

**Universidade do Minho**  
Escola de Ciências

Cecília Eduarda Coelho Machado da Cruz Martins

## **Machine Learning and Image Processing**

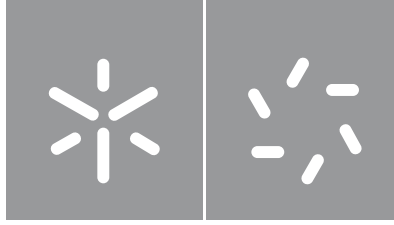
**Machine Learning and Image  
Processing**

Cecília Eduarda Coelho

UMinho | 2020

julho de 2020





**Universidade do Minho**

Escola de Ciências

Cecília Eduarda Coelho Machado da Cruz  
Martins

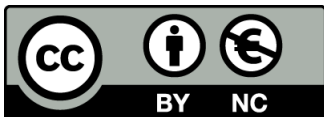
## **Machine Learning and Image Processing**

Dissertação de Mestrado  
em Matemática e Computação

Trabalho efetuado sob a orientação do  
**Professor Doutor Luís Jorge Lima Ferrás**  
e da  
**Professora Doutora Maria Fernanda Pires da  
Costa**

## Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos. Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada. Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.



**Atribuição-NãoComercial-CompartilhaIgual**

**CC BY-NC-SA**

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

## Acknowledgements

First of all, I would like to acknowledge my supervisors, Professor Doctor Luís and Professor Doctor Fernanda for all their availability, concern and kindness during this process and for the flexibility given, by supporting and approving the concepts I wanted to use to achieve the goal of this thesis, after a lengthy session of questions. I am also grateful for the help of Professor Doctor Ana Jacinta for teaming up with my supervisors and contributing in the same terms.

Furthermore, I want to express my gratitude to Accenture for the experience and for suggesting to address the detection of swimming pools in satellite images based on the theme of this thesis. Also, I want to thank my two supervisors provided by Accenture, Cristiana and Guilherme. Although they were busy they never forgot to check on me and their advice and opinions can be found scattered through all this work.

To Diana and Luís for accompanying me in this journey. Thank you for your help, patience and motivation. No day would pass without some good laughter. To Ozzy, the pseudo engineer and future physicist that recently discovered the wonderful world of Machine Learning. Could not forget Fernando and João for all the hardships we overcame together in this master's degree and for being the best work group I have ever encountered. Finally, I want to express my greatest appreciation for my mother and for all the people that have always supported me in all my choices, if I could go back in time I wouldn't change a thing.

*To my grandfather and to the child that dreamt about working with cutting edge technology.*



## **Statement of Integrity**

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.





## Resumo

### Machine Learning and Image Processing

A legislação Portuguesa declara a obrigatoriedade da comunicação de novas construções, como piscinas, à Autoridade Tributária e Aduaneira. Esta comunicação permite o ajustamento do Imposto Municipal sobre Imóveis a pagar anualmente pelo proprietário. De acordo com o *Technavio* e o *MarketWatch*, irá ocorrer um aumento significativo do número de piscinas devido a vários fatores como a redução do custo da construção, o aumento da consciência para a adoção de um estilo de vida saudável, entre outros. Isto leva à necessidade de um reforço na inspeção de forma a garantir que todas as novas construções foram devidamente comunicadas à autoridade competente. Atualmente, estas inspeções são realizadas com a distribuição de recursos humanos pelo terreno, o que traz um elevado custo operacional e temporal, impedindo uma catalogação a uma taxa próxima da de construção.

Hoje em dia, a automatação de tarefas está a tornar-se muito requisitada devido a permitir o aumento da eficiência e a redução de custos. Recentemente, os algoritmos de *Deep Learning* tem demonstrado resultados incríveis quando usados para deteção de objetos.

O objetivo desta dissertação é o estudo dos vários algoritmos de deteção de objetos existentes e a implementação de um modelo de *Deep Learning* capaz de detetar piscinas em imagens satélite. De forma a obter os melhores resultados na tarefa em questão, o algoritmo RetinaNet foi usado. Além disso e com o intuito de melhorar a experiência na utilização do modelo desenvolvido, foi construída uma interface gráfica simples.

**Palavras-chave:** Visão por Computador, *Deep Learning*, Deteção de Objetos, *ResNet*, *RetinaNet*



## Abstract

### Machine Learning and Image Processing

Portuguese legislation states the compulsory reporting of the addition of amenities, such as swimming pools, to the Portuguese tax authority. The purpose is to update the property tax value, to be charged annually to the owner of each real estate. According to *Technavio* and *MarketWatch*, this decade will bring a global rise to the number of swimming pools due to certain factors such as: cost reduction, increasing health consciousness, and others. The need for inspections to ensure that all new constructions are communicated to the competent authorities is therefore rapidly increasing and new solutions are needed to address this problem. Typically, supervision is done by sending human resources to the field, involving huge time and resource consumption, and preventing the catalogue from updating at a rate close to the speed of construction.

Automation is rapidly becoming an absolute requirement to improve task efficiency and affordability. Recently, Deep Learning algorithms have shown incredible performance results when used for object detection tasks.

Based on the above, the objective of this thesis is to study the various existing object detection algorithms and implement a Deep Learning model capable of recognising swimming pools from satellite images. To achieve the best results for this specific task, the RetinaNet algorithm was chosen. To provide a smooth user experience with the developed model, a simple graphical user interface was also created.

**Keywords:** Computer Vision, Deep Learning, Object Detection, ResNet, RetinaNet



# Contents

<b>1 Introduction</b>	<b>1</b>
1.1 State-of-the-Art	3
1.2 Motivation and Goal	6
1.3 Dissertation's Structure	7
<b>2 Deep Learning for Object Detection</b>	<b>9</b>
2.1 Object Detection	11
2.2 Artificial Neural Networks	13
2.3 Convolutional Neural Networks	16
2.4 Region-based Convolutional Neural Network	20
2.5 Fast Region-based Convolutional Neural Network	24
2.6 Faster Region-based Convolutional Neural Network	26
2.7 You Only Look Once	29
2.8 Single-Shot Detector	31
2.9 RetinaNet	31
2.9.1 Focal Loss	31
2.9.2 Feature Pyramid Network	32
2.9.3 Network Architecture	34
<b>3 Development Tools and Dataset</b>	<b>37</b>
3.1 Programming Language	39
3.2 Graphical design tools	40
3.3 Dataset	40
<b>4 Training RetinaNet</b>	<b>43</b>
4.1 Data Preprocessing of the Training Set - Pascal VOC to CSV	45
4.2 Training of RetinaNet Models	46
<b>5 Testing RetinaNet - Results and Discussion</b>	<b>49</b>

5.1 Performance Metrics	51
5.2 Data Preprocessing of the Test Set	53
5.3 Test Set from Kaggle	55
5.3.1 Model using ResNet50	55
5.3.2 Model using ResNet101	57
5.3.3 Model using ResNet152	58
5.3.4 Conclusions	59
5.4 Test set from Google maps	60
5.4.1 Model using ResNet50	62
5.4.2 Model using ResNet101	63
5.4.3 Model using ResNet152	64
5.4.4 Conclusions	66
<b>6 Graphical User Interface</b>	<b>67</b>
<b>7 Conclusions and Future Work</b>	<b>75</b>
7.1 Conclusions	77
7.2 Future Work	77
<b>A</b>	<b>79</b>
A.1 Intersection over Union (IoU)	79
<b>B</b>	<b>81</b>
B.1 Region Proposal Networks	81
<b>C</b>	<b>83</b>
C.1 ResNet	83
<b>D</b>	<b>87</b>
D.1 ImageNet	87

# List of Figures

1.1	Representing Images on Multiple Layers of Abstraction in Deep Learning.	6
2.1	An RGB image is composed of three colour channels (red, green and blue) that when blended together, the image is computed.	12
2.2	(a) Traditional computer vision methods vs. (b) Deep learning workflow [1].	13
2.3	Components of a biological neuron network [2].	14
2.4	Components of an artificial neuron, designated by perceptron [2].	14
2.5	Composition of an Artificial Neural Network with a single-hidden layer.	15
2.6	Composition of a Deep Artificial Neural Network.	15
2.7	A $6 \times 6 \times 3$ RGB image example is seen by the computer as a matrix. When a filter is applied to each colour channel, the output is a convoluted matrix. The result is a $4 \times 4 \times 1$ matrix. (In the figure only one colour channel is shown.)	17
2.8	A Max Pooling layer applied to the input image in figure 2.7, results in the reduced feature map of $2 \times 2 \times 1$ . It was used a stride of 3 and a Pooling size of $3 \times 3 \times 1$ .	18
2.9	Components of a convolutional neural network [3].	19
2.10	Input image given to the convolutional neural network [4].	19
2.11	Division of the input image in several new independent images.	19
2.12	The new images are given to the network.	20
2.13	Original image with detected objects.	20
2.14	Input image [4].	21
2.15	Random regions generation (not all bounding boxes are displayed).	21
2.16	Aggregation of small regions by patterns identification (not all bounding boxes are displayed).	22
2.17	Final aggregation of regions and identification of regions of interest (not all bounding boxes are displayed).	22
2.18	Detected region of interest.	23
2.19	One SVM for each class to be detected and one for background classification.	23
2.20	Original image with detected objects.	24
2.21	Input image [4].	24

2.22	Detected regions of interest.	25
2.23	The reshaped regions with a RoI pooling layer are passed into a fully connected network.	25
2.24	Bounding boxes reshaped and classified.	26
2.25	Image given as input [4].	26
2.26	The input is given to convolutional layers that produce a feature map.	27
2.27	RoI are detected from the feature maps.	27
2.28	To create the input for a Fully-connected Network, the regions are reshaped using a RoI pooling layer.	28
2.29	Bounding boxes reshaped and classified.	28
2.30	Input image [4].	29
2.31	Input image divided into an $S \times S$ grid.	29
2.32	Input image with randomly drawn bounding boxes.	30
2.33	Filtered bounding boxes according to the threshold value chosen by the user.	30
2.34	Graph that shows the loss functions values for each prediction. The cross entropy represented by the blue line and the other full-lines are the focal loss function with a $\gamma$ varying from 0.5 to 5 [5].	32
2.35	Illustration of the bottom-up and top-down pyramid along with a lateral connection scheme [6].	33
2.36	RetinaNet network architecture composed of four main components: a) bottom-up pathway; b) top-down pathway; c) classification subnetwork; d) regression subnetwork; [5].	34
3.1	Four training images of the chosen dataset [4].	40
5.1	Testing set image annotation using <i>LabelImg</i> [7].	53
5.2	Two testing images, containing swimming pools, of the chosen dataset used for evaluating the model's performance [4].	55
5.3	Two testing images without swimming pools, of the chosen dataset, used for evaluating the model's performance [4].	55
5.4	Model's predictions of two testing images of the dataset.	56
5.5	Model's predictions of two testing images without swimming pools.	56
5.6	Model's predictions of two testing images of the dataset using ResNet101 as backbone architecture.	57
5.7	Model's predictions of two testing images without swimming pools.	57
5.8	Model's predictions of two testing images of the dataset using ResNet152 as backbone architecture.	58
5.9	Model's predictions of two testing images without swimming pools.	59



5.10 Eight test images taken from google maps with different sizes and resolutions. The images are numbered in yellow on the top right corner.	61
5.11 Eight test images taken from google maps with different sizes and resolutions. The detected objects are enclosed by blue boxes. The images are numbered on the top right corner.	62
5.12 Eight test images taken from google maps with different sizes and resolutions. The detected objects are enclosed by blue boxes. The images are numbered on the top right corner.	63
5.13 Eight test images taken from google maps with different sizes and resolutions. The detected objects are enclosed by blue boxes. The images are numbered on the top right corner.	65
6.1 Desktop application main window.	69
6.2 "Save Path" button action.	70
6.3 "Upload Image" button action.	70
6.4 "Upload Folder" button action.	71
6.5 "Run" button action.	71
6.6 "Show" button action.	72
6.7 Error prompt shown where the saving path and/or at least one image are not provided.	72
6.8 An error message prompts if the previous (left arrow) or next (right arrow) image is clicked but there are no more images with bounding boxes to show.	73
B.1 Region Proposal Networks architecture [8].	81
C.1 Adding more layers to a neural network in which the best possible state was attained results on the following layers learning the identity function.	83
C.2 Training error (left) and test error (right) on CIFAR-10 [4] with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error [9].	84
C.3 A residual block where the layer's input is added to the output [9].	84
C.4 Classical neural networks on the left, ResNets on the right. Dashed lines denote training error, and bold lines denote testing error (training on CIFAR-10 [4] [9]).	85



# List of Tables

5.1	The confusion matrix. . . . .	51
5.2	Confusion matrix computed with the results obtained by comparing predictions and ground truth for the model that uses ResNet50 as backbone architecture. . . . .	56
5.3	Confusion matrix computed with the results of comparing predictions and ground truth for the model that uses ResNet101 as backbone architecture. . . . .	58
5.4	Confusion matrix computed with the results of comparing predictions and ground truth for the model that uses ResNet152 as backbone architecture. . . . .	59
5.5	Confusion matrix values and computed evaluation quantities for each of the trained models. . . . .	59



# List of Abbreviations

<b>ANN (or NN)</b>	Artificial Neural Network
<b>CNN</b>	Convolutional Neural Network
<b>CPU</b>	Central Processing Unit
<b>CSV</b>	Comma-Separated Values
<b>CV</b>	Computer Vision
<b>DL</b>	Deep Learning
<b>DNN</b>	Deep Neural Network
<b>EUSA</b>	European Union of Swimming Pool and Associations
<b>FCL</b>	Fully-Connected Layer
<b>FCN</b>	Fully-Connected Network
<b>FPN</b>	Feature Pyramid Network
<b>GUI</b>	Graphical User Interface
<b>IMI</b>	Imposto Municipal sobre Imóveis
<b>IoU</b>	Intersection over Union
<b>MCC</b>	Matthews Correlation Coefficient
<b>ML</b>	Machine Learning
<b>RCNN</b>	Region-based Convolutional Neural Network
<b>ReLU</b>	Rectified Linear Unit
<b>RGB</b>	Red, Green, Blue
<b>RoI</b>	Region of Interest
<b>RPN</b>	Region Proposal Network
<b>SVM</b>	Support Vector Machine
<b>UK</b>	United Kingdom
<b>XML</b>	Extensive Markup Language



# Chapter 1

## Introduction





## 1.1 State-of-the-Art

Fifty years ago, Machine Learning was still science fiction. Today it's an integral part of our lives, helping us do everything from finding photos to driving cars.

Machine Learning has evolved along time, mainly due to philosophers, filmmakers, mathematicians, and computer scientists who fuelled the dream of learning machines. One can distinguish three milestones.

**1642:** Blaise Pascal was 19 when he made an *arithmetic machine* for his tax collector father. It could add, subtract, multiply, and divide. Three centuries later, Machine Learning is used to combat tax evasion in income taxes.

- 1679: German mathematician, philosopher, and occasional poet Gottfried Wilhelm Leibniz devised the system of binary code that laid the foundation for modern computing;
- 1847: Philosopher George Boole created a form of algebra in which all values can be reduced to true or false. Essential to modern computing, Boolean logic helps a CPU (Central Processing Unit) decide how to process new inputs;
- 1936: Inspired by how we follow specific processes to perform tasks, English logician and cryptanalyst Alan Turing theorised how a machine might decipher and execute a set of instructions. His published proof is considered the basis of computer science;

**1943:** A neurophysiologist and a mathematician co-wrote a paper on how human neurons might work. To illustrate the theory, they modelled a neural network with electrical circuits. In the 1950s, computer scientists would begin applying the idea to their work.

- 1952: Machine Learning pioneer Arthur Samuel created a program that helped an IBM computer get better at checkers the more it played. Machine Learning scientists often use board games because they are both understandable and complex. Arthur Samuel is considered by many as the father of Machine Learning;
- 1959: In computing, a neural network is a system modeled on the human nervous system. The first neural network applied to a real world problem, Stanford's MADALINE used an adaptive filter to remove echoes over phone lines. It's still in use today;
- 1985: Invented by Terry Sejnowski and Charles Rosenberg, this artificial neural network taught itself how to correctly pronounce 20,000 words in one week. Early outputs sounded like gibberish, but with training its speech became clearer;
- 1997: IBM's Deep Blue beat chess grandmaster Garry Kasparov, it was the first time a computer had bested a human chess expert. Kasparov demanded a rematch, but IBM declined and immediately retired Deep Blue;
- 1999: Computers can't cure cancer, but they can help us diagnose it. The CAD Prototype Intelligent Workstation, developed at the University of Chicago, reviewed 22,000 mammograms and detected cancer 52% more accurately than radiologists did;

**2006:** When his field fell off the academic radar, computer scientist Geoffrey Hinton rebranded neural net research as *Deep Learning*. Today, the internet's heaviest hitters use his techniques to improve tools like voice recognition and image tagging.

- 2009: In 2006, Netflix offered \$1M to anyone who could beat its algorithm at predicting consumer film ratings. The BellKor team of AT&T scientists took the prize three years later, beating the second-place team by mere minutes;
- 2012: A neural network created by Google learned to recognise humans and cats in YouTube videos - without ever being told how to characterise either. It taught itself to detect felines with 74.8% accuracy and faces with 81.7%;
- 2015: When PayPal set out to fight fraud and money laundering on its website, it took a hybrid approach. Human detectives define the characteristics of criminal behaviour, then a Machine Learning program uses those parameters to root out the bad guys on the PayPal site;
- 2016: Oxford team develops read-lips. This artificial-intelligence system identified lip-read words with an accuracy of 93.4%;
- 2020: How far this technology will take us remains to be seen, but applications in the works span everything from improving in-store retail experiences with Internet of Things (IoT) to boosting security with biometric data to predicting and diagnosing disease;

Now the question is: what is Machine Learning? How do machines can learn?

Machine Learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine Learning algorithms build a mathematical model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.

It is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; the computer follows the algorithm and no learning is needed. For example, the instruction *sum the two numbers provided by the user* has no learning involved. If the computer follows the instructions, when the user inserts the values 3 and 4, the output given by the computer is 7.

For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than have human programmers specify every needed step. For example, imagine a set with several nature images, and some of those images have cats in it. It would be nice if the computer could automatically detect the cats, otherwise one would have to check picture by picture the existence of cats. How to *teach* the computer what is a cat?

The discipline of Machine Learning employs various approaches to help computers learn to accomplish tasks where no fully satisfactory algorithm is available. In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid (in the

example above, this technique consists in labelling some of the pictures with cats, so that the computer knows these pictures have cats). This set of pictures can then be used as training data for the computer to improve the algorithm(s) it uses to determine correct answers (if the picture has a cat or not). This technique is known as Supervised Learning.

**Supervised learning:** The computer is presented with example inputs and their desired outputs (for example, when the computer *sees* a picture of a cat - input, it should draw a rectangle/bounding box around the cat - output), given by a teacher, and the goal is to learn a general rule that maps inputs to outputs. For the example given above, this means that the more information given to the computer (the bigger the training set), the higher the probability to correctly detect pictures with cats.

Other ways a computer can learn are:

**Unsupervised Learning:** No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

**Reinforcement learning:** A computer program interacts with a dynamic environment in which it must perform a certain task (such as driving a vehicle or playing a game against an opponent). As it navigates its problem space, the program is provided feedback that is analogous to rewards, which it tries to maximise. For example, rewarding the computer when it correctly identifies a picture of a cat.

Other approaches exist in the literature that don't fit into this three-fold categorisation. For example topic modeling, dimensionality reduction or meta learning. As of 2020, Deep Learning (DL) has become the dominant approach for much ongoing work in the field of Machine Learning.

**Deep Learning** is a class of Machine Learning algorithms that uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces. Therefore, Deep Learning is the technique of choice for detecting specific objects in images. The word *deep* in Deep Learning refers to the number of layers through which the data is transformed (figure [1.1](#)).

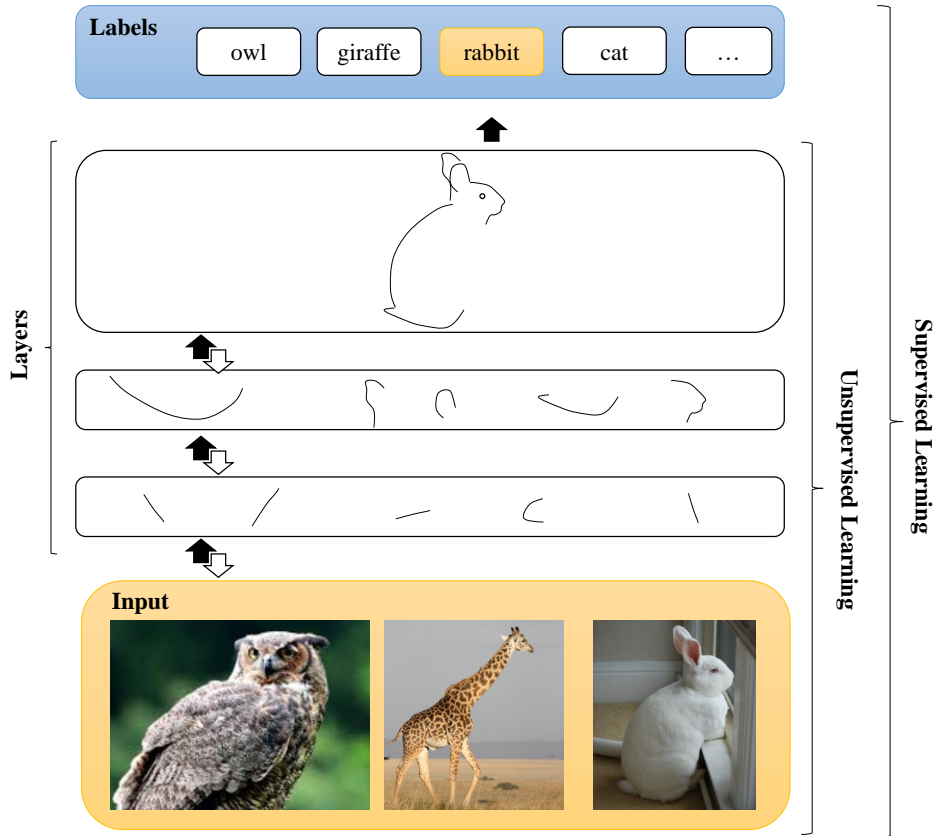


Figure 1.1: Representing Images on Multiple Layers of Abstraction in Deep Learning.

In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognise that the image contains a face. Importantly, a DL process can learn which features to optimally place in which level on its own. (Of course, this does not completely eliminate the need for hand-tuning; for example, varying numbers of layers and layer sizes can provide different degrees of abstraction.)

Based on the objective of this work, which is the detection of swimming pools by using satellite images, Deep Learning seems to be the ideal technique to be used in order to achieve the goal of this work.

## 1.2 Motivation and Goal

According to *Technavio* [10] and *MarketWatch* [11], there will be a global rise in the number of swimming pools in this decade. This may be related to the increasing number of residential construction projects, awareness of the benefits of swimming, the decreasing cost of building swimming pools, or the increasing demand for attractive amenities in residential complexes.

Data collected by the European Union Swimming Pool and Spa Association (EUSA), which gathers the latest pool sector volume and market characteristics, released in Barcelona in 2009, show that among a population of 339,6 million people, there were 3,73 million swimming pools manufactured till 2009. In Portugal, there was one pool per 133 people, being 90% inground. It beats Italy and the United Kingdom (UK) with 290 and 261 people per pool, respectively [12]. Although these statistics indicate the presence of a large number of swimming pools in Portugal, in fact there are many more that were unaccounted for due to the failure to communicate these constructions. Portuguese legislation, since the fourth of September of 2018 by law number 60/2007 [13], states that it is mandatory to request permission to the local town hall before constructing a swimming pool. It is necessary to communicate the addition of this property to the Portuguese tax authority. Otherwise the structure will be considered illegal [14].

There are many possible reasons for the existence of so many illegal swimming pools. For instance, lack of regulation information and common knowledge of poor inspection by the authorities, avoidance of 6% increment in the Portuguese properties tax, called *Imposto Municipal sobre Imóveis* (IMI), when this amenity is added [15]. The existence of a pool in a property is translated into a positive variation of a coefficient used in the formula to compute the tax value [16]. This culminates in huge losses to the government of the country, as each year proper taxes are not collected correctly. To detect illegal swimming pools, inspection companies are hired. These companies mostly rely on private complaints and human resources that do door-to-door inspection to verify whether everything is accounted for. This method consumes a lot of human resources and time, becoming impossible to detect every single illegal pool.

Although satellite images aren't updated often, these are a great alternative to sending people door to door. It allows the detection of pools from several areas without leaving the office. This task is very repetitive and is influenced by each person's ability to recognise the structures. To overcome this issue in an efficient manner, in this thesis is proposed the use of a model capable of detecting swimming pools using satellite images without human intervention. To achieve this, Deep Learning algorithms, powered by a collection of data composed by satellite images (dataset), were analysed. Furthermore, a user friendly graphical user interface was developed for detecting illegal swimming pools.

### 1.3 Dissertation's Structure

This dissertation is structured into seven chapters: Introduction; Deep Learning for Object Detection; Development Tools and Dataset; Training RetinaNet; Testing RetinaNet - Results and Discussion; Graphical User Interface; and Conclusions and Future Work. In addition, four appendices are provided: Intersection over Union (IoU); Region Proposal Networks; ResNet; and ImageNet. The contents of each chapter are shortly described in the following topics:

- Chapter 1: Introduction

In this chapter is presented the problem addressed in this thesis: illegal swimming pool detection using satellite images. It starts by giving a brief description of the history of

Machine Learning, followed by the motivation and goal of this work;

- Chapter 2: Deep Learning for Object Detection

This chapter focus on objected detection and it gives a brief description of the the most promising object detection Deep Learning algorithms. The advantages and disadvantages of these algorithms are analysed when detecting swimming pools using satellite images;

- Chapter 3: Development Tools and Dataset

In this chapter is presented the programming language (*Python* and its libraries) used for the development of the Deep Learning scripts and the framework (*Electron*) used in the development of the Graphical User Interface. Finally, it is presented the dataset used in this work;

- Chapter 4: Training RetinaNet

In this chapter is described the data preprocessing, needed to fulfil the RetinaNet input requirements. Three separate models were trained, each with a different RetinaNet backbone architecture: ResNet50, ResNet101 and ResNet152;

- Chapter 5: Testing RetinaNet - Results and Discussion

In this chapter are presented the metrics for evaluating the model's performance. The three models, were tested using the dataset test satellite images as well as a dataset test extracted from *Google Maps*. The results obtained are discussed;

- Chapter 6: Graphical User Interface

In this chapter is presented the Graphic User Interface developed in order to make the process of detecting illegal swimming pools (using satellite images) user friendly;

- Chapter 7: Conclusions and Future Work

In this chapter are presented the conclusions and future work of this thesis.

## Chapter 2

# Deep Learning for Object Detection





In this chapter, a brief description on object detection and the most prominent Deep Learning techniques for object detection are presented.

## 2.1 Object Detection

When looking at an image, the human brain has the ability to instantly locate and recognise different objects on that image (object classification). Nowadays, the extraction of information from a digital image or video is becoming a necessity due to the increasing number of applications that rely on it. For instance, face recognition [17], autonomous driving [18], and pedestrian detection [19]. In the field of Computer Vision (CV), this challenging problem is called object detection [20].

In object detection, the goal is to locate and classify objects in an input image by outputting bounding boxes for each object, and a class [1] label for each box. That is, given an input image  $X$ , the output is expected to be in the form  $(X, Y)$  where  $Y$  is an array containing the class label and the box edge's coordinates.

An image is a visual representation of a real-life object. Each image is made from small boxes called pixels. Each pixel's colour is represented by an RGB (Red, Green, Blue) number from a colour space where each colour's intensity is in the range of 0 to 255 [21]. This colour space is used because it mimics the way the human eye works (the human eye has cone cells located in the retina that are able to identify three colours (red, green and blue) and their combinations [22]).

Computers store RGB images in the form of three matrices (or channels), one for each main colour, where each pixel is converted to a number between 0 and 255 for each channel. When three colour matrices are blended, by adding the values of each pixel, the image is computed. An example is shown in figure 2.1

---

<sup>1</sup>A class is a group formed by objects with common attributes or characteristics and usually it refers to the object's name, "chair" for example.

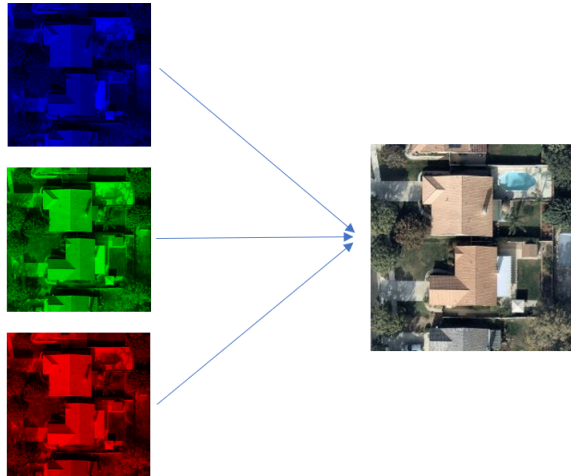


Figure 2.1: An RGB image is composed of three colour channels (red, green and blue) that when blended together, the image is computed.

Traditionally, the CV techniques used for object detection are based on feature descriptors. These are a part of the broader family of Machine Learning methods. These have the ability to identify/extract features for each object class, a small list of interesting points that describe an image's content (points, edges or colour variations) and that allow for image classification <sup>2</sup>. To accomplish this, several CV algorithms are used, namely edge detection, corner detection or threshold segmentation <sup>23</sup>. For example, during training for image classification, all the detected interesting points (the most important features) form a definition's list of the object class. When classifying new images, if a significant number of points belong to the definition list, then the image is classified as belonging to that class (for more information on feature descriptors refer to <sup>24</sup>). If an image contains several objects (classes) to be identified, feature extraction becomes computationally expensive. This problem will be aggravated with the increasing number of features per class.

To prevent from this, the user needs to go through a trial and error process to determine which features best describe the different classes <sup>25</sup>.

With the use of Deep Learning methods, the computer receives several images, in which the classes present have been annotated, and the DL method <sup>3</sup> is able to discover the most descriptive features of each class by itself, acting as a human in feature descriptors. This allows for better performance than the traditional methods with the disadvantage of the user not being able to know which features were given higher emphasis (it is used as a black box procedure). The difference between a ML and a DL workflow is represented in figure <sup>2.2</sup>

---

<sup>2</sup>Image classification is a computer vision process in which an image is classified based on its visual contents.

<sup>3</sup>A DL method is a computer program developed to simulate the outcome of a situation.

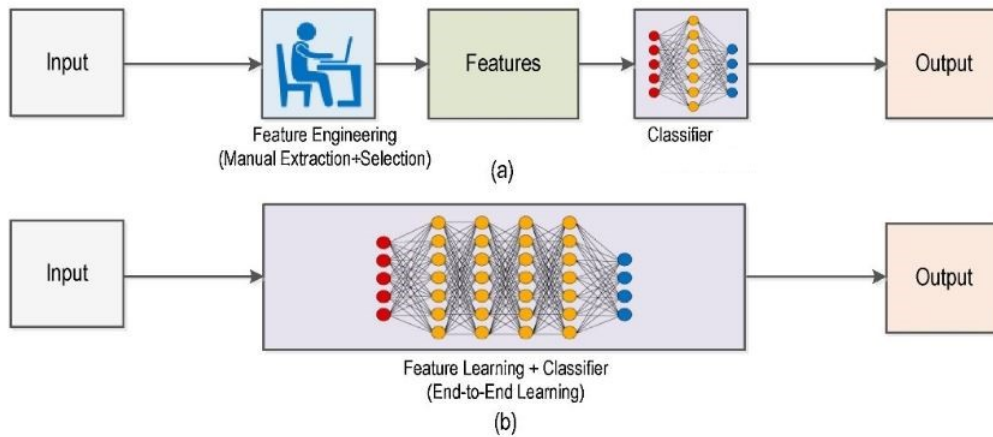


Figure 2.2: (a) Traditional computer vision methods vs. (b) Deep learning workflow [1].

## 2.2 Artificial Neural Networks

As already mentioned, Deep Learning [26] is a part of a broader family of Machine Learning methods based on Artificial Neural Networks (ANN or NN) with feature learning (also known as representation learning). ANN were inspired by information processing and distributed communication nodes in biological systems [27]. ANN have various differences from biological brains. Specifically, NN tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analog (encodes information as a continuum).

ANN may contain many layers of neurons that are used to train models by using an extensive amount of labelled data. This can be described as mathematical models of the human brain with the purpose of processing nonlinear relations between inputs and outputs [27].

It should be remarked that Deep Learning architectures have produced results comparable to and in some cases surpassing human expert performance [27].

### Artificial Neurons

Artificial neurons are the most elementary unit of an ANN. In the human brain, when an electrical signal is transmitted due to an external stimuli, through the dendrites, a neuron receives signals as inputs and sends output signals to other neurons via synapses [28], this is pictured in figure 2.3.

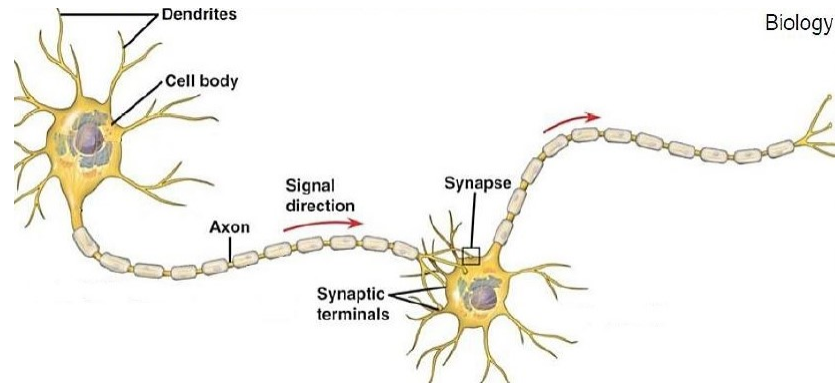


Figure 2.3: Components of a biological neuron network [2].

Similarly, in an artificial neuron (often called a perceptron) the received signals ( $x_i$ ) interact in a multiplicative way with the dendrites based on certain weights ( $w_i$ ) (expressing the strength of the synapse). The weights are learned by the network and control the impact that one neuron has on another. This impact can be excitatory or inhibitory according to positive or negative weights respectively. Additionally, a bias,  $b$ , that stores a value is added to the sum of the weighted inputs. This bias is not influenced by the weights but it contributes to the final output result, allowing the user to control the behaviour of the layer [4] [29]. A representation of an artificial neuron can be viewed in figure 2.4.

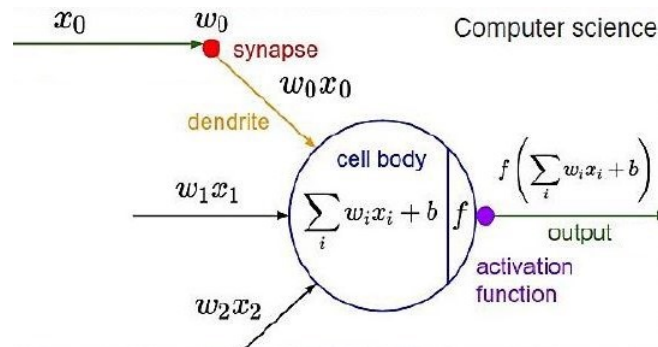


Figure 2.4: Components of an artificial neuron, designated by perceptron [2].

If there are multiple inputs to the network ( $x_0, x_1, x_2, \dots, x_n$ ), each is multiplied by a weight (synapse) ( $w_0, w_1, w_2, \dots, w_n$ ). In order to generate a result, the products are summed and fed to an activation function [29].

An activation function introduces non-linear properties to a NN . It determines whether a neuron is activated or not if its value is above a certain threshold. Based on the final result, the next layer of neurons may be activated [30]. This leads the NN to learn complex connections between input and output, this process is often designated by training.

An ANN can be built by stacking numerous perceptrons into layers, as seen in figure 2.5. Thus,

<sup>4</sup>A layer refers to a group of neurons that work together at a certain depth of a NN, figure 2.5

the most basic ANN has three components: an input layer that receives the provided data to the NN; a hidden layer that has the task of discovering relationships between the features in the input; and an output layer that produces the result of the given inputs [30].

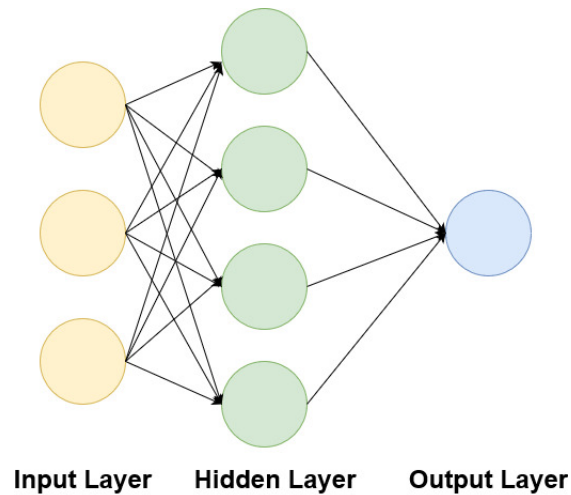


Figure 2.5: Composition of an Artificial Neural Network with a single-hidden layer.

Deep learning is introduced when there is more than one hidden layer, meaning that the difference between a single-hidden layer ANN and Deep Learning lies in the depth of the model [31]. Figure 2.6 shows an example of a Deep Neural Network. The activation function must be a non-linear function so that adding more hidden layers contributes to obtain a better model. If a linear function is used, the neural network will behave like a perceptron due to the fact of the sum of several linear functions being a linear function. This means that adding more hidden layers won't produce a higher performance Deep Learning model and will result in a neural network lacking the ability to learn complex patterns from the data.

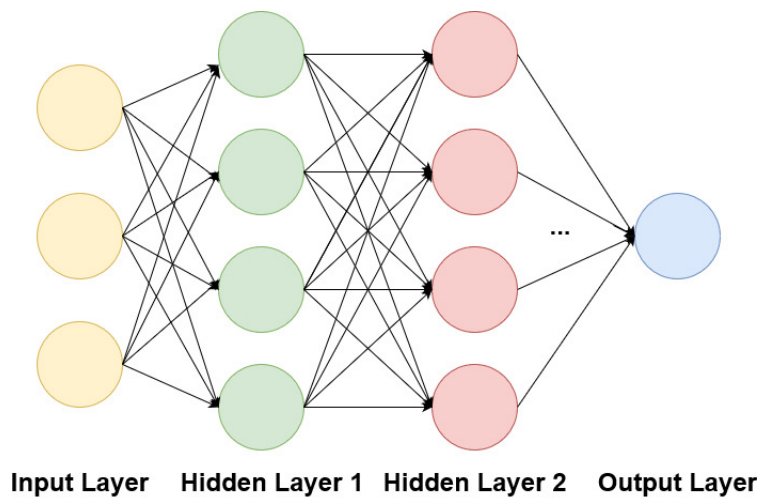


Figure 2.6: Composition of a Deep Artificial Neural Network.

The most common ANN are structured in a way where each neuron is connected to every other neuron in the next layer, named Feed-Forward ANN. However, better results can be achieved by connecting neurons to other neurons in different patterns, meaning that every possible combination can be studied (for more information on the most used patterns and their applications refer to [32], [29]).

When a Deep Neural Network (DNN) is used for classification, the last layer is given by a function which computes the probability of the input belonging to the classes in study (it is known by logistic regression). Two of the most used functions are the Softmax and the Sigmoid functions [33]. The Softmax function extends the logistic regression idea into a multi-class problem by normalising the input vector into a vector of values in which the total sum is 1. This allows the usage of as many classes as needed for the problem we are dealing with. Unlike the latter, the Sigmoid function outputs the predicted probabilities in the range of 0 and 1 being the total sum of these not necessarily 1. In a DNN it is necessary to define weights for each connection. During the DNN's training, the aim is to optimise the weights that minimise a loss function that measures the error, i.e., the difference between the expected values and those obtained by the network. There is no universal loss function and there are various factors to consider when choosing one for a given problem [34]. In this thesis, two loss functions will be presented, namely the cross entropy and focal loss.

## 2.3 Convolutional Neural Networks

Object detection can be achieved by ANN, but a computational problem arises due to the proportional increasing number of parameters of the ANN with the increasing number of the input features associated to the image (image's size). For instance, the two-dimensional image must be converted into a one-dimensional vector which is the input format required by an ANN.

For example, if the dataset has images of size  $64 \times 64 \times 3$  (since the image is RGB there will be 3 colour channels), then there will be 12288 features, but if the size is  $1000 \times 1000 \times 3$  then 3000000 features will be provided. This leads to an exponential increase of time, computational cost and to the loss of the spatial features (pixels arrangement) of an image [35]. An alternative that fixes these problems is the Convolutional Neural Networks (CNN).

A Convolutional Neural Network (CNN) takes an image as input, defines a weight matrix (explained later in this subsection), designated by filter, and then the input is convolved to extract specific features without losing the information about its spatial arrangement. In contrast to the ANN, the convolved matrices are smaller in size and consequently the CNN will have a smaller number of parameters. Thus, leading to a dramatically reduction of the time and computational cost. Just like the ANN, a CNN's number of parameters increases with the size of the input but at a slower rate since the input image is reduced to a feature matrix [35][31].

To set up a basic Convolutional Neural Network it is necessary to define three basic components:

1. **Convolutional layer:** Every image can be represented as a matrix of pixel values. The concept behind this layer is to define a weight matrix (filter) in order to compute the dot

product between the values of the filter and the image pixel matrix. This is accomplished by sliding the filter across the width and height of the input matrix. With this approach, it is possible to extract certain features from the images (for a detailed explanation on filters refer to [36]). The next example illustrates how this process works.

Suppose the input is an RGB image of size  $6 \times 6 \times 3$ . The weight matrix, a  $3 \times 3 \times 1$  filter, for example, runs across each channel of the image matrix in such a way that all the pixels are covered at least once (stride of 1), resulting in a convolved output. Each  $6 \times 6 \times 1$  input channel matrix is now converted into a  $4 \times 4 \times 1$  matrix as represented in figure 2.7

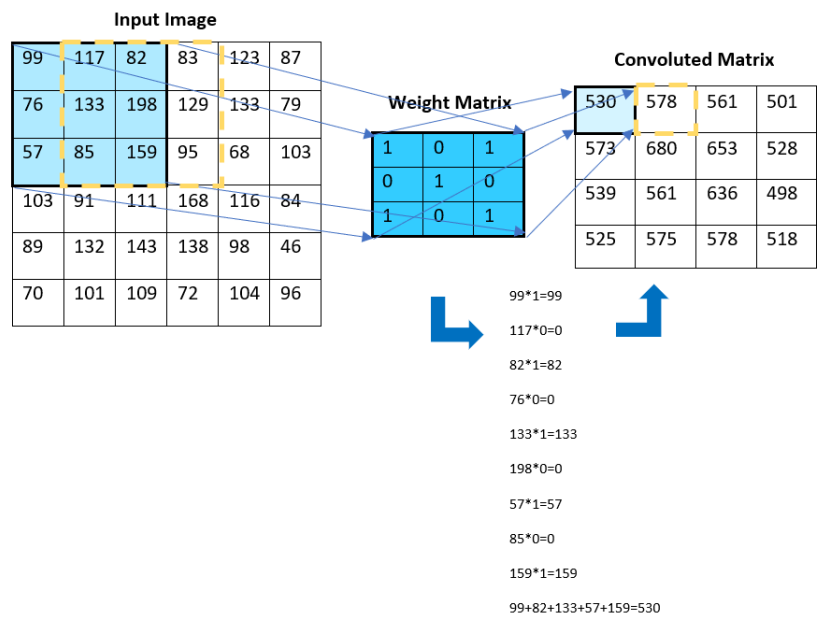


Figure 2.7: A  $6 \times 6 \times 3$  RGB image example is seen by the computer as a matrix. When a filter is applied to each colour channel, the output is a convoluted matrix. The result is a  $4 \times 4 \times 1$  matrix. (In the figure only one colour channel is shown.)

Several filters can be used as a weight combination with the aim of extracting different information, such as edges, a particular colour (one of the RGB channels as represented in figure 2.1, for example) or to remove unwanted noise. These filters have been studied and documented in the literature, the way the weights are distributed in the matrix is already predefined according to user needs. However, it is possible for the user to create a new filter. The output of the convolutional layer is called the activation map (or feature map) whose depth is equivalent to the number of filters applied. These filters slide across the input image with a jump of 1 or more pixels, the size of this jump is called stride. When applying several filters, these are applied to the input image, meaning that there will be the same number of feature maps as filters applied. It is important to mention that if it is desired to keep the output with the same size of the input image, the “zero” padding technique needs to be applied to the input image [37]. This technique appends a border of zeros around the image

and depending on the application it may have advantages [38].

2. **Pooling layer:** Due to the size of some inputs, it can be useful to periodically introduce pooling layers between subsequent convolution layers in order to reduce the spatial size of an image. This is achieved by reducing the dimension of the feature map. One of the most used pooling layers, Maximum Pooling (or Max Pooling), computes the maximum value for each patch (defined by the stride and pooling size, represented by the different colours in figure 2.8) of the feature map [39]. Given a stride and pooling size of 3, after the max operation is applied to each depth dimension of the convolved output, the  $6 \times 6 \times 1$  feature map becomes a  $2 \times 2 \times 1$  matrix, see figure 2.8.

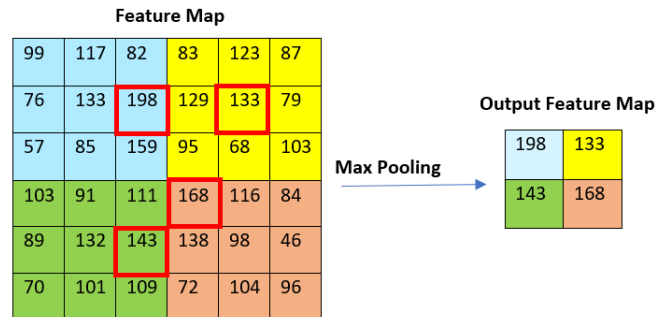


Figure 2.8: A Max Pooling layer applied to the input image in figure 2.7 results in the reduced feature map of  $2 \times 2 \times 1$ . It was used a stride of 3 and a Pooling size of  $3 \times 3 \times 1$ .

3. **Output layer:** The convolution and pooling layers are only able to extract features and reduce the number of features from the original images. In order to generate the desired output, it is required the use of Fully-Connected Layers (FCL). These layers are identical to the traditional DNN (see figure 2.6) with the addition of an activation function in the output layer. This output layer performs the task of classification by using the features extracted by the convolutional and pooling layers. The term “Fully-Connected” means that every neuron in the previous layer is connected to every neuron in the following layer. It is important to point out that if the output arriving to the FCL has more than one-dimension (more than one filter was applied) and since the last layer only receives a single vector of numbers, the last pooling layer output must be flattened into a single vector. Then, the flattened feature map is given as input to the FCL which gives the classes probability as output. [28] [31] [40].

The architecture of a CNN is schematised in figure 2.9.



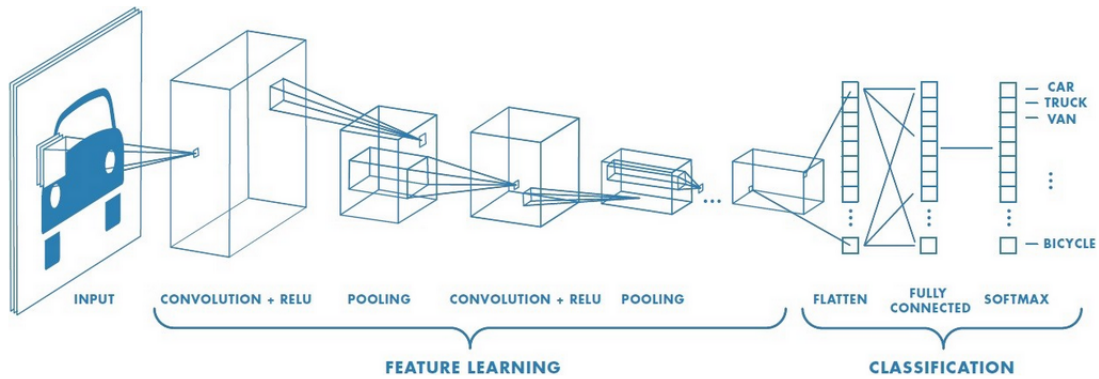


Figure 2.9: Components of a convolutional neural network [3].

The CNN algorithm steps will be illustrated considering the image detection of swimming pools in satellite images. It is assumed that the CNN was previously trained to detect swimming pools. The steps are the following:

1. Take an image as input (figure 2.10);

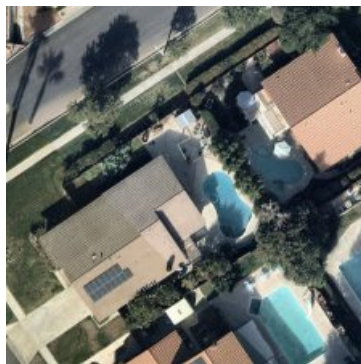


Figure 2.10: Input image given to the convolutional neural network [4].

2. Divide the image into various regions (figure 2.11);

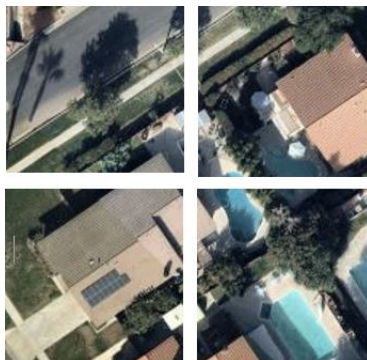


Figure 2.11: Division of the input image in several new independent images.

3. Consider each region as a separate image;
4. Pass all the new images to the CNN and get the classifications (figure 2.12);

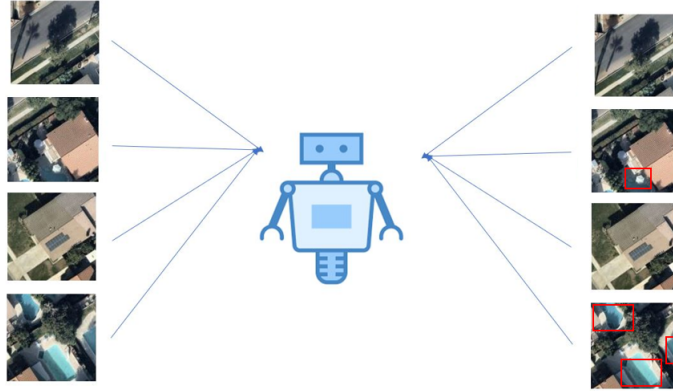


Figure 2.12: The new images are given to the network.

5. Combine all the images to get the original image with the detected objects (figure 2.13);



Figure 2.13: Original image with detected objects.

Since the objects in an image can have different aspect ratios and spatial locations, in order to improve the accuracy of the model for detecting smaller objects, a very large number of small regions would be required, resulting in high computational cost. The Region-based CNN comes to solve this problem and reduce the number of regions by locating objects using a proposed method.

## 2.4 Region-based Convolutional Neural Network

Rather than working on a massive number of regions, the Region-based Convolutional Neural Network (RCNN) algorithm proposes the creation of boxes (regions) in the image, obtained by selective search, and analyses whether those boxes contain any object [41].

There are four regions that form an object: these regions are based on varying scales, colours, textures and enclosure [42]. Selective search is able to identify these patterns in an image and

propose various regions based on that. Below is described how the selective search algorithm works when considering detection of swimming pools [42].

The steps of the selective search algorithm are the following:

1. First take an image as input (2.14);



Figure 2.14: Input image [4].

2. Generate multiple random regions to form the image (figure 2.15);

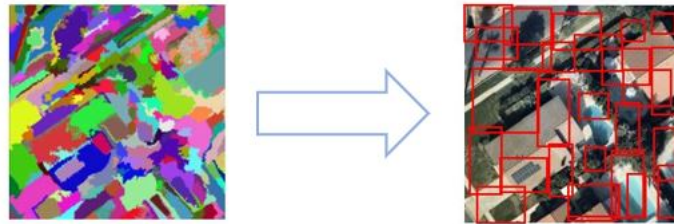


Figure 2.15: Random regions generation (not all bounding boxes are displayed).

3. Combine similar regions, based on colour, texture and size similarity and shape compatibility, to form a larger region (figure 2.16);

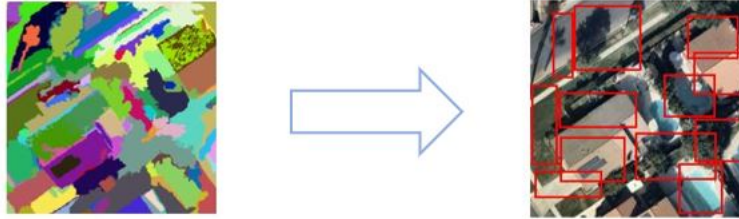


Figure 2.16: Aggregation of small regions by patterns identification (not all bounding boxes are displayed).

4. Produce the final object locations, Regions of Interest (RoI) (see figure 2.17);

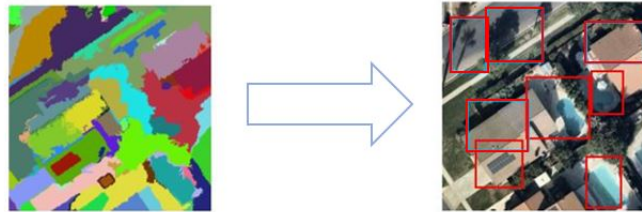


Figure 2.17: Final aggregation of regions and identification of regions of interest (not all bounding boxes are displayed).

The RCNN algorithm steps will be illustrated considering again the image detection of swimming pools. The steps are the following:

1. Get a pre-trained (explanation on pre-trained networks can be found in appendix D.1) Convolutional Neural Network;
2. Based on the number of classes to be detected, train the last layer of the network;
3. Get the RoI for each image and reshape it, into a unique vector, in order to be passed to a CNN (figure 2.18);



Figure 2.18: Detected region of interest.

4. For each class, train one binary Support Vector Machine (SVM) in a N-dimensional space (where N is the number of features of the data), that separates each class data points with maximum distance [43] to classify objects and one for background (figure 2.19);

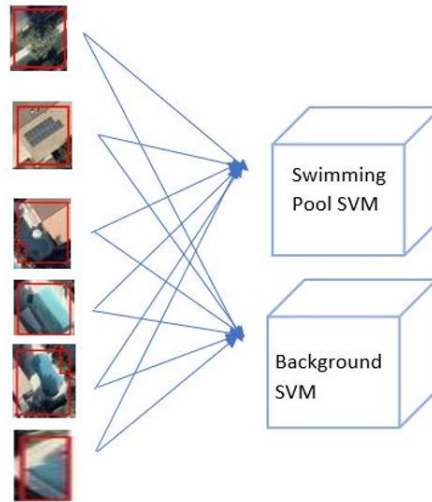


Figure 2.19: One SVM for each class to be detected and one for background classification.

5. For each identified object, train a linear regression model to generate tighter bounding boxes, usually referred as Bounding Box Regressor (figure 2.20);



Figure 2.20: Original image with detected objects.

The RCNN training is expensive and slow due to the expensiveness of the selective search algorithm and the need to have one CNN to perform feature extraction for each object detected. Using RCNN for object detection involves three different algorithms: a CNN for feature extraction; a SVM for objects classification; and a regression model for improving the fit of the bounding boxes.

## 2.5 Fast Region-based Convolutional Neural Network

The major improvement that comes with a Fast Region-based Convolutional Neural Network (Fast RCNN) over a RCNN is the combination of the three different methods used in the RCNN (CNN, SVM and regression) into one single architecture [4]. The steps involved in this method are the following:

1. Receive an image as input (figure 2.21);

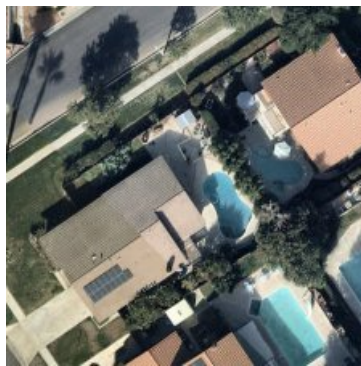


Figure 2.21: Input image [4].

2. Similarly to RCNN, generate the RoI by passing the input into a CNN (see figure 2.22);



Figure 2.22: Detected regions of interest.

3. Apply a RoI pooling layer to reshape the regions and pass each of them on to a Fully-connected Network (figure 2.23);

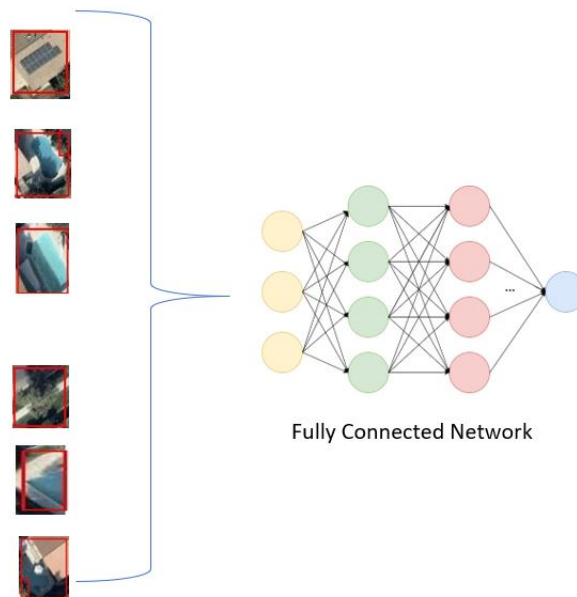


Figure 2.23: The reshaped regions with a RoI pooling layer are passed into a fully connected network.

4. The Fully-connected Network has two layers, used in parallel, with one output each. One decides the class of the object using a softmax layer and the other gives the bounding box coordinates for each object with a Bounding Box Regressor (figure 2.24);



Figure 2.24: Bounding boxes reshaped and classified.

To summarise, Fast RCNN uses a single model (CNN) for feature extraction, classification and bounding box fitting. This architecture, like RCNN, uses the selective search algorithm to find the RoI making this algorithm slow, so its usage is not advised for large real-life datasets. This problem was addressed in the literature, leading to the development of the Faster Region-based Convolutional Neural Network [8].

## 2.6 Faster Region-based Convolutional Neural Network

The Faster Region-based Convolutional Neural Network (Faster RCNN) is an adjusted version of Fast RCNN. The main difference between the two versions is that, instead of the selective search algorithm, Faster RCNN uses a Region Proposal Network (RPN) [8].

A RPN generates  $k$  anchor boxes of various random sizes and shapes by sliding a window over the feature maps outputted by the CNN. The RPN predicts, for each anchor box, the probability of that anchor being an object and the best-fit bounding box for that object (for more information on RPN see appendix B.1).

For performing object detection, the Faster RCNN approach uses the following steps:

1. Receive an image as an input (2.25);

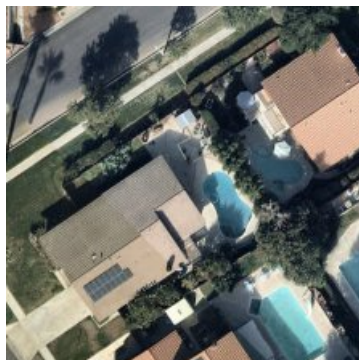


Figure 2.25: Image given as input [4].



2. Pass the input to the CNN and get the image's feature map (figure [2.26](#));

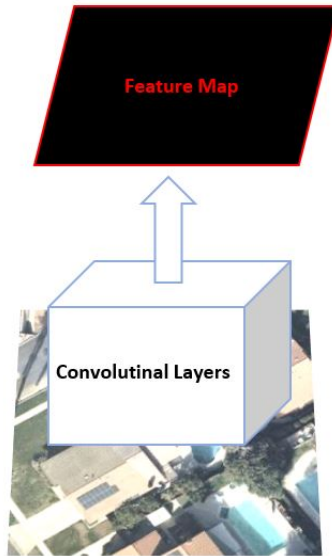


Figure 2.26: The input is given to convolutional layers that produce a feature map.

3. Apply the RPN on the feature map, which returns the object proposals and their corresponding objectness score that gives the probability of a box enclosing an object (figure [2.27](#));



Figure 2.27: RoI are detected from the feature maps.

4. Reshape the proposals using a RoI pooling layer, so all feature map's sizes match (figure [2.28](#)) (for more information on how this layer works see [45](#));

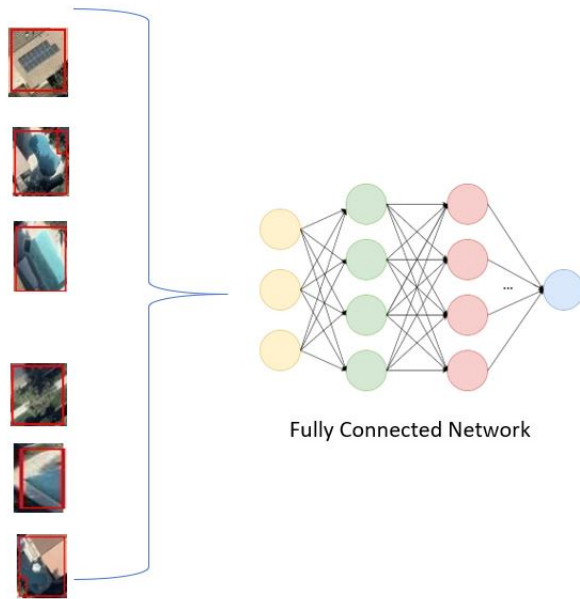


Figure 2.28: To create the input for a Fully-connected Network, the regions are reshaped using a RoI pooling layer.

5. Pass the resized proposals onto a Fully-connected Network with two layers. One softmax layer for classification and one linear regression to output the bounding boxes (figure 2.29);



Figure 2.29: Bounding boxes reshaped and classified.

Like every object detection algorithm discussed above, Faster RCNN identifies objects using regions. This implies that the network requires more than one pass through a single image in order to extract all the objects. Another downside of this algorithm is the handling of different systems working in sequence, making the performance of a system to depend on how the previous systems performed (error propagation).

## 2.7 You Only Look Once

The You Only Look Once (YOLO) algorithm uses the entire input image and predicts the bounding box and each corresponding class probability using a neural network similar to a CNN. The advantage of this algorithm is its remarkable speed.

The object detection using the YOLO algorithm is performed by considering the following steps:

1. Receive an image as input (figure 2.30);



Figure 2.30: Input image [4].

2. Divide the image into an  $S \times S$  grid of cells (figure 2.31);



Figure 2.31: Input image divided into an  $S \times S$  grid.

3. For each cell, predict  $N$  bounding boxes and the corresponding confidence scores. These scores report how confident the model is, i.e., it encloses an object and its prediction accuracy. Each bounding box is described by the vector  $(x, y, w, h, c)$  where:  $x$  and  $y$  are the coordinates of the centre relative to the grid cell;  $w$  and  $h$  are the width and height relative to the whole image, respectively;  $c$  is the confidence. The confidence is given by equation 2.1 [46]

$$c = P(\text{object}) \times IoU \quad (2.1)$$

where  $P(object)$  is the probability of the bounding box enclosing an object and  $IoU$  is the Intersection over Union (see appendix [A.1](#)) between the predicted box and the ground truth. In parallel, the network computes all the classes probability conditioned by the existence of an object in the bounding box,  $C = P(class_i|object)$ . Notice that only one set of class probabilities will be computed per cell, despite the number of existing bounding boxes (figure [2.32](#)). This step output is usually referred to as a feature map [46](#).

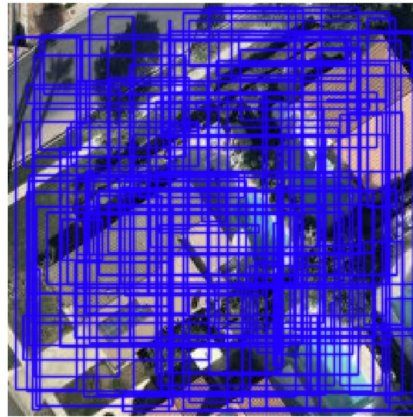


Figure 2.32: Input image with randomly drawn bounding boxes.

4. In order to get class-specific confidence scores for each box, the conditional class probabilities and the box confidence predictions are multiplied [46](#). At the end, according to a class probability threshold value provided by the user, the boxes that have the highest class probabilities are filtered in order to have one box for each detected object (figure [2.33](#)).



Figure 2.33: Filtered bounding boxes according to the threshold value chosen by the user.

## 2.8 Single-Shot Detector

The Single-Shot Detector (SSD) algorithm follows the same strategy as YOLO but instead of using a single feature map (for prediction of classes and bounding boxes) it uses several activation maps with different scales. Therefore, SSD may achieve higher precision due to the improved ability to detect different sized objects on an image.

The SSD algorithm divides the image into different sized  $S \times S$  grids of cells instead of just one. This allows the SSD to find smaller objects (smaller cells) or bigger objects (bigger cells) [47].

Despite the SSD precision, YOLO would be a better choice if speed is preferred over accuracy.

## 2.9 RetinaNet

The RetinaNet algorithm is a single-stage detector that has two new improvements over YOLO and SSD: Focal Loss and Feature Pyramid Network (FPN).

### 2.9.1 Focal Loss

Single-stage detectors, like YOLO and SSD, classify the whole image. In general, the background of an image represents a big part of the whole image, these detectors experience class imbalances since most of the bounding boxes do not contain any object. Unlike YOLO and SSD, two-stage detectors like RCNN, Fast-RCNN and Faster-RCNN, start by predicting a few object locations and then use a CNN to classify each of these objects. The RetinaNet algorithm was proposed in order to fix the class imbalances by slightly changing the loss function, as can be seen in figure 2.34

The loss function used by the other algorithms is commonly called cross-entropy and has the following expression, when computed for all the data (equation 2.2):

$$CE(p, y) = - \sum_t y_t \log p_t, \quad (2.2)$$

where  $t$  is the class index,  $y_t$  the label (1 if the object belongs to class  $t$ , 0 otherwise) and  $p_t$  is the probability of the object belonging to class  $t$ . Consider the following example: in an image, there are 10000 bounding boxes and the neural network predicts, with high accuracy, 9990 are background. If the other 10 boxes contain objects and the network isn't sure to which class they belong to, the loss of these few true objects will be much smaller than the background loss.

Therefore, the network won't focus on the minority in which it had difficulties due to the overpowering of the large number of easily classified boxes over the minority, which reflects in a low value of the loss function.

The creators of RetinaNet modified the cross-entropy loss function so that the contribution of the more easily classified examples (such as the background) is reduced and a more focused learning on the few interesting cases is performed (figure 2.34). This new loss function is called Focal loss and is given by [5]:

$$FL(p, y) = - \sum_t y_t (1 - p_t)^\gamma \log p_t, \quad (2.3)$$

where  $\gamma \in [0, 5]$  is a modulating factor that reduces the loss contribution from easily classified examples, as seen in figure 2.34. According to Tsung-yi Ling et al. [5],  $\gamma = 2$  is the best choice. Notice that a well (or easy) classified example is characterised by having a predicted probability above 0.6.

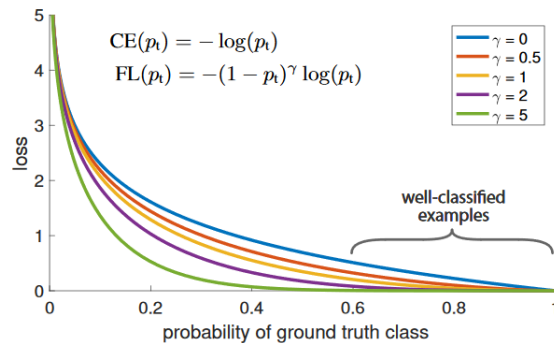


Figure 2.34: Graph that shows the loss functions values for each prediction. The cross entropy represented by the blue line and the other full-lines are the focal loss function with a  $\gamma$  varying from 0.5 to 5 [5].

The graph in figure 2.34 shows how the focal loss is able to give less importance to easy classified examples by using a modulating factor,  $\gamma$ . If a part of the image is easily classified then the  $(1 - p_t)$  factor will be close to zero, which induces a very low or no learning, giving more attention to the cases of interest. Note that equations 2.2 and 2.3 expressions are computed for all the dataset while in figure 2.34 the goal is to study the loss variation in a single example.

## 2.9.2 Feature Pyramid Network

Detecting objects in different scales is a difficult task, specially for smaller objects. To overcome this problem, a possible solution is to give the network several copies of the same image at different scales (resembling a pyramid). Processing multiple scale images is computationally expensive as well as time consuming, and therefore, a Feature Pyramid Network (FPN) is proposed.

The Feature Pyramid Network is a feature extractor that generates multiple multi-scale feature maps with focus on both accuracy and speed. The FPN comprises a bottom-up and a top-down pathway, represented in figure 2.35. The bottom-up pathway uses ResNet, and, for each layer, the spatial resolution is decreased by a factor of 1/2 by doubling the stride used for the next convolution block (which may contain several layers), allowing more high-level structures to be detected (there is an increase of semantic value). The output of each layer is a feature map that, by lateral connection, will be used for enriching the top-down pathway. In the top-down pathway, the spatial resolution is increased by a factor of 2 (for each layer) using the nearest neighbour

upsampling<sup>5</sup>. Although semantically strong, the reconstructed layers display imprecise object locations. To prevent this from happening, lateral connections are added between reconstructed layers and the corresponding feature maps by applying a  $1 \times 1$  convolution<sup>6</sup> to the feature maps and add them to the reconstructed layers, element-wise. A scheme is shown in figure 2.35<sup>6</sup>.

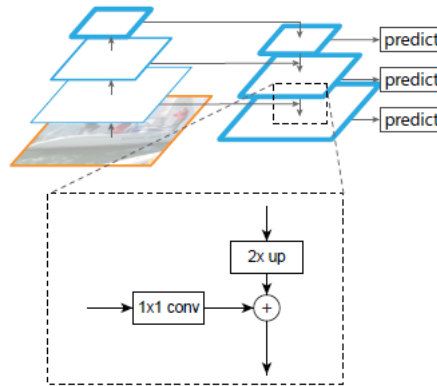


Figure 2.35: Illustration of the bottom-up and top-down pyramid along with a lateral connection scheme<sup>6</sup>.

<sup>5</sup>The nearest neighbour upsampling increases the size of images by assuming the new pixels have the same intensity as the closest pixel.

<sup>6</sup>A  $1 \times 1$  convolution collapses all the input pixel channels into one pixel, allowing for a reduction of the number of feature maps while retaining the salient features. This is useful to reduce the number of depth channels so as to reduce the computational cost by reducing the number of parameters that will be passed to the next phase of a neural network. For example, if the input has a size of  $64 \times 64 \times 3$  and a  $1 \times 1$  convolution is applied ( $1 \times 1 \times 3$  filter), the output will have the same size of the input but only one channel,  $64 \times 64 \times 1$ .

### 2.9.3 Network Architecture

In this subsection the main components of RetinaNet are summarised:

- **Bottom-up pathway:** Backbone network called Feature Pyramid Network (in this project, built on top of a ResNet) which computes several convolutional feature maps at different scales of an entire image, regardless of the input image size;
- **Top-down pathway and lateral connections:** Upsampling of the spatially coarser feature maps from higher pyramid levels. Same size top-down and bottom-up layers are associated by lateral connections;
- **Classification subnetwork:** Fully-Connected Network (FCN)<sup>7</sup> responsible for predicting the probability of an object to belong to an anchor box and to perform the classification of the given objects. This subnetwork is composed of four  $3 \times 3$  convolutional layers with 256 filters (number arbitrarily chosen by the authors for design simplicity and robustness<sup>6</sup>) followed by a Rectified Linear Unit (ReLU) activation function<sup>8</sup>. Finally, one more  $3 \times 3$  convolutional layer with  $K \times A$  filters is applied with a Sigmoid activation function, outputting  $(W, H, K \times A)$  feature maps (where  $W$  and  $H$  are the width and height of the classification subnetwork input map,  $K$  and  $A$  are the number of classes and anchor boxes, respectively), figure 2.36<sup>9</sup>. The activation function used in this stage can be either Sigmoid or Softmax, in this thesis the Sigmoid function was used;
- **Regression subnetwork:** Responsible for executing bounding box regression with the purpose of approximating the box to the ground-truth. The architecture is identical to the classification subnetwork with the exception of the last convolutional layer for which  $K = 4$ , outputting feature maps with shape  $(W, H, 4 \times A)$ , figure 2.36<sup>10</sup>;

Figure 2.36 shows the architecture of the RetinaNet.

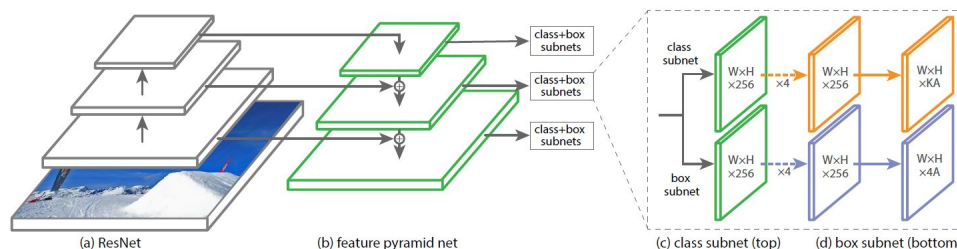


Figure 2.36: RetinaNet network architecture composed of four main components: a) bottom-up pathway; b) top-down pathway; c) classification subnetwork; d) regression subnetwork; <sup>5</sup>.

<sup>7</sup>A fully-connected network is an artificial neural network which makes usage of FCL.

<sup>8</sup>The ReLU activation function is a linear function that outputs the input if it is positive and 0 otherwise.

<sup>9</sup>For example, in a  $4 \times 4$  feature map, for each 16 grid cells, 16 different anchor boxes will be used by RetinaNet. Since each box will be looking for  $K$  classes and there are  $A$  boxes per grid, the output map of this subnetwork will have  $K \times A$  channels.

<sup>10</sup>This subnetwork outputs 4 coordinates that characterise each anchor box. Since there are  $A$  boxes per grid, the output feature map of the regression subnetwork will have  $4 \times A$  channels



As shown in figure [2.36](#), after the pyramid is concluded, a  $3 \times 3$  convolution is applied to each layer's map in order to generate the final feature maps with a reduced aliasing effect (caused by the upsampling). Then, a classification and a regression networks are applied, in parallel, to each FPN level. From these results classified bounding boxes from all levels, which may result in an object being predicted in more than one layer. To prevent this from happening, the boxes with IoU greater than 0.5 are filtered by keeping the one with the highest confidence score.

Based on the Deep Learning techniques presented in this chapter, and their advantages and disadvantages that affect the detection of swimming pools in satellite images, one concluded that the RetinaNet algorithm is the most adequate to solve this problem.



## Chapter 3

# Development Tools and Dataset



### 3.1 Programming Language

The programming language used in the development of the Deep Learning and auxiliary scripts was *Python*. *Python* is a prominent, simple yet potent, and flexible programming language. Since it has a huge number of resources and high-quality documentation, it becomes very attractive for both beginners and advanced programmers. *Python* also has minimal and consistent syntax that instigate easy implementations and debugging. Moreover, *Python* has an extensive selection of tools and libraries built specifically for Deep Learning problems, that allow to save time in the development of new scripts.

For the development of this work the following libraries were used:

- **Tensorflow:** Open-source Machine Learning library developed by the *Google Brain* researching team for the implementation and deployment of large-scale models. Due to its simplicity and flexible tools, it allows for quickly experimenting Machine Learning and Deep Learning models [48];
- **Keras:** Open-source neural network oriented library and able to run on top of other libraries, such as TensorFlow. Developed by a *Google* engineer, for a research project, with focus on simplicity and a modular approach to building and training Deep Learning algorithms. Furthermore, it enables fast experimentation with neural networks [49].

Other tools were used for auxiliary tasks such as preprocessing the data:

- **Numpy:** It is a fundamental package for scientific programming since it supports high-performance multidimensional array objects and supplies the tools and functions to manipulate them [50];
- **BeautifulSoup:** Useful for retrieving information from HTML and XML files saving hours of work. It provides the user with the ability to navigate, search and modify a parse tree by dissecting documents and extracting needed information. This can be done by a few, simple lines of code [51];
- **Math:** Module that gives access to the most popular mathematical functions and constants (without complex numbers) so that more complex mathematical computations can be easily done [50];
- **OS:** Package that allows to interact with the operating system by using built-in functions to perform several tasks. It provides functions, such as, for creating, removing and changing directories [50];
- **Sys:** Provides variables, functions and methods of the *Python* interpreter with the purpose of manipulating the runtime environment. This module allows to pass user inputs from the command line to *Python* [50].

## 3.2 Graphical design tools

In order to make the process of detecting illegal swimming pools (using satellite images) user friendly, a Graphical User Interface was developed. The tool used for this development was *Electron*, an open-source framework, developed and maintained by *Github*, that enables the creation of cross-platform desktop applications. Since it utilises *Chromium* and *Node.js*, the applications can be written using *HTML*, *CSS* and *Javascript*. Several notorious projects were built with *Electron*, namely *Visual Studio Code*, *Facebook Messenger*, *Twitch* and *Slack* [52].

## 3.3 Dataset



The dataset (a set of images) used for this work can be found on *Kaggle*, an online community where users can find and publish datasets, explore and create Data Science and Machine Learning models, and enter competitions [4]. The chosen dataset has 3748 training and 2703 test  satellite images with bounding box annotations and labels for cars (denoted by 1) and swimming pools (denoted by 2) classification. The images are in RGB and have  $224 \times 224$  pixels. In figure  are represented four image examples from the training set.



Figure 3.1: Four training images of the chosen dataset [4].

---

<sup>1</sup>The training set is the data the Deep Learning algorithm will learn the weights and biases from. The testing set is the data used to make the model's predictions and compare with the real values, allowing to verify the performance of the model.

Every training image that contains swimming pools or cars has a corresponding labels file where several essential information is given: the name of the corresponding image in the filename section (denoted by `<filename>...</filename>`); the size of the image in the size block (denoted by `<size>...</size>`); and the detected objects class and bounding box coordinates in the object section (denoted by `<object>...</object>`). The labels file has a XML (Extensible Markup Language) format and follows the Pascal VOC structure:

```
<?xml version="1.0"?>
<annotation>
  <filename>000000012.jpg</filename>
  <source>
    <annotation>ArcGIS Pro 2.1</annotation>
  </source>
  <size>
    <width>224</width>
    <height>224</height>
    <depth>3</depth>
  </size>
  <object>
    <name>2</name>
    <bndbox>
      <xmin>149.53</xmin>
      <ymin>196.11</ymin>
      <xmax>193.97</xmax>
      <ymax>224.00</ymax>
    </bndbox>
  </object>
  <object>
    <name>2</name>
    <bndbox>
      <xmin>120.24</xmin>
      <ymin>212.77</ymin>
      <xmax>158.87</xmax>
      <ymax>224.00</ymax>
    </bndbox>
  </object>
</annotation>
```

In the above XML labels file example, the image has two objects of class name 2 meaning that this image contains two swimming pools located inside the bounding boxes depicted by the values given in the bndbox sections (denoted by `<bndbox>...</bndbox>`).

In order to accomplish the task of swimming pools detection, it is necessary to exclude the cars labels from the files since the goal of this work doesn't include car detection. A *Python* script was implemented to filter the training images by considering the following approach.

Open every training labels file and for each file do:

- Count the number of pools (denoted by 2) and cars (denoted by 1) labels;
- If the image contains only cars, remove from the training set the labels file and the corresponding image.

Since there are training images with pools that also have cars, the information relative to cars must be deleted from the labels file. To remove it, a *Python* script was implemented using the following steps:

- Open every training labels file;
- if the label is 1, then delete all the information related to that label.

After applying this Python script to the training dataset, the number of training images is 1993.



## Chapter 4

# Training RetinaNet



The deep learning algorithm chosen to achieve the detection of swimming pools in satellite images was RetinaNet. The main reasons to chose this algorithm was the fact that the satellite images for detection may have different sizes and a huge amount of background.

Several implementations can be found in the literature, being free to use and modify. The implementation used in this thesis can be found on Github [\[53\]](#).

## 4.1 Data Preprocessing of the Training Set - Pascal VOC to CSV

In order to use the RetinaNet algorithm, the input data must have the correct format. Thus, to train a model, it must be given as input a training set of images and two CSV files, namely: one CSV (Comma-separated Values) class mapping format file with the classes corresponding labels and one CSV with the annotations.

In this case, the mapping file with the existing classes has only one label since the only objects to be detected are swimming pools. In case there were more, the file should have one class per line. The following structure must be used:

```
class_name_0 , id_0  
class_name_1 , id_1
```

where the class id must start at 0. Since there is only one class, it must have the first available id value, 0. Thus, the CSV mapping file has the following structure:

```
pool , 0
```

The CSV file with annotations must have one box annotation per row. Therefore, images in which there are multiple objects (multiple bounding boxes) must have one row for each bounding box. Thus, each row must have the following specific format:

```
path/to/image.jpg , x1 , y1 , x2 , y2 , class_name
```

An example from the chosen dataset must resemble the following:

```
000000012.jpg , 149.53 , 196.11 , 193.97 , 224.00 , pool  
000000012.jpg , 120.24 , 212.77 , 158.87 , 224.00 , pool  
000000014.jpg , 211.71 , 156.41 , 224.00 , 196.16 , pool
```

Since the training dataset is not in the required format by the RetinaNet algorithm, it is imperative to convert the data labels files in XML format to the CSV format described above.

As mentioned before, the PASCAL VOC provides a standardised form to annotate image datasets for object detection. The main attributes of this type are the existence of one annotations file per

image in the XML format. A *Python* script was written to transform the dataset annotations files to the input required format (CSV format), as follows:

Create a new CSV file, open the PASCAL VOC files one by one and for each file do:

- find the information under the tag “name”;
- find the information under the tag “xmin”;
- find the information under the tag “xmax”;
- find the information under the tag “ymin”;
- find the information under the tag “ymax”;
- write a new row in the CSV file with the name, xmin, ymin, xmax, ymax and class name info for each bounding box.

## 4.2 Training of RetinaNet Models

After the preprocessing of the training dataset, the data is ready to be used to train the Deep Learning model given by the RetinaNet algorithm. This algorithm allows a certain level of training customisation by having available certain options:

- **Backbone architecture:** The FPN can be built on top of several architectures. The default is ResNet50 but both ResNet101 and ResNet152 are also available. The choice between the various ResNet influences on the model’s results since there is a difference of depth between them, represented by the numbers 50, 101 and 152, being ResNet50 the shallowest and ResNet152 the deepest network (for more information on ResNet see appendix [C.1](#) [\[5\]](#));
- **Weights:** To initialise the weights of the DNN given by the RetinaNet algorithm, an available weights file can be used. This means the training is not done from scratch, speeding up the training process (the algorithm will start to converge earlier) [\[31\]](#). By default, ImageNet weights are used (more information on ImageNet and weights in appendix [D.1](#));
- **Epochs:** One epoch is when an entire dataset is passed through the neural network only once. By default, RetinaNet sets this option to 50 [\[31\]](#);
- **Batch size:** The number of training examples in a single batch, by default, is 1. A dataset can be divided into batches (subsets of the training dataset) if it is not possible to pass the entire training dataset into the neural network at once due to its size [\[29\]](#);
- **Steps:** (or iterations) 10000 by default, steps is the number of batches needed to complete one epoch [\[29\]](#);

The Deep Learning models training was done using the default settings with the exception of the epochs, iterations and backbone architecture parameters. The first two parameters were 1 and

10000 by default, respectively. Due to the high computational cost, the training takes a huge amount of time. Therefore, 2 epochs and 500 iterations were used, which lowered the training time to less than a day, for all models. Since the training dataset, from *Kaggle*, has 1993 images and a batch size of 1 and 2 epochs were chosen, the dataset will be divided into 1993 batches, each with one image. This means that one epoch will involve 1993 batches in which the model weights will be updated after each batch. Three different backbone architectures were used in order to analyse which would give better results. Therefore, using the training dataset with 1993 images, three separate models were trained, each with a different architecture, namely: ResNet50, ResNet101 and ResNet152.



## Chapter 5

# Testing RetinaNet - Results and Discussion





## 5.1 Performance Metrics

To analyse the performance of each RetinaNet model, a confusion matrix (square matrix of order 2) was computed. This matrix allows a better understanding of the types of errors a model is making. Based on the confusion matrix it is possible to determine the following measures: accuracy, precision, sensitivity, specificity and the Matthews Correlation Coefficient (MCC) of the model. Additionally, two error types, Type I and Type II, can also be computed to deepen the understanding of the model's fouls.

To compute a confusion matrix it is necessary to have a test dataset with the respective expected outcome values. The expected outcome values and predictions are compared to get the four values of the confusion matrix. These values are obtained by counting the number of results in each of the following categories [54] [55]:

- **True Positive (TP):** The expected and the predicted values are positive. In this case, the image has a swimming pool in a certain location and the model accurately identified it. For each swimming pool, correctly detected, is added 1 to the TP value (note that an image with 3 correctly classified objects contributes with 3 to the TP value);
- **True Negative (TN):** The expected and the predicted values are negative. In this case, the image has no swimming pools and the model doesn't classify any part of the image as one. For each image, correctly classified as not having pools, is added 1 to the TN value;
- **False Positive (FP):** The predicted value is positive but the expected is negative. In this case, the image has no swimming pools but the model identifies a part of the image as being one. For each pool detected but not being one, is added 1 to the FP value;
- **False Negative (FN):** The predicted value is negative but the expected is positive. That is, the image has swimming pools but the model isn't able to detect any. For each image, wrongly classified as not having pools, is added 1 to the FN value.

The confusion matrix is organised as it follows in table 5.1,

Predicted value  Ground Truth	Positive	Negative
Positive	$TP$	$FP$
Negative	$FN$	$TN$

Table 5.1: The confusion matrix.

Using the four values that form the confusion matrix, it is possible to compute five performance metrics relative to the model, defined by equations 5.1, 5.2, 5.3, 5.4 and 5.5 and two error types, given by equations 5.6 and 5.7.

The accuracy (equation 5.1) is given by the division between the total of correctly classified examples and the total of predictions made. However, using only this measure to assess the model's performance is not recommended since it assumes equal weight for both types of errors (FN and FP). For instance, if only 1% of the images encompass swimming pools and the model

predicts no pools in every image, then the accuracy would be 99%. In spite of the almost perfect accuracy, the cases that were significant for this study were disregarded making this model useless. Thus, the accuracy must be used carefully and desirably with other performance metrics [54].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.1)$$

Precision (equation 5.2) is obtained by dividing the number of correctly classified positive examples by the number of predicted positive examples. A high precision expresses that an example is truly positive when predicted positive [54] [55].

$$Precision = \frac{TP}{TP + FP} \quad (5.2)$$

Sensitivity (equation 5.3) is defined by the division between the number of correctly classified positive examples and the number of positive examples. In other words, it is the number of positives correctly predicted out of all the positive classifications. A high sensitivity value indicates the class is being correctly classified [54] [55].

$$Sensitivity = \frac{TP}{TP + FN} \quad (5.3)$$

Specificity is computed by the division between of correctly classified negative examples and the number of negative examples. It reports the ratio of true negative examples predicted correctly by the model to all the negative examples and is given by equation 5.4. A high value of specificity indicates that the model is predicting the negative examples well.

$$Specificity = \frac{TN}{TN + FP} \quad (5.4)$$

As presented before, the accuracy is sensitive to class imbalance. Furthermore, precision, sensitivity and specificity are asymmetric by only considering the positive or negative class. Because of this, if it is desired to evaluate the model's performance on both classes simultaneously (existence or not of a swimming pool, in the case studied in this thesis), the Matthews Correlation Coefficient (MCC) can be used. This metric is given by equation 5.5

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (5.5)$$

Giving an in depth look to equation 5.5, one reaches the conclusion that the MCC gives a value of 1 if the FP and FN values are equal to 0 (perfect classifier), meaning a value close to 1 indicates the classes are well predicted. On the other hand, a classifier with TP and TN values equal to 0 (a classifier with no correct predictions) has a MCC value of  $-1$ . Thus, the MCC value always falls between  $-1$  and  $1$ , being a value of  $0$  indicating that the classifier being evaluated has a random behaviour (equivalent to a random flip of a coin).

In addition, two error types can be analysed to provide a more in-depth look at the wrong predictions made by the model. These errors are named Type I and Type II. The Type I error refers to the rejection of a true null hypothesis and, for this reason, is also named false positive rate. In

the swimming pools detection application, a Type I error corresponds to the model detecting a pool in an image where there is not one. This error can be computed using equation 5.6

$$Type\ I = 1 - Specificity = \frac{FP}{FP + TN} \quad (5.6)$$

The Type II error refers to the acceptance of a false null hypothesis (also named false negative rate). In the application studied in this work, a Type II error indicates that the model was unable to detect a swimming pool in a testing image. The Type II error is given by equation 5.7

$$Type\ II = 1 - Sensitivity = \frac{FN}{TP + FN} \quad (5.7)$$

## 5.2 Data Preprocessing of the Test Set

The chosen dataset has 2703 testing images without labels. In order to verify if the model is correctly detecting swimming pools, it was necessary to classify and draw bounding boxes by hand. Another possibility was training the model with fewer images and using the rest for testing, since all the training examples have an annotations file. However, due to the small number of training images available (1993), this option was not considered.

Thus, to create the annotations for the testing set images, *LabelImg* [7], a graphical image annotation tool was used (figure 5.1). This tool is written in python and its graphical interface was developed using *Qt* [56]. The annotations are saved as XML files in the PASCAL VOC format.

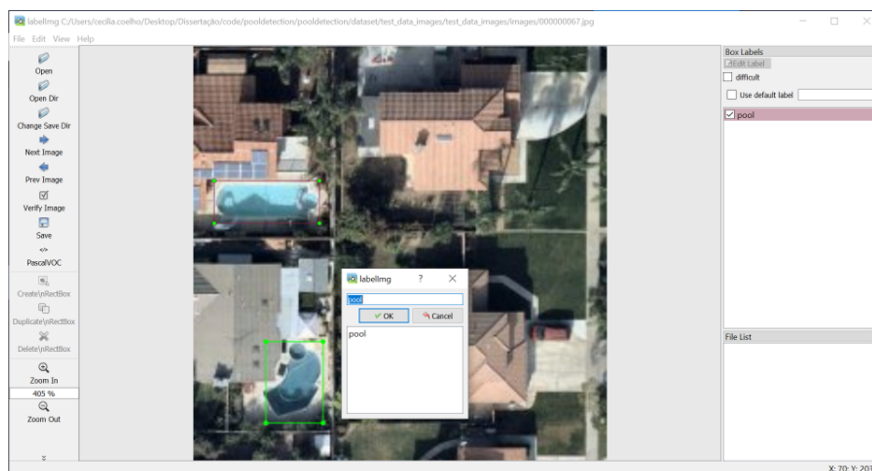


Figure 5.1: Testing set image annotation using *LabelImg* [7].

As shown in figure 5.1, the tool provides a pointer to select the region that contains the targeted object. After the box being drawn, a prompt box enables to create and choose the label. When all the wanted objects, in an image, are selected and labelled, the save button stores the information in the PASCAL VOC format. For the image in figure 5.1 the resulting file is:

```

<annotation>
  <folder>images</folder>
  <filename>000000067.jpg</filename>
  <size>
    <width>224</width>
    <height>224</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>pool</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>11</xmin>
      <ymin>72</ymin>
      <xmax>68</xmax>
      <ymax>95</ymax>
    </bndbox>
  </object>
  <object>
    <name>pool</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>36</xmin>
      <ymin>160</ymin>
      <xmax>71</xmax>
      <ymax>204</ymax>
    </bndbox>
  </object>
</annotation>

```

Given the fact that the CSV format used for training the model is the one required by the RetinaNet algorithm and to uniformize the used files format, all the annotations written by the tool were converted to this format using the *Python* script described in section [2.9.3](#).

## 5.3 Test Set from Kaggle

The trained RetinaNet models make use of three different backbone architectures, namely ResNet50, ResNet101 and ResNet152. After labelling the testing images provided by the chosen test dataset, these were given to the models to get the predictions. This test dataset has 2703 satellite images with a total of 620 swimming pools, distributed across a total of 524 images.

Before analysing the results of the whole testing images set, first, four examples will be studied (represented in figure 5.2 and figure 5.3). The full list of predictions will then be analysed.



Figure 5.2: Two testing images, containing swimming pools, of the chosen dataset used for evaluating the model's performance [4].

As expected, not all the testing images had swimming pools. In figure 5.3 are represented two examples of this case.



Figure 5.3: Two testing images without swimming pools, of the chosen dataset, used for evaluating the model's performance [4].

### 5.3.1 Model using ResNet50

When the images in figure 5.2 are given to the model that uses a backbone structure built on top of a ResNet50, it is possible to see in the naked eye that the model was able to detect all the swimming pools correctly, as can be seen in figure 5.4



Figure 5.4: Model’s predictions of two testing images of the dataset.

The model’s certainty of the objects enclosed by the bounding boxes being swimming pools has a minimum of 0.780, maximum 0.898 and mean 0.844 values, which indicates the model’s reasonably high confidence when identifying the objects.

The images with no swimming pools (figure 5.3) were also given. The result is shown in figure 5.5



Figure 5.5: Model’s predictions of two testing images without swimming pools.

As it can be seen in figure 5.5, no bounding boxes were drawn which indicates that no swimming pools were detected by the model in the given images.

The entire dataset test images collection, composed of 2703 images, was given to the model with the objective of getting the predictions. The model’s bounding boxes were compared with the human annotated ones (ground truth) in order to compute the performance metrics based on the confusion matrix, see table 5.2.

Predicted   Ground Truth	Positive	Negative
Positive	471	18
Negative	131	2200

Table 5.2: Confusion matrix computed with the results obtained by comparing predictions and ground truth for the model that uses ResNet50 as backbone architecture.

From table 5.2, this model was able to correctly classify 471 objects as swimming pools, from the

test images, (represented by the TP value) and 2200 as not (TN value). 18 objects were classified as pools while not being one (FP value) and 131 images were wrongly classified as not having a pool (FN value). With these values and using the equations [5.1](#), [5.2](#), [5.3](#), [5.4](#) and [5.5](#) it was computed an accuracy of 0.9472, a precision of 0.9631, a sensitivity of 0.7824, a specificity of 0.9919 and a MCC of 0.8380. The type I and II errors, given by equations [5.6](#) and [5.7](#), were also computed being 0.0081 and 0.2176, respectively.

### 5.3.2 Model using ResNet101

When the images in figure [5.2](#) are given to the model that uses a backbone structure built on top of ResNet101, it is able to detect all the swimming pools correctly, as can be seen in figure [5.6](#).



Figure 5.6: Model's predictions of two testing images of the dataset using ResNet101 as backbone architecture.

The model's certainty of the objects enclosed by the bounding boxes being swimming pools has a minimum value of 0.688, maximum 0.948 and mean 0.839. When compared with the model using a backbone architecture built on top of a ResNet50, the ResNet101 provides a higher maximum confidence score but also a lower minimum.

The images with no swimming pools (figure [5.3](#)) were also given, resulting in figure [5.7](#)



Figure 5.7: Model's predictions of two testing images without swimming pools.

Similarly to the ResNet50 backbone architecture, this model is able to correctly detect the non-existence of swimming pools in the images, although the mean confidence scores is a bit lower.

The 2703 test images were given as input to the model which gave the predictions as outputs. The model's bounding boxes were compared with the corresponding ground truth in order to evaluate the model's performance using the confusion matrix, see table 5.3

Predicted   Ground Truth	Positive	Negative
Positive	480	7
Negative	133	2285

Table 5.3: Confusion matrix computed with the results of comparing predictions and ground truth for the model that uses ResNet101 as backbone architecture.

From table 5.3 this model was able to correctly classify 480 objects as swimming pools, from the test images, (represented by the TP value) and 2285 as not (TN value). 7 objects were classified as pools while not being one (FP value) and 133 images were wrongly classified as not having a pool (FN value). With these values and using the equations 5.1, 5.2, 5.3, 5.4 and 5.5, and it was computed an accuracy of 0.9522, a precision of 0.9626, a sensitivity of 0.7830, a specificity of 0.9969 and a MCC of 0.8519. The type I and II errors, given by equations 5.6 and 5.7, were also computed being 0.0031 and 0.2170, respectively.

### 5.3.3 Model using ResNet152

Finally, the images in figure 5.2 were passed to the model that uses a backbone structure built on top of a ResNet152. This model was able to detect all the swimming pools correctly, as can be seen in figure 5.8



Figure 5.8: Model's predictions of two testing images of the dataset using ResNet152 as backbone architecture.

The model's certainty of the objects enclosed by the bounding boxes being swimming pools has minimum 0.674, maximum 0.871 and mean 0.783 values being the model with the lowest scores, obtaining the lowest minimum, maximum and mean confidence values of all three trained models. The images with no swimming pools (figure 5.3) were also given, resulting in figure 5.9





Figure 5.9: Model’s predictions of two testing images without swimming pools.

Once again, all the images were correctly classified however this model got the lowest confidence scores. This fact could indicate the other models are more suitable for the task. Nevertheless, the cases studied don’t represent a significant number of tests as to make conclusions about performance. Therefore, the 2703 test images were given as input to the model which gave the predictions as outputs. The model’s bounding boxes were compared with the corresponding ground truth in order to evaluate the model’s performance using a confusion matrix, see table 5.4

Predicted   Ground Truth	Positive	Negative
Positive	503	6
Negative	111	2241

Table 5.4: Confusion matrix computed with the results of comparing predictions and ground truth for the model that uses ResNet152 as backbone architecture.

Table 5.4 shows that this model was able to correctly classify 503 objects as swimming pools, from the test images, (represented by the TP value) and 2241 as not (TN value). 6 objects were classified as pools while not being one (FP value) and 111 images were wrongly classified as not having a pool (FN value). With these values and using the equations 5.1, 5.2, 5.3, 5.4 and 5.5, it was computed an accuracy of 0.9591, a precision of 0.9882, a sensitivity of 0.8192, a specificity of 0.9973 and a MCC of 0.8766 . The type I and II errors, given by equations 5.6 and 5.7, for this model are 0.0027 and 0.1808, respectively.

### 5.3.4 Conclusions

The results obtained with the three models are shown in table 5.5

	TP	TN	FP	FN	Accuracy	Precision	Sensitivity	Specificity	MCC	Type I	Type II
ResNet50	471	2200	18	131	0.9472	0.9631	0.7824	0.9919	0.8380	0.0081	0.2176
ResNet101	480	2285	7	133	0.9522	0.9626	0.7830	0.9969	0.8519	0.0031	0.2170
ResNet152	503	2241	6	111	0.9591	0.9882	0.8192	0.9973	0.8766	0.0027	0.1808

Table 5.5: Confusion matrix values and computed evaluation quantities for each of the trained models.

Analysing the results in table [5.5](#), it can be seen that every model’s accuracy is high and remarkably close to each other, indicating that the classifiers are correct most of the times (approximately 0.95 of accuracy). All RetinaNet models have high precision (above 0.95) indicating that an image labelled as positive is truly positive, as verified by the small number of false positives reported in the table. The model using a ResNet152 backbone architecture is capable of better recognising the class in study (lowest number of false negatives). This is also corroborated by the highest sensitivity value obtained for this model. The three models have high specificity, being above 0.99 for all three models, which means the models predict negative examples well (images without swimming pools). The Matthew’s Correlation Coefficient (MCC) is higher for the model built on top of a ResNet152, demonstrating, out of the three models, the effectiveness in classifying both classes (presence or absence of pools in an image). Moreover, the ResNet152 has the smallest Type I and Type II errors suggesting that the predictions are correct most of the times.

A closer look at the false positive and false negative classifications, exposes the type of difficulties faced by each model and the kind of objects that compromise the swimming pools detection:

- Every model detects blue rectangles, that may be canopies, for instance, as swimming pools;
- All models have difficulties in detecting empty pools;
- Both ResNet50 and ResNet101 models identify blue basketball fields as pools;
- The model that uses a ResNet50 as backbone architecture wrongly detects circular blue looking canopies;

From this analysis one may conclude that the model using a ResNet152 backbone architecture outperforms the models using ResNet50 and ResNet101. Furthermore, all three models have similar computational cost.

## 5.4 Test set from Google maps

As to further test the limitations of the trained models, a few *Google Maps* cropped images were used. This images were taken with various zoom percentages and from areas with different image quality. The eight images chosen for testing are shown in figure [5.10](#).



Figure 5.10: Eight test images taken from google maps with different sizes and resolutions. The images are numbered in yellow on the top right corner.

The test images in figure [5.10](#) were specifically chosen for enclosing objects very similar to swimming pools and swimming pools without the usual appearance. It is listed below the characteristics of each image:

- **image 1** -  $433 \times 340$  pixels, a lake and a blue tag are displayed;
- **image 2** -  $442 \times 258$  pixels, encloses two swimming pools and a swimming pool look-alike glass structure on top of a building;
- **image 3** -  $298 \times 241$  pixels, a pool without one of the top characteristics, the blue water;
- **image 4** -  $100 \times 118$  pixels, the image is very small with high zoom;
- **image 5** -  $366 \times 270$  pixels, unusual shape swimming pool present;
- **image 6** -  $1351 \times 1364$  pixels, very high-quality 3-dimensional picture with 2 pools and a blue basketball court;
- **image 7** -  $1099 \times 1333$  pixels, the same court as in image 6 and a green water swimming pool;
- **image 8** -  $732 \times 613$  pixels, the basketball court seen in images 6 and 7 but with higher zoom.

### 5.4.1 Model using ResNet50

The images in figure 5.10 were given to the RetinaNet model built on top of a ResNet50. The detected swimming pools were enclosed by a blue bounding box as shown in figure 5.11



Figure 5.11: Eight test images taken from google maps with different sizes and resolutions. The detected objects are enclosed by blue boxes. The images are numbered on the top right corner.

It is listed below the comments on the results obtained for each image:

- **image 1** - The model was able to detect no swimming pool, even though the image contains a lake and a blue tag;
- **image 2** - It correctly detected the two swimming pools and did not identify the glass structure as a swimming pool;
- **image 3** - The unclean swimming pool was detected by this model;
- **image 4** - Several detections were made for a single pool. The very low resolution and size of this image might explain the multiple bounding boxes due to the pixels being easily distinguished;
- **image 5** - Three pools were detected at the real location of a single pool. The error might be explained by the unusual shape;
- **image 6** - The model exceeded the expectations by correctly identifying the swimming pools and not mistaking the blue basketball court by one;

- **image 7** - Similarly to what happened with image 3, the unclean pool was not identified. This may be the reason why the lake in image 1 was not detected. Also, the previously not detected court (image 6) was, this time, classified as a pool. The difference between the pictures indicate that this model is sensible to zoom percentages;
- **image 8** - The test done on this image supports the statement presented in the previous item. The higher zoom contributed to deceive this model;

#### 5.4.2 Model using ResNet101

The images in figure 5.10 were given to the RetinaNet model built on top of a ResNet101. The detected swimming pools were enclosed by a blue bounding box as represented in figure 5.12



Figure 5.12: Eight test images taken from google maps with different sizes and resolutions. The detected objects are enclosed by blue boxes. The images are numbered on the top right corner.

It is listed below the comments on the results obtained for each image:

- **image 1** - The model was able to detect no swimming pool, even though the image contains a lake and a blue tag;

- **image 2** - It was not capable of detecting any of the swimming pools, falling behind the ResNet50 model for this image;
- **image 3** - The unclean swimming pool wasn't detected by this model;
- **image 4** - Several detections were made for a single pool. Again, the very low resolution and size of this image might explain the multiple bounding boxes (due to the pixels being easily distinguished);
- **image 5** - Similarly to the previous model, three pools were detected at the real location of a single pool;
- **image 6** - The model correctly identified the swimming pools and did not mistake the blue basketball court by one;
- **image 7** - Both the unclean pool and court were correctly classified. The only difference between this image and image 3 is the quality. Image 7 has higher quality and presents 3-dimensional textures;
- **image 8** - The bigger zoom didn't influence the model's detection;

### 5.4.3 Model using ResNet152

Finally, the images in figure [5.10](#) were given to the RetinaNet model built on top of a ResNet152. The detected swimming pools were enclosed by a blue bounding box as represented in figure [5.13](#)



Figure 5.13: Eight test images taken from google maps with different sizes and resolutions. The detected objects are enclosed by blue boxes. The images are numbered on the top right corner.

It is listed below the comments on the results obtained for each image:

- **image 1** - The model was able to detect no swimming pool, even though the image contains a lake and a blue tag;
- **image 2** - The ResNet152 model was capable to accurately detect the objects in this image;
- **image 3** - The unclean pool was properly identified;
- **image 4** - Unlike the previous models, it wasn't able to detect the swimming pool in this image;
- **image 5** - In contrast to the ResNet50 and ResNet101 models, the model using ResNet152 correctly detected a single pool in this image;
- **image 6** - It correctly identified the swimming pools and did not mistake the blue basketball court by a swimming pool;
- **image 7** - Both the unclean pool and court were correctly classified;
- **image 8** - The basketball court was not able to trick this model;

#### 5.4.4 Conclusions

The results obtained with the three models are now summarised.

- The models using ResNet50 and ResNet101 are not able to correctly classify a swimming pool when the water has an unusual colour (green as in the tested examples);
- The results indicate that the ResNet50 model is sensitive to the image's zoom (wrong detection of objects when the zoom percentage is high);
- Classification of exceptionally small images may be difficult for the model using ResNet152.

In conclusion, the model that had the best performance was the one built on top of a ResNet152. It was able to correctly detect a bigger number of swimming pools, except for the smallest image.



## Chapter 6

# Graphical User Interface



Other main goal of this dissertation was to develop a Graphical User Interface (GUI) in order to make the process of detecting swimming pools using satellite images user friendly, for instance, to be used by a Government entity to detect this type of illegal constructions.

This GUI was created using the *Electron* framework. It was built using *HTML*, *CSS* and *Javascript* in connection to the RetinaNet model, with a backbone architecture built on top of a ResNet152, python script and auxiliary scripts. The desktop application's main window is shown in figure 6.1

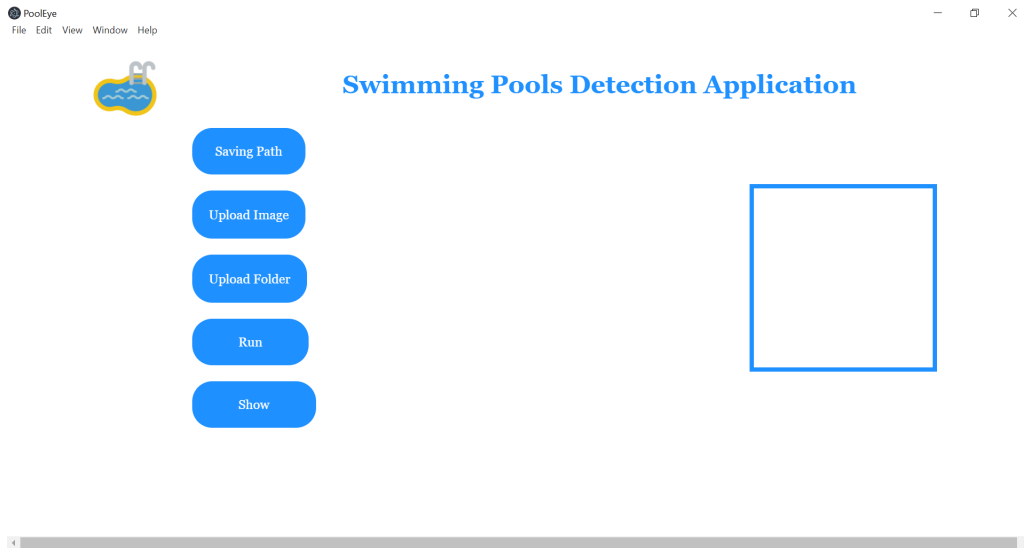


Figure 6.1: Desktop application main window.

As seen in figure 6.1, on the right side there is a blue frame where the images in which swimming pools were detected will be shown. On the left side, five buttons with different functionalities are displayed. A brief description of the five menu options shown in the left side of figure 6.1 is given from the top (Saving Path) to the bottom (Show).

- **Saving Path:** Allows the user to choose the path where a CSV file containing the bounding boxes information will be saved;



Figure 6.2: “Save Path” button action.

- **Upload Image:** Several single images can be selected for detection of swimming pools;

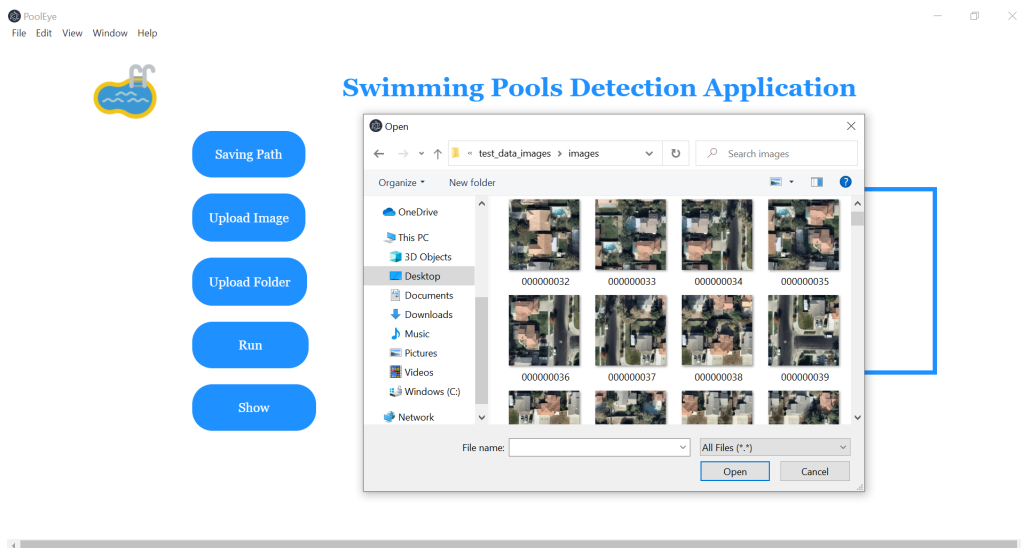


Figure 6.3: “Upload Image” button action.

- **Upload Folder:** When a folder is selected all the images inside will be given to the Deep Learning model for classification;

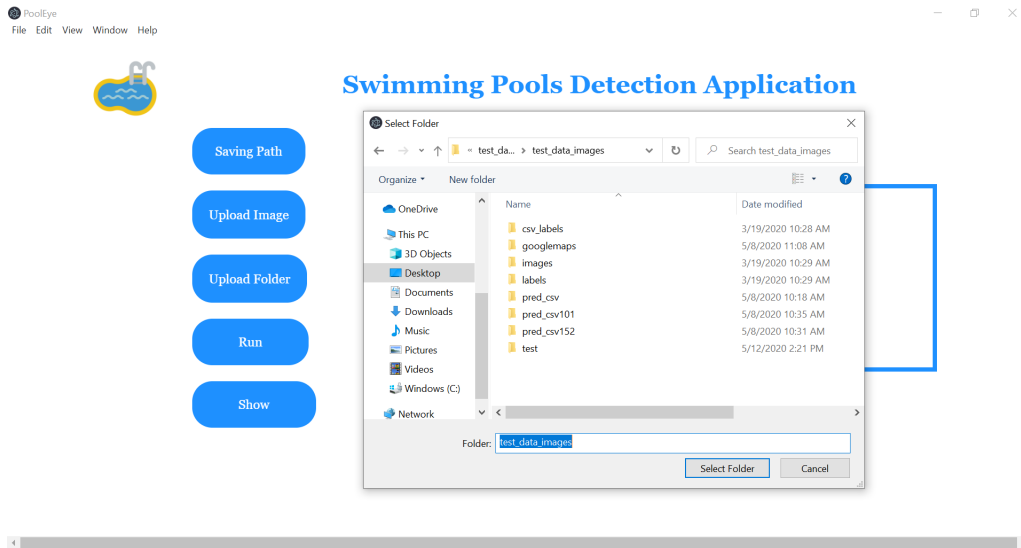


Figure 6.4: “Upload Folder” button action.

- **Run:** The images selected with the “Upload Image” or “Upload Folder” buttons are given as input to the RetinaNet model. A completion bar appears till the model ends running all the images. At the end, a CSV file with each image’s detection information will be available at the path chosen using the “Saving Path” button;

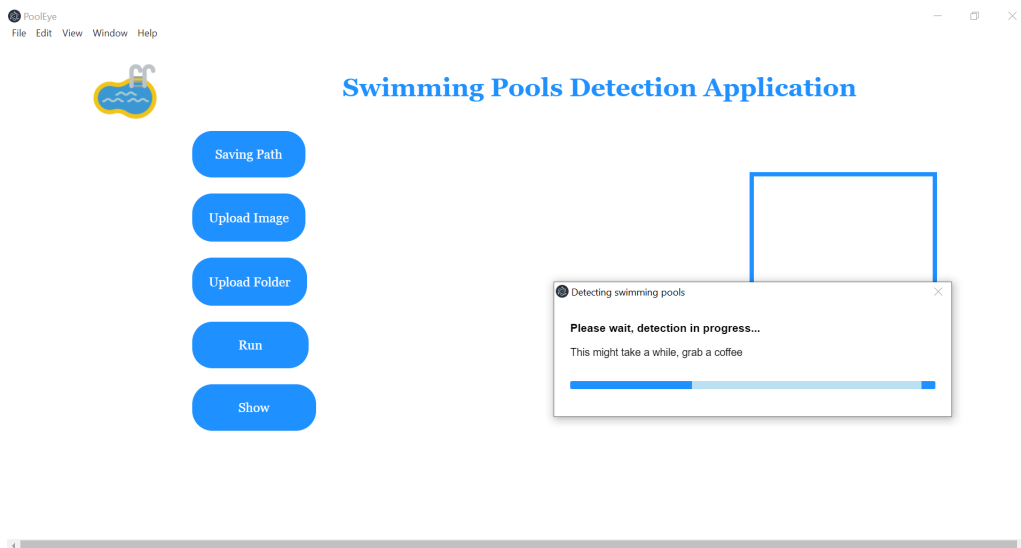


Figure 6.5: “Run” button action.

- **Show:** When clicked, a python script that draws the bounding boxes in the images using the CSV file created by the model is run. Only one box per image is shown. If the user wants to see more images with bounding boxes, it should click the arrows under the frame where the images appear;

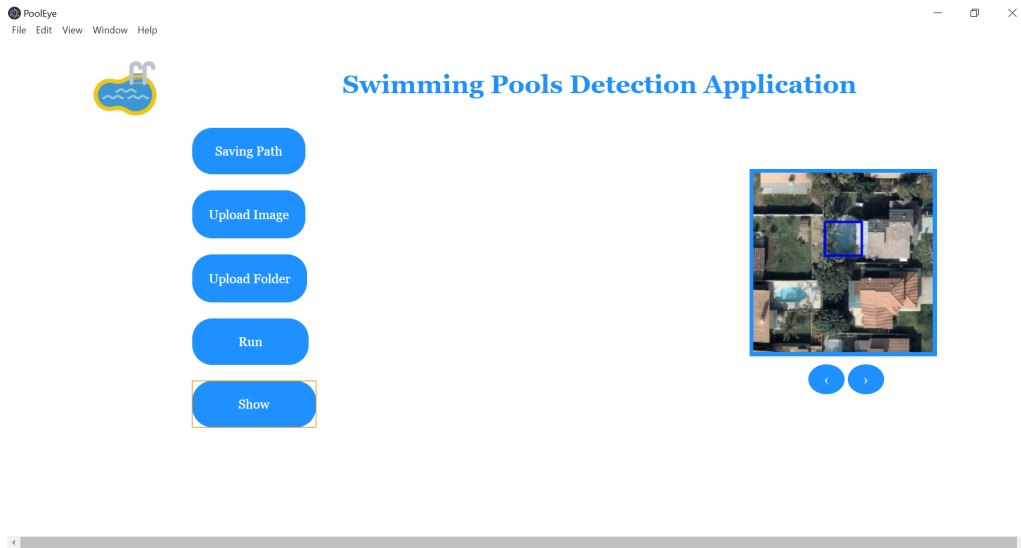


Figure 6.6: “Show” button action.

In order to assist the user, when an attempt to run the model before providing a saving path or at least one image is made, an error prompt is shown, figure 6.7.

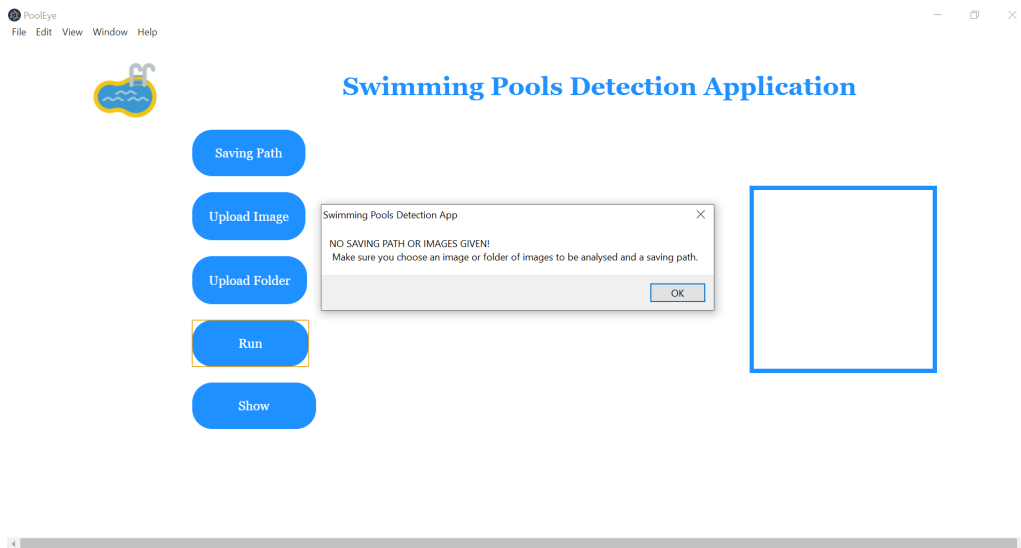


Figure 6.7: Error prompt shown where the saving path and/or at least one image are not provided.

Moreover, an error message prompts if there are no more bounding boxes to show, when the arrow buttons are clicked (figure 6.8).

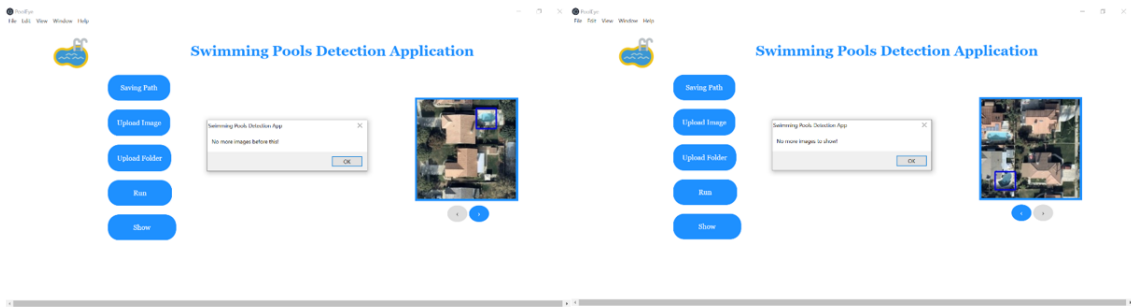


Figure 6.8: An error message prompts if the previous (left arrow) or next (right arrow) image is clicked but there are no more images with bounding boxes to show.





## Chapter 7

# Conclusions and Future Work



## 7.1 Conclusions

The main goal of this work was to develop a software application to identify swimming pools in satellite images using a Deep Learning approach. After an intensive study on the advantages and disadvantages of the available algorithms used for object detection, the RetinaNet algorithm was chosen.

Three RetinaNet models were trained, each built on top of a differing backbone architecture, using a dataset from *Kaggle*. After the training, the models were tested using the test set composed of 2703 images with 620 swimming pools distributed by 524 images.

The results were analysed using seven performance metrics, that are computed based on confusion matrices. From these results, one may conclude that the model using a ResNet152 backbone architecture outperforms the models using ResNet50 and ResNet101.

The models were further tested using images obtained from Google Maps. The idea is to test the models under extreme conditions. These images contain objects similar to a swimming pool, such as a blue basketball court, and also swimming pools with unusual colours, several uncommon shapes and sizes. The results indicate that the ResNet50 model is sensitive to the image's zoom, while the ResNet101 model isn't consistent with the characteristics of the pools detected. Furthermore, the classification of really small images may be difficult for the model using ResNet152. According to the results obtained, the model with a backbone architecture built on top of a ResNet152 has the best performance when used for the detection of swimming pools in satellite images. Therefore, the ResNet152 model was adopted to integrate the interface.

A Graphic User Interface was developed in order to make the process of detecting illegal swimming pools using satellite images user friendly.

The interface allows to select single images or a folder of images for detecting swimming pools. The classification information is stored in a CSV file and saved in the chosen path. Additionally, blue bounding boxes are drawn around the detected objects if the user desires to have a visual representation of the classifications.

## 7.2 Future Work

Several improvements can be considered in the future. The most challenging one involves training the model using more epochs and iterations (steps) in order to achieve a better performance. To do that, one must use a cluster or a supercomputer so that the training computational time becomes feasible.

Regarding the newly developed GUI, new functionalities can also be added. Namely, one can create a python script to perform the automatic acquisition of the satellite images and give the geographic coordinates of the swimming pools as an output.

Furthermore, other recent object detection algorithms can be analysed and compared with the RetinaNet performance, for this specific problem. Namely, the EfficientDet, the Sniper and the Recurrent YOLO.



# Appendix A

## A.1 Intersection over Union (IoU)

Intersection over Union (also known as the Jaccard index) is the most used metric to evaluate the accuracy of an object detection algorithm on a specific dataset. Essentially it calculates the similarity between the predicted box and the ground truth.

In order to apply the IoU metric two variables are needed, the ground truth bounding boxes (hand labelled bounding boxes from the testing set) and the predicted bounding boxes (output of a neural network). With this two information, the IoU is computed as follows (equation [A.1](#) [\[57\]](#)):

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{A.1}$$

where  $A$  and  $B$  are the prediction and ground truth bounding boxes, respectively. In the numerator is computed the area of overlap between the predicted and ground truth bounding boxes, denoted by  $|A \cap B|$  and in the denominator is the area of union (the area enclosed by both the predicted and ground truth bounding boxes), denoted by  $|A \cup B|$ .



# Appendix B

## B.1 Region Proposal Networks

Region proposal networks (RPN) is an algorithm that generates region proposals for the locations of each object in an image by sliding a convolution layer (the authors chose a  $3 \times 3$  window) [8]. The RPN uses two different layers: a regression layer where a convolution layer strides through the image and predicts  $k$  anchor boxes (since the sliding window is  $3 \times 3$ , there will be 9 boxes for each pixel) at each anchor point location (note that is called an anchor to the central point of the sliding window); a classification layer that predicts the probability of an object being contained by a box [8]. The architecture of RPN is represented in figure B.1

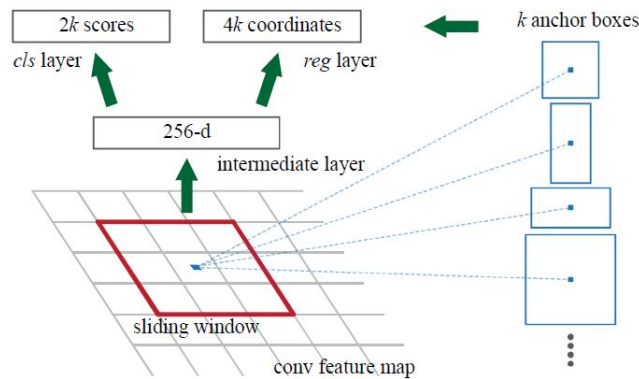


Figure B.1: Region Proposal Networks architecture [8].

As seen in figure B.1, the classification layer outputs 2 scores for each  $k$ , giving two probabilities, one for object and another for not object. The regression layer outputs  $4 \times k$  coordinates that represent the bounding box spatial coordinates.





# Appendix C

## C.1 ResNet

ResNet is considered a powerful deep neural network having obtained first place in *ILSVRC* and *COCO* competitions in 2015 [9]. The ResNet architecture has several versions in which the only change is the number of layers, this number is indicated by a two or more digit number following the name ResNet. In this project, three variants were studied with 50, 101 and 152 layers named ResNet50, ResNet101 and ResNet152, respectively [9].

In theory, adding layers to a neural network has two possible effects on the performance, it either increases or remains the same. Accordingly, more layers are never a disadvantage since once the neural network is in its best possible state (achieved 100% accuracy and the loss function is at the global minimum), the new layers added should learn the identity function,  $g(x) = x$ ; to retain the best possible state (figure C.1).

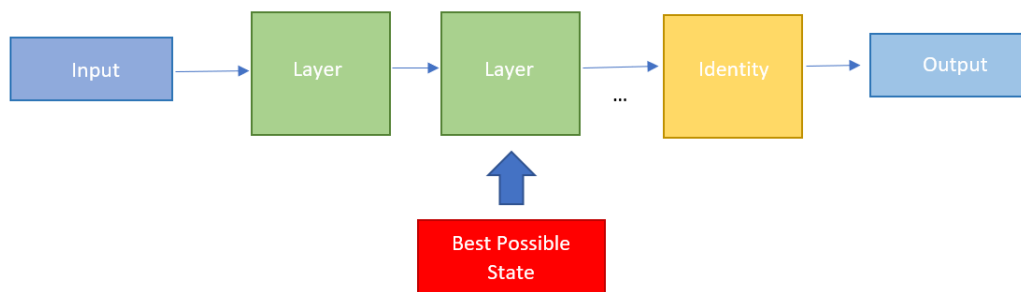


Figure C.1: Adding more layers to a neural network in which the best possible state was attained results on the following layers learning the identity function.

Nonetheless, when this concept is tested, the conclusion may be different. Adding more layers to a neural network might lead to a decreasing accuracy in training and testing, as seen in the graphs in figure C.2 [9].

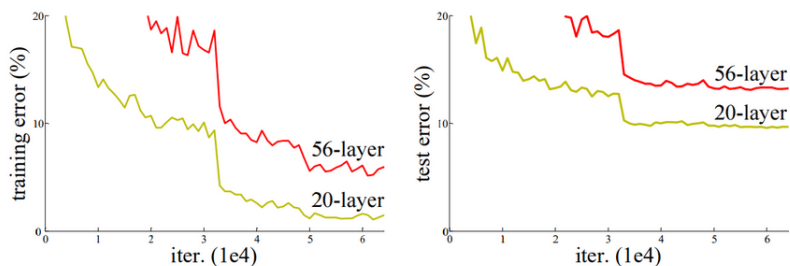


Figure C.2: Training error (left) and test error (right) on CIFAR-10 [4] with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error [9].

This counter-intuitive experimental results demonstrate that the learning of the identity function is exceptionally difficult since the combination of weights and biases are massive, thereby making the probability of finding the correct combination of parameters extremely small in a finite amount of time and data. So, to make the learning of the identity function easier so that it is possible to add more layers without worrying about the diminishing of the accuracy, a *Microsoft Research Asia* team proposed a solution by adding the input of a layer (or block of layers) to its output (called a residual block, that may have an arbitrary number of layers, figure C.3) so that the network learns using the residuals, hence the name ResNet [9].

With this solution, the layers, instead of attempting to learn the identity function, seek for the very simple “zero” function, denoted by  $F(x) = 0$ , this is schematised in figure C.3.

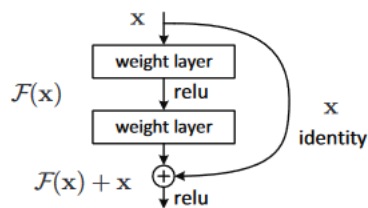


Figure C.3: A residual block where the layer’s input is added to the output [9].

Note that the number of layers that compose a residual block can be customised [9].

From the graphs in figure C.4, it is verified that the ResNet (graph on the right) has the capacity of taking advantage of the addition of several new layers without taking any performance risks, in contrast to the “plain” neural networks.

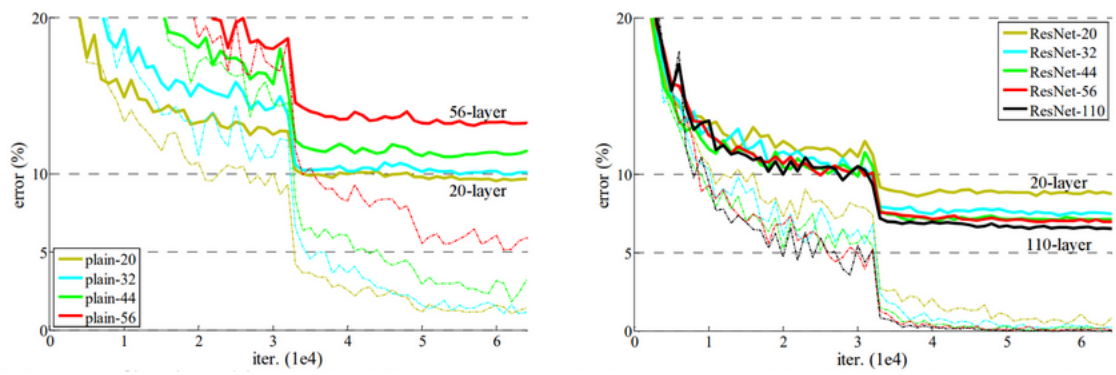


Figure C.4: Classical neural networks on the left, ResNets on the right. Dashed lines denote training error, and bold lines denote testing error (training on CIFAR-10 [4] [9]).



# Appendix D

## D.1 ImageNet

*ImageNet* is a project that provides a dataset composed of a large collection of human annotated images, created by academics at *Princeton*, *Stanford* and other American universities, for aiding the development of computer vision algorithms. It contains more than 14 million images of more than 20000 classes and bounding box annotations for 1 million images.

Several neural network models have been trained using the *ImageNet* dataset. The weights learned by the networks that won object detection contests (namely ResNet, InceptionV3, MobileNet, etc) are available to the community and can be downloaded with *Keras*. The usage of pre-trained models for training on a different task is very useful since the features learned while training are helpful for the new goal.

Deep neural networks possess a huge number of unknown parameters used to learn different features. In order to discover the optimum values of all the parameters, a huge training set is crucial. Since it is difficult to acquire such huge labelled datasets and due to the high time and computational cost to train a network, approved model's weights files, obtained by training on millions of images and for hundreds of hours on powerful GPUs, are used as starting weights for a model, instead of training from scratch [58] [23].



# Bibliography

- [1] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh. Deep learning vs. traditional computer vision. (Computer Vision Conference (CVC), Las Vegas, Nevada, United States), 04 2019.
- [2] F. Li and A. Karpathy. Convolutional neural networks for visual recognition. (online resource - <https://cs231n.github.io/neural-networks-1/>), last accessed on 2020/03/03), 2015.
- [3] Prabhu. Understanding of convolutional neural network (cnn) - deep learning. (online resource - <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>), last accessed on 2020/05/26), 2018.
- [4] Google Alphabet Inc. Kaggle. (online resource - <https://www.kaggle.com>), last accessed on 2020/02/01), 2010.
- [5] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 10 2017.
- [6] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. pages 936–944, 2017.
- [7] Tzutalin. Labelimg. (online resource - <https://github.com/tzutalin/labelImg>), last accessed on 2020/04/20), 2015.
- [8] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 06 2015.
- [9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 7:770–778, 12 2015.
- [10] J. Maida. Global swimwear market 2020-2024| rising number of swimming pools to boost market growth. (online resource - <https://www.businesswire.com/news>), last accessed on 2020/02/26), 2020.

- [11] MarketWatch. Swimming pool construction market (3.8% cagr) 2018-2026: Global business growth, size and forecast. (online resource - <https://www.marketwatch.com/press-release/swimming-pool-construction-market-38-cagr-2018-2026-global-business-growth-size-and-forecast-2019-11-05>), last accessed on 2020/02/26), 2019.
- [12] European Union of Swimming Pool and Spa Associations. Sector volume and market characteristics. (online resource - <https://www.eusaswim.eu/>, last accessed on 2020/02/25), 2009.
- [13] Assembleia da República. Lei n.º60/2007. *Diário da República n.º 170/2007, Série I*, pages 6258–6309, 2007.
- [14] Ministério do Equipamento do Planeamento e da Administração do Território. Decreto-lei n.º 555/99. *Diário da República n.º 291/1999, Série I-A*, pages 8912–8942, 1999.
- [15] Decreto-Lei n.º 287/2003. Código do imposto municipal sobre imóveis e o código do imposto municipal sobre as transmissões onerosas de imóveis. pages 7568–7647, 2003.
- [16] A. Cabrita. Imi 2019: Um guia com tudo o que precisa saber. (online resource - <https://www.doutorfinancas.pt/impostos/>, last accessed on 2020/02/26), 2019.
- [17] Z. Yang and R. Nevatia. A multi-scale cascade fully convolutional network face detector. *Proceedings - International Conference on Pattern Recognition*, pages 633–638, 2016.
- [18] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. pages 6526–6534, 07 2017.
- [19] C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34:743–61, 07 2011.
- [20] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32:1627–45, 09 2010.
- [21] H. Singh. *Practical Machine Learning and Image Processing For Facial Recognition Using Python*. Apress, Berkeley, CA, 2019.
- [22] S. J. Ryan, C. P. Wilkinson, S. R. Sadda, and P. Wiedermann. *Retina*. Elsevier, 6th edition edition, 2017.
- [23] E. Salahat and M. Qasaimeh. Recent advances in features extraction and description algorithms: A comprehensive survey. *Proceedings of the IEEE International Conference on Industrial Technology*, pages 1059–1063, 2017.
- [24] A. I. Awad and M. Hassaballah. *Image Feature Detectors and Descriptors: Foundations and Applications*. Springer International Publishing, 1st edition edition, 2016.
- [25] B. Ghogh, M. N. Samad, S. A. Mashhadi, T. Kapoor, W. Ali, F. Karray, and M. Crowley. Feature selection and feature extraction in pattern analysis: A literature review. 2019.



- [26] A. Buetti-dinh, V. Galli, S. Bellenberg, O. Ilie, M. Herold, S. Christel, M. Boretska, I. V. Pivkin, P. Wilmes, W. Sand, M. Vera, and M. Dopson. Deep neural networks outperform human expert’s capacity in characterizing bioleaching bacterial biofilm composition. *Biotechnology Reports*, 22:e00321, 03 2019.
- [27] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.
- [28] J. Chapmann. *Neural Networks: Introduction to Artificial Neurons , Backpropagation Algorithms and Multilayer Feedforward Networks (Advanced Data Analytcs) (Volume 2)*. CreateSpace Independent Publishing Platform, 2017.
- [29] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.
- [30] J. Feng and S. Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 1237(2), 06 2019.
- [31] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. (online resource - <http://www.deeplearningbook.org>, last accessed on 2020/03/02), 2016.
- [32] D. Anderson and G. McNeill. Artificial neural networks technology. (online resource - <http://andrei.clubcisco.ro/cursuri/>), last accessed on 2020/03/20), 1992.
- [33] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *Journal of the American College of Radiology*, (online resource - <https://d2l.ai>, last accessed on 2020/04/05), 2020.
- [34] W. Di, A. Bhardwaj, and J. Wei. Deep learning essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling. 2018.
- [35] A. Khan, A. Sohail, U. Zahoor, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, pages 1–62, 2020.
- [36] L. Shapiro and G. Stockman. *Computer Vision*. Pearson, 1st edition edition, 2001.
- [37] L. Shapiro. Images and filters notes. *CSE/EE 576: Computer Vision - University of Washington*, (online resource - <https://courses.cs.washington.edu/courses/cse576/>), last accessed on 2020/04/21), 2018.
- [38] M. Hashemi. Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Journal of Big Data*, 6(1), 12 2019.
- [39] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6354 LNCS(PART 3):92–101, 01 2010.
- [40] A. Amidi and S. Amidi. Convolutional neural networks cheatsheet. *C2 230 - Deep Learning University of Stanford*, (online resource - <https://stanford.edu/shervine/teaching/cs-230/>, last accessed on 2020/04/07), 2019.

- [41] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.
- [42] J. R. R. Uijlings, K. E. A. Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013.
- [43] M. Awad and R. Khanna. Efficient learning machines: Theories, concepts, and applications for engineers and system designers. *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, 04 2015.
- [44] R. Girshick. Fast r-cnn. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 12 2015.
- [45] M. Sewak, M. R. Karim, and P. Pujari. *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing, 2018.
- [46] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:779–788, 2016.
- [47] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016.
- [48] G. Zaccane. *Getting Started with Tensorflow*. Packt Publishing, 2016.
- [49] A. Gulli and S. Pal. *Deep Learning with Keras: Implementing deep learning models and neural networks with the power of Python*. Packt Publishing, 2017.
- [50] Python Software Foundation. Python standard library. (online resource - <https://www.python.org/>), last accessed on 2020/05/15), 2020.
- [51] L. Richardson. Beautiful soup documentation. (online resource - <https://readthedocs.org/projects/beautiful-soup-4/>), last accessed on 2020/05/16), 2019.
- [52] GitHub. Electron.js. (online resource - <https://www.electronjs.org/>), last accessed on 2020/05/05), 2013.
- [53] FIZYR. Keras implementation of retinanet object detection. *GitHub*, (online resource - <https://github.com/fizyr/keras-retinanet>), last accessed on 2020/04/24), 2017.
- [54] D. M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.
- [55] C. Sammut and G. Webb. *Encyclopedia of Machine Learning*. Springer US, 2 edition, 2017.
- [56] H Nord and Eirik Chambe-Eng. Qt. (online resource - <https://www.qt.io/>), last accessed on 2020/05/14), 1995.

- [57] N. Tsoi, J. Gwak, I. Reid, and S. Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. 02 2019.
- [58] M. Zabir, N. Fazira, Z. Ibrahim, and N. Sabri. Evaluation of pre-trained convolutional neural network models for object recognition. *International Journal of Engineering & Technology*, 7(3.15):95–98, 2018.