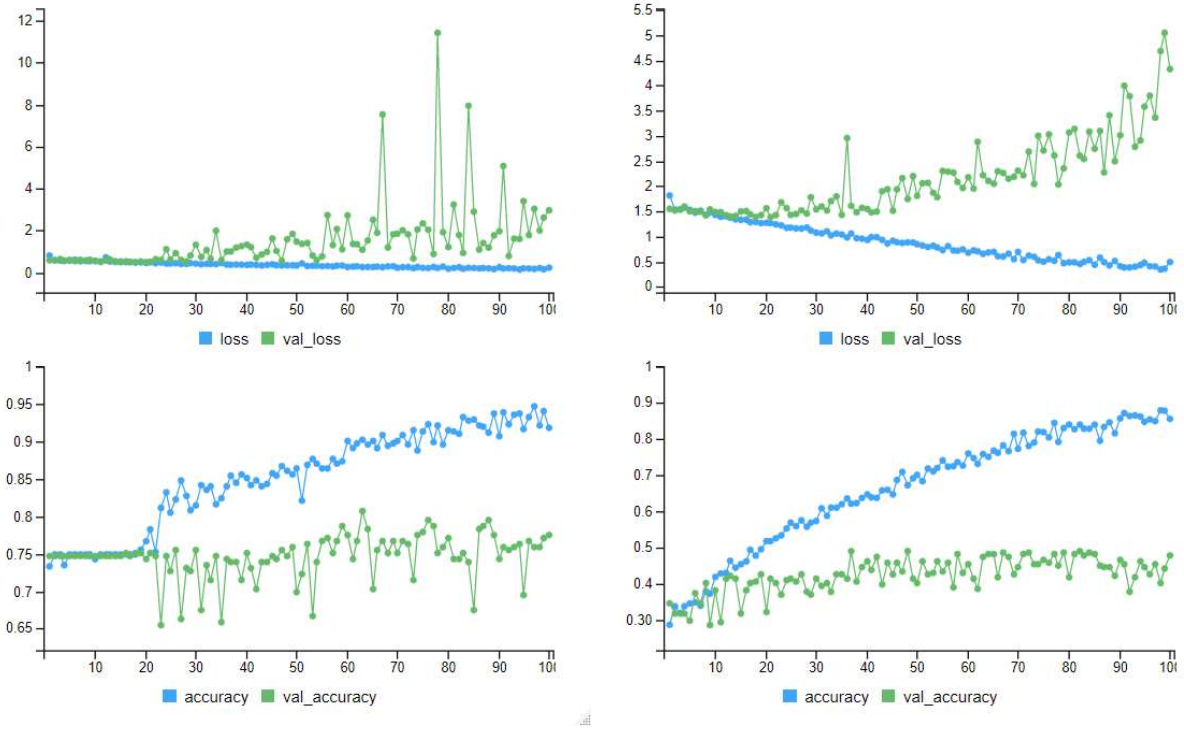
Note that now there are 2 convolutional layers before the pooling operation. That is because this new layout should let the network learn more complex patterns before downsizing with the max pooling layer. Regarding the number of filters, the maximum has been set again at 256 to avoid memory runouts.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu") %>%
  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 64, kernel_size = 3, activation = "relu") %>%
  layer_conv_2d(filters = 128, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_conv_2d(filters = 128, kernel_size = 3, activation = "relu") %>%
  layer_conv_2d(filters = 256, kernel_size = 3, activation = "relu") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dense(256, activation = "relu") %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(5, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.14: Model 4 architecture

From the plots shown in figure 5.15, several conclusions can be extracted. The first one is that overfitting is naturally delayed when compared to model 3, meaning that, without padding or added dropout layers the validation and test loss scores don't tend to diverge until after 20 to 30 iterations, depending on the case. What's more, for the 5-class case the overall loss values slightly decrease.

On another note, although less relevant, the accuracy for the 5-class case increases, while in all previous attempts it seemed to be stuck around 0.3 with occasional peaks around 0.35.



(a) 2 classes

(b) 5 classes

Figures 5.15: Model 4, accuracy and loss evolution during training and validation

When analysing the metrics in table 5.8 one can observe an overall improvement when compared to those of model 3. It's also remarkable the fact that, although the third variation of model 3 also had more convolutional layers, it wasn't able to achieve the same improvement as model 4. The reason behind this particularity may lay on the layer stacking, thus proving the point that the model is capable of learning slightly more complex patterns.

Table 5.8: Accuracy and loss values achieved with Model 4

	2 classes		5 classes	
	Acc	Loss	Acc	Loss
Test	0.7719	0.5365	0.3316	1.5175
Validation (best)	0.751	0.4945	0.4137	1.3872
Validation	0.7751	2.9648	0.4779	4.3272
Training	0.9185	0.2309	0.8562	0.4955

In spite of the small improvements explained before, other models like the first variation of model 3 or model 2 are capable of performing better in terms of test, validation or even both. Additionally, although less important, the computation time increases from approximately 40 seconds per epoch for models 2 and 3, to 100 seconds per epoch. While it's true that time hasn't been considered as a determining factor in this study, for this particular case spending twice as much is not considered to be worthy.

5.5 Model comparison

Once all tests have been performed, and before deciding on a final model, all the results obtained are summarized and compared in this section. For that purpose, only the test and best validation metrics are taken into consideration, as they are the most relevant ones. The reason for that is the fact that the training metrics are altered by the ability of the model to eventually fit with high accuracy only the training data, which leads to the phenomenon of overfitting previously discussed.

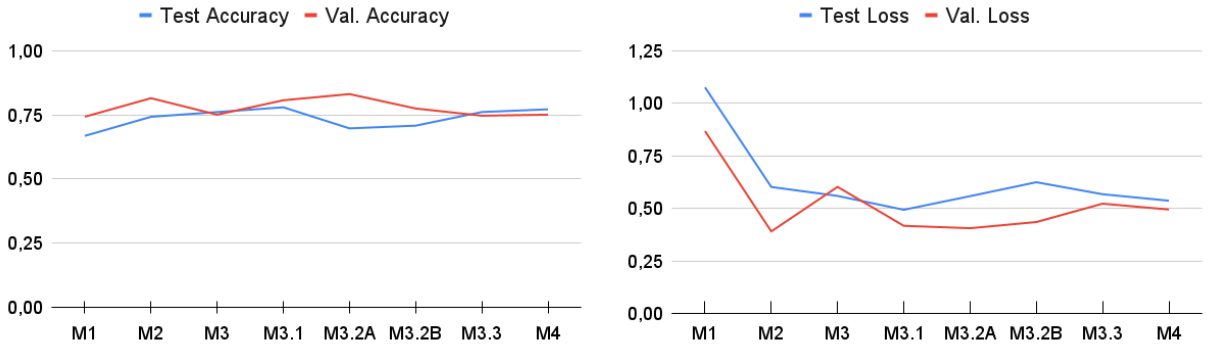
Glancing over the tables and plots from the previous sections, one can already tell that the models behave differently for the 2-class classification problem and for the 5-class one. Therefore, the following analysis is performed separately and, if deemed necessary, two final models will be developed.

5.5.1 2-class classification case

For each of the models, figure 5.16 shows in red the best validation scores and in blue the test ones. In particular, the plot on the left refers to the accuracy while the one on the right belongs to the loss. As mentioned before, the most important parameter is the loss, and therefore it will drive most of the decisions taken.

When looking at the validation loss, it is clear that model 2 is the one with the lowest value, although it's closely followed by variations 1 and 2A from model 3. On the other hand, the minimum loss score during test is achieved by the first variation of model 3.

Before moving on, and only as a quick reminder, model 2 was the one with one convolutional layer on top of the classifier, while model 3 considered convolutional and pooling layers. As for the variations, the first one added padding, whereas 2A had dropout layers with low ratios.



(a) Accuracy

(b) Loss

Figures 5.16: Model comparison, 2-class case

As a consequence of the previous statements, a good option for this classification case would be an architecture like the one from model 2, but that also adds a max pooling layer and considers padding as well as more dropout layers with ratios around 0.25.

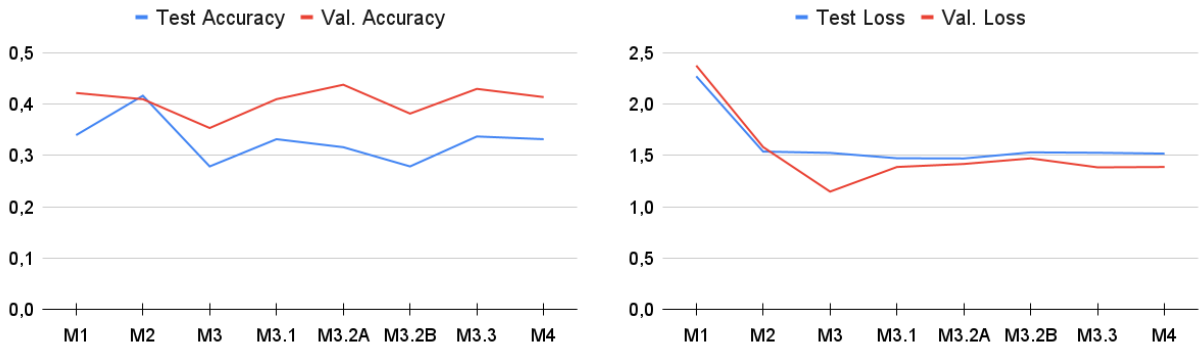
Having a look at the accuracy it can be seen that it remains rather stable, which is something that doesn't happen for the loss, as model 1 yields a notably higher value. For this reason, it is safe to say that the overall accuracy of the model will be more independent of the architecture, and therefore it's an indication of the loss being the right parameter upon which the decisions should be based.

5.5.2 5-class classification case

The classification by damage type has proven to be a much more difficult task than only discerning whether there are damages or not. Clear evidence of this fact can be seen in figure 5.17. In it, the loss is shown on the right, in blue for test and red for validation, and the accuracy on the left with the same colour code.

The thing that stands out the most from these plots is how stable the loss score is, especially during test. In the case of validation, it takes a dive for model 3, although given the random factor inherent of neural networks it might be only coincidence, and still it remains well above 1, which wouldn't lead to a very reliable performance.

When it comes to the accuracy, there is more variation although the tendency is for it to be rather low. Also, it stands out the fact that the lowest accuracy value for both testing and validation is obtained with model 3, that also had the lowest loss score and therefore the performance should be slightly better.



(a) Accuracy

(b) Loss

Figures 5.17: Model comparison, 5-class case

All in all, the multiple attempts carried out for this project have turned out to be unsuccessful in the job of classifying pictures of aircraft damages into 5 different categories in a reliable manner. Several architectures have been tested and their parameters have been tuned, but none of them has shown an acceptable performance or a noticeable improvement whatsoever. Therefore, the remaining steps of this study will be completed only for the 2-class case.

This doesn't mean that the 5-class case is unsolvable, but rather that more pictures are needed to train the model. Future work should aim at gathering more images to enlarge the initial data set, and then perform again the tests explained before, especially the ones with the deeper models.

5.6 Final Model

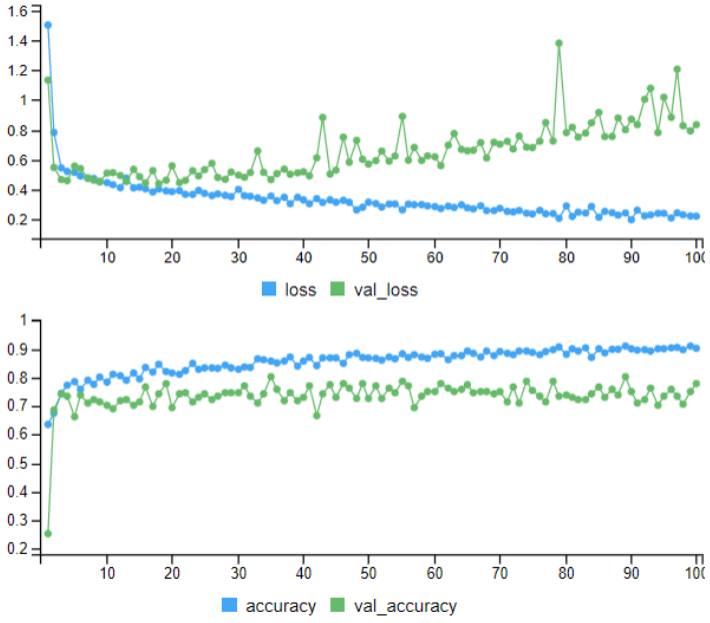
After all the tests and analysis performed, the model described in this section is a balance of everything that has been learned so far. Note that in Machine Learning there are no right answers but rather more accurate ones, which is why the architecture developed for this problem may not work well when facing others, just like it happened at the beginning when the "cats vs. dogs" model wasn't useful for classifying aircraft pictures (refer to Chapter 4).

Figure 5.18 shows the final appearance of the model. It's based on model 2, as it isn't as deep as some of the other options, but it has some additional features. It considers padding in the convolutional layer and it adds two additional layer types: a dropout one with a ratio of 0.25, and a max pooling layer right after the convolution.

```
# Instantiate model
inputs <- layer_input(shape = c(180, 180, 3))
outputs <- inputs %>%
  data_augmentation() %>%
  layer_rescaling(1 / 255) %>%
  layer_conv_2d(filters = 32, kernel_size = 3, activation = "relu", padding = "same") %>%
  layer_max_pooling_2d(pool_size = 2) %>%
  layer_dropout(0.25) %>%
  layer_dense(128, activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(0.5) %>%
  layer_dense(2, activation = "softmax")
model <- keras_model(inputs, outputs)
```

Figures 5.18: Final model architecture

Once ready, the model is trained for 100 epochs. Just like all the others, the loss function chosen is categorical cross-entropy and the optimizer is *rmsprop*. Figure 5.19 shows the resulting metrics.



Figures 5.19: Final model validation and training scores

As expected, the appearance of overfitting is delayed until approximately the 20^{th} iteration. Additionally, the overall loss values are significantly lower than in the previous tests. When it comes to accuracy, it doesn't seem to improve significantly for the validation, but since it ranges between 0.7 and 0.8 it's already better than in earlier architectures.

The most relevant metrics are shown in table 5.9. When comparing them to those of the previous models, one can see an overall improvement in the loss scores, especially during test. For instance, although the best validation loss value hasn't beaten that obtained with model 2, which was the lowest of all, the test loss is smaller for the final model. The underlying reason for these results can be the fact that since the validation loss is the value used for weight optimisation, some information may leak from the validation data set into the training process, thus leading to exceptional metrics which won't be reproduced during test.

Table 5.9: Accuracy and loss values achieved with the final model

	2 classes	
	Acc	Loss
Test	0.7082	0.5467
Validation (best)	0.743	0.4409
Validation	0.7791	0.8357
Training	0.9026	0.223

Note that, at the end of the day, accuracy and loss are only two parameters that try to describe the performance of a model, but they aren't an undeniable truth. For this reason, all the work done in this project is an initial study, and the actual behaviour of a model can be best assessed when tested on a production environment during weeks or even months.

The metrics obtained with this final model show a good combination of loss and accuracy scores both during validation and training, but note that other options like the first variation of model 3 also show similar behaviour. This is a reassurance of the fact that multiple methodologies can lead to similar solutions in Machine Learning, and they are both equally right. Only in further testing stages a more accurate assessment could be provided.

5.7 Bringing the model into a useful tool

So far several architectures have been analysed and tested, and a final model has been defined. Once the study has reached this point, it's time to transition from the drawing board to an actual tool that implements the chosen model and allows its deployment in a more real-life-like scenario.

This is the final stage that will enable further testing with new pictures, and that will allow the distribution of the model in the shape of an app so that it can be shared with other stakeholders interested in evaluating this technology.

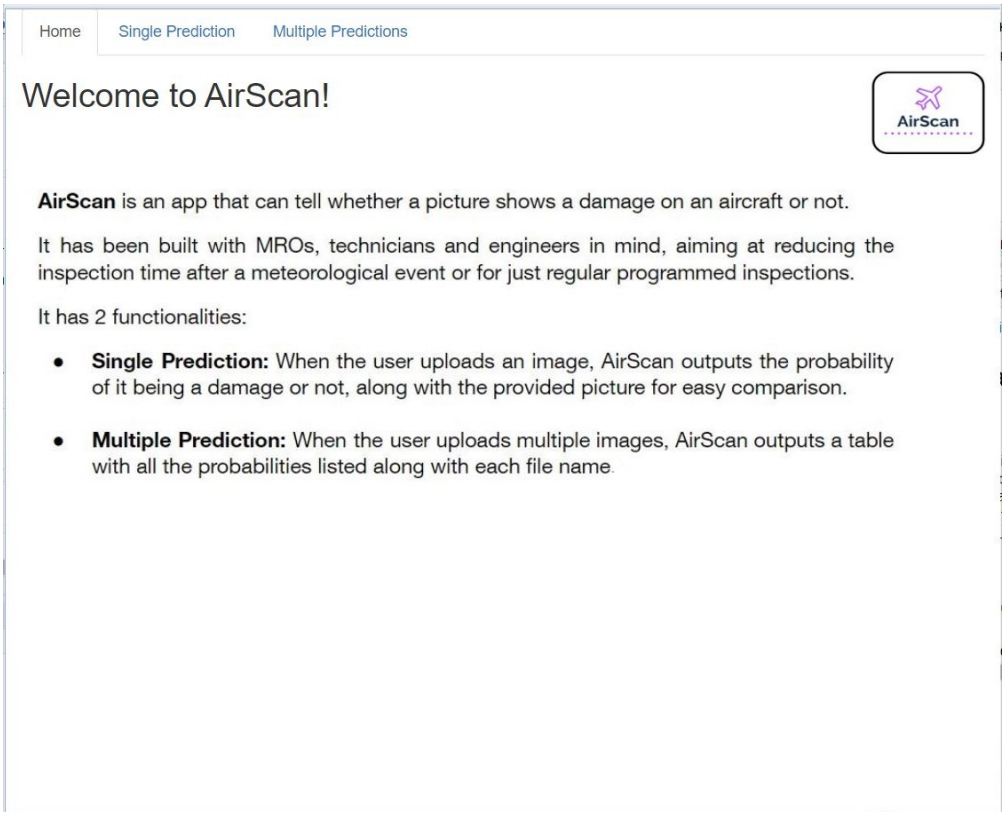
As a result of that, **AirScan** is born, an app for aircraft damage detection. The following sections outline how it has been created and which are its main functionalities.

5.7.1 Building an app in R

Just like the rest of the code, AirScan has been developed with R. In particular with Shiny, an open source package for creating interactive web-based applications that can execute R code on their back-end [17] [18].

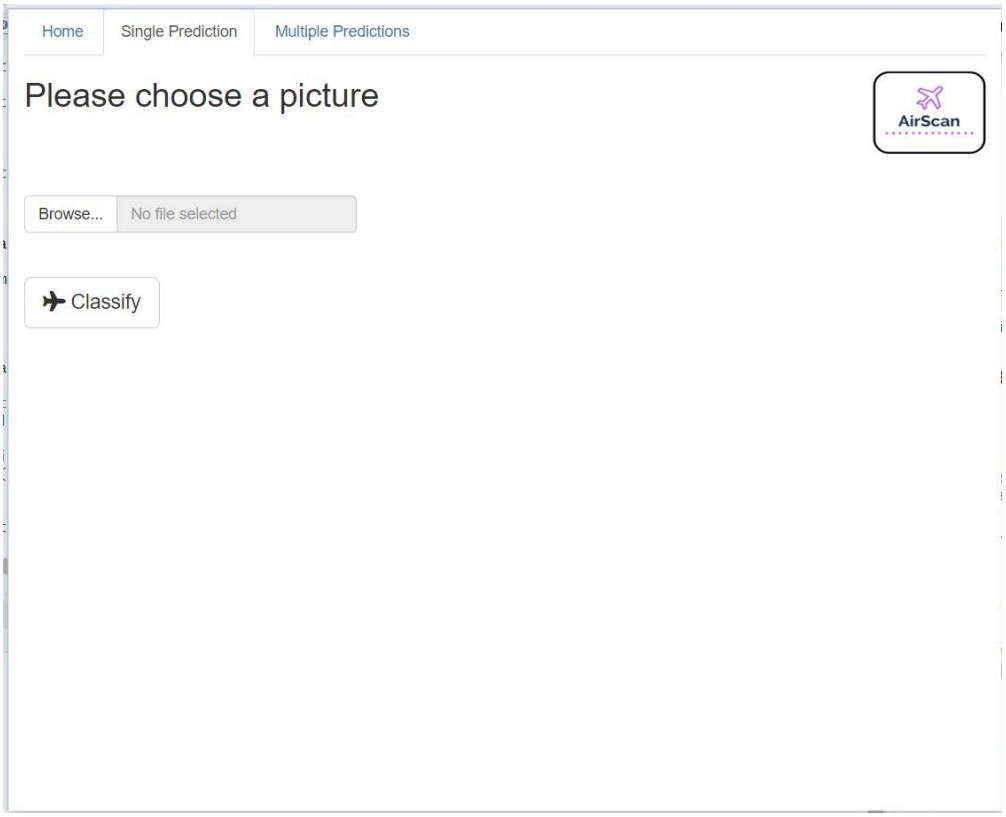
What this app basically does is load the model with its weights, which were saved during the previous tests, and makes predictions for the images that the user uploads. As output it gives the probability of being a damage or not, which can be displayed in various ways, depending on the mode the user has chosen.

When it comes to its appearance it has been kept clean, functional and easy to use, always bearing in mind its main goals. Figure 5.20 shows AirScan's homepage, which offers a brief description of its purpose along with its functionalities.



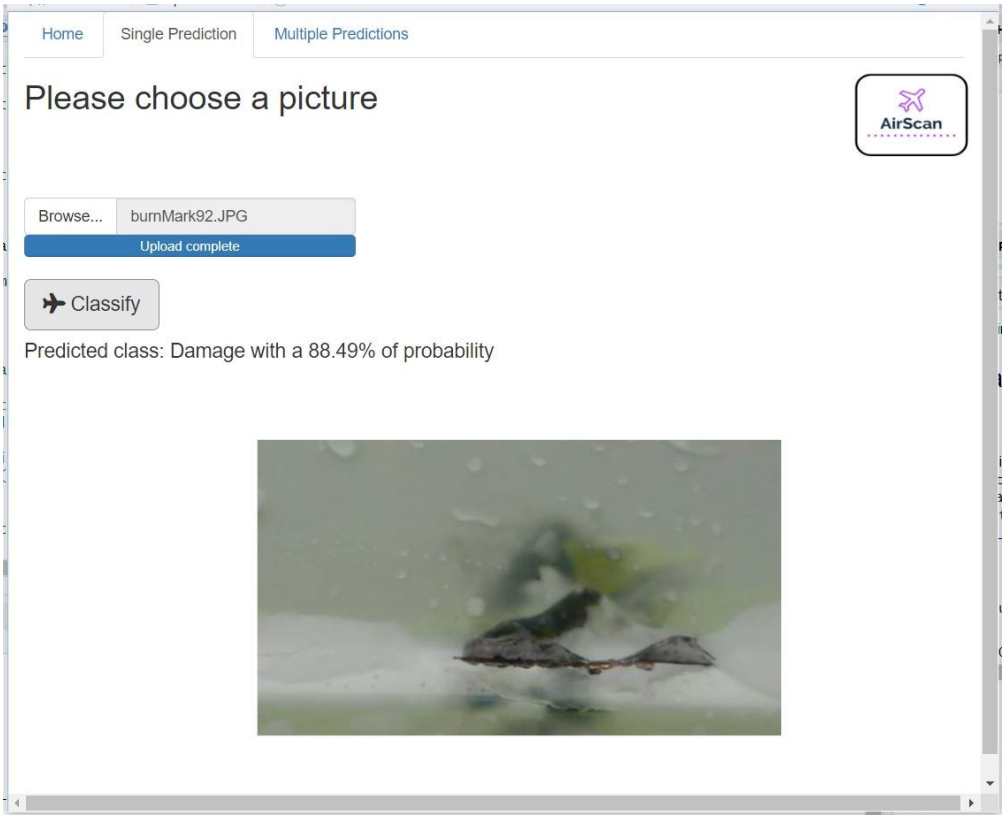
Figures 5.20: AirScan homepage appearance

Diving right into the core tabs, the Single Prediction tab (figure 5.21) allows the user to make predictions for only one image at a time. By clicking on "Browse" a dialogue will open where they can navigate within the folders and choose a picture. Once selected, a progress bar will appear indicating when the file has been successfully uploaded. Then by clicking on "Classify" the model will start running.



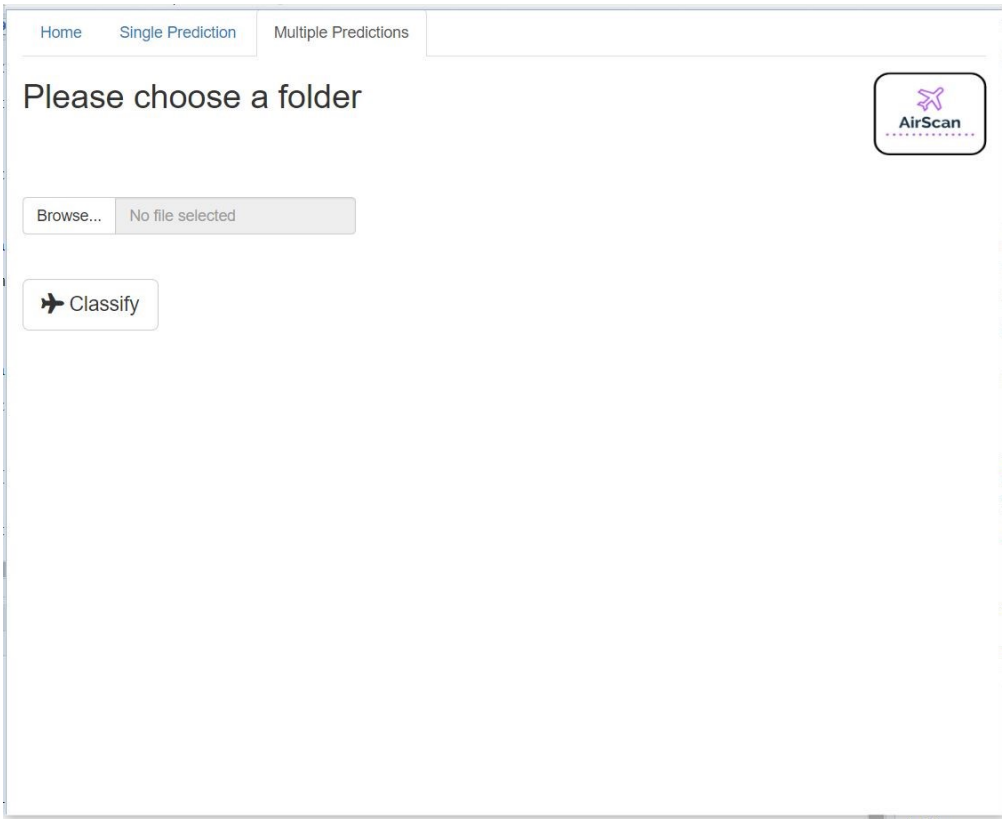
Figures 5.21: AirScan single prediction page

Figure 5.22 shows how the output is presented by the app. On one side it tells the most likely predicted class and the percentage of probability, while on the other side it shows the selected image. In that way, the user will be able to determine by themselves whether the model is providing accurate guesses or not. The main purpose of this tab is for the user to make sure that the model is working right, as the layout makes it very easy to check and compare the prediction with the actual image.



Figures 5.22: AirScan single prediction page output

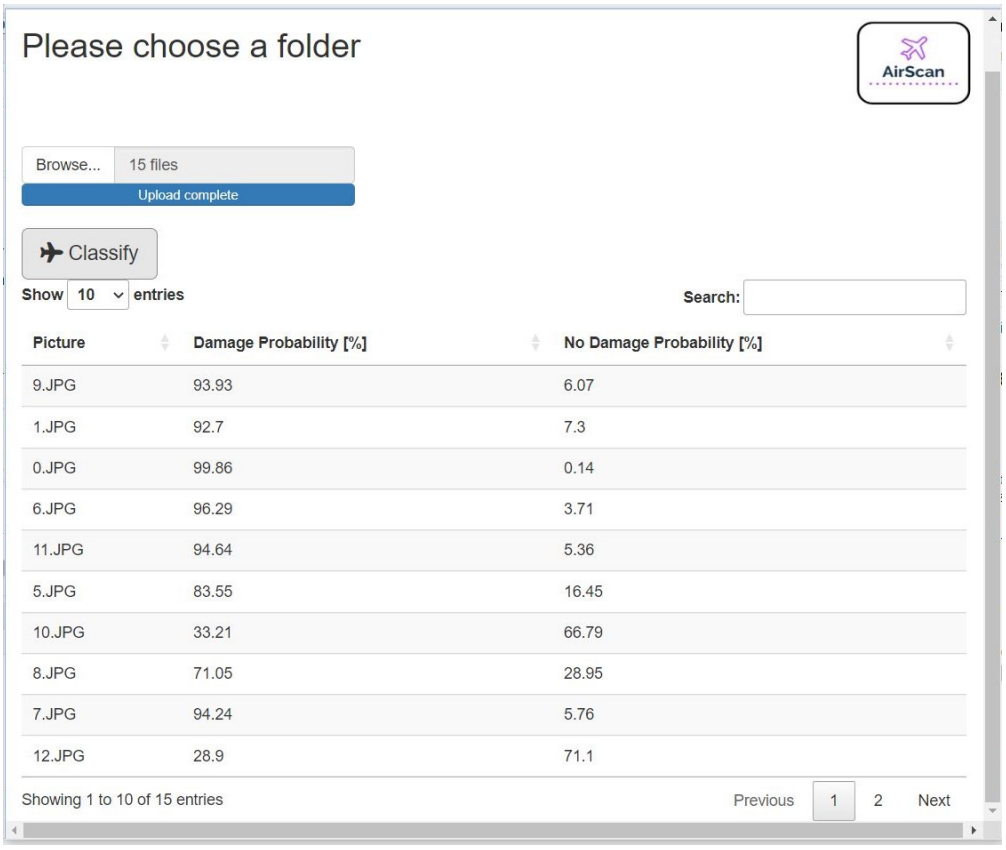
Lastly there is the Multiple Predictions tab, shown in figure 5.23. Its initial arrangement is very similar to that of the Single Prediction. When clicking on "Browse" a dialogue will open for the user to navigate within the folders, although this time it will let them choose many different files. Again a progress bar will tell when all the pictures have been uploaded. After that, when the "Classify" button is clicked, the model will make its predictions for all of the selected images.



Figures 5.23: AirScan multiple prediction page

Figure 5.24 shows the disposition of the output. Since this time there will be multiple results, they are all gathered and presented within a table for clarity. Instead of only presenting the most likely class, the percentages for both of them is displayed, along with the name of the file. Additionally, the table can be configured to show as many rows per page as desired and the user can easily navigate through the pages.

Note that this time the pictures aren't displayed, as it would be too much information at one glance. Instead they are cross-referenced with the file name, which should make it easier to search for those containing damages.



Figures 5.24: AirScan multiple prediction page output

As a summary, AirScan has been developed with the objective of creating an easy to use interface that provides a friendly environment to interact with the model. In that way, engineers and technicians will be able to upload their pictures and make predictions which, at an early stage of deployment, should be monitored to evaluate the model's performance. As a result, users will be able to determine which kind of pictures are harder for the app to differentiate so that they can be gathered and used for further training.

Once AirScan is up and running it's time for the next stage, its actual deployment for testing in a real-life work environment with aircraft maintenance companies. Although this is out of the scope of this project, the next chapter provides some theoretical indications to enhance this next step's success.

Chapter 6

Results discussion and further steps

Up to this point, research has been done on Machine Learning techniques so as to achieve a final model that can perform predictions successfully. It has been shown that it can distinguish between damaged and undamaged aircraft structure, and it has been included in an app, AirScan, for easy interaction.

With everything achieved so far, this chapter aims at taking a look further into the future to see all the possibilities that technologies like AirScan can offer to the aircraft maintenance industry.

6.1 Testing with new images

All throughout the previous chapters, the performance of the models has been presented in terms of accuracy and loss scores. Even though they provide important information for adjusting the depth of the network and the types of layers within it, one can find it hard to translate this data into more comprehensible metrics and extrapolate them to a real world scenario.

As a consequence, the objective of this section is to provide completely new and unseen pictures to the model and compare its predictions to the actual condition of the aircraft's structure. Although it will be done with only a small sample, this small evaluation can help to better understand the strengths and weaknesses of the model.

Tables 6.1 and 6.2 show the images selected for this small test. They have been carefully chosen so as to assess and evaluate specific features, as well as to challenge AirScan as much as possible. For example, pictures 1 and 2 show undamaged fuselage skin, both from the lower section of an aircraft. While the first is a clear white surface, the second shows some signs of dirt that could be misunderstood as defects. Taking into account that airplanes fly constantly and are cleaned only once every several months, it's important that the model can differentiate dirt and stains from actual damages, as otherwise too many false alerts would be triggered and the technicians would still need to check most of the aircraft.

The third picture chosen is a burn mark, the category with the most representation within the database. Therefore, one would expect a high accuracy from the prediction. Quite the opposite could be anticipated

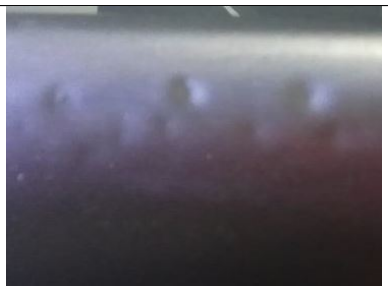
for the fourth and fifth pictures, which are dents. This kind of damage is usually hard to spot, even for the human eye, because usually dents are superficial and need light coming from a specific angle in order to distinguish them clearly. Additionally, during the image gathering process many were discarded for not being representative enough, so it is very interesting to see how AirScan will behave when facing these damages.

The last picture given is from a gouge, which differentiates from a scratch by being deeper and wider, and also by entailing a higher loss of material. Although the damage is clear, it doesn't belong to any of the major categories of defects, which is why the model may have not been exposed to enough data to spot it easily.

Table 6.1: Comparison between Airscan predictions and actual status

Picture	**Prediction**	**Actual status**
	No damage with a 77.21% of probability	Undamaged fuselage
	No damage with a 74.16% of probability	Undamaged but dirty fuselage
	Damage with 98.09% of probability	Burn mark on fuselage skin
	No damage with a 54.95% of probability	Dent on a flap

Table 6.2: Comparison between Airscan predictions and actual status - Continued

Picture	Prediction	Actual status
	Damage with a 93.55% of probability	Dented area on a slat
	Damage with a 95.52% of probability	Gouge on fuselage skin

From the previous tables one can extract some conclusions. Regarding the pictures showing no damages, AirScan has provided correct predictions. Note that, as could be anticipated, the image from a dirty fuselage shows less probability than the clean one, although the guess is still correct.

Regarding the burn mark on image 3, the model has made a right guess and with the highest percentage of probability, 98.09%. This fact is coherent taking into account that burn marks were by far the type of damage with more representation in the data set.

When it comes to the dents, the results aren't as clear. For the picture with the single dent on a flap, AirScan isn't capable of providing the right guess. Note that the probability is of 59.95%, the lowest one of all, which means that it isn't that clear for the model that there's no damage. Probably, with more exposure to images of dents during further training, the model could learn the appropriate patterns that would let it distinguish this type of defects. On the opposite end there's picture 4 of a dented area. In this case the prediction is correct and with a high percentage of 93.55%. This could probably mean that, out of mere coincidence, during training the model was shown more pictures of dented areas than of single dents, which is why it may have learnt to distinguish more accurately this kind of pattern.

Lastly, the picture of the gouge has been accurately classified as damaged structure, although with a lower probability than burn marks. Therefore, one can conclude that AirScan shows its best performance on burn marks but it can also correctly differentiate undamaged structure and defects that imply a loss of material. Nevertheless, deformations on the structure, such as dents, are its week spot and they need to be reinforced with further training upon such damages.

6.2 Guidelines for the AirScan's deployment

From everything exposed so far, one can appreciate that AirScan is showing promising results. This project aims at setting the foundations for the optimisation of aircraft maintenance inspections and thus it constitutes a first step in the right direction. The ultimate goal would be to integrate AirScan with a drone equipped with a camera that can perform the inspections while the model is providing predictions.

In fact, Airbus' Aircraft Maintenance Manual (AMM) already considers the possibility of carrying out with drones the inspections after a lightning strike. They even provide their own drone which comes with a dedicated software, although it's only for image visualisation. Up to this point they haven't developed any algorithm that can autonomously detect damages and classify the images taken by the drone. Among its main advantages, Airbus highlights the fact that an inspection on an aircraft of the A320 family can be performed in only 30 minutes [19].

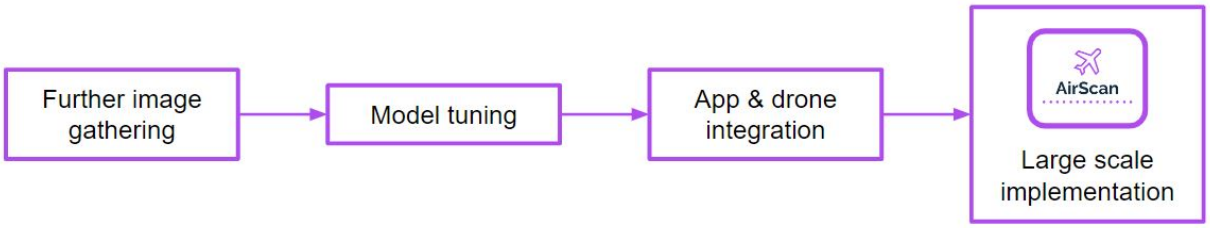
Figures 6.1: Airbus' drone for aircraft inspections. Source: [20]

Taking into consideration that so far the model has only been evaluated under a very controlled environment and with relatively few pictures (a bit more than 1200), the first thing to do would be to ask several users to upload their aircraft images for a few weeks and manually check with Airscan if the predictions given are right or not. When they are not, those images should be kept in order to comprehend which features are hard for the AI to understand. In this way, engineers and technicians would be able to see for themselves the potential of such a tool, and also will be able to trust and rely more on the app.

When a few hundreds of new images are collected, say for example 200, the model should be retrained again to see whether its metrics improve or not. Special attention should be put to those wrong predictions and

more images of this certain category should be gathered to provide them in further training iterations.

Once the model becomes more reliable, this process can be automatised so that the algorithm itself can provide metrics of its performance, like for instance the percentage of right and wrong predictions. Then once every few days an engineer would need to check them to monitor their evolution. Note that training shouldn't stop because, even though aircraft damages aren't likely to change their features (a dent will always be a dent), variations on, for example, the aircraft livery might lead to undesired results if training is discontinued.



Figures 6.2: Future work outline

The next stage of the process would be to integrate the app with a drone equipped with a camera that can provide real-time images so that, while the inspection is happening, the model is already making predictions on the first images sent. Thanks to the GPS technology embedded in the drone, the technicians will know right away where those defects are and go straight to perform their evaluation and possible repair.

Since the maintenance manuals such as AMM and SRM are organised according to the ATA chapters, a further functionality that could be added to the app would be to link the location of the damage to the corresponding chapter in the manual or even to a particular task for its evaluation and repair. This could be done thanks to the fact that the aircraft's structure is contained within the ATA chapters 50 to 57, and they are divided by location as follows:

Table 6.3: ATA chapters related to the aircraft's structure

Chapter	**Concerned area**
51	Standard practices
52	Doors
53	Fuselage
54	Nacelles and pylons
55	Vertical and horizontal stabilisers
56	Windows
57	Wings

Once all the steps described above are successfully accomplished, the time will have come for the large scale deployment of the system. Among its many benefits stands out the cost reduction driven by the decrease in the time required to perform an inspection. Furthermore, the room for human error will be reduced as it will be less likely for the drone to miss a damage due to stress and time pressure. As a result, the status of an aircraft can be assessed much faster after a meteorological event like a lightning strike or a hail storm, and it can fly again much sooner than with the current methods.

Chapter 7

Budget summary

Given the fact that this project requires mainly image gathering and code development the main costs derive from the labour plan and the laptop used. As a result, the total projected cost is 4007.70 €. The full breakdown and detailed calculations can be found in the document provided separately.

Note that all the software needed, mainly RStudio and its packages and libraries, are open source and thus not accounted for. Last but not least, no investment has been needed to obtain the pictures of aircraft damages, as they have been kindly given by an airline for this study. Nevertheless they have a huge value as it is very hard to find them online, and without them this project wouldn't have been possible.

Chapter 8

Analysis and assessment of the social and environmental implications

One of the main outcomes of this project is the possibility of enhancing and optimising aircraft inspections by introducing new technologies to the process. With a computer and Machine Learning algorithms, many pictures can be gathered and processed at once thus reducing the time spent.

When looking at the UN's Sustainable Development Goals, one can see how this project could contribute to Goal 9: Build resilient infrastructure, promote sustainable industrialization and foster innovation. In particular, its targets aim at promoting research and innovation in all industrial sectors so as to upgrade their technological capabilities [21].

Thanks to proposals like this project, a drone or some other sort of device equipped with a camera will inspect an aircraft, take detailed pictures of its outer structure and send them in real time to a computer which will classify them according to whether there's damage or not. In that way, the technicians will know where they are located in a matter of minutes and won't need to go around the airplane for long hours working with lifting platforms and the risk they entail. Additionally, the lifting platforms used are sometimes powered by fuel, which is why a reduction of their use will have a positive outcome in terms of CO_2 emissions and also costs. At most, when damages are located at heights, they will need to go up there once to perform the repair, and not multiple times to check all areas, thus also reducing the risk of a possible injury.

Chapter 9

Conclusions

All throughout this report, different Machine Learning models have been studied in order to assess the feasibility of developing an AI capable of detecting aircraft structural damages. In particular, 4 different architectures have been considered, which have eventually led to a fifth one, the final model.

For that purpose, a database of aircraft pictures has been built, containing images from different damage types and also of healthy structure. In particular, the models proposed have been trained, validated and tested with 1268 pictures. Additionally, with the objective of expanding the database, data augmentation techniques have been implemented. Since the success of the system in distinguishing between damage types isn't guaranteed, two cases have been considered: 2-class case (damage/no damage) and 5-class case (dent, scratch, burn marks, other and no damage).

From all the tests performed, one can conclude that only a linear classifier is too simple to solve a problem like the one in hand, and thus can be discarded as a standalone algorithm. Nevertheless, when combined with convolutional layers to form a convolutional neural network, the overall performance increases notably.

Once established that convolutions are the best ally for computer vision problems, the amount of layers and their configuration is assessed. To fulfill the requirements of this project, deep neural networks with up to 6 convolutional layers have proved to be inefficient, because they don't manage to provide better results than shallower networks, and the computation time is doubled.

When it comes to ways for fighting underfitting and, specially, overfitting, padding has been proven to be the most successful one. As a result, the system is capable of learning more complex patterns and the global loss scores improve. Other regularisation techniques like dropout also contribute to a better performance, although one has to be very careful with the rate. The value that has shown the best behaviour is 0.25, while ratios of 0.5 and above result in an excessive loss of information which actually harms the model. Additionally, the models that include max pooling layers in their architecture are able to achieve lower loss scores than those that don't have them.

For all the models tested, the 2-class case has been showing improvements when changing some of the layers and characteristics. Nevertheless, the same can't be said for the 5-class case, as the evolution of the metrics during training and validation, and the scores attained haven't shown a significant improvement. Therefore, one can confirm that the problem isn't so much on the model itself, but rather on the amount of images available for training.

As a result of all the exposed above, the final model is a combination of model 2, and the first and second variations of model 3. In particular it features a convolutional layer with padding, a max pooling layer, a dropout layer and a linear classifier. Once it's trained, it is exported to an app, AirScan, that allows users to perform predictions with single images or multiple images.

Future lines of work should aim at gathering more pictures of all the damage types considered so as to perform further training in the model. Once this is achieved, the app should be distributed to multiple stakeholders so that it can be tested in a production environment. Eventually, AirScan should be integrated with a drone or device equipped with a camera that can provide real-time images and therefore real-time predictions.

All in all, although the goal of classifying pictures according to the damage type shown hasn't been achieved, the developed model can successfully differentiate between damaged and healthy structure. This partial achievement of the objectives is already a big step in the right direction, as the most important for an engineer is to know whether there are defects or not. As a result, the maintenance personnel will already know exactly where to look at, and the total inspection time will be significantly reduced, thus decreasing the ground time and the costs it implies for airlines.

References

1. *AMOS mobile, Digital momentum for technicians.* Available also from: https://www.swiss-as.com/amos-mro/amosmobile.
2. (EASA), European Aviation Safety Agency. *Commission Regulation (EU) No 1321/2014.* 2014. Available also from: https://www.easa.europa.eu/en/document-library/regulations/commission-regulation-eu-no-13212014.
3. FRANÇOIS CHOLLET Tomasz Kalinowski, J.J. Allaire. *Deep Learning with R, Second Edition.* 2022. Available also from: https://livebook.manning.com/book/deep-learning-with-r-second-edition/front-matter/.
4. SÁNZ, José; FERRE, Manuel; ESPADA, Alvaro; NAROCKI, Matias Collar; FERNÁNDEZ-PARDO, José. Robotized inspection system of the external aircraft fuselage based on ultrasound. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2010, pp. 2612–2617. Available from DOI: 10.1109/IROS.2010.5653073.
5. WANG, Congqing; WANG, Xianfeng; ZHOU, Xin; LI, Zhiyu. *The aircraft skin crack inspection based on different-source sensors and support Vector Machines - Journal of Nondestructive Evaluation.* Springer US, 2016. Available also from: https://link.springer.com/article/10.1007/s10921-016-0359-3.
6. GARBUNO, Daniel Martinez. *LATAM cuts aircraft inspections to 40 minutes with drones.* 2022. Available also from: https://simpleflying.com/latam-drone-aircraft-inspections/.
7. STANFORD VISION LAB Stanford University, Princeton University. *ImageNet Large Scale Visual Recognition Challenge (ILSVRC).* 2020. Available also from: https://www.image-net.org/challenges/LSVRC/.
8. ROBINSON, Rob. *Convolutional Neural Networks - Basics An Introduction to CNNs and Deep Learning.* 2017. Available also from: https://mlnotebook.github.io/post/CNN1/.
9. *Input layer, R Documentation.* Available also from: https://rdrr.io/cran/keras/man/layer_input.html.
10. *Convolutional layer, Keras Documentation.* Available also from: https://keras.io/api/layers/convolution_layers/convolution2d/.
11. *Pooling layer, Keras Documentation.* Available also from: https://keras.io/api/layers/pooling_layers/max_pooling2d/.
12. *Dropout layer, Keras Documentation.* Available also from: https://keras.io/api/layers/regularization_layers/dropout/.

13. *Flatten layer, Keras Documentation*. Available also from: https://keras.io/api/layers/reshaping_layers/flatten/.
14. *Dense layer, Keras Documentation*. Available also from: https://keras.io/api/layers/core_layers/dense/.
15. BAELDUNG. *Training and Validation Loss in Deep Learning*. 2023. Available also from: https://www.baeldung.com/cs/training-validation-loss-deep-learning.
16. TENSORFLOW. *MNIST database of hand-written digits*. 2023. Available also from: https://www.tensorflow.org/datasets/catalog/mnist?hl=es-419.
17. *Shiny Documentation*. Available also from: https://shiny.posit.co/.
18. WICKHAM, Hadley. *Mastering Shiny*. 2020. Available also from: https://mastering-shiny.org/index.html.
19. AIRBUS. *Innovation takes aircraft visual inspections to new heights*. Available also from: https://www.airbus.com/en/asset-preview/86171.
20. AIRBUS. *Drone for aircraft inspections*. Available also from: https://www.airbus.com/en/asset-preview/61821.
21. *Sustainable Development Goals*. Available also from: https://www.un.org/sustainabledevelopment/infrastructure-industrialization/.