NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FRANCISCO VALADAS DE ALBUQUERQUE

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

MASTER'S THESIS

AUTONOMOUS POWERLINE INSPECTION BASED ON COMPUTER VISION

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
, June 23, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL
AND COMPUTER ENGINEERING

MASTER'S THESIS

AUTONOMOUS POWERLINE INSPECTION BASED ON COMPUTER VISION

FRANCISCO VALADAS DE ALBUQUERQUE

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

Adviser: Doutor André Teixeira Bento Damas Mora
Assistant Professor, NOVA University Lisbon

Examination Committee

Chair: Doutor João Francisco Alves Martins
Full Professor, NOVA University Lisbon

Rapporteur: Doutor Dário Filipe Romana Pedro
CEO - Beyond Vision , (PhD) - NOVA University Lisbon

Adviser: Doutor André Teixeira Bento Damas Mora
Assistant Professor, NOVA University Lisbon

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
, June 23, 2023

MASTER'S THESIS

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.9.15) [1].

To all my loved ones,

Acknowledgements

First and foremost, I wish to express my sincere gratitude to professor André Mora who was my dissertation supervisor and always helped me unconditionally with every problem I have faced within my work by giving me his most honest and truthful opinion. Furthermore, this thesis would have never been finished without his commitment to helping me with his professional expertise and knowledge.

Also, I would like to thank Beyond Vision, specifically Fábio Azevedo and Dário Pedro, for their outstanding effort in helping me find a thesis theme that would suit the best way possible my preferences as well as for giving me enough data to work on to create something unique ultimately.

I also want to thank Nova School of Science and Technologies for teaching me essential subjects in different areas and helping me to achieve my dreams.

Last but not least, I want to thank my girlfriend for always believing in me no matter the circumstances, helping me surpass all the problems I have faced in life, and making me achieve my goals. Also, I would like to thank my family, without their unconditional support, I would never be the man I am today. Finally, I would like to thank all my friends in Portugal and around the world who have always been there for me.

Abstract

Due to recent technological advancements and increased dependence on electricity, there is now a greater demand for this resource than ever before. Consequently, new and improved methods are needed to ensure the safe and reliable functioning of the electrical grid. Over time, the components of powerline grids get damaged due to wear and tear or even accidents. If these components are not properly addressed, the consequences can be significant financial losses for electric companies and even the endangerment of human lives during blackouts. Traditional methods of inspecting Powerlines, such as a helicopter or foot patrols, are outdated because of their high financial costs and time inefficiency. However, advances in UAV technology led to be possible the mounting of these devices with computer vision systems that were once impractical due to their high prices and memory limitations.

To address the challenge of inspecting Electric Powerlines, an autonomous powerline tower detection system that uses computer vision algorithms such as deep learning YOLOv5 and Faster-RCNN algorithms is proposed. The system enables drones to fly close to the towers so that a later image capturing of components that may be faulty or at risk of malfunctioning in the future can be made. About 1500 images were collected, pre-processed and augmented in order to replicate real-life scenarios, allowing these computer vision algorithms to be trained to detect High Voltage Powerline Towers.

The usage of these algorithms has shown promising results on validation outputted metrics and processing time, with values of mAP@0.5:0.95 of 0.667 and 27.609 FPS of inference time, re-enforcing this way their potential for real-life deployment.

Keywords: Computer Vision, High Voltage Powerlines, Object Detection

Resumo

Devido aos recentes avanços tecnológicos e ao aumento da dependência da eletricidade, há agora uma maior procura por esse recurso do que nunca. Consequentemente, são necessários métodos novos e melhorados para garantir o funcionamento seguro e fiável da rede elétrica. Com o passar do tempo, os componentes das redes elétricas ficam danificados devido ao desgaste ou até mesmo acidentes. Contudo, se esses componentes não forem devidamente tratados, podem haver consequências tais como perdas significativas financeiras para empresas elétricas ou mesmo vidas humanas podem ficar em risco durante apagões. Os métodos tradicionais de inspeção de Linhas de Transmissão , como helicópteros ou patrulhas a pé, estão desatualizados devido aos seus altos custos financeiros e ineficiência em tempo. No entanto, os avanços na tecnologia de UAV tornaram possível a montagem desses dispositivos com sistemas de visão computacional que antes eram impraticáveis devido aos seus altos preços e limitações de memória.

Como solução para a inspecção de Linhas de Transmissão é proposto um sistema autónomo de detecção de Torres de Linhas de Transmissão que utiliza algoritmos de visão computacional, como o YOLOv5 e o Faster-RCNN. O sistema permite que drones voem perto das torres para que seja possível a captura posteriores de imagens dos componentes que possam estar com falhas ou em risco de mau funcionamento num futuro próximo. Cerca de 1500 imagens foram coletadas, pré-processadas e aumentadas para replicar cenários reais, permitindo que estes algoritmos de visão computacional sejam treinados para a deteção de torres de alta tensão.

O uso destes algoritmos mostrou resultados promissores em métricas de validação e tempo de processamento, com valores de mAP@0.5:0.95 de 0,667 e tempo de inferência de 27,609 FPS, reforçando assim o seu potencial para implantação na vida real.

Palavras-chave: Visão Computacional, Linhas de Transmissão de Alta Tensão, Deteção de Objectos

Contents

List of Figures

List of Tables

Acronyms

AP	Average Precision *(pp. 47–50)*
BB	Bounding Box *(pp. x, 26–30, 32, 33, 35, 36, 38–40, 42–44, 49–51, 63, 73, 83, 84)*
CL	Convolutional Layers *(pp. 30–32, 36)*
CPU	Central Processing Unit *(pp. 61, 85)*
CV	Computer Vision *(pp. viii, 1, 6, 7, 11, 17, 18, 20, 21, 24, 26, 33–35, 37, 45, 49, 50, 56, 58–60, 62–64, 66, 79, 80, 83, 84)*
DLNN	Deep Learning Neural Network *(pp. 65, 66)*
FCL	Fully Connected Layer *(pp. 30, 32)*
FPS	Frames Per Second *(pp. 7, 50, 84)*
GPU	Graphics Processing Unit *(pp. 56, 61, 70, 71, 75, 84, 85)*
IoU	Intersection Over Union *(pp. 27, 30, 40, 42, 43, 49, 50)*
lr0	Initial Learning Rate *(pp. 52, 53, 81)*
lrf	Final Learning Rate *(p. 53)*
mAP	Mean Average Precision *(pp. x, 48–50)*
mAP@0.5	Mean Average Precision with 0.5 Intersection Over Union *(pp. 49, 50)*
mAP@0.5:0.95	Mean Average Precision at 10 different values of Intersection Over Union [0.5,0.55,...,0.95] with 0.5 step *(pp. 7, 49, 50, 70–73, 75–79, 81, 82, 84)*
MPL	Max Pooling Layer *(p. 31)*
NMS	Non Maximum Suppression *(p. 29)*

PL Powerline *(pp. 1, 6–8, 10, 70, 73, 82–85)*

RCNN Regional-Based Convolutional Neural Network *(pp. x, 7, 26, 35–41, 44, 49–51, 56–58, 60–65, 68, 70, 77–79, 81, 82, 84)*

RoW Right of Way *(p. 8)*

RPN Region Proposal Network *(pp. x, 38–40, 51)*

SGD Stochastic Gradient Descent *(pp. 36, 52, 53, 76)*

UAV Unmanned Aerial Vehicle *(pp. x, 1, 6, 17, 18, 21, 23, 24, 41, 82–85)*

YOLO You Only Look Once *(pp. x, xii, 7, 26, 27, 30–34, 44, 49, 50, 52–58, 60–65, 68, 70, 71, 73–82, 84)*

1 Introduction

Since the late 20th century, electricity has taken a lead role on our everyday lives. It can be found in homes, hospitals, factories, and almost everywhere [2]. As a result, it has become almost impossible to live without this resource.

However, with this dependence comes a significant responsibility to ensure its reliable and safe delivery so that access to energy is uninterrupted.[3]

The existing methods to address this issue reveal different problems such as time, financial costs, accuracy, and dangers that they may cause to human and animal lives.[3] With this, a more significant push for a newer and better solutions started to unveil.

With the greater evolution of UAV systems and hardware for these same systems, it is becoming easier each day to implement UAVs with autonomous decisions. The hardware embedded into the UAV is becoming smaller and smaller, and they are also getting better computational resources that have enough power to run CV Algorithms in real-time and get feedback that can be used for the UAV to move appropriately and achieve the final goal of detecting Powerline (PL)s Towers.

1.1 Background of Power Lines

1.1.1 History

Nowadays, we take energy for granted, and living without this vital and helpful resource is almost impossible. Every time we drive near a highway or remote field, we usually see high-voltage and middle-voltage grids. However, around 150 years ago, all of this great technology was in its early beginning operation.

In the 1860s, viable and trustworthy DC electromechanical generators were invented. First, Werner Von Siemens created the *dynamo* to Power his telegraph systems. After this invention, he proposed two fundamental principles, the self-excitation and the reversible principle, which explains the idea of a device functioning as a motor and a generator[4, 5]. Later, in 1869, Gramme, with the help of the principles stated by Siemens, introduced the first dynamo capable of generating higher currents and voltages than ever before[5]. This way, the road to large-scale electrical power production was finally open.

At the end of 19^{th} Century, a well-known inventor, Thomas Edison, invented the earliest electricity distribution system from a centralized Power Station in Pearl Street Station, Lower Manhattan. This small step for humanity leads to serving about 80 clients of Edison's American company with Electricity in an area of about 1.6 Kilometers[5, 6]. Due to the little knowledge of this field at the time, this early distributed energy system was very inefficient since that Power plants had to be placed near the consumer.

With the help of many different inventors, by the 1890s, a new and better-distributing method of Power was employed known as AC(Alternating Current) to deliver energy over much longer distances than what was done back in the start with the DC distribution network created by Edison. After this invention, it was possible at last the usage of high-voltage distribution[7]. With the usage of AC, we would have an increase in transmission capacity proportional to the square of the voltage, and to help even more, it would come with a decrease in cost per unit of electricity delivered[6]. In such a manner, between the 1890s and Today, the voltage that Power Lines Began grew roughly from 2Kv to 1600Kv as shown in Figure 1.1. This way, the number of Kilometers that AC Power Lines can work grew significantly from a few to hundreds of them.

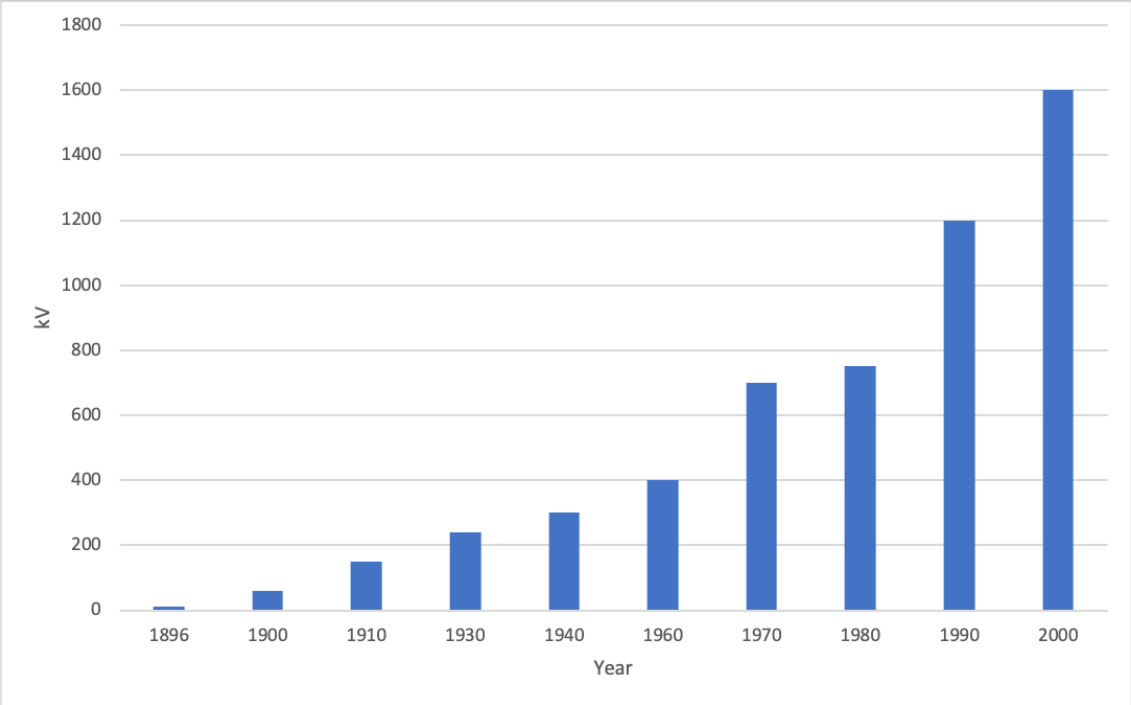


Figure 1.1: Maximum AC Voltage Rated on Power Lines per Year

1.1.2 Structure of Power Systems and Demand of Electricity

As we know, Electricity, over time, has become a favorite over other sources of energy that compete with it, not only because of its fast delivery of Power, for instance, on electric cars but also because of its versatility in its usage. Furthermore, this kind of energy has a very peculiar setback: the inconvenience in terms of price that comes with storing Electricity on a big scale. Therefore Electric Producers tend to produce energy according to the real-time demand needed, although it is possible to store energy in batteries[6].

An essential condition for electricity generation, transmission, distribution, and supply is the requirement that generation and demand be continuously and instantaneously balanced. Attached to this comes the idea that a disruption anywhere on the system might jeopardize the dynamic balance overall, which would have adverse effects on the supply of energy across a large area, possibly on a small scale like a region or even a more extensive scale that leads to significant consequences such as a whole country[6].

Nowadays, to deliver Electricity from its production to consumption points, there is a need for both a transmission grid and a distribution grid to create a unified system that operates as smoothly as possible.[6–8]. To obtain Electricity, there is an organization in terms of how Electricity flows from the producer to the consumer. The energy produced starts its journey in the **Generating Station**, flows to the **Generator Step-Up Transformer** where the Power is transformed to a much higher value to obtain better long-distance transmissions and flows through the **Transmission Lines** until it reaches a **Substation Step-Down Transformer** that will decrease the amount of Power so that it can go to

Highly Energy Intensive Businesses such as Railways also known as **Sub-transmission Costumers** that still work under voltages in the orders of 69kV to 28kV, or to **Primary Costumers** that can be Small Factories who use voltages between 4kV to 13kV.

Finally, this energy can also go to **Secondary Costumers** such as house customers who use much smaller voltages than the ones being transported by the producer in the order of 120V to 240V. To enhance the understanding of the concepts being discussed, a small diagram in Figure 1.2 is presented.[6, 7].

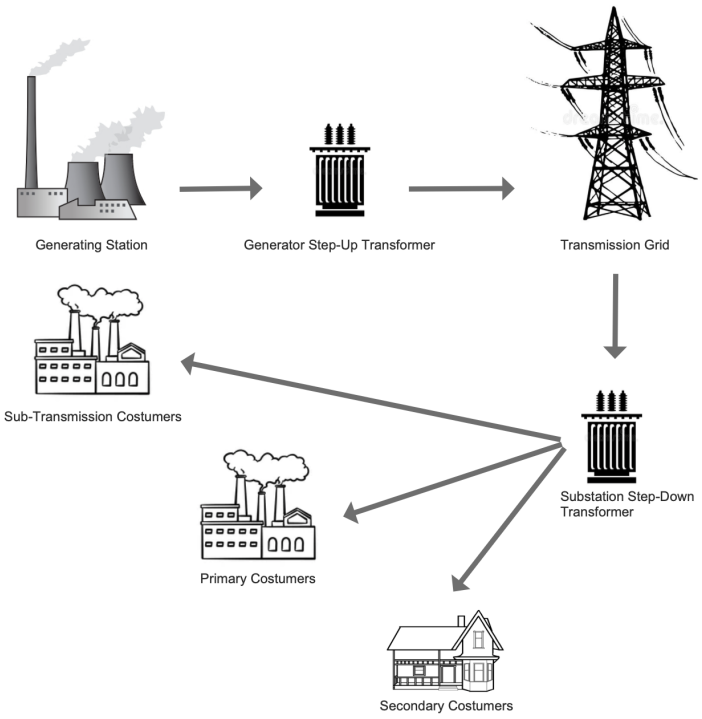


Figure 1.2: Diagram of the Key Elements in a Transmission Grid, adapted from [9–13]

Over time, there has been a significant increase in demand for electric energy. As a result, electric companies have upgraded and expanded their facilities, either by improving existing infrastructure or constructing new transmission lines and facilities. The growth has been so substantial that the investment in Transmission and Distribution of Energy is projected to reach 351 billion annually by 2026[14]. Moreover, according to the International Energy Agency's 2007 assessment, developing nations must invest a minimum of 165 billion dollars annually to meet their energy requirements[15].

As it can be seen on Figure 1.3, there has been a clear need for Electricity over the last few years. One of the reasons for this is the necessity of evolution, the creation of robots to work in factories, and the lower demand for human workers over the years being replaced by machines. Additionally, the higher usage of air conditioning and the replacement of gas to cook for electricity are factors that also changed the demand curve for Electricity[6, 15].

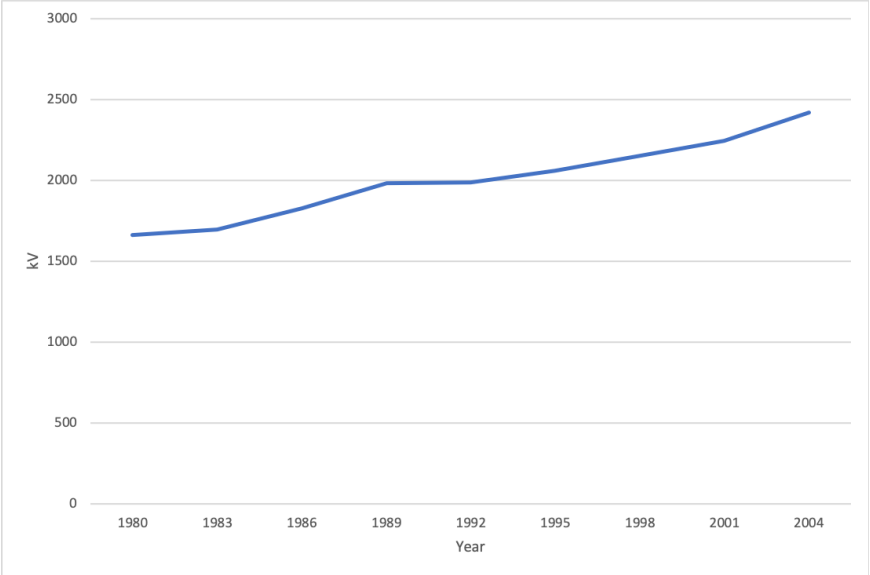


Figure 1.3: Graph of the demand for Electricity in the World in kWh per Year

1.2 Problem Statement and Contributions

To guarantee a stable delivery of electricity, maintenance and inspection of Powerline (PL)s must be done periodically. However, over the years, they can get damaged due to different factors such as Vegetation, Natural Disasters, or even the simple wear out of components within the PLs Grid. Vegetation and wear out of components is a threat that takes time to strike the PLs. Thus, natural disasters present a significant challenge due to their unpredictability and ability to cause rapid and severe damage. Although the dangers of natural disasters are significant, they are so rare and spontaneous that it is only possible to predict the other stated threats in advance.

Over the years, many different implementations have been used. At its most primordial, PLs fault detection and monitoring used to be done by foot-patrolling, but other implementations started to emerge because of its high time inefficiency. Helicopters came right after foot patrolling inspection as a possible viable solution, but due to their high cost of implementation, the path for mitigating the problem was not yet carved.

Unmanned Aerial Vehicle (UAV)s, in the past few years, have become an emerging technology in the field of Power Line inspection, and although their significant advantages are mainly their blazing pace since they lessen human risk while also making it possible to collect data from various perspectives, positions and distances from the different obstacles.[3] However, they also face some complex and challenging problems that must be overcome[16]. With the usage of this kind of vehicle, there is a tremendous challenge of avoiding all kinds of obstacles that may come in their way while performing their remote sensing job, these obstacles can vary in size and shape, but the most common ones that UAVs face are trees, Power Line conductors, buildings, branches, leaves, overhead communication cables, among others [3, 17].

Nonetheless, a key factor enters all of the solutions using UAVs, which is Computer Vision (CV). This is a very challenging factor when hiring this kind of technology. CV is responsible for automatically moving the camera in different directions to keep the electric tower in the camera's field of vision and to identify and classify the various effects and failures of the PLs infrastructure [18]. Besides this, CV is also responsible for the perception and evaluation of data that can be disturbed by various redundant obstacles such as (trees, grass, roads, fences, etc.), which can also be called background noise or illumination challenges as well as viewpoints variations and which may lead to an increase of the difficulty of the task[16, 18, 19].

CV Algorithms are Algorithms that need a special and careful attention. To properly identify the PLs Towers many different CV Algorithms can be used to this scope, however, by using Deep Learning Object Detection Algorithms, it becomes easier to have an Algorithm that can respond to its surrounding in real time because of the high inference times that they prove to have.

In order to achieve this goal, it is needed a large Dataset that perfectly mirror the reality so that important image features can be learned by the Algorithm making it powerful

and capable of being deployed on a real life scenario as well as a careful look into the training of the Algorithms, changing its parameters so that the best possible solutions can be achieved.

To do all of this gathered images of this PLs Towers need to be gathered and labeled with a careful attention so that no object misses its identification to be done during training. Just like humans, these Algorithms learn with their mistakes meaning that the commonly the longer and the better trained they are, the best models they will output.

By knowing the problem under study on this document it is possible to understand the approach taken to solve it. The main idea under study here is that Computer Vision Algorithms such as You Only Look Once (YOLO)v5 and Faster-Regional-Based Convolutional Neural Network (RCNN) are used to correctly identify PL Towers so that UAV can use this type of Algorithm embedded on their Hardware to do this Object Detection in a real life scenario. By gathering different images that replicate this real-life scenario it was then possible to successfully train an algorithm with a Precision higher than 90%, Mean Average Precision at 10 different values of Intersection Over Union [0.5,0.55,...,0.95] with 0.5 step (mAP@0.5:0.95) of 66.7% and a Frames Per Second (FPS) value of 27.609 on a T4 GPU.

By doing that, contributions such as the ones described bellow were obtained.

- Autonomous Inspection of PL Towers.
- Risk minimizing of PL inspection.
- Financial Cost minimizing of PL inspections.
- Dataset arranging for CV Algorithms usage. [1]
- Setbacks and Advantages of using Faster-RCNN and YOLOv5 CV Algorithms.

[1]The arranged and duplication verified Dataset can be found and downloaded using the following link: https://universe.roboflow.com/highvoltagetowers/tese_high_voltage_allvo8ee

1.3 Hazards to Power Lines

As previously stated, Power Lines must always work, and even the slightest problem in a Power Line can induce the producer economic damages[20]. Therefore, some central aspects must have meticulous care, such as Vegetation, Snow, Nest of birds, and Buildings, among other threats[21].

1.3.1 Trees and Buildings

In order to be possible to understand why vegetation is such a big threat to Power Line's safe operation, it is essential to understand some important Terms such as Right of Way (RoW). This important term refers to the minimum width under the PLs on which any tree that grows or any building that might be built can threaten the Power Line. Usually, in this zone, everything that happens to grow there or to be built is cleared so that the Power Line suffers no interference and has no risk of failure in its everyday work[21–23].

In the case of growing trees, they are a significant threat to Power Lines because they can damage the infrastructure or even create a Wildfire, resulting in a Power Failure[20, 24]. These kinds of electric failures can be classified into two groups, failures related to tree growth and failures related to tree failures.

According to Guggenmoos in [25], trees can lead to 3 different kinds of failures:

- Breaking the conductors of the Power Lines or bringing them to the ground
- Leading Phases to contact with each other
- Creating a significant link between phases, allowing a carbon route to form which leads to a short

Depending on the kind of tower, the precise amount of land expropriated surrounding each tower is set to a specific value. In the case of the study that I pretend to do here, I will only consider lattice towers since that is common for High Voltage Transmission.

First, to better understand the problem associated with the RoWs, we need to understand the design of the Electric Power Line Transmission Towers. According to [21, 26] the Electric Power Lines are made of the following components: Lightning Arresters, Shield wires Insulators, Power Lines, and Conductors.

- Lightning Arresters are used to shield the transformers(costly items) and other electrical apparatus from voltage surges.
- Shield Wires are smaller conductors wires compared to the standard conductor wires in Power Lines. They are the top wires in transmission lines and are used to shield the leading conductors in Power Lines from direct lightning strikes.
- Insulators are used to electrically isolate transmission lines from one another and the supporting structure. They are typically made of polymer nowadays.

- Lightning Arresters are used to shield the transformers(costly items) and other electrical apparatus from voltage surges.

- Conductors in Power Lines give a path for energy transmission from the electric producer to the electric consumer.

To summarize better what is being stated here, in Figure 1.4 there is a Power Line with the respective titles of each component.

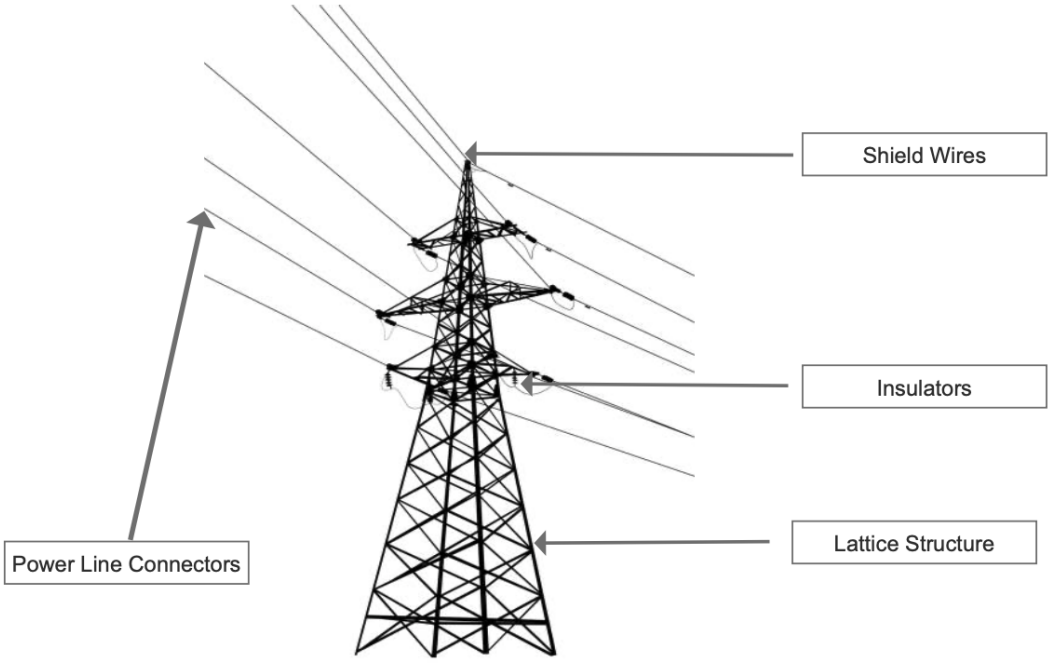


Figure 1.4: Design of an Electric Power Line, adapted from [27]

In order to warranty a safe and reliable function of the Power Lines, the RoW must maintain its normal width without having any vegetation or other potential threat[28]. However, the exact value of the RoW can vary according to different factors such as the height of the tower, size of the conductors, the distance between the towers, and the dimension of the conductors, among other different factors[21]. Although specifying the right value for the RoW is quite difficult, there are some minimum values for this piece of land that requires to be respected. Those are the ones that we can see in the table below[21].

Voltage in Kilo Volts	Width Ranges in meters
<230	15m to 37m
230	23m to 40m
345	23m to 46m
500	38m to 54m

Table 1.1: Minimum RoW underneath the Power Lines

According to Gazzea et al., when trees grow under or close to the electric power lines, they must have at least 2.0 meters of distance from the conductors so that no accidents take place.

1.3.2 Birds and Animals

Birds, among other animals, find their way of creating a home for themselves, usually at non-man-made places such as cliffs or trees or other natural structures. However, sometimes they consider Electric Power Line Lattice Towers an excellent place to build a nest and to live their own lives, reproducing there and even perching in the Power Line Conductors[22, 28]. Although this does not have a direct negative effect on Power Lines, it is essential to state that when trees are not adequately trimmed underneath the Power Lines leaving a safe RoW, they can be ignited by birds electrocutions because once again a bird touching a tree when is perching in a Power Line conductor leads to a safe path for energy to flow to ground, creating this way a short circuit and causing a fire to be ignited[22]. Therefore, it is essential to have extra care with trimming trees underneath the Power Lines so that there are no losses in wild lives.

1.4 Dissertation Outline

In this dissertation, we can find five chapters that provide a deeper description of the developed work. The first three chapters focus mainly on the theory and current state of the art regarding works with similar or the same objectives as the one under study here. The remaining chapters, however, focus on the results and the proposed tool itself, with some concepts deeper explained so that the results can be better understood.

Introduction. This chapter presents a comprehensive account of the study conducted and the tool developed, along with an exposition of the problem that motivated the research. A thorough analysis of the history of power line systems and the risks faced by power lines are also presented, focusing on highlighting the drivers that led to the creation of the proposed tool.

State of the Art. On this chapter presents an in-depth examination of existing algorithms for detecting malfunctions in PLs. Various solutions developed by researchers and practitioners are reviewed, highlighting the strengths and limitations of each approach. A

thorough analysis of different algorithms and implementations using diverse technologies focuses on understanding their underlying principles and operational characteristics.

Concepts. This chapter provides an in-depth explanation of the algorithms utilized in this study. It includes a thorough discussion of each algorithm's structure, detecting method, and evolution, which provides a comprehensive understanding of their implementations and advantages. Additionally, it contains a detailed description of the Metrics, Hyperparameters, and Common issues associated with the different Algorithms. These details help us differentiate the better solution among all trained models presented in the Results chapter.

Results. Here an explanation of the implementation methods used to obtain the results presented in this document is given. First, we describe the data collection process, including Dataset preparation and augmentation techniques. Additionally, we discuss the training process of each computer vision algorithm used in the study, leading to a final discussion on the effectiveness of each model for the object detection task at hand.

Conclusions and Future Work. The conclusion of this study presents the results obtained from the various CV Algorithms used. Additionally, the study acknowledges the limitations and areas where further work could be done to improve the research in the future.

2

State of the Art

2.1 Power Line Inspection Tools

2.1.1 Helicopters

Video surveying over Electric PLS has been, since the beginning of its usage, an excellent method to assure that the safety requirements for the normal functioning of PLS are being respected.

One type of surveying usually employed by companies that hold Electric PLS is helicopter-assisted inspections. Usually, a crew is hired to verify the Power Line's different components but nowadays it hasn't been having wide usage because of the highly impractical costs that come with this kind of inspection, both in hardware and personal expenses[29, 30]. Helicopter inspections were the first automated attempt to automate Power Line inspections[31].

For the helicopter's surveying, there are two types of possible surveying:

- Surveying using a pilot to drive the helicopter and a camera operator to record images of the PLS
- Surveying using a pilot to guide the helicopter and a camera with different sensors mounted to detect the PLS and keep recording the Power Line under its field of view.

Although the first type of Power Line surveying stated here was a massive evolution in the direction of automated Power Line inspection, it is not very reliable since it is dependent on the human vision from a long-distance and, therefore, can lead to a sparse amount of data[29, 32]. Typically, during this type of examination, the camera operator is responsible for keeping the power line in view of the camera. However, this task can be very challenging due to the need to compensate for the helicopter's movement, leading to operator fatigue and disorientation [33].

Furthermore, as pointed out by Jones et al. in [33], manually adjusting the camera's sight line is impractical as it must quickly capture and continuously track the Power Line towers as the helicopter moves along the line.

On the upper hand, the second type of surveying can be done with the help of various sensors the Differential Global Positioning System(DGPS), Geographic Information System(GIS), IR and Corona detection cameras, ALS(Airborn Laser Scanner), IR(Infra-red), among other different technologies[24, 30, 31, 33, 34]. In this type of video inspection, these sensors can conduct extensive, long-term power line examination since that sensors do not have the drawback of fatigue that humans experience[35].

On Figure 2.1 we have an image from INMR of a Helicopter doing a Power Line inspection.

Figure 2.1: Helicopter doing an inspection of Over-Head PLS, obtained from [36]

Locating poles and transmission towers is a more demanding challenge when we use sensors because different tactics need to be employed to process the data that sensors retrieve, such as understanding the shapes and particularities in towers further to identify an object as a transmission Tower or not. On the other hand, it is quite easy for a human to track a Tower since he can understand with his experience and knowledge if an object is not what it is being looked for during an inspection[33].

Helicopter surveying was indeed a good start for automatic video surveying of PLS, but at the same time, it has some great drawbacks, such as:

- The cost of employing this kind of inspection. According to [32] this can be on average 870$ per each kilometer inspected.
- Having to avoid places where low-flying helicopters can burden people[29, 33].

- Helicopters need to fly quite close to the PLS making it a dangerous task not only for the helicopter crew but also for humans living nearby[29, 30, 37–39].

- Helicopters have limited accuracy due to the high speed at which they usually travel[24, 35].

During the past few years, different approaches to using a helicopter to automate the mapping of PLS have come to life.

First, Jones et al. in [33], proposed a method where he uses active vision techniques to automatically direct the camera's sight line by identifying distinctive features of the wood support poles in the image. In this technique, they would first get the pole in the camera's Field of View by doing a geometric transformation between the helicopter's position with the help of a Differential Global Positioning System (DGPS) and the position of a particular pole that is given by a Geographic Information System(GIS) after an operator selects the spatial coordinates of the pole of interest. Subsequently, an image processing program would be employed to find the pole in the scene once it is in the field of vision.

In order to assess the picture degradation caused by camera and chopper motion during video inspection of overhead wires, the same author of the previous method, in the paper [29] devised a kinematic model of sight line geometry. The model's equation for the image Jacobian, which demonstrates that the velocity flow field of the picture is mostly uniform, links the velocity of a point on the image plane to the apparent velocity of the target object. The degree of picture blur that develops during an inspection operation on a flight path parallel to a High Voltage overhead wire is also predicted using the image Jacobian.

2.1.2 Climbing Robots

Another kind of inspection to perform the task of PLS inspection is the usage of Climbing Robots. Here instead of flying above the PLS and recording the conditions in which they are, it is used instead a Climbing Robot that travels suspended along the Power Line Conductor[31, 35, 40]. This type of inspection comes with a great inspection precision because it is made at a close range[35, 37]. Also, in high-precision jobs over a long period, this kind of robot end up doing a better job than human operators[34].

Here we could separate the robot system into two parts: the vision system and the robot system. The vision system is equipped with many different sensors to comprehend the environment around it and to plan the path for the robot[35]. However, while doing this, the vision system is also responsible for handling the duties involved in defect detection[35].

As we can see in the image 2.2, we have an example of a climbing robot doing a Power Line inspection named Explainer, produced by a Japanese Company named HiBot.

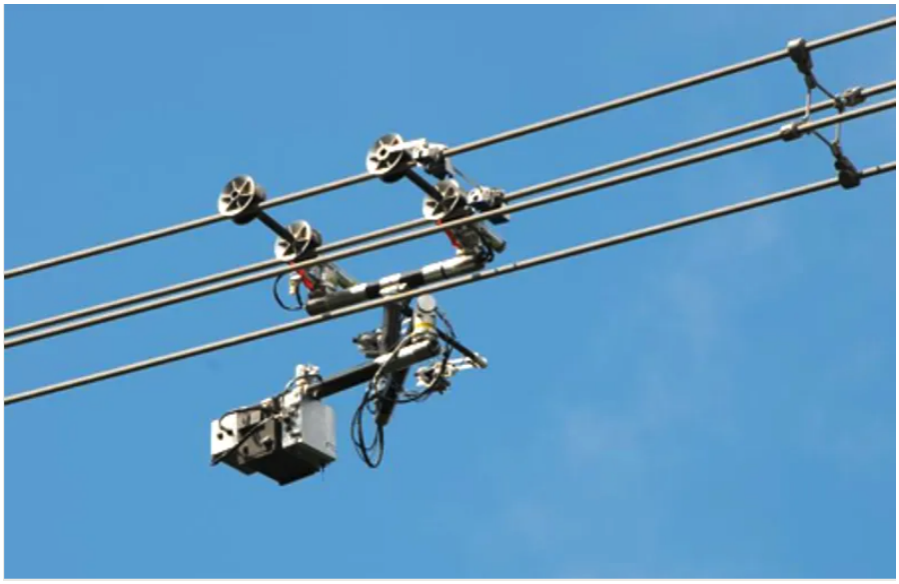


Figure 2.2: Climbing Robot produced by HiBot doing an inspection of Over-Head PLS, obtained from [41]

Climbing robots have great clear advantages over helicopters and man-walking inspections because they can be safer than hiring human workers on dangerous tasks and also cheaper than hiring not only humans but also helicopters[30, 34].

Nevertheless, according to [34, 35, 37, 40], they have some great and hard challenges that they need to overcome, such as:

- Overcoming Obstacles in the Power Line Conductors
- Shield the robot against the electric field inert in the Power Line Conductors
- Blurry data because of swinging in windy climates
- Weight of the robot

This kind of robot reveals to be hard to construct and complex in order to be able to overcome obstacles on the Power Line[31, 37].

Furthermore, due to the extremely high electric fields on the surface of the Power Line Conductors, which can reach up to 1.5 Mega Volt per meter or even higher in the presence of defects in certain Power Line components, ensuring that the electronics and sensors in climbing robots are properly shielded presents a significant and intricate challenge [31, 37].

The Blurry data spoken above is obtained because of the challenging weather conditions that this kind of robot has to deal with at times. At the same time, this data is the primary obtained data for the Power Line evaluation, and when it is blurry, it becomes unreliable. Also, these unpleasant weather conditions that lead to undesired vibrations in the Power Line Conductors cause issues with the robot's navigation, which mainly depends on a visual system[34].

Nevertheless, talking about the robot's weight in this kind of method of Power Line surveying seems like a trivial problem to deal with. However, it is always important to note that a robot body that is too heavy might endanger the electrical lines and reduce the climbing robot's ability to check them effectively[35].

Over the past few years, different designs and methods have been employed for this robot. One of them was proposed by [42] where a climbing robot would be equipped with a multi-spectral camera composed of a visible camera, an infrared system, and an ultraviolet system in order to obtain results on the Power Line components. Here the results prove to be better than using a climbing robot with the traditional viewing methods making it a reliable, cost-effective, safer, and better option.

Also, another approach to the usage of climbing robots was made by Yang et al. in [43]. Here a Bi-brachiate inspection robot was designed to inspect a High Voltage Power Line, and it would implement an adaptive fuzzy logic method that was used to control the inspection robot so that it could navigate obstructions on the power lines.

Another method was proposed by Huang et al. in [44]. They created a light climbing robot for the high-voltage grid's insulator identification. It could achieve insulator detection with zero resistance value by moving alongside the power insulators with the help of different technologies. However, it had a significant setback because of the light mechanism structure, and because of this, it had a weak capacity to pass obstacles.

Today many different institutions are already developing Power Line maintenance robots, such as the Canadian Hydro-Québec Research Institute, which is creating a multi-purpose mobile robot that can perform Power Line conductors inspection and do live maintenance of the same. According to them, the control and operating systems were built in a way that could be easy to understand for the buyer.

A different robot was built with HiBot Corporation, Kansai Electric Power Corporation, and the Tokyo Institute of Technology in 2002. Here, in order to circumvent obstructions found on Power Lines, functions were devised. Additionally, the counterbalancing box in the navigation for obstacle avoidance was operated to sustain the robot.

There was also a group of institutions that built this kind of Robots. Those were the Chinese academy of sciences (CAS) and universities (Wuhan University, and Shanghai University, among others). They have built many prototypes, such as AApex-D, AApex-C, among others.

Finally, the University of KwaZulu-Natal in South Africa created an inspection robot to avoid obstacles by changing the angle between two V-shaped arms.

As we can see, climbing robots can be a good and reliable resource for power line inspection. They not only substitute human working men for hazardous tasks but also can provide a good amount of data to verify the conditions of the different components in a Power Line.

2.1.3 UAVs

With the rapid development of technologies and the development of aerial robotics as a subject of study, UAVs started to be an inspection method [19]. This method of Power Line inspection proves to be a better solution than most of the other ones already stated in this document because it comes as a cheaper solution and has better mobility than most of the others. Of course, the traditional human walking alongside the Power Line method is safer than the other methods and comes with a great inspecting efficiency on the PLS since they can gather information on various positions, angles, and distances from the assets[3, 16, 35, 45–47].

As we can see, UAVs have many different vantages compared to their competitors on PLS surveying. Compared to helicopters, they decrease significantly not only the operating costs but also the danger to human beings in flying accidents[38]. Also, according to Liu et al. when using the standard human inspection method or the UAVs transmission line inspection technology, it is explained that the latter has a more productive operation.

As the studies on UAVs inspection of Power Line's different components increase also, the idea that comes to be more and more searched on this kind of technologies is a UAV mounted with CV in order to be able to identify different faults on PLS. In figure 2.3, we have an example of a Multi-rotor UAV produced by DJI, inspecting a Power Line.

Figure 2.3: Multi-Rotor UAV produced by DJI doing an inspection of Over-Head PLS, obtained from [49]

In this area of inspection where UAVs have been employed, we can highlight some different types of UAVs for different types of inspections on PLS. According to [46, 48] we can enumerate the following:

- *Fixed Wing UAV*, this kind of UAV is mainly used for Vegetation monitoring and

Rough Inspection of PLS.

- *Multi-rotor UAV*, this kind of UAV is mainly used for Inspection of the Condition of Towers by acquiring detailed images.

When discussing Fixed-Wing UAVs, it's crucial to note that they typically have higher speeds and fly at greater altitudes than Multi-Rotor UAVs, as per [46]. On the other hand, they pose a greater challenge in terms of landing and take-off than the latter, as mentioned in [48].

However, when we look at Multi-Rotor, it is important to highlight that the endurance of this kind of UAV, when compared to the Fixed Wing ones, is smaller, and therefore they require that the inspector transport with him numerous batteries so that the operation under different tasks by this kind of UAVs can be done[17]. Nevertheless, this option of UAVs is cheaper than the latest but proves to be harder for them flying at high altitudes and dealing with certain amounts of wind[45].

UAVs by themselves have the advantage of being able to carry many different sensors and technologies with them in order to perform their job well. This can be lightweight digital cameras, thermal cameras, GPS receivers, IMU, barometers, magnetometer, TIR(Thermal Infrared), color cameras, accelerometers, LIDAR, among others [19, 37–39, 46].

Even so, just like any other technology, they have some setbacks although they might be the best option nowadays to do Power Line detection when looking at the different types of components they can detect and give feedback. According to [16, 18, 50] this setbacks can be:

- There are intricate power line networks in the operational area that the UAV's mission must detect and avoid. This, together with the fact that numerous background noises (such as grass, trees, fences, and highways), can compromise the visual perception of the surroundings, making the job of locating PLS manually or automatically challenging.

- Some UAVs payload restrictions help build up the challenge of adding heavy sensors such as LIDAR to detect electrical lines.

- The autonomy of the UAVs sometimes cannot comply with what is expected in Power Line's inspection, making it a challenging problem to deal with.

- Need for a Complex and Flawless Algoritm to do accurate checks, especially in bad weather circumstances like a strong lateral wind.

- Viewpoint variations while flying the UAV as well as illumination changes.

The challenges we have discussed require a solution that is both adaptable and dependable. To address them, various CV techniques have been utilized. These techniques

involve different Algoritms that are designed to achieve specific objectives, such as detecting Power Line Conductors amidst noise sources like fences, trees, grass, and roads. Examples of these Algoritms include PLInED [16], LIDAR-based methods [19], and Power Line Inspection using Deep Learning [30].

2.2 Computer Vision

In order to better understand what kind of CV algorithms exist, it is fundamental first to understand what CV is.

CV is a branch of Artificial Intelligence that enables machines to analyze visual input, such as pictures and movies, and to produce patterns for identifying, tracking, and categorizing things[51, 52]. Unlike computer graphics, the field of CV focuses on extracting information from pictures[52]. Most CV jobs have to do with feature extraction from input scenes (digital pictures) to get information about events or descriptions[52]. In this area, many different Image Recognition processes can be employed. These are Image Classification, Object Localisation, Object Detection, and Image Segmentation.

- Image Classification is a Machine Learning problem that consists of labeling with one or more labels and input using pre-determined labels[53]. In other words, it consists of identifying the class to which an image belongs[54].
- Object Localization, this method consists of locating the approximate whereabouts of an object in an image, giving as output the coordinates of the object's location to further be possible to draw a bounding box around it[55].
- Object Detection, on the upper hand, is a combination of both previous methods. This method involves two tasks. The first task is known as the position task, which involves locating an object and drawing a bounding box around it. The second task is the classification task, where the goal is to identify the category of the object within the bounding box and determine the probability that the predicted category is correct. [53, 56]
- Image Segmentation is a method that can be formulated as a classification problem where just like Object Detection, a class is given to an object of interest. However, instead of drawing a bounding box around the object, the next step is pixel-level labeling of an object of interest, returning a delineation pixel by pixels of the object of interest located on a picture[57, 58].

This way, we can now understand that different processes can output different types of recognition around an image. These different types of processes are used for different purposes, such as autonomous driving, analyzing medical images, video surveillance, and face recognition, among many other possible applications[54, 56, 57].

2.3 Computer Vision on Power Line Fault Detection

CV has played a critical role in automatically identifying power line components, including cables, insulators, and electric towers, since its inception in identifying faults in PLS [59]. Also, according to Miralles, Pouliot, and Montambault in [59], given that both robots and Unmanned Aerial Vehicles (UAVs) require CV for guidance, navigation, and control in addition to inspection, CV looks to be one of the most crucial technologies for autonomous vision-based power line inspection[59].

According to Pagnano, Höpf, and Teti, some of the biggest challenges in the area of UAV autonomous inspection of electrical PLS are the following:

1. *Visual Servoing*: Many different sensors are needed to reach autonomous navigation that follows the PLS. More than just using a sensor like GPS, it is required to do it reliably.

2. *Identification and Avoidance of Obstacles*: To achieve an autonomous system that inspects PLS, it must not crash into the Power Line so that no dangers and shortages come from this inspection.

3. *Robust Control Algoritms*: They ensure excellent stability and detailed, in-depth close-range investigations while suffering adverse weather conditions.

As we know, in order to overcome these complex challenges, it is imperative to use CV as the main solver. According to Martinez et al. in [30], UAVS must be mounted with CV to overcome problems such as:

- Detect and Locate the Towers under the FOV of the UAV.
- Keep tracking the Tower on the following frames.
- Aim the camera to bring and keep the Tower in the center of the FOV.
- Depending on the kind of inspection, move the UAV and camera when the Tower is in the FOV to focus on the Tower's inspection targets.

2.3.1 Usages of Computer Vision

2.3.1.1 Detection Of Power Lines

The detection of PLS is a well-researched topic. Compared with the others stated here, we can even consider it the most researched one[59].

PLS can be detected with various technologies such as Visible Cameras, Visible Enhanced with Infrared Cameras, or even a Millimeter Wave Radar[59]. There is also another way of detecting PLS, which is detecting them by the magnetic field. However, knowing

that during power shortages, there is no energy passing the wire, the detection of the magnetic field becomes compromised, and therefore this process will not detect above-ground cables and wires[59].

In the end, the result of using this kind of technology is always a 2D image of the ground that ultimately can have some PLS on it.

After this, each method employs the same fundamental approach: after creating a binary representation of a collection of line segments, a higher-level processing stage attempts to separate non-power line segments and combine and link line segments that are part of PLS[59].

PLS are small in diameter. EHV(Electric High Voltage) PLS usually have 4 centimeters in diameter. Therefore, an image usually only occupies the width of 1 to 3 pixels[46, 59]. Knowing this, there is usually no room for texture detailing on the picture to do a grouping based on this parameter. Therefore, the categorization and detection algorithms are only based on geometric hypotheses. Sometimes even a third stage is included to ensure that the frames are time consistent and therefore grouping based on this parameter as well[59].

2.3.1.2 Inspection and Detection of Tower Components

As we have seen, Power Line Towers have different components, such as Insulators, Catenary Split Pins, and Support Components. These components have a life span on which they can work. Therefore, they can wear off and get damaged in many different ways. For this same reason, inspecting the Tower is compulsory to check all of these components. This inspection is usually done with visual data and with the help of other technologies, such as Infrared or Ultra-Violet Cameras, depending on which component is usually meant to be inspected.

When assessing the effectiveness of a method for performing this task, speed and precision are typically the key factors used to gauge its quality and robustness.

2.3.1.3 Power Lines Inspection

In order to detect early signs of possible failures, a Power Line Inspection is imperative. With this, those early symptoms of wear-off to different components in PLS can be detected, preventing a possible future failure of the mechanism.

If the outwards layer of the PLS is affected, proper Visual Inspection can be used to detect this problem. After the data is acquired, it can be checked by the proper system to qualify and quantify the threat in terms of how dangerous it may be.

2.3.1.4 Power Lines RoW Maintenance

As we know, maintaining a proper Power Line Corridor well trimmed is very important in order to prevent any Power outages or any other problems with the PLS that can cause financial damage to Power Line companies.

In the field of tree monitoring, many different approaches were already stated. Most of them would track the growth of the trees in height and width and track their location.

2.3.2 Different Implementations with Computer Vision

Here we can highlight different approaches expressed by different authors to understand better what is being stated. First, we can highlight Yan et al. in [60], the authors proposed a method where first they would do a Power Line Pixel Detection on which one of two different low-level approaches could be applied.

The first approach was a Line Detection Mask, where the goal was to get a strong response from the filter in 4 different orientations (0º, 45º, 90º, and 135º) to classify the object of interest as a Power Line.

The second approach instead would be a Ratio Line Extraction where the idea was to get the response of the edges detector between two different regions. If the response of the line detector were good enough, it would be considered a line object.

We would get a Binary Image of Edges using any of these previous operands.

After using one of the two stages stated, a Radon Transform would be employed to detect the PLS by computing the line integrals from several sources along parallel routes in a specific direction. Subsequently, a Line Segment Grouping was made along the approximate orientation of the PLS, segmenting it according to co-linearity and proximity to endpoints. With this step, it is possible to have most of the disconnected line segments rejoined nicely.

Later, as a final step to join the few gaps missing, a Kalman filter was applied in the lines left by the previous step. Every line was thought of as the path a moving point takes along a straight line. Every time a gap was detected, the broken-off pixels along the tracking direction had a predetermined number of assignments to join the gaps effectively.

Another different approach was presented by Zhang et al. in [46]. This paper proposed the usage of Block Bundle Adjustment to acquire the external orientation of images from UAV photos and ground control points. After this, an automated measurement technique for PLS based on epipolar restrictions named PLAMEC was proposed to obtain the PLS spacial position. This technique would extract power line 3D vectors with stereo picture pairings of corresponding photographs from several flight strips.

Subsequently, another technique to Semi Patch Match based on epipolar constraints would be employed. This technique was also known as SPMEC to automatically obtain dense point clouds from the path of the PLS. With this technique, a 3D reconstruction could be built of all the group's objects. Finally, the spatial separation between the Power

Line and the ground point cloud collected from the optical imaging would be calculated as the last step to identify and locate barriers automatically.

The results of this method proved to be great, achieving a success rate of 93.2% on Power Line's recognition.

Martinez et al. proposed a solution named PoLIS(Power Line Inspection Software) [30] where cooperation between CV and Machine Learning was made in order to reduce the amount of work that operators have to deal with when analyzing video data acquired by a UAV or a Helicopter on PLS.

The main idea here would be to give only keyframes to the operating inspector to reduce the quantity of data the inspector would need to analyze drastically.

The most important part of this technique would be the Tower Detection Stage. In this stage, the following methods were applied:

1. *Hough Transform*: This was applied to detect vertical structures in the input image.
2. *Sliding Algorithm*: Here, a supervised classifier would be applied to all the areas of the picture to classify something as a tower or not.
3. A *K-means* clustering algorithm was applied to remove the false towers wrongly detected as such.

At the same time the previous stage was being held, another stage called Tower Tracking Stage was also working. In this stage, a strategy named Hierarchical Multi-Parametric and Multi-Resolution Inverse Compositional Image Alignment Algorithm (HMPMR-ICIA) was employed to determine the Tower's location more accurately in various frames.

After passing the Tower Detection stage, the Tower Tracking would put HMPMR-ICIA algorithm into action to estimate the Tower's position on the next frame. If the tracker stopped tracking the Tower for some reason, the Detecting Stage was employed again until a tower was positively detected so that the Tracking Stage could start its job again.

In the end, the PoLIS software entered the Key Frames selection Stage, where some of the frames were considered Key Frames and stored to be inspected by the operator. In this stage, different Frames at different distances were stored so that the operator had different angles and positions to the Tower. Therefore, he could at least find the information he intended to inspect in one of the frames.

A different approach was taken by Santos et al. in [16], where a vision-based power line detection algorithm was created, named PLineD. This algorithm aimed to improve the detection robustness with noise in the images.

In this algorithm, the following techniques were employed in order to reach the pretended goal:

1. *Edge Drawing*: Here, a 5x5 Gaussian kernel was first used and then a Sobel Operator, followed by an Anchors detection to finally get the edges with the help of the direction of progress for the gradient direction.

2. *Segments Cut*: In this section, the edge segments were separated into horizontal and vertical.

3. *Segment Covariance*: At this stage, the idea was to delete the uncut little and small curved segments not deleted in the previous section.

4. *Group Segments*: In this section, the goal was to group segments according to different pre-noted restraints so that a possible Power Line could be detected.

5. *Parallel Segments*: There was a search for parallel segment groups according to different constraints to finally consider something in an image as a Power Line or not.

According to the authors of PLineD, this approach gave excellent results in Power Line detection in pictures with many different noises.

As we can see, many different approaches were presented in this inspection field.

3

Concepts

When we are on the street walking on the sidewalk, and something fetches our attention, like the traffic or an interesting animal, or even a crosswalk that has different colors from the typical black and white, we can right away identify what kind of object, animal or construction we are looking at.

As we know, we can clearly identify if a firetruck or an ambulance is passing the cars in the traffic, if the animal we just came across 5 minutes ago was a cat or a dog, and even what kind of a dog or cat it was. Therefore we can understand that humans have a tool that we take for granted and that we think is easy to replicate. However, for a computer, that is not the case.

In this chapter, it will be possible to find different approaches that have been proposed to solve this Object Detection problem over the years and also some background about Metrics, Hyperparameters, and Common Problems within the different algorithms that will help us to understand better the decisions taken on the next chapter.

3.1 YOLO Family

YOLO, which is short for "You Only Look Once," is a Deep Learning CV Algorithm that has gained notoriety over the years for its impressive speed and accuracy in detecting objects, as well as its comparatively small size and memory footprint when compared to other Deep Learning Algorithms used for the same task.[61–63].

YOLO algorithm is a One-Stage Detector and a Regression-Based Detector, so it essentially discards the inquiry of the region recommendation, which is the first step in the two-stage detectors[64]. In one run, it predicts classes and Bounding Box (BB)s for the entire image rather than just the relevant portion. In other words, it gets directly to BB coordinates and class probabilities from picture pixels[62, 65]. This family of algorithms is so simple in design that it can be implemented in machines with scarce system resources[63]. Unlike the RCNN family of algorithms described in Section 3.2, YOLO performs object classification and detection simultaneously[66].

Even while the YOLO family algorithms outperform all two-stage detectors in speed,

this speed is at the expense of accuracy. Nevertheless, this Algorithm is an excellent real-time object detector because this accuracy is not much lower than the one linked to the two-stage detectors with PASCAL VOC 2007 Dataset.[66].

3.1.1 Detection

To comprehend how this Algorithm operates, it is imperative first to understand its earliest version. In 2015, Joseph Redmon introduced YOLOv1 as a novel approach to Object Detection, as described in [62]. This Algorithm was a pioneer in employing one-stage detectors during the Deep Learning Era[67].

This model functioned in a completely different way from the two-stage detectors. Here, a single neural network was employed for the full image[67].

The concept behind YOLOv1 is to divide an image into a grid of cells, each with a size of S x S (the default size being 7 x 7). When the center of an object falls within a grid cell, that cell gets in charge of detecting the presence of that object. This means that other cells disregard the object's presence, even if partially visible within their area[62, 68]. After dividing the input image into a grid, the model predicts B BBs and confidence scores for each box within each grid cell[62]. These confidence scores, as shown in equation 3.1, indicate whether an object is present in the BB.

$$Confidence = Pr(\text{Object}) \times \text{IOU}\left(\frac{\text{Predicted}}{\text{Truth}}\right) \tag{3.1}$$

Here, the value of Pr(Object) represents the probability that an object exists in the grid cell, while the value of IOU($\frac{Predicted}{Truth}$) is the Intersection Over Union (IoU) between the BBs of the Ground Truth Object and the Predicted Object, on Section 3.3.1 a detailed description of this last metric is given[62, 68].

The Object's Probability, represented as Pr(Object), ranges from 0 to 1. Therefore, the value from the Confidence score can only be between 0 and the value of the previously mentioned IoU. Therefore, a Confidence score of 0 indicates the absence of an object within the BB. At the same time, the highest value implies that the BB perfectly aligns with its Ground Truth BB[62, 68].

The concept of object detection within a grid cell is based on the notion that the object's center must be situated within that specific grid[65]. Every time a BB is prompted around an object, a target vector like 3.2 is created:

$$\left[p_c, \quad b_x, \quad b_y, \quad b_w, \quad b_h, \quad p_{c_1}, \quad \ldots, \quad p_{c_n}\right] \tag{3.2}$$

The components of this target vector as described:

1. The confidence score that an object is inside the BB, as explained previously, is represented by p_c[65, 68].

2. The center of the predicted BB is represented by the values (b_x, b_y)[65, 68].

3. The width and height of the BB are represented by (b_w, b_h)[65, 68].

4. The likelihood of each cell containing an object of a specific class, among the various classes c_1 to c_n provided to the Algorithm, is represented by the probabilities p_{c_1} until p_{c_n}.[65, 68]

This approach enables the detection of multiple objects of different classes within the same grid cell, each with a corresponding probability value. Hence, whenever a BB is generated, there is always a probability associated with it for each possible object fed into the algorithm.[62, 65].

All input images undergo normalization to address potential issues stemming from variations in image size. This ensures that all images are later resized to a fixed size, preventing problems such as attempting to access nonexistent pixels. The values mentioned before are determined through the following equation:

$$b_x = \frac{\text{X Coordinate}}{\text{Image's Width}} \tag{3.3}$$

$$b_y = \frac{\text{Y Coordinate}}{\text{Image's Height}} \tag{3.4}$$

$$b_w = \frac{\text{Width of the Object}}{\text{Image's Width}} \tag{3.5}$$

$$b_h = \frac{\text{Height of the Object}}{\text{Image's Height}} \tag{3.6}$$

The values mentioned above regarding the position of the BBs are normalized by utilizing the equations 3.3 until 3.6, which ensure that all values are confined within the range of 0 to 1. Figure 3.1 [62, 68] is an example of values normalization. This standardization makes it easier for the Algorithm to process the data accurately without encountering issues related to mismatched image sizes.

- size = 640x640
- (x,y) = (452,320)
- bx = 452/640
- by = 320/640
- bw = (594-310)/640
- bh = (640-0)/640

Figure 3.1: Normalization of BBs

Once all the target vectors for the BBs in each grid cell of the image have been collected, an output predictor tensor similar to the one in Equation3.7 is produced[62].

$$Output = S \times S \times (B \times 5 + C) \tag{3.7}$$

In this context, the size of the processing grid for the detector is determined by the values of S × S. The B × 5 notation represents the number of BBs retrieved per cell, where each BB is defined by the first five variables outlined in the target tensor shown in 3.2. Lastly, the number of Classes predicted within the architecture is represented by the variable C, as described in [68].

As we can see on the target tensor 3.2 a B amount of BBs is gathered for the Output tensor. These B Bounding Boxes are suggested per grid cell and known as Anchor Boxes. They have different shapes and sizes, so a better job of gathering the object within the image can be made[62, 65, 68].

As depicted in Figure 3.2, many proposed BBs with varying confidence values are obtained after the final step. However, many of these BBs are either mismatched with the object or overlap with other BBs containing the same object, as noted in [65, 68].

In order to solve this same problem, two steps are taken to lower the amount of obtained BBs.

The first step is to apply a Confidence threshold, which filters out proposed BBs with values lower than the threshold. Typically, the Threshold is set at 0.5, but it may vary based on the Algorithm's intended implementation.[62, 65, 68].

Even after suppressing imprecise BBs, many BBs may remain, necessitating further action [62, 65, 68]. As a result, a second step known as Non Maximum Suppression (NMS) is employed to address these BBs. This process involves selecting the BB with the highest

Confidence probability and then discarding all other BBs around it that overlap the earlier with an IoU higher than a threshold, typically set at 0.5, but may vary depending on the application.

This last step is repeated for all the BBs until no BB is left, which verifies the second step, leading to a Final Detection like the one seen in Figure 3.2.

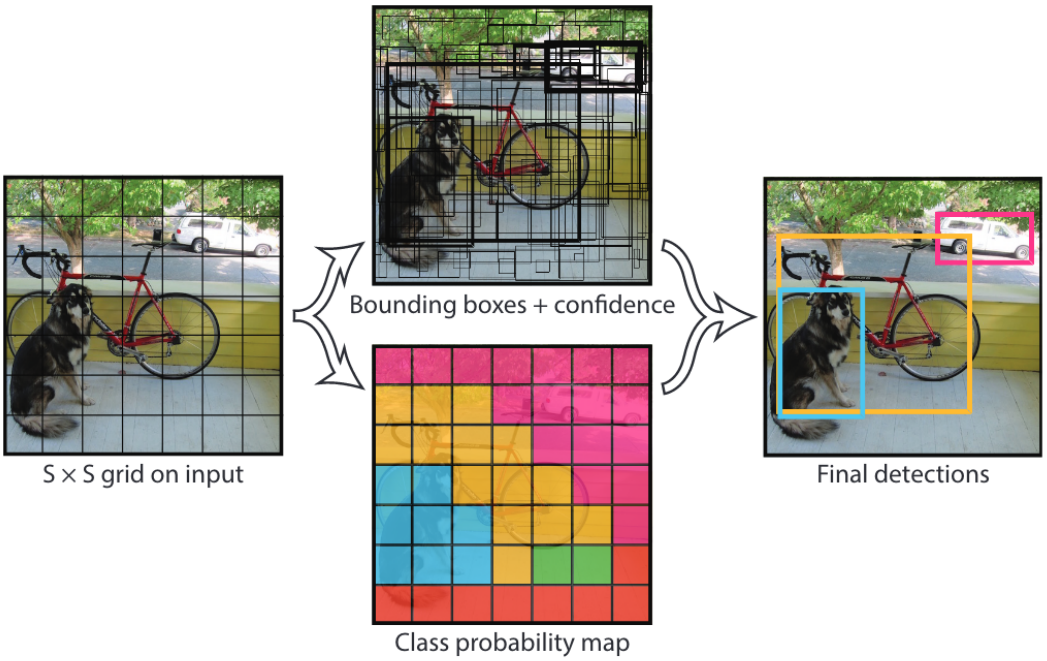


Figure 3.2: YOLOv1 Working Method, obtained from [62]

3.1.2 Architecture

YOLO architecture was inspired at first on GoogleNet architecture[62, 65]. On YOLOv1, we have 24 convolution layers, 4 max-pooling layers and 2 Fully Connected Layer (FCL)s on the end of the architecture[65].

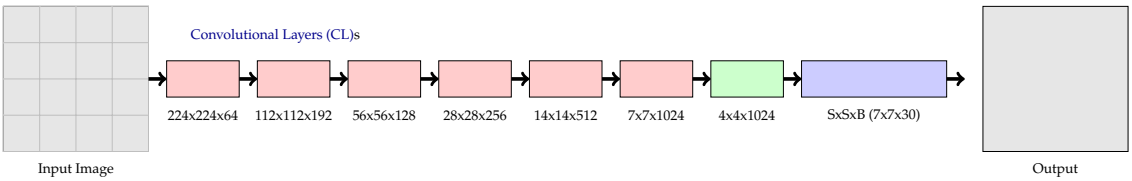


Figure 3.3: YOLOv1 Architecture

The CLs in YOLOv1 extract essential features from the input image and reduce its size. These layers apply filters to the image to detect features such as edges, corners, and textures. These detected features are then processed through multiple layers to generate

Layer Type	Output Shape	Filter Size	Stride	Number of Filters
Input	$448 \times 448 \times 3$	-	-	-
Convolutional	$448 \times 448 \times 64$	7×7	2	64
Max Pooling	$224 \times 224 \times 64$	2×2	2	-
Convolutional	$224 \times 224 \times 192$	3×3	1	192
Max Pooling	$112 \times 112 \times 192$	2×2	2	-
Convolutional	$112 \times 112 \times 128$	1×1	1	128
Convolutional	$112 \times 112 \times 256$	3×3	1	256
Convolutional	$112 \times 112 \times 256$	1×1	1	256
Convolutional	$112 \times 112 \times 512$	3×3	1	512
Max Pooling	$56 \times 56 \times 512$	2×2	2	-
Convolutional	$56 \times 56 \times 256$	1×1	1	256
Convolutional	$56 \times 56 \times 512$	3×3	1	512
Convolutional	$56 \times 56 \times 256$	1×1	1	256
Convolutional	$56 \times 56 \times 512$	3×3	1	512
Convolutional	$56 \times 56 \times 256$	1×1	1	256
Convolutional	$56 \times 56 \times 512$	3×3	1	512
Convolutional	$56 \times 56 \times 256$	1×1	1	256
Convolutional	$56 \times 56 \times 512$	3×3	1	512
Convolutional	$56 \times 56 \times 512$	1×1	1	512
Convolutional	$28 \times 28 \times 1024$	3×3	1	1024
Convolutional	$28 \times 28 \times 512$	1×1	1	512
Convolutional	$28 \times 28 \times 1024$	3×3	1	1024
Convolutional	$28 \times 28 \times 512$	1×1	1	512
Convolutional	$28 \times 28 \times 1024$	3×3	1	1024
Convolutional	$28 \times 28 \times 1024$	3×3	1	1024
Convolutional	$28 \times 28 \times 1024$	3×3	2	1024
Convolutional	$14 \times 14 \times 1024$	3×3	1	1024
Convolutional	$14 \times 14 \times 1024$	3×3	1	1024
Convolutional	$14 \times 14 \times 1024$	3×3	1	1024
Flatten	50176	-	-	-
Fully Connected	4096	-	-	-
Fully Connected	1470	-	-	-
Reshape	$7 \times 7 \times 30$	-	-	-

Table 3.1: Layers in YOLOv1

even more complex features, ultimately resulting in a feature map that can be used for object detection.

We also have Max Pooling Layer (MPL)s on this architecture, just like in Figure 3.3 and Table 3.1.2. On the upper hand, these Layers are used for down-sampling the feature maps generated by the CLs. Therefore, their primary purpose is to reduce the size of the feature maps and the number of parameters in the network. Max pooling works by selecting the maximum value from a set of adjacent pixels within the feature maps and retaining that maximum value as the representation for that region. This reduces the spatial dimensions of the feature maps, enabling the network to learn more abstract representations of the

data while retaining important information.

At the end of the architecture, we finally have FCLs. These Layers use the information extracted from the Convolution Layers to make predictions about the presence of objects in the image. They are used to perform the final classification of the object in the image and predict the BB coordinates for the detected object. Here, they receive the output from the previous layer and process it to produce the final predictions, which include the class probabilities and the BB coordinates. Finally, the final output from the FCLs generates the final detection results.

According to Thuan in [68], on YOLOv1, the initial 20 CLs of the architecture were utilized for training the feature extractor, which was fed an input image of size 224 x 224. Subsequently, the architecture was augmented with an additional 4 CLs and 2 FCLs, and the input image size was increased to 448 x 448 to serve as the object detector. In addition, the two FCLs previously mentioned are also responsible for predicting the output probabilities and coordinates[62].

3.1.3 YOLOv5

This YOLO version was selected for this dissertation because it was the only version written in PyTorch at the time of this research. It also had ample documentation available on the internet, making it an easily implementable model compared to its predecessors, which were written in C using Darknet.

3.1.4 YOLO Evolution

Inspired by the GoogleNet Architecture, the YOLO algorithm, like every other Object Detection Algorithm, evolved a lot over time[65, 66].

This first version of YOLO was off-charts at the time because it could produce about 45 frames per second, giving this way no competition to its opponent algorithms in terms of speed[69, 70]. However, this Algorithm, when compared to the other State of the Art methods, proved to outperform them, even though it had a higher rate of localization errors and a lower propensity to forecast false positives[62, 70].

About a year later, YOLOv2 and YOLO9000 were made by the same authors as the previous version. The need for a new version came after YOLOv1 made many localization errors and had a low Recall rate when compared with other Algorithms[61, 71].

This way, YOLOv2, just like the original paper states, became better and faster than its predecessor. These improvements were made with the following changes to the Algorithm:

- Batch Normalization
- High-Resolution Classifier
- Convolutional With Anchor Boxes

- Dimension Clusters
- Direct Location Prediction
- Multi-Scale Training
- Darknet-19
- Training for classification
- Training for Detection
- Hierarchical Classification
- Dataset Combination with WordTree
- Joint Classification and Detection

In 2018 the last update from the original creators was made. This last update was named YOLOv3, and Joseph Redmon, the original creator of YOLO, decided to stop research on CV because of the repercussions this Algorithm had on Military Applications[68].

YOLOv3 saw some improvements compared to earlier versions, including multi-scale detection, multi-label classification, and the addition of an objectivity score for BB prediction[61, 70, 72].

After YOLOv3, the following versions were made by other authors beside the original ones.

First came YOLOv4. In this version, several improvements were made, those were mainly:

- Improved Network Architecture: YOLOv4 has a new network architecture combining various architectures (ResNet, SPP-Net, EfficientNet) for improved accuracy and efficiency in object detection. The use of EfficientNet remarkably improves speed and accuracy[73].
- Better anchor scaling: YOLOv4 uses anchor scaling that adapts to the shapes and sizes of the objects in the image, providing better detection accuracy.
- Better class imbalance handling: YOLOv4 uses a combination of focal loss and label smoothing to handle class imbalance better, improving detection accuracy for rare classes[73].
- Improved Data Augmentation: YOLOv4 uses a new data augmentation technique called "Mosaic data augmentation" that leads to improved generalization and accuracy[73].

Approximately one month after the release of YOLOv4, Glen Jocher introduced YOLOv5. This version was highly similar to YOLOv4, and surprisingly, no technical paper was published to accompany it[68].

The introduction of YOLOv5 marked a significant milestone in the evolution of object detection algorithms. Unlike its prior versions, coded in C, YOLOv5 was written in Python, offering several advantages. Firstly, it made implementing and integrating the Algorithm on IoT devices much more accessible. Secondly, the larger PyTorch community, as compared to the Darknet community, which was associated with the previous YOLO versions, opened up the potential for broader development and improvement of the algorithm[68].

Glenn Jocher significantly improved YOLOv5 by incorporating the anchor box selection process into the architecture. As a result, the network can determine the optimal anchor boxes for any input Dataset without requiring manual adjustment. This eliminates the issue of anchor boxes being unable to adjust to Datasets with classes not found in the COCO Dataset, which was previously a common challenge in CV[68].

On September 7^{th} 2022, another version of YOLO, YOLOv6, was released while this project was under development. This new version boasts several significant improvements, including advancements in the network design, label assignment, loss function, industry-friendly features, and quantization and deployment processes[74].

Interestingly, YOLOv7, the latest version of YOLO, had its paper released about 3 months before YOLOv6. On this version, according to [75], the main contributions that were made to the Algorithm were mainly:

1. A new design of bag-of-freebies methods
2. New methods to address difficulties related to the evolution of object detection methods(re-parameterized modules and dynamic label assignment strategy)
3. The introduction of both "extend" and "compound scaling" methods for real-time object detectors.

3.2 RCNN Family

In this section, similar to the previous one, an explanation of the RCNN family of algorithms will be given at the start. Then, a review of each version of these algorithms, starting with the first one and moving on to subsequent versions, culminating in the Faster-RCNN CV Algorithm, which was selected for use in this work, will be given.

The RCNN family of object detection algorithms was one of the early adopters of deep learning for object detection[67].

RCNN stands for Region-Based Convolutional Network, and it is a two-stage approach, where the first stage involves proposing regions of interest (ROIs) that could potentially contain objects. The second stage involved classifying these ROIs and refining the object BBs[65]. In other words, this innovative Algorithm at the time had a similar approach to object detection as the human brain does, first looking at the whole image that the eyes are perceiving and then starting to look for the parts that are most relevant[56]. This approach was innovative at the time and remained a benchmark for object detection algorithms until today.

As Girshick et al. describes in the original paper, this is composed of three main modules:

1. Selective Search Algorithm, this module was charged with generating candidate regions that could contain an object of interest.

2. Feature Extraction Module. To create a fixed-sized feature map, this module takes the region proposals produced by the preceding module and crops them from the input image. A fixed-length feature vector representing the region suggestion is then extracted from the feature map using a Convolutional Neural Network (CNN).

3. Classification, Localization, and BB Regression Model. This module takes the feature vector obtained from the previous module. It uses it to classify the region proposal into one of the pre-defined object classes and predict the BB coordinates that enclose the object in the region proposal. These two tasks are performed by a fully connected layer followed by a softmax activation function for classification and a linear layer for regression.

In order to comprehend the workings of this model, it is crucial to have a deeper understanding of the critical terms mentioned earlier. At first, the model's operation may appear complex, but delving into the key components makes it easier to grasp.

The Selective Search module adopted a unique approach to proposing different regions, as detailed in Figure 3.4.

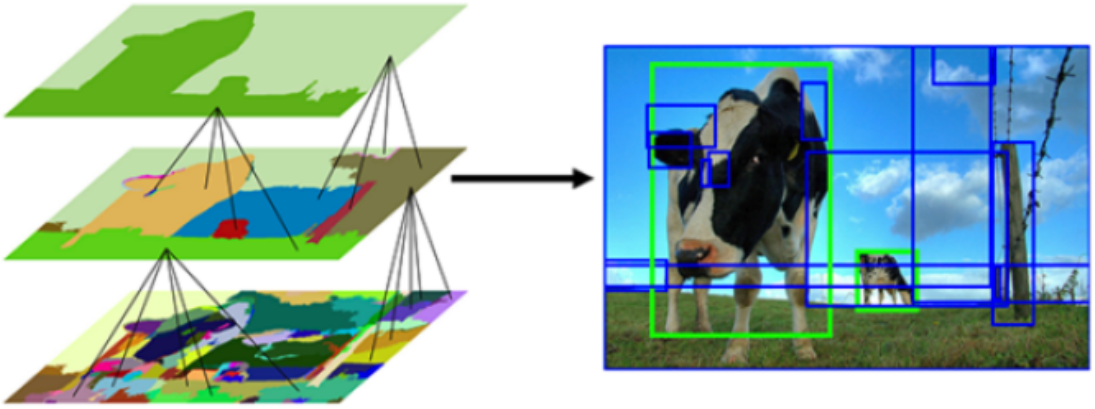


Figure 3.4: Selective Search Algorithm, image obtained from [77]

During the process of Feature Extraction, a 227x227 RGB image is input into a CNN, which consists of 5 CLs and 2 fully connected layers.[76].

During the Classification, Localization and BB Regression Model, different region proposals have a score given to them. Here, the authors of the original paper label all the proposed BBs for training as positive regions when the BBs have an IoU over 0.5 regarding the ground truth and negative(background regions) when the IoU is below 0.3. The proposals that fall into the value of IoU between 0.3 and 0.5 are ignored[78, 79].

To optimize the CNN in RCNN, the Stochastic Gradient Descent (SGD) Optimizer is utilized to iteratively update the model parameters by minimizing the loss function.

Over time, the original version of RCNN encountered several significant issues. Firstly, the Fully Connected Layers required the input image to be 227x227 pixels because of the region proposal around the object of interest. However, this demand posed problems as each region had to be resized, leading to high computational costs[56].

Furthermore, another issue that surfaced was the slow training that the Algorithm experienced. Predictions for an image could take up to 40 to 50 seconds. This occurred because each region of interest was trained and processed independently, and the system was only fine-tuned in the end. In addition, since about 2000 regions of interest were generated on each new image, many CNN features had to be processed, as shown in equation 3.8.

$$N\text{-CNN} = N \times 2000 \tag{3.8}$$

Where N refers to the number of images and N-CNN instead is the Total number of CNN features that had to be processed[56, 76, 80].

However, the issue with the original RCNN was the time it took to generate a prediction and the need to store the extracted features from each region proposal in separate files on the disk. This requirement led to a high demand for storage space, making the Algorithm time-consuming and storage-intensive[56, 78].

3.2.1 Evolution of RCNN Family

The first version of the RCNN algorithm had several significant drawbacks, prompting a push for new and improved versions. This effort resulted in the creation of evolutionary algorithms in the RCNN family, including Fast-RCNN and later Faster-RCNN, developed by the author of the original RCNN.

With the introduction of these new algorithms, significant improvements were made to the RCNN family. One such improvement was the reduced computational complexity, leading to faster processing times and making these algorithms more practical for real-time applications. Additionally, the new algorithms were designed to eliminate the need for storing intermediate results on the disk, thus reducing storage requirements. These advancements made the new versions of the RCNN family of algorithms more efficient and more accessible for a broader range of applications, which led to widespread adoption in CV.

3.2.1.1 Fast-RCNN

In 2015, Ross Girshick, the author of the original RCNN Algorithm, published an upgraded version called Fast-RCNN. This newer model brought about significant changes, including:

- Number of regions passed to the Convolutional Network changed from 2000 to 1 per image[80].
- Implementing a multi-task loss function in Fast R-CNN, allowing all layers in the network to be updated during training without the need for disk storage to cache features[65].
- The entire image and object proposals are inputted into the Fast-RCNN at once[65].
- The Fast-RCNN introduced the RoI pooling layer, which helped with simultaneous feature extraction and classification[53].

According to Girshick, the upgrades to the original Algorithm in Faster-RCNN resulted in a speed improvement of approximately nine times over its predecessor[80].

By reducing the number of regions passed to the convolutional neural network (CNN) and inputting the image and object proposals as one, Fast R-CNN achieves faster processing times than its predecessor, mainly because of the convolutional operation is only performed once instead of many times per image. This leads to only one feature map being generated from it. This way, the testing and training times are reduced significantly.

Introducing the RoI Pooling Layer in Fast R-CNN allowed the CNN to receive a feature map containing all Regions of Interest (RoIs) simultaneously, rather than requiring each RoI to be wrapped in a separate feature map to be fed to individual RoI-specific CNNs. This change resulted in further speed improvements and increased accuracy of the algorithm[78].

Although Fast R-CNN was faster than its predecessor, RCNN, it still used the region proposal method and was not fast enough for large Datasets, as noted in [80]. The need for a faster version led to the creation of Faster R-CNN, which addressed this issue and improved compared to the previous Algorithm.

3.2.1.2 Faster-RCNN

Faster R-CNN was also introduced in the same year as the publication of Fast R-CNN. This version of the R-CNN family solved the bottleneck issue that existed in Fast R-CNN.

An Region Proposal Network (RPN) was introduced with this version resolving once and for all the problem of a dull and obtuse process that the selective search algorithm proved to have[80, 81].

As the authors describe, The RPN in Faster-RCNN is a fully convolutional network (FCN) type and can be trained end-to-end to generate detection proposals[81].

In Faster-RCNN, a new neural network is incorporated after the backbone, typically ResNet or VGG [81]. Here, the backbone functions as a feature extractor and efficiently extracts visual features like color gradients, edges, and texture.

The earlier versions of RCNN employed a Selective Search algorithm to generate proposals for candidate regions that may contain an object of interest. However, due to its slow speed, this part of the Algorithm was replaced with RPN in Faster-RCNN version[80].

To better understand how RPN exactly works, we have to look at the following enumerated steps:

1. RPN gets as input the feature maps outputted by the Backbone[80, 81].

2. A sliding window technique is utilized on the feature map generated by the backbone to create a set of K anchor boxes with various shapes and sizes at each location, as described in [79–81]. The typical configuration consists of 9 anchor boxes produced by combining 3 different scales with 3 different aspect ratios[81].

3. An Anchor Refinement Step is then used to adjust the location and size of the anchor boxes, improving their fit to the object of interest. This step is crucial for accurately localizing and identifying objects in the image[79–81].

4. An Object Score is generated to indicate the likelihood that an object is present within each proposed box. This score is a probability value between 0 and 1, and is used to rank the anchor boxes according to their level of confidence in containing an object[80, 81].

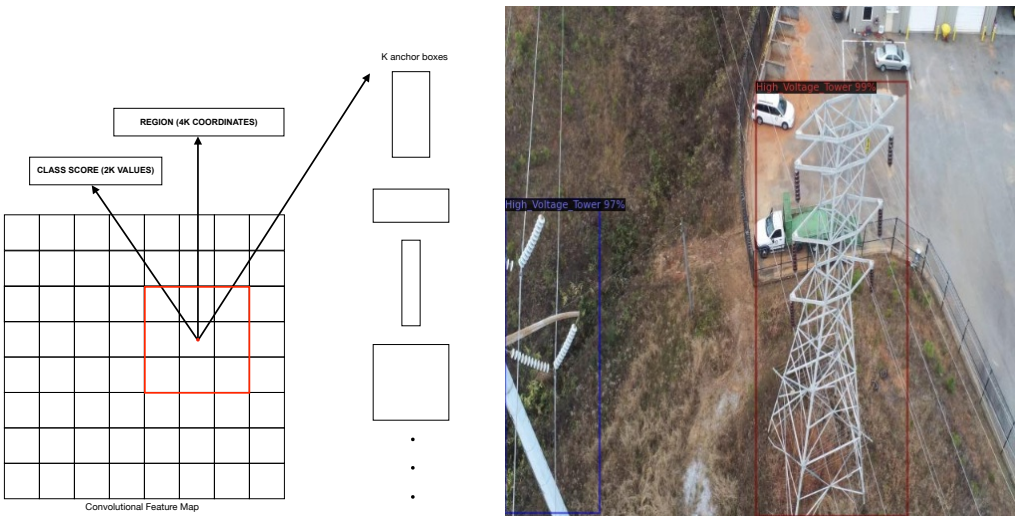
In the Anchor Refinement Step of RPN, it is crucial to note that the shape and aspect ratio of the BBs has been predetermined and fixed beforehand. Therefore, the purpose of the refinement step is to optimize the position and size of the anchor boxes so that they can more accurately fit the object of interest.

We can understand from this that the RPN produces two outputs, the Object Score and the Anchor Box Refinement Coordinates. These outputs are:

- 2n. n anchor boxes and two values per each. These two values are the probability that an object is inside that anchor box and that an object is not inside that anchor box[79, 81].
- 4n. n anchor boxes and four values that refer to the BB coordinates already regressed being those the coordinates for x_{center}, y_{center},height,width[79, 81].

Interestingly, as Ren et al. describes, the anchors on this version of the RCNN family algorithms are translation invariant, meaning that the Algorithm ensures that it is capable of detecting objects at different scales and orientations and is not affected by minor variations in object position within the image.

In Figure 3.5, we have an image describing better how this RPN step of Faster-RCNN works.



(a) Region Proposal Network (b) Detection Using Faster-RCNN

Figure 3.5: RPN and further Faster-RCNN Detection

By incorporating RPN and sharing convolutional features, Faster-RCNN improved significantly over the original RCNN and Fast-RCNN. This resulted in a 250x speedup compared to the original RCNN and a 10x speedup compared to Fast-RCNN using the PASCAL VOC 2007 Dataset during training time, making it a significant breakthrough in the field of object detection [65].

Such a thing was made possible by combining all the separate steps described in the original RCNN paper into a single neural network[53]. As the author of the original paper, Ren et al., points out, "The entire system is a single, unified network for object detection". Integrating the separate steps into a single network resulted in significant speed improvements over the previous RCNN and Fast-RCNN models[65].

The proposed approach for Faster R-CNN involves a RPN and Fast R-CNN object detection network. This leads to quick network convergence, resulting in a single architecture that shares convolutional features for both tasks[81].

However, only some things can be perfect, and this variant of the RCNN family was no exception. This version still faced some challenges that could be further refined, including:

- Computational Expensive. Firstly, it uses a Deep Neural Network backbone like VGG, ResNet, and others, requiring significant computational resources. Secondly, it involves a two-stage training process, adding to its computational complexity. Consequently, the Algorithm's runtime is limited due to the substantial computational requirements[82].

- Poor performance on Small Scale Objects. This happens primarily due to the evaluation metric used to measure detection quality based on IoU (IoU). This metric can cause detections slightly offset from the actual object to have a significantly lower IoU score, potentially falling below the 0.5 thresholds required to consider an object as present in the image. When the object is larger and occupies more pixels in the image, it is easier for a slightly offset detection to have a high enough IoU to be considered a correctly detected object within the BB[79].

- Impact on performance due to Anchor Boxes. Using anchor boxes in the RPN can slow down the refining of the anchor boxes to accurately fit the object of interest. This is because the RPN needs to optimize the location of the anchor boxes during training.

- Sliding Window Redundancy. Using a sliding window technique in RPN causes objects that appear multiple times in an image to be processed redundantly. Each region of the image is processed independently, resulting in duplicate processing of the same object[80].

- Anchor Boxes Suppression. When two anchor boxes with an object inside overlap, the Non-Maximum Suppression (NMS) algorithm may suppress the anchor box with the lower object probability, even if it contains an object, resulting in the loss of potential object detections.

Detecting small objects accurately can be challenging for several reasons, and Faster-RCNN is not immune to this issue. One reason not previously described is that the feature maps fed to the RPN undergo several max-pooling layers, reducing their spatial dimensions. This downsampling poses a challenge for detecting smaller objects. Furthermore, it results in a loss of spatial information and can make the network less precise at localizing such objects in the image[79].

3.2.2 Applications and Further Creations

This algorithm family has been applied to many areas over the past years. These applications were mainly:

1. Object Detection. This Algorithm can be embedded inside the hardware of a Drone leading to Object Detection for many reasons.
2. Face recognition. This Algorithm can be employed to recognize faces in a picture or a video.
3. Text Detection and Recognition. This Algorithm can detect and recognize text on videos and images.
4. Image Segmentation. It can divide an image into different regions or segments based on similarities in color or texture.
5. Autonomous Driving. This algorithm family can detect and classify objects on the road, such as cars, pedestrians, and traffic signs, which is helpful for autonomous driving.
6. Medical Analysis. This Algorithm can be used for medical image analysis, such as detecting and classifying tumors in MRI or CT scans.

Several other algorithms have been developed based on or utilizing the algorithms described in this family. These algorithms include:

1. Google Lens[65]
2. Rosetta[65]
3. FPN [53, 56]
4. Mask-RCNN [53, 56]
5. R-FCN [53, 56]
6. Faster-RCNN+++ [53]
7. G-RMI [53]
8. SPP-net [56]

In the context of object detection, which is the primary focus of my thesis, this kind of Algorithm is intended to be integrated into the hardware of a UAV (Unmanned Aerial Vehicle), allowing the UAV to perform real-time predictions and respond accordingly. The Algorithm's feedback will guide the UAV's movements and enable it to perform its intended functions[65].

3.3 Metrics

Metrics end up being just like the dashboard gauges in a car. In the same way that a driver needs to monitor the speedometer, fuel gauge, and other gauges to ensure safe and efficient driving, a machine learning engineer needs to monitor metrics such as accuracy, loss, and learning rate to ensure the Deep Learning Algorithm is training correctly and producing accurate results. Just as a driver can adjust their speed or route based on the information provided by the gauges, the engineer can adjust Hyperparameters or modify the model architecture based on the metrics to improve the performance of the deep learning algorithm.

3.3.1 Intersection over Union

When comprehending the metrics for evaluating an algorithm, we typically start with the IoU. However, this seemingly "basic" metric has additional critical evaluation methods. These methods relate to the detection process and include:

- True Positives. The algorithm correctly identified the object's BB[83].
- True Negatives. The algorithm correctly ignored an object that is not of interest by considering it as background. Still, this method is often ignored since the algorithm needs to be designed to identify background objects.
- False Positives. The Algorithm detects an object that does not actually exist, or it detected an object but incorrectly localized it in the image[83].
- False Negatives. The algorithm wrongly ignored an object as if it was the background when it was not[83].

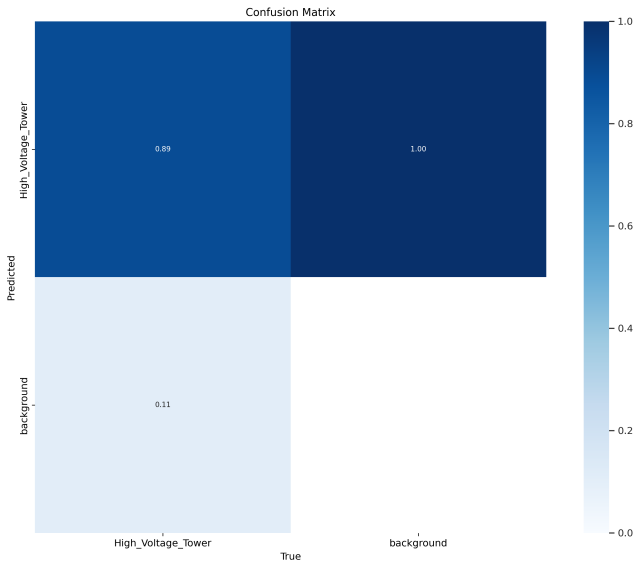


Figure 3.6: Confusion Matrix of a trained model

Usually, these metrics explained before are represented on a confusion matrix, like the one shown in Figure 3.6.

However, they are not enough to tell us how well the algorithm identified the values because they end up being binary, which means that they are either right or wrong, leaving no space for when it was close to identifying the object and how close it was.

Hence, IoU is crucial in evaluating object detection algorithms as it quantifies the degree of overlap between predicted and ground truth BBs[83].

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} = \frac{B_{pred} \cap B_{gt}}{B_{pred} \cup B_{gt}}$$

In the equation above, B_{pred} represents the predicted BB, and B_{gt} represents the BB of the ground truth. With the diagram presented in Figure 3.7 it is possible to understand this concept better: the green area represents the union of the two BBs, and the red area represents their Intersection.

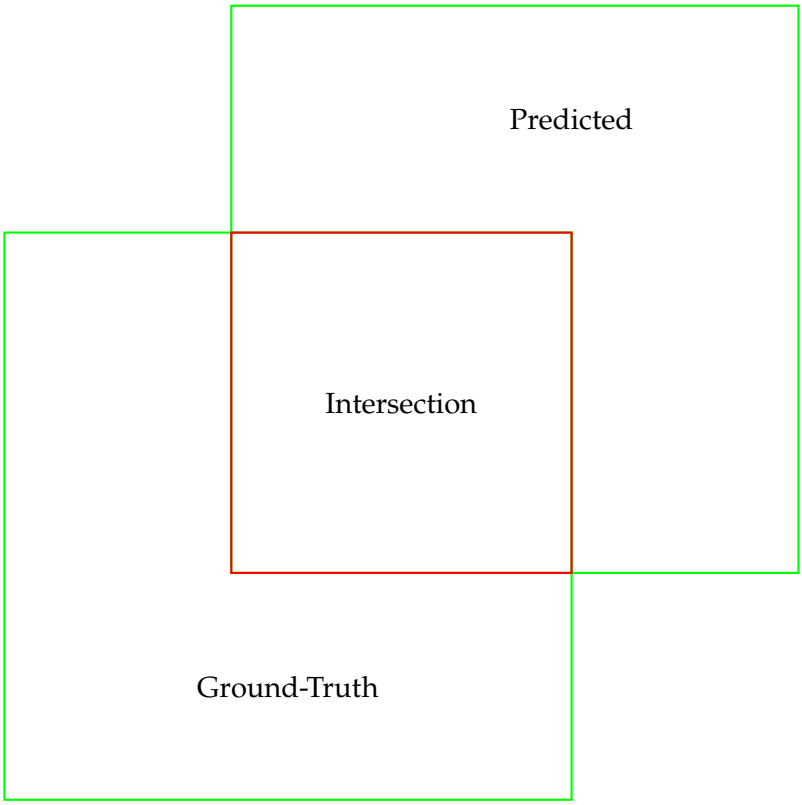


Figure 3.7: Intersection and Union of Ground-Truth and Predicted BBs

The IoU metric, sometimes also noted as the Confidence value, is helpful for thresholding as it ranges between 0 and 1. By setting a threshold, we can determine whether a Predicted BB is considered a Positive or Wrong detection based on whether the IoU between the Predicted and Ground-Truth BBs exceeds the threshold. Below are the equations

for each scenario.

$$\begin{cases} \text{Positive Detection}, & \text{if } IoU(gt, pd) \geq \alpha \\ \text{Negative Detection}, & \text{if } IoU(gt, pd) < \alpha \end{cases} \tag{3.9}$$

When the IoU equals 1, it indicates a perfect match between the Predicted and Ground Truth BBs with complete overlap. On the other hand, an IoU value of 0 means that both BBs do not intersect or touch each other at all.[83]

Once we comprehend these concepts, we can grasp that while training the YOLO or RCNN algorithms, we have a Hyperparameter known as IoU, which can be fine-tuned by specifying the minimum amount of overlap between the predicted and ground-truth BBs to be considered a positive or negative prediction. This allows the deep learning algorithm to train itself accordingly[83].

After completing the training process and testing the algorithm using new images, we need to specify a threshold value α to determine whether a predicted BB should be considered a valid detection. In other words, we need to set a minimum value of IoU between the predicted and ground-truth BBs required for a prediction to be considered accurate. We will explore this concept in greater detail later on.

The IoU threshold is usually set to a value between 0 and 1. However, choosing a higher threshold can reduce false positives but may also miss some true positives. On the other hand, lowering the threshold can increase false positives but may also detect more true positives[83].

3.3.2 Precision and Recall

Precision is a performance metric commonly used in deep learning algorithms to assess the model's ability to correctly identify and classify relevant objects while minimizing the number of false positives.[83–85] As we can see in Equation 3.10, Precision measures the proportion of accurate positive predictions over the total number of positive predictions made by the model, resulting in a value between 0 and 1[83, 84].

Recall is another commonly used evaluation metric in deep learning algorithms and its values are in the same range as Precision. The metric is quantified by the ability of the model to identify all relevant cases, as shown in Equation 3.11[83–85].

As we can see below, Recall and Precision are given by the following equations:

$$Precision = \frac{TP}{\text{All Predictions}} = \frac{TP}{TP + FP} \tag{3.10}$$

$$Recall = \frac{TP}{\text{All Ground Truths}} = \frac{TP}{TP + FN} \tag{3.11}$$

While Precision and Recall are valuable metrics for evaluating the performance of Deep Learning Object Detection Algorithms, it is crucial to exercise caution because relying on either metric alone can be misleading.

Precision is a crucial metric in deep learning applications where correctly identifying positive examples is critical. A high Precision score signifies that the model correctly identifies true positive examples and minimizes false positive predictions. However, relying solely on Precision may cause the model to reject some true positive examples as false negatives, resulting in potential information loss.

Similarly, Recall can also lead to incorrect conclusions because it does not reflect the model's ability to detect False Positives.

For this reason, it is essential to view Precision and Recall as complementary metrics that the human evaluator must analyze together to accurately assess the quality of Deep Learning algorithms.

From this, we can infer that the ultimate goal is to achieve a Precision and Recall score of 1, indicating that the algorithm has made no False Positive or False Negative Detections. Unfortunately, although achieving a perfect model is the ideal goal, it is often not feasible.

3.3.2.1 Precision-Recall Curve

On CV algorithms such as those studied in this document, we can use the IoU metric described earlier to vary the confidence score and examine how it affects both the Precision and Recall metrics. By adjusting the threshold, we can achieve different Precision and Recall values. However, there is often a trade-off between Precision and Recall - increasing one metric may cause a decrease in the other.

Engineers often use the Precision-Recall curve to determine the optimal threshold value for balancing Precision and Recall. This graphical representation displays the relationship between Precision and Recall across various confidence thresholds. The curve is generated by plotting the Precision-Recall pairs at different confidence score thresholds.

By examining the curve, engineers can identify the threshold that best balances Precision and Recall. This threshold is often chosen based on the specific application and its associated costs and benefits. For Example, a high Precision threshold may be more appropriate for applications where false positives are exceptionally costly. In contrast, a high Recall threshold may be more important for applications with harmful false negatives.

The optimal scenario for this metric is typically represented by a curve that closely approximates the ideal one, where the Precision and Recall are at their maximum values for all confidence or IoU thresholds.

In Figure 3.8, we have an example of the Precision-Recall graph where the red curve shows the ideal scenario, while the blue line represents a more realistic curve.

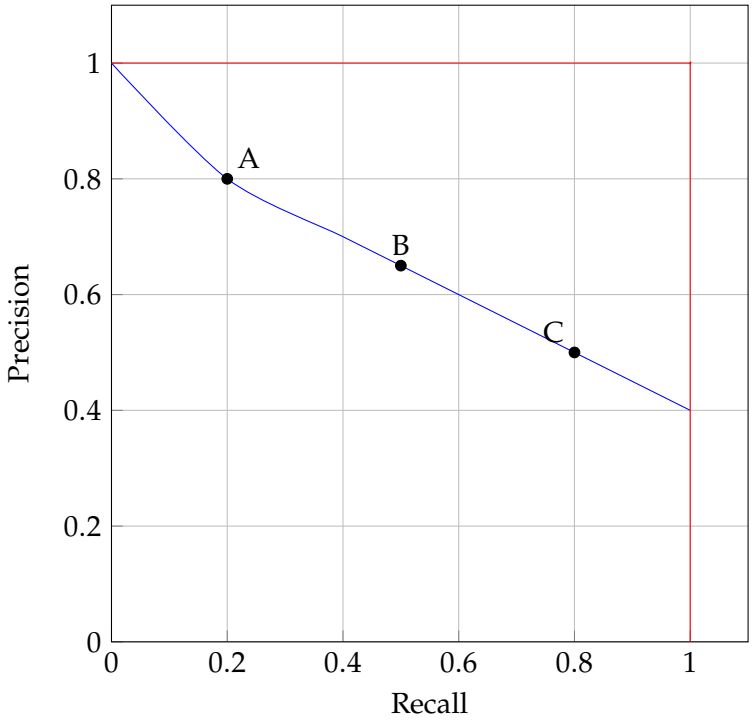


Figure 3.8: Precision-Recall Curve

When interpreting the Precision-Recall graph, it is crucial to remember that the curve should usually be downward-sloping. This is because as the model attempts to identify more positive examples, the likelihood of false positives increases, leading to a decrease in Precision. Therefore, Precision typically decreases as Recall increases, resulting in a curve that slopes downwards.

If the Precision-Recall curve is not downward-sloping, it may indicate a problem with the model or the evaluation process. Conversely, the upward-sloping Precision-Recall curve may indicate that the model is not making good predictions or that the evaluation process is not sensitive enough to detect errors. Generally, a downward-sloping Precision-Recall curve is expected for most classification tasks, as there is usually a trade-off between Precision and Recall.

3.3.3 F1 Curve

The $F1$ score that we can see in equation 3.9 is a value between 0 and 1 that measures the trade-off between Precision and Recall by recurring to their harmonic mean.[83–85]

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.12}$$

This metric is typically most useful in scenarios where both Precision and Recall are equally important, and the goal is to optimize both simultaneously. However, other evaluation metrics may be more appropriate in situations where one of these metrics is more important than the other.

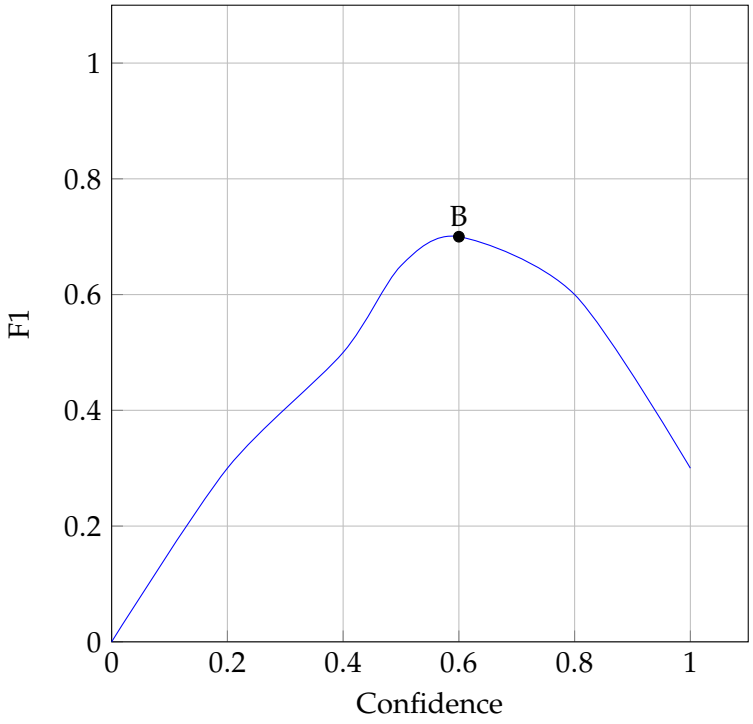


Figure 3.9: F1 Curve

We can observe an example of these types of curves in Figure 3.9. As depicted, the optimal Confidence value that yields the highest combined Precision and Recall values is 0.6.

3.3.4 Average Precision

Average Precision (AP), like the F1 curve, has much to do with the Precision-Recall curve because AP is a metric that measures the quality of a model's predicted ranking of items by calculating the area under the Precision-Recall curve[83].

To calculate AP, we first calculate Precision and Recall values at different confidence thresholds for the model's predictions, just like we can see in Equation 3.13.

$$AP = \int_{r=0}^{1} p(r)dr = \sum_{k=1}^{n} Precision(k) \cdot (Recall(k) - Recall(k+1)) \tag{3.13}$$

In this formula, $p(r)$ represents the Precision-Recall curve, n represents the total number of thresholds, $Precision(k)$ represents the Precision at k and $(Recall(k) - Recall(k+1))$ represents the change in Recall from k to $k+1$.

At the n^{th} value, the Recall shall be equal to 0 so that the calculation can be made for all the values of Recall and Precision.

In essence, AP summarizes how well a model can identify relevant items while avoiding irrelevant ones across different confidence levels in the model's predictions. It is commonly

used to evaluate object detection models, where the goal is to identify the presence and location of objects in an image.

Usually, when models are already trained and if the engineer has evaluation metrics such as AP for each model, the model that ideally he should use is the one that gets the highest value of AP, regardless of how long it took to converge since the model has already converged.

However, suppose the engineer has multiple models with similar APs. In that case, the idea is to consider other factors, such as computational resources required for inference, inference speed, model complexity, and ease of deployment, to decide which model to use.

The decision on which model to use is subjective and ultimately up to the engineer's discretion.

3.3.5 Mean Average Precision

The Mean Average Precision (mAP), as we can see in Equation 3.14, is widely connected to the AP. This metric consists of the mean of the AP scores computed for each class in the Dataset.[83, 84]

$$mAP = \frac{1}{k} \sum_{i}^{k} AP_i \tag{3.14}$$

Here we can infer that :

- k is the number of classes in the Dataset. This is the total number of classes that the algorithm is trying to predict.
- i refers to the index of the class for which AP is being calculated. This goes from 1 to k.
- AP_i is the AP for class i.

The mAP is commonly used as a performance metric since it provides a global evaluation of the Algorithm's performance across all classes in the Dataset. It is the Average of the AP scores computed for each class. This makes it a comprehensive measure of the model's ability to detect objects of interest in the Dataset.

Similar to the AP, the mAP also does not provide a definitive answer as to which model is the best to use, as this decision depends on various factors.

Choosing the best model to use is similar to selecting a mode of transportation for a trip. Just as there are different modes of transportation (car, train, bus, etc.) with different strengths and weaknesses, there are different models (Neural Networks) with different capabilities and trade-offs.

If we look at the previous example, deciding the mode of transportation to use depends on various factors such as distance, time, cost, comfort, and convenience. Similarly,

choosing the best model to use depends on various factors, such as the Dataset, the task, the available computational resources, the deployment environment, and the desired performance metrics.

In both cases, the decision ultimately lies once again with the user, who needs to balance the different factors and make a subjective judgment based on their expertise, experience, and preferences.

3.3.5.1 mAP@0.5

mAP (mAP) at 0.5 value of IoU, also known as Mean Average Precision with 0.5 Intersection Over Union (mAP@0.5), is a commonly used performance metric in object detection tasks. For Example, this metric is widely used in CV to report results on algorithms that are trained and tested on benchmarked Datasets, such as PASCAL VOC 2007 (with Faster-RCNN) or COCO (with YOLOv5)[83]. It measures the AP of the model with a 0.5 Detection Threshold, which means that a predicted BB is considered correct if it has an IoU of 0.5 or higher.

To compute mAP@0.5, the AP (AP) is first calculated for each class in the Dataset with a 0.5 IoU Threshold. Then, the mean of these AP values across all classes is taken to obtain the mAP@0.5. The mAP@0.5 gives a measure of the overall object detection performance of the model.

Typically, this metric is displayed on a graph where the X axis represents the Epoch, and the Y axis represents the mAP@0.5, as illustrated in Figure 3.10.

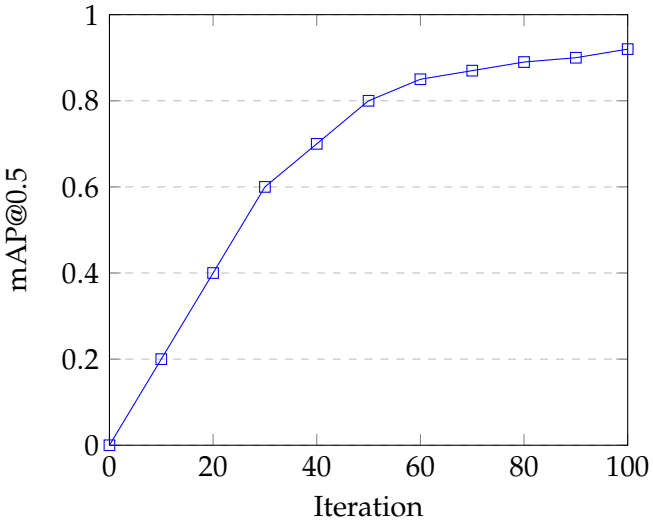


Figure 3.10: mAP at 0.5 IoU

3.3.5.2 mAP@0.5:0.95

The mAP@0.5:0.95 is a metric that evaluates the performance of a model across different thresholds ranging from 0.5 to 0.95 with a step of 0.05. Unlike mAP@0.5, which measures

performance only at a single IoU threshold of 0.5, mAP@0.5:0.95 considers ten different thresholds. The final mAP score is the mean of the AP values across all classes.

This metric is widely used in CV algorithms to report results of algorithms that are trained and tested on benchmarked Datasets, such as COCO (with YOLOv5).[83]

This metric provides a more comprehensive evaluation of the model's performance compared to the mAP@0.5 because it considers the accuracy of the predicted BBs at different levels of overlap with the ground truth. A higher mAP@0.5:0.95 score indicates better object detection performance of the model.

3.3.5.3 Frames Per Second

When we want to know how fast is a model on processing each image, we use a metric named FPS[84]. This metric shows how fast the Algorithm predicts BBs on an image. Herefore, this metric is given by the following equation:

$$FPS = \frac{Number\ of\ Frames}{Processing\ Time} = \frac{1}{T} \tag{3.15}$$

In the above equation, the value given by T refers to the Processing time of one single Frame.

This way, it is evident that when implementing Algorithms such as YOLOv5 and Faster-RCNN in real-time, the FPS metric will show how well the Algorithm will be capable of Identifying an Object of Interest in a short time, showing, therefore, how practical the Algorithm will be for a Real-life implementation.

However, it is important to remember that the FPS metric is also influenced by the hardware configuration used to run the model, meaning that different hardware setups may result in different FPS values. Therefore, it is crucial to thoroughly consider the hardware used when evaluating the FPS of Deep Learning CV Algorithms.

3.3.6 Loss Function

As we know, the purpose of the loss function is to train the neural network to make accurate predictions by minimizing the difference between the predicted output and the actual output. This function measures the error between the predicted output and the ground truth, which is then used to adjust the neural network weights during the backpropagation process. The ultimate goal is to minimize the loss function over the training data so that the neural network can make accurate predictions on unseen data.

3.3.6.1 YOLOv5

On this version of YOLO, the loss function combines three main parts:

- Object Loss: Confidence of the presence of an object on a BB.
- BB Regression Loss: Regression of predicted box coordinates.

- Class Loss: Class probability prediction error.

Objectness Loss is used to predicting an object's presence or absence in each grid cell. Next, the BB Regression Loss is used to refine the predicted BB coordinates. Finally, the Class Loss is used to calculate the loss associated with classifying an object into one of the predefined categories based on the predicted class probabilities. These three loss components are summed to compute the final loss.

3.3.6.2 Faster-RCNN

Faster R-CNN uses a multi-task loss function to jointly train the RPN and the Fast R-CNN network. The loss function has two main components: the RPN loss and the Fast R-CNN loss.

The RPN loss is a binary cross-entropy loss that predicts if a proposed region is an object or background. The RPN also has a regression loss term that predicts the offset between the proposed box and the ground-truth box.

The Fast R-CNN loss is a multi-class cross-entropy loss that is applied to the RoI (region of interest) classification task, which predicts the object class label of each RoI. The Fast R-CNN also has a BB regression loss term, which predicts the offset between the RoI and the ground-truth box.

The purpose of the loss function is to optimize the weights of the neural network to minimize the overall loss between the predicted and ground-truth outputs, thereby improving the accuracy of the object detection system.

3.4 Hyperparameters

Just as a sound system has various knobs and dials that can be adjusted to optimize the sound quality for a specific room, audience, or music genre, Hyperparameters in deep learning algorithms are adjustable parameters that can be tuned to optimize the performance of the model for a specific task, Dataset, or hardware architecture.

Hyperparameters are parameters that are not learned from the data but are set by the user before the training of the deep learning model. These parameters control the learning process and the behavior of the model.

Selecting the appropriate Hyperparameters is essential to achieve good performance and generalization of the model.

Typically, we evaluate a model's performance based on its metrics before making any changes to the model. However, numerous parameters can be modified to enhance a model's efficiency and improve performance.

One of the initial issues an engineer should address is the consistency of BB assignments across the model. Specifically, whether the BB rules applied to one image are consistently applied to all other images.

After this rule has been discarded, we can fine-tune the model by changing its Hyperparameters.

3.4.1 Optimizers

An Optimizer is a crucial part of the training process of deep learning models, and its purpose is to minimize the loss function during training by updating the model's parameters. The Optimizer achieves this by calculating the gradients of the loss function concerning the model's parameters and then using these gradients to update the parameters in the direction that reduces the loss. On Deep Learning Algorithms, we typically have three popular Optimizers:

- SGD. This is a classic optimization Algorithm that updates the parameters based on the gradient of the loss function concerning the parameters. This Optimizer is the default Optimizer for YOLOv5, and it works by updating the weights of the neural network in small steps based on the gradient of the loss function concerning the weights. This is done iteratively for a fixed number of epochs or until the convergence criteria are met. The learning rate is usually scheduled to decrease over time to improve convergence. Other Hyperparameters, such as momentum and weight decay, can be applied to improve optimization further.

- Adam. This adaptive learning rate optimization algorithm combines ideas from momentum-based and adaptive learning rate methods.

- AdamW. This optimization algorithm is similar to Adam's but with the added benefit of weight decay. Weight decay is a method to prevent Overfitting by penalizing the model for having large weights. AdamW combines the adaptive learning rate of Adam with weight decay to achieve better performance on large-scale training tasks, such as object detection in YOLOv5. It works by adjusting the learning rate for each weight in the model based on the weight updates' gradients and history while adding a term to the weight updates that discourage large weights.

According to the Creator of YOLOv5, he says: "Adam can work well on smaller custom Datasets and can provide good initial results on larger Datasets, whereas SGD tends to outperform in the long run, especially on larger Datasets, and seems to generalize better to real-world results".

3.4.2 Initial Learning Rate

The Initial Learning Rate (lr0) in YOLOv5 serves as a set starting point for the learning rate schedule. During training, the learning rate is adjusted based on the Epoch or iteration number to find an optimal rate that allows the model to converge quickly and accurately. The lr0 determines the base value for this adjustment.

The Hyperparameter lr0 typically depends on the choice of Optimizer used on the Dataset. Just like it is described on the hyp.scratch-low.yaml file provided by the YOLOv5 creator, if the SGD Optimizer is selected, it is recommended to set lr0 to 0.01, while for the ADAM Optimizer, the recommended value is 0.001.

3.4.3 Final Learning Rate

Conversely, the Final Learning Rate (lrf) is used to gradually decrease the algorithm's learning rate during the later epochs of training to ensure that the model continues to make manageable updates to the weights at later stages of training. This helps the model to converge to the optimal values for the weights, as decreasing the learning rate allows the model to take smaller steps towards the optimal values and fine-tune the weights. By doing this, the model can continue to make smaller weight adjustments to fine-tune its performance without destabilizing the training process. It also prevents the weights from changing too much once their metrics are stabilized.

In the last Epoch of the algorithm, the learning rate will be given according to equation 3.16.

$$Llr = lrf \times lr0 \tag{3.16}$$

The Last Epochs Learning Rate, denoted as Llr, is typically set to 10% of the total number of training Epochs. The Final Epoch's Learning Rate, denoted as lrf, is the minimum learning rate value set after all adjustments. In contrast, the lr0, denoted as lr0, is the starting learning rate value.

3.4.4 Momentum

Momentum is a Hyperparameter commonly used in optimization algorithms like SGD and ADAM or even AdamW to speed up the convergence of the model during training.

When we look at the Optimizers, we know their objective is to reach the best weight values. However, there is a chance that the algorithm gets to a local minimum and gets stuck there because the learning rate needs to be bigger to make the weights get updated incorrectly.

To better understand this, in Figure 3.11, we have a straightforward example of this problem.

The function represented in Figure 3.11 has two critical points, a local minimum, and a local maximum. If an optimization algorithm begins near the red point, representing the local minimum, it will become trapped there and converge to that point instead of discovering the global minimum. This happens because the function's gradient is zero at the local minimum, and the algorithm cannot identify a direction toward the global minimum.

In order to prevent this, one of the methods that are employed is the usage of momentum. This Hyperparameter, in simple terms, allows the Optimizer to continue moving in its current direction even if the gradient changes direction.

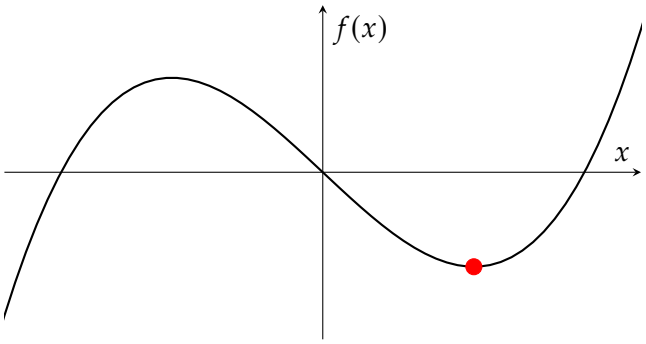


Figure 3.11: Example of a local minimum

Momentum operates like inertia, favoring the continuation of movement along the previous trajectory and making it slightly more difficult to change direction.

This Hyperparameter value is usually between 0.9 and 0.99 on YOLOv5, set to 0.937 by default.

3.4.5 Weights

Typically, when beginning to train a deep neural network, pre-trained weights are utilized to expedite the training process and facilitate the discovery of suitable weights for the given Dataset.

Training a Deep Neural Network from scratch is also an option. However, it may change significantly during the initial epochs, essentially serving as a warm-up period for the weights to converge gradually. Since the weights are initialized randomly, they will initially result in a high loss value, and the model will undergo significant weight updates during the early epochs. However, the weights will gradually stabilize as training progresses, resulting in smaller weight updates and better convergence toward optimal values.

On YOLOv5, we have different-sized weights, and when we choose each one of the possible weights, we are also choosing the architecture in terms of neurons. However, even though the bigger models can prove to have better accuracy, they come with a trade-off of worse speed, leading to training that might take much longer to get finished.

In Figure 3.12, we have the different possible model sizes for YOLOv5.

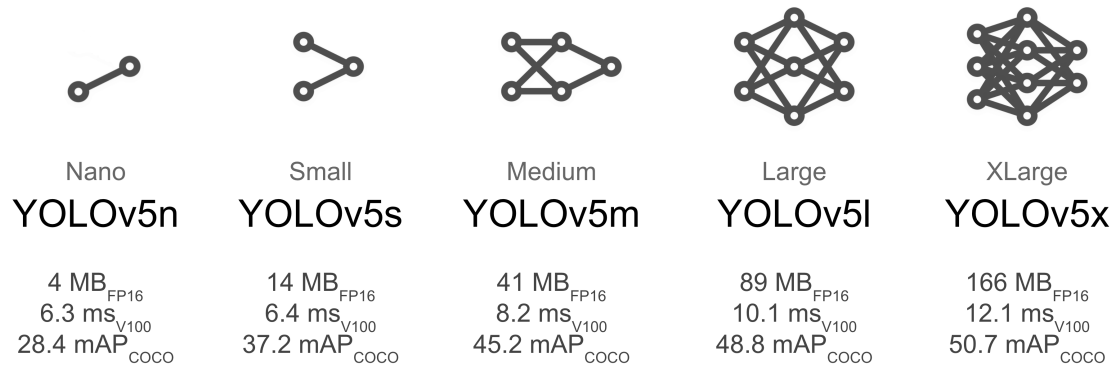


Figure 3.12: Different Model Sizes for YOLOv5, image obtained from [86]

3.4.6 Image Size

The image size is a Hyperparameter that must be specified when training the algorithm. The algorithm needs to know how much the input images will be resized.

The default image resizing values for YOLOv5 are typically set at 640x640 pixels. However, when setting the image size for training, it is vital to be cautious with the chosen dimensions. If the images contain small objects critical for detection, it may be necessary to use larger image sizes to avoid losing information during the resizing process. This ensures that the algorithm can learn patterns from these small objects instead of blurring them out or losing their details during training.

In YOLOv5, it is typically recommended to resize images to dimensions that are multiples of 32, which ensures compatibility between the feature maps produced by the backbone network. However, when selecting the image size, it is essential to consider that larger values require more computational resources and can significantly increase training time.

3.4.7 Batch

Training a Deep Neural Network involves feeding it with batches of input data, and one of the essential Hyperparameters that need to be specified is the batch size. With a batch size, the network would train on one image at a time, updating the weights after each input, which is computationally efficient.

The batch size refers to the number of images that are processed in parallel during each iteration of the training process. The batch size is a crucial Hyperparameter because it can significantly impact the training speed and the final performance of the model.

One of the main benefits of using a larger batch size, or in other words, a batch with a larger amount of pictures, is that it can significantly reduce the amount of time required to train the model. This is because the model is updated less frequently, which reduces the overhead associated with data loading and backpropagation. However, using a larger

batch size also requires more Graphics Processing Unit (GPU) memory, which may only be available on some systems.

On the other hand, using a smaller batch size can lead to slower training times but can also result in more stable training and better generalization performance. This is because smaller batch sizes lead to more frequent updates, which can help the model avoid getting stuck in poor local minima and can also help prevent Overfitting.

In general, the optimal batch size for a given model and Dataset will depend on various factors, including the Dataset's size, the model's complexity, and the available computational resources.

In YOLOv5, setting an appropriate batch size is crucial for efficiently training the deep neural network. The author recommends starting with the maximum batch size possible that the working environment can take to accelerate training, then adjusting it to a smaller value later. This approach speeds up the training process and provides a baseline for comparing the quality of models trained with different batch sizes.

3.4.8 Epochs

The Epoch is a very important Hyperparameter in CV Deep Learning Algorithms because they determine the duration of the training process for Deep Learning models.

A more significant number of epochs can improve the model's performance by allowing it to learn more complex patterns and features from the training data. However, training for too many epochs can lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new, unseen data.

Therefore, it is essential to carefully choose the number of training epochs, balancing the need for improved performance with the risk of Overfitting. This can be done by monitoring the model's performance on a validation set during training and stopping the training process when the validation loss stops decreasing or starts to increase.

As we can see, this Hyperparameter must always be very well-tuned according to the Dataset the model trains himself with.

3.4.8.1 Patience

As previously mentioned, training a model for too many epochs may lead to metrics convergence at a certain point, rendering the remaining epochs unnecessary in improving the model while increasing the computational resources required.

In object detection algorithms such as YOLOv5 and Faster-RCNN, the patience Hyperparameter is crucial in preventing Overfitting and enhancing model accuracy. By observing the validation loss during training, we can determine if the model is still improving or has reached a plateau. If the validation loss does not improve for a specified number of epochs, training can be stopped early to avoid wasting computational resources on further training. Figure 3.13 illustrates this concept.

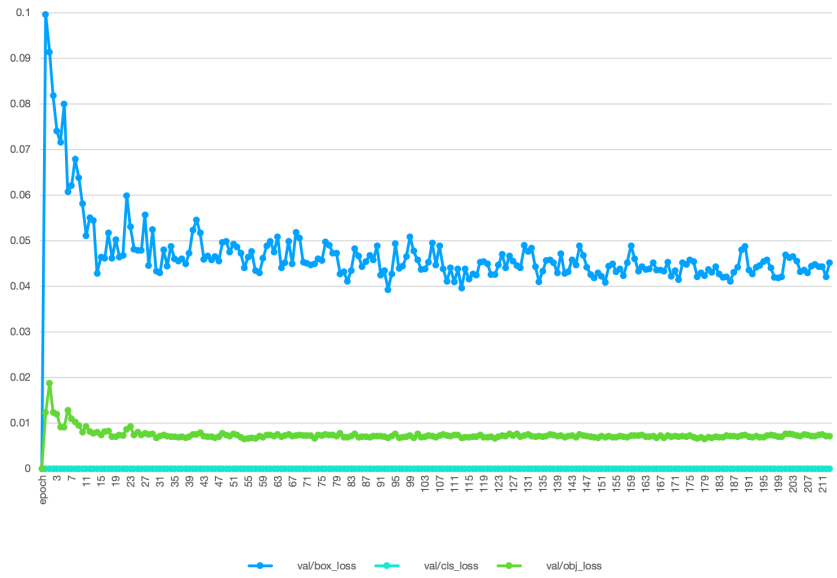


Figure 3.13: An example of a Model using Patience to stop its training

Figure 3.13 shows that the model's validation losses for box, object, and class converged to specific values. Since the model had a patience value of 100 epochs, training was stopped when the validation loss, the sum of the three losses, did not improve for the specified number of epochs. This prevented wasting computational resources and save time for the engineer.

3.4.9 Freeze

The freeze Hyperparameter in models like YOLOv5 and Faster-RCNN determines whether to freeze the weights of the pre-trained layers during training or not. Therefore, when we use a pre-trained model, we can freeze the pre-trained layers during the initial training process. This means that the weights in the pre-trained layers are not updated during training, and only the weights in the new layers are updated.

By freezing the pre-trained layers, we can accelerate the training process and reduce the chance of Overfitting. The pre-trained layers already contain learned features relevant to the task. By freezing them, we ensure the network remembers these features while learning new ones.

Once the new layers have been trained, we can unfreeze the pre-trained layers and fine-tune the entire network to improve its performance on the target task. This allows us to leverage the transfer learning benefits of the pre-trained model while still adapting it to our specific use case.

However, Freezing layers typically do not improve the model's performance. Its primary purpose is to conserve computer resources and accelerate the training process.

3.5 Common Problems on CV Algorithms

When we look at the metrics outputted by an algorithm like Faster-RCNN or YOLOv5, many problems can happen that make these algorithms have their performances not good enough for the engineer. Those can be :

1. Overfitting: This occurs when the model performs well on the training data but fails to generalize to new, unseen data. This can happen when the model is too complex or needs more data to train on.

2. Underfitting: This is the opposite of Overfitting and occurs when the model is too simple to capture the complexity of the data. This can result in poor performance on both the training and test data.

3. Poor quality data: If the data used to train the model is of poor quality, then the model may not be able to learn the correct features and may perform poorly.

4. Imbalanced data: This occurs when the classes in the Dataset are not equally represented, resulting in the model being biased towards the majority class.

5. Model architecture: Choosing the exemplary architecture for the problem is crucial. Using an architecture that is too complex can lead to Overfitting, while using an architecture that is too simple can lead to Underfitting.

6. Hyperparameters: Choosing the correct Hyperparameters, such as learning rate, batch size, and number of epochs, among others, can have a significant impact on the model's performance.

7. Optimization: Choosing the suitable optimization algorithm can also impact the model's performance. Using an Optimizer that is too aggressive can cause the model to converge too quickly, while using an Optimizer that is too slow can cause the model to take too long.

3.5.1 Overfitting

As we saw earlier, Overfitting occurs when a machine learning model is trained too well on the training data to the point where it begins to memorize the data instead of generalizing patterns. In other words, the model becomes too complex and fits the noise in the training data instead of the underlying relationships between the features and the target variable. This leads to poor performance when the model is tested on new data.

Overfitting can occur in any machine learning model, including CV algorithms, and is a common problem that needs to be addressed during model training.

In Figure 3.14, we can see an example of how Overfitting may look when training a CV Deep Learning Algorithm.

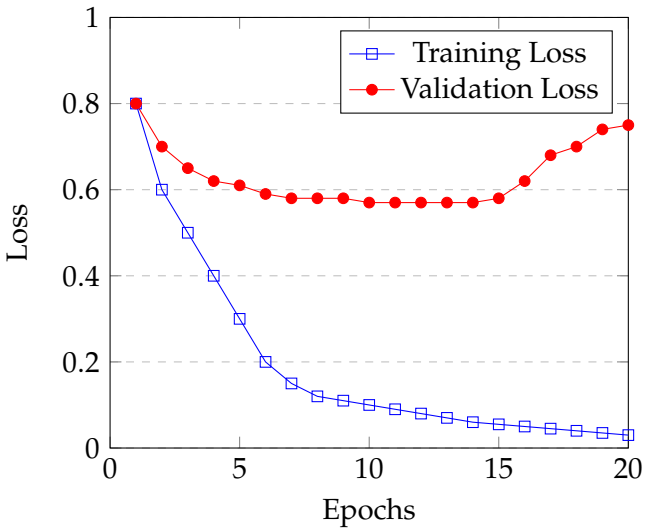


Figure 3.14: Example of a model suffering Overfitting

In order to address this problem, we have some techniques that can be employed. These are Early Stopping, Regularization, or Data Augmentation.

Here, only Regularization will be talked about because Data Augmentation is discussed in Section 4.2.1.5, while Early stopping is implemented through the Patience Hyperparameter, explained in Section 3.4.8.1.

3.5.2 Regularization

This technique involves adding a penalty term to the loss function during training to discourage the model from relying too heavily on certain features or weights that may be specific to the training set but not generalizable to new data.

Regularization can take different forms, such as L1 or L2 Regularization, which respectively add a penalty proportional to the absolute value or square of the weights. By using regularization techniques, the model is encouraged to generalize better to new, unseen data and to avoid Overfitting to the training set.

Another important Regularization form is Dropout. It randomly sets a fraction of the neuron outputs in a given layer to zero during each training epoch. This helps to prevent Overfitting and improve the generalization of the model. All neuron outputs are used during testing, but a fraction of the active neurons scale their values during training[87]. Dropout is implemented by adding Dropout layers to the model architecture, specifying the fraction of neurons to be dropped out during training.

When implementing Dropout, a percentage value between 0 and 1 is used to specify the percentage of all the neurons that will be dropped out per Epoch. This ensures that no neuron receives the proper attention or importance and promotes even generalization across the training and validation Datasets. In addition, by doing so, the problem of overfitting the training data is avoided.

3.5.3 Underfitting

Underfitting is the opposite of Overfitting and happens when the model needs to be more complex to capture the patterns in the data. In other words, the model needs to be more complex and learn the essential features of the data. Figure 3.15 shows how Underfitting can occur. This tends to happen when the model is not trained long enough or when there is insufficient data to train the model.

In the case of CV models like YOLOv5 and Faster-RCNN, Underfitting can occur when the network architecture is not deep enough to capture the complexity of the images or when the model is not trained with enough data. For Example, suppose a YOLOv5 model is trained on a small Dataset of images. In that case, the model may be unable to capture all the variations and complexities of the images in the Dataset, resulting in Underfitting.

To avoid Underfitting, choosing an appropriate network architecture and training the model with sufficient data is essential. In addition, using Data Augmentation techniques such as Rotation, Scaling, and others discussed in Section 4.2.1.5 can also help increase the amount of data and reduce Underfitting.

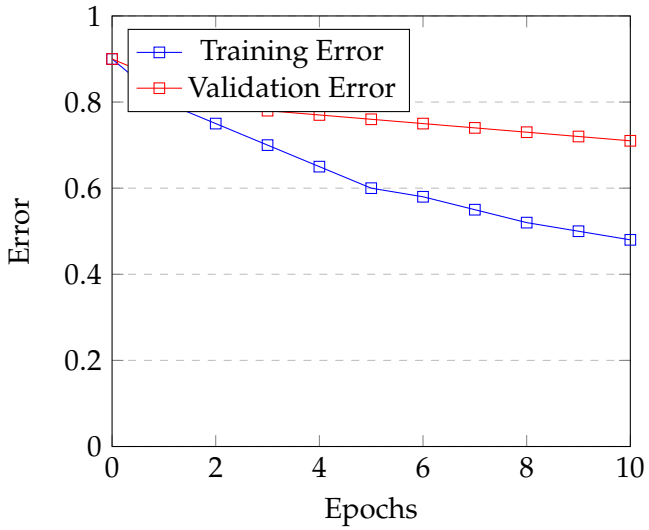


Figure 3.15: Example of a model suffering Underfitting

In Figure 3.15, we can see two lines, the blue line representing the training error and the red line representing the validation error. In an ideal scenario, both lines would decrease as the number of epochs increases, indicating that the model is improving in its ability to generalize to new data. However, this graph shows that both lines are decreasing, but they still have very high error values. This indicates that the model is not generalizing well to new data and is Underfitting.

This happens because the model needs to be more complex to capture the underlying patterns in the data or insufficient data to train the model adequately. Therefore, we need to adjust the model architecture, increase the amount of data, or use Data Augmentation techniques to reduce Underfitting.

4

Results

This chapter will be divided into three main concepts: Dataset Preparation, Tests, and finally, Results and Conclusions.

4.1 Experimental Environment

Both offline and online Applications and Units were utilized to complete this work. This was necessary because GPUs were required to train deep neural networks such as YOLO and RCNN due to their ability to perform numerous calculations simultaneously, unlike Central Processing Unit (CPU)s which perform calculations sequentially.

Initially, the training of YOLOv5s on Jupiter Notebook with CPU was tried, but the training processor took about 6 hours to do 30 epochs.

Afterward, Google Colab was used, which provides up to 12 hours of free GPU usage. This minor adjustment enabled the completion of the same task, which had previously taken a CPU-based approach, in an impressive 6-minute time. As a result, there was a shift from Jupiter Notebook to Google Colab, which provided superior features and faster training speed.

In the rest of this section, the features utilized were categorized into two groups: online and offline.

4.1.1 Offline

For this work, a Macbook Pro late 2016 with a dual-core 2GHz Intel Core i5 Processor, 8 GB 1867 MHz LPDDR3 RAM Memory, and an Intel Iris Graphics 540 1536 MB Graphic Card was used. Everything is described in the following table:

Computer Settings	
Unit	Values
CPU	Dual Core 2GHz Intel Core i5
RAM	8 GB 1867 MHz LPDDR3
Graphic Card	Intel Iris Graphics 540 1536 MB

Table 4.1: Offline Specifications

4.1.2 Online

In order to optimize the working speed, the Roboflow platform was used, and Google Colab was used to perform tasks that could be faster and better done than offline.

Platform	Values
GPU	40 Core 1.59GHz 16GB Tesla T4
Tools	matplotlib, Detectron2, Roboflow, ClearML, OpenCV, Tensorboard, GoogleColab, GoogleDrive
Framework	PyTorch

Table 4.2: Online Specifications

Here, independent platforms such as Detectron2 and ClearML were used to be parsed into the Pytorch environment.

In a very short sentence, Detectron2 is a high-performance, open-source object detection framework built on PyTorch, built by FAIR (Facebook AI Research). It provides many state-of-the-art object detection algorithms and models, which can be easily customized and trained on user-specific Datasets. In this work, Detectron2 was used in order to implement Faster-RCNN.

On the upper hand, ClearML is an open-source platform designed to help developers and data scientists streamline their machine-learning workflows. ClearML provides a centralized interface for managing and monitoring deep learning projects, making it easier to keep track of experiments and analyze results. This platform was used to track real-time metrics of YOLOv5 to avoid waiting until the end of YOLOv5 training to assess the performance. By doing this, a lot of training time was saved. In addition, the training was stopped as soon as it was evident that the chosen path produced unwanted or surprisingly harmful results.

4.2 Dataset

In CV, the quality and size of the Dataset used to train a model are crucial to its performance and ability to generalize to new, unseen data. Generalization is the ability of a model to

predict new data that it has not previously seen accurately.

A diverse and comprehensive Dataset covering a wide range of real-world scenarios can help a model learn robust and representative features that can then be used to detect and classify objects in images or videos[87].

On the case of popular object detection models like YOLO and RCNN, these algorithms rely on large and diverse Datasets to learn the characteristics of different objects in a scene, such as their shape, texture, and color. By exposing the model to many images with different objects, poses, and background conditions, the Algorithm can learn to recognize these objects and their corresponding features with greater accuracy and efficiency[87].

However, it is important to note that the quality of the Dataset is equally essential to its size. A biased Dataset with low-quality images can result in a model that is overfitting to the training data and, therefore, cannot generalize well to new and unseen data[88]. Therefore, it is important to carefully curate and pre-process the Dataset to ensure that it is diverse, comprehensive, and representative of the real-world scenarios in which the model is expected to operate.

As part of this thesis, a platform called Roboflow was used for data collection, labeling, pre-processing, augmentation, and deployment. Roboflow is an emerging web application that provides an open-source solution for these crucial tasks in deep learning CV algorithms. Furthermore, Roboflow completes these fundamental steps more efficiently and with greater accuracy, allowing the focus on other vital aspects of the work to be done deeper.

4.2.1 Data Collection

The process of collecting and preparing data is crucial for the success of object detection algorithms like YOLO and RCNN. As we already understood, the quality and quantity of data available can directly affect the performance of the algorithms. Data collection typically involves finding images or videos that contain the objects of interest, annotating them to indicate the presence and location of the objects, and then preparing the annotated data for use in training the algorithms.

In this case study, pictures from many different real-life scenarios were gathered with the help of Beyond Vision, Roboflow users, and Kaggle users.

The Dataset was roughly composed of 1453 images to which later an augmented step was applied, turning them to 3497 images and 6885 instances of BB.

During the data collection process, a problem arose with duplicated images.This may seem like an innocuous problem, however, duplicated images in a Dataset can lead to overfitting and bias in the training process of a CV algorithm like YOLO or RCNN. Overfitting occurs when the Algorithm memorizes the Dataset instead of learning the underlying patterns, leading to a poor generalization of new, unseen data. In Section 3.5.1, we can find a more precise explanation of this concept. Furthermore, duplicated images can bias the model towards specific classes or objects, leading to an imbalanced model

that performs poorly on some classes. Therefore, having a diverse and balanced Dataset without duplicated images is critical to ensure the model can generalize well to new data.

This issue may appear trivial initially, as image duplication in a Dataset can be easily prevented by verifying each image during data collection. However, ensuring no duplicates exist can be challenging, particularly when merging Datasets from various sources or dealing with large image sets. The risk of inadvertently including duplicates increases as the number of images grows, which can negatively impact the performance of CV algorithms such as YOLO or RCNN, making it crucial to address this problem to ensure optimal model performance.

While manually removing duplicate images from a Dataset might seem feasible, it can be tedious and error-prone, mainly when dealing with large Datasets. Additionally, ensuring that all duplicates have been removed can be challenging since some images may look alike but still possess subtle differences that distinguish them. Such biases and inaccuracies could lead to poor results in the resulting model. By contrast, automating the process of detecting and removing duplicates can save time, improve Precision, and enhance the integrity of the Dataset.

4.2.1.1 Removing Duplicates

In order to tackle the previously stated problem the best way possible, A python file named *detect_and_remove.py* addresses this issue by hashing each image of a given Dataset to later make a hash comparison between an image and all of the other images of the Dataset to check for duplicates was created.

First, the following processes were done:

1. Convert the image into a grey scale and resize it to have one more column of width than height.

2. Compute the relative horizontal gradient between adjacent column pixels

3. Compute the hash value of the previous step

A function with a variable inside is a 2-dimensional array that compares each pair of adjacent columns from left to right on all the picture rows until the last column is created. If the column further away from the start has greater greyscale intensity, the position will be computed as True on the 2D array, and if not, it will be computed as False.

Here, we add a column to the grey scale picture to calculate the horizontal gradient on all the columns otherwise would not be possible because the last column would end up being ignored.

When we get to the return of this same function, we compute the relative horizontal gradient hash value, a unique value each picture will have. Even the slightest difference from one picture to another will make them not duplicate, avoiding the human error of deleting two similar but not duplicate images.

On that return, as we can see, first, the variable previously mentioned gets flattened from a 2D array to a 1D array making it easier to scroll that same array further to compute the hash value. Then using the built-in Python function "enumerate", the I index from each v value of the flattened array that has its value computed as accurate is fetched. Finally, we sum up the indexes according to the following equation in order to get a unique hash value for each image :

$$\text{Hash value} = \sum_{i=0}^{n-1} 2^i$$

4.2.1.2 Data Organization

To achieve the first goal of detecting High Voltage Power Line Towers, it was first necessary to organize the Dataset, which consisted of 1332 images of towers and various background images that could cause errors in the Algorithm. First, five different Datasets were created, each with the same number of labeled pictures but differing numbers of background pictures (40, 80, 120, 13, and 0). Then, these Datasets were trained with the same Hyperparameters to determine the optimal number of background pictures for accurately identifying the towers. After this, it was decided to use 70% of all the images as training images, 20% for validation, and finally, 10% for testing. Unfortunately, there is no standard value for these percentages, which can vary from job to job.

Background images play an essential but sometimes forgotten role in training a Deep Learning Neural Network (DLNN).

During training, the model must learn to distinguish between objects and the background. This is achieved by showing the model many images with a wide range of objects in different backgrounds. The model must learn to differentiate between the objects and the background to classify them accurately.

If the model is trained only on images with objects, it may be unable to distinguish between the object and the background when presented with new images during testing. This can lead to false positives or false negatives, where the model identifies objects that are not present or fails to detect objects that are present.

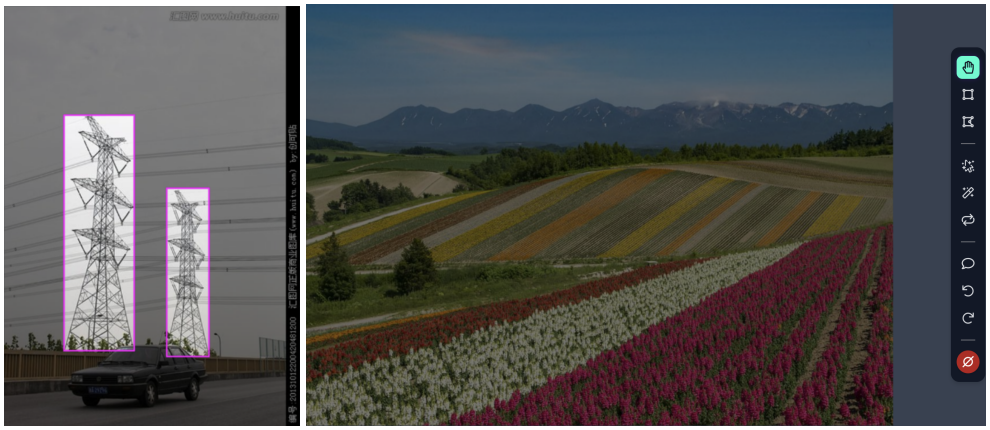
To prevent a high number of false positives detected by YOLO or RCNN, the recommendation of the YOLOv5 creator to use 0-10% background images on the Dataset was followed.

4.2.1.3 Data Annotation

To train an Object Detection Algorithm, it is crucial that all images containing objects of interest be appropriately annotated. This process requires manual annotation of each image, enabling the Algorithm to process and learn from the given data to detect the objects of interest accurately.

Although annotation may seem easy at first glance, it is a slow and tedious process requiring great care and attention to ensure precise and accurate labeling across the entire Dataset.

In this work, Roboflow was utilized. It is an open-source online software that streamlined the annotation process, accelerating it significantly. As a result, it was possible to annotate each image by uploading the Dataset to the app, ensuring Precision and accuracy. In Figure 4.1, we have an image that depicts this pivotal process.



(a) High Voltage PowerLine Tower

(b) Background Image

Figure 4.1: Images Labeling Example

4.2.1.4 Data Pre-Processing

It is essential to employ this small step to prepare a Dataset for training a Deep Neural Network.

Data Pre-processing consists in preparing and transforming raw data into a format that a DLNN can easily consume. This is an essential step in deep learning because raw data is often noisy, unstructured, and contains missing or irrelevant information that can negatively impact the model's performance.

Many pre-processing techniques were not employed on the Dataset used for this project because only a small amount of this technique was required since it was already quite well pre-processed. However, three pre-processing techniques were used in the Dataset: Resizing, Auto-Orientation, and Class modifications.

All the pictures were resized to 640x640, so feeding them to the Deep Learning CV Algorithms would be easier. This was the chosen size beforehand because these algorithms have this size of images as a standard input size.

Another reason for resizing the picture before feeding them to Algorithm was so that it could be possible the glimpse and understand what is the Algorithm seeing as input instead of letting the images go with their size and later letting the Algorithm perform this task by itself, leaving a worse knowledge of what the Algorithm is getting as input.

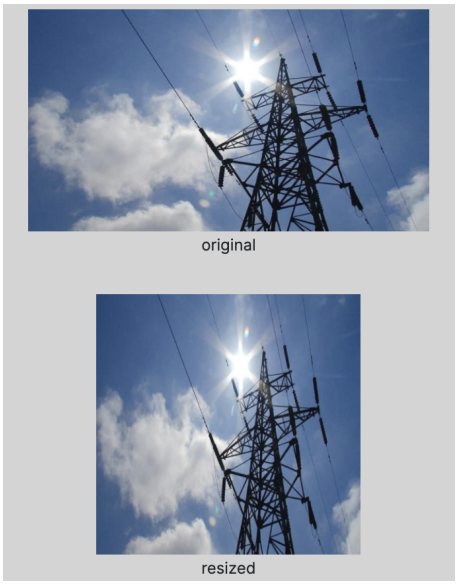


Figure 4.2: Resizing of an image in Roboflow Platform

Another Pre-processing method that was employed on the Dataset was Auto-Orientation. This step helps standardize the orientation of images in a Dataset. This can be particularly useful when dealing with images captured from different devices, as the orientation of the image may vary depending on how the device was held. By auto-orienting the images, we can ensure that they all face the same direction, which can help the model learn more effectively. Even though Auto-Orienting was done to the images and then the rotation on the Augmentation step, it was done to ensure that the images did not get over-rotated. This will be easier to understand in Section 4.2.1.5.

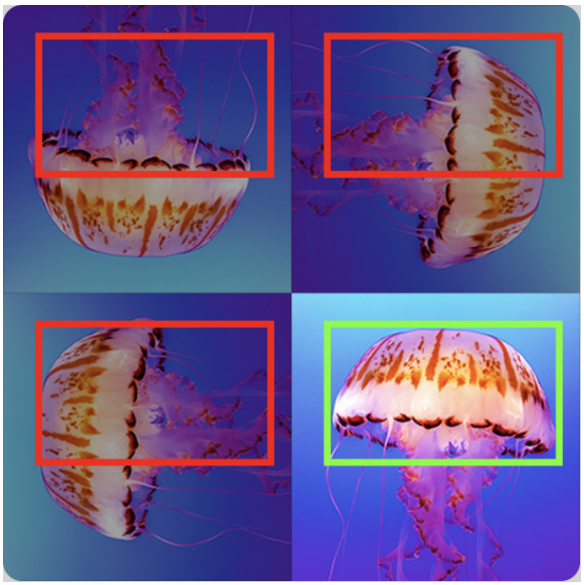


Figure 4.3: Auto-Orienting of an image in Roboflow

The Classes Modification Step was implemented to address the issue of different classes referring to the same object in the merged Datasets. Whenever two or more Datasets are

combined, each Dataset's classes are treated as separate, even if they refer to the same object. Therefore, the classes corresponding to the same object were merged using the pre-processing technique illustrated in Figure 4.4 to address this.

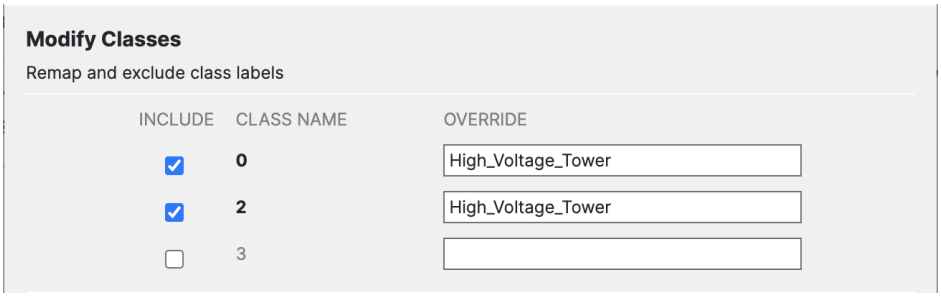


Figure 4.4: Modification of classes of an image in Roboflow Platform

4.2.1.5 Data Augmentation

As a final step in preparing the data for training, it was decided that Data Augmentation techniques would be used, as depicted in Figure 4.5. This term refers to a technique used to artificially expand the size of a Dataset by creating new, modified versions of the original data. This technique benefits deep neural networks like YOLO and RCNN, as they require large amounts of data to train effectively.[88]

Data Augmentation involves applying various transformations to the original data, such as rotations, translations, scaling, flipping, adding Noise, changing lighting conditions, and more.[88] These transformations create new examples that are similar to the original data but with slight variations, effectively increasing the size of the Dataset and providing more varied examples for the network to learn from.

By increasing the size and diversity of the Dataset through Data Augmentation, the deep neural network becomes more robust and can better generalize to new, unseen examples. Additionally, it helps prevent Overfitting.[88]

In this crucial step, the following technique for the images was employed:

1. Crop between 0 and 20%. This was done to have images closer and further away from the object of interest.

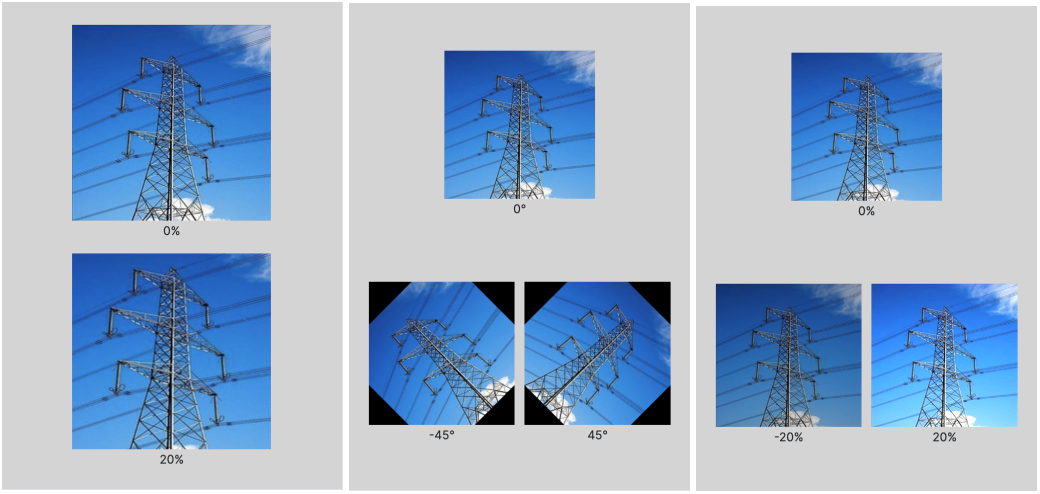
2. Rotation between -45º and +45º. This was done so that the Algorithm could deal with images taken sideways.

3. Brightness between -20º and +20º. This was done so that we could artificially create pictures that mimic photos taken close to sunset time and sunrise time so that the Algorithm could become stronger.

4. Exposure between -10 and 10%. This step can be easily confused with the Brightness step. However, this is linked to the picture's exposure to the sun. This step mimics the differences in sun exposure to the camera, ensuring a better reality cloning to generalize the Algorithm to the real world.

5. Blur up to 0.5 px. Here a Blur effect is employed to the image by adding a random amount of Gaussian blur with a maximum deviation of 0.5 pixels to each pixel. A standard deviation of 0.5 pixels for the Gaussian blur in Roboflow means that each pixel's color value is replaced with a weighted average of its neighboring pixel's color values, where the weights are determined by a Gaussian distribution with a standard deviation of 0.5 pixels. This standard deviation will later enter the equation for the Gaussian Blur via the σ value.

6. Noise with 1% value. This step is used to make the Dataset more resilient to camera artifacts. For that reason, Noise is used to make the Dataset bigger and stronger for training, reducing adversarial attacks and avoiding Overfitting. However, Noise should not be used on the validation and testing sets.



(a) Cropping (b) Rotation (c) Brightness

(d) Exposure (e) Blur (f) Noise

Figure 4.5: All the Data Augmentation processes implemented on the Dataset

4.3 Training and Testing

To accurately identify the position and class of the PL Towers within an image, various pieces of training of the Algorithms on both YOLOv5 and Faster-RCNN were conducted.

Prior to the ultimate pieces of training of the object detection algorithms, preliminary training was carried out on the Dataset to gain familiarity with optimizing the Hyperparameters based on the evaluation metrics. Unfortunately, due to the limited GPU resources provided by Google Colab, some of them were impossible to conduct. Furthermore, even after upgrading to the premium version, which offered priority access to GPUs, certain models training remained impractical because of their significant memory requirements.

This section presents the pieces of trainings conducted on YOLO and Faster-RCNN algorithms separately and compares and evaluates the best-performing algorithms across both approaches.

On every training made for the YOLOv5 and Faster-RCNN Algorithms, the primary metric used for comparison was the mAP@0.5:0.95. This metric was chosen as the primary metric because if we have a high mAP@0.5:0.95, we can consider that the model has a low rate of detecting false negatives and false positives and how good is the Precision across different levels of Recall. Consequently, it was then possible to have the assurance that the model would have a lower chance of identifying a tree or any other object that would not be important as an object of interest.

As we can understand by now, numerous Hyperparameters can be adjusted to enhance the performance of an algorithm for a specific task. Tuning these Hyperparameters ensures that the Algorithm is tailored to meet the unique needs of the task at hand, resulting in better outcomes. An algorithm can be customized to improve accuracy, speed, or other performance metrics by finetuning them, depending on the task's requirements.

This may look simple at first, but the combinations we can make can be on the scale of thousands making it very hard to achieve a great outcome. Furthermore, all of the Hyperparameters present on both YOLOv5 and Faster-RCNN interact and affect the Algorithm differently. Therefore it is mandatory to have a training plan that discards each possible value for each parameter so that we end up with only one possible set of Hyperparameters.

Given the complexity of optimizing Hyperparameters for YOLOv5, a hierarchical approach was adopted to narrow down the possible combinations. Specifically, a focus on the primary Hyperparameters of each Algorithm, including the image size, batch size, and network size, was first employed. By training one combination where only one Hyperparameter would have its value at a time, it was possible to eliminate the values that do not result in optimal performance, identifying, therefore, the best possible values for each parameter. Only after completing this process the exploration of the other Hyperparameters available values was done. This approach made it possible to simplify the training process and focus on the most significant factors, ultimately achieving outstanding results for the specific Dataset. By selecting the Hyperparameters that yield

the best value for the mAP@0.5:0.95 metric, it was then ensured that its value remained stable or increased across various trainings rather than decreasing. This approach helped me avoid any decrease in this metric's value between each training.

4.3.1 YOLOv5

1. Noise on the Dataset.
2. Percentage of Background Images on the Dataset.
3. Image Size.
4. Batch Size.
5. Model Size.
6. Optimizer.
7. Hyperparameter Size.
8. Dropout

As a starter, it was decided that the initial training would have the following values of Hyperparameters and Dataset constraints:

- Batch = 64. A high Batch value will ensure that the Algorithm is not always changing its weights and will make the training faster.
- Epochs = 30. Here it is best to have the highest possible Epochs so that the Algorithm gets to a clear plateau and eventually even stops when the Patience, which was already explained in Section 3.4.8.1, is surpassed. However, this idea works well in theory, but in real life, if that were to be implemented, there would be a very demanding quantity of computer resources. For that same reason, it was decided that 30 epochs of training would be the standard value. Using this value would be enough to understand where each metric curve would plateau and the number of computer units needed would not be too high.
- Image Size = 640x640. This value was chosen based on the tips given by the original creator of YOLOv5, where he advises the usage of a 640x640 image size as a good starting point.
- Model Size = Small. The model's size is a critical Hyperparameter that affects various factors, including the training and inference speed of the algorithms and the demand for GPU resources. By choosing the Small size network, it was then possible to choose many different Hyperparameters such as Batch and Images Sizes, without it being slow or even eventually crashing because of insufficient processing memory.

- % of Backgrounf Images = 3%. In order to have a good algorithm, it is mandatory to use some Background Images on the Dataset so that the Algorithm becomes strong. As a basis, it was decided to be used 3% of Background Images on the Dataset so that there would not be too many Background Images or too few that could lead the Algorithm to a possible uninteded value of mAP@0.5:0.95.

- Hyperparameter Size = Small. For the case studied here, it was best the usage of Hyperparameters that would be the same size as the model. Because the base model was small, the Hyperparameters were chosen with the same size.

4.3.1.1 Noise

As described in Section 4.2.1.5, many different Data Augmentations can be employed on a Dataset to increase the number of pictures and quality so that the Dataset can better clone a real-life scenario.

One possible Data Augmentations is Noise. This augmentation step can be used to strengthen the Algorithm so that it can better process pictures that might have artifacts done for different reasons such as poor lighting conditions, high ISO settings, long exposure times or even heat. This way, the Algorithm may become more resilient and better prepared for real-life scenarios where this problem can occur.

In order to understand if it was viable and better to employ this Data Augmentation process, the training that was done is described in Figure 4.6, where we can see only the value for each of the metrics rounded with two decimal places when the highest value on mAP@0.5:0.95 metric was achieved.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
No Noise	26	0.921	0.809	0.881	0.538	95.177
Noise	29	0.909	0.831	0.896	0.551	86.729

Figure 4.6: Comparison of the Best weights of Datasets with and without Noisy images

As we can see from the values in the table 4.6, the mAP@0.5:0.95 is higher with the Algorithm that has no Noise. That result was expected because Noise is a Data Augmentation procedure that leads to artifacts behind placed on an image. Therefore, with artifacts being placed on an image, both the quality and the features of the image are lost, leading this way to a smaller image quality compared to an image without any of these artifacts. With this, it is easy to understand why the Dataset without Noise has a higher mAP@0.5:0.95.

There needs to be more, to sum up why the Dataset without any Noise should be used. Not only is its mAP@0.5:0.95 higher on one Dataset, but also because when we think of implementing the Algorithm on a real-life scenario, usually only a few of the images

fetched with the drone have this problem associated with it. Therefore it is not the best option to use the Noisy Dataset when it is not the scenario that best describes reality.

4.3.1.2 Background Images

When an algorithm is being trained every time an object is wrongly identified as another object, a BB is drawn around it, which is later considered a False Positive. To avoid these situations, it is a good practice to have a certain amount of background images, in other words, to have a certain amount of images that do not have the Object of interest so that the model gets used to what is considered as a PL Tower and what is not of greater interest, in other words, Background.

Hence, as the original author of YOLOv5 stated, it is a good practice to have a number of Background images that fall between 0-10% of the total images within the Dataset so that the Algorithm becomes more robust and better trained. However, it is not straightforward what is the value of Background images that should be used on a Dataset. With that said, to decide the amount of background images that should be used on this Dataset, it was decided to be done a training with five different Datasets where only the amount of Background Images would differ. Here, the Algorithm was trained using 0,1,3,6 and 9% of Background images.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
9%	24	0.91	0.833	0.897	0.565	91.041
6%	27	0.881	0.811	0.873	0.506	93.779
3%	29	0.909	0.831	0.896	0.551	86.729
1%	27	0.9	0.837	0.886	0.556	86.605
0%	27	0.916	0.821	0.885	0.535	90.156

Figure 4.7: Comparison of the Best weights of Datasets with different percentages of Background Pictures

As we can see by the table 4.7, we can understand that the one Algorithm that yielded the best mAP@0.5:0.95 is the one that had 9% of the total amount of Images within the Dataset as Background Images even though the Datasets that had 3% and 1% of Background images also ended up having a value of mAP@0.5:0.95 that was remarkable. It is essential to identify each Tower and distinguish it from its surroundings accurately, and this Dataset's performance indicates that it is best suited for achieving that objective.

One reason why the Dataset with the highest number of Background Images was chosen for further training is that it contained more images of scenarios where the Algorithm could get confused. For instance, the scenario under study involved landscapes without any PLs. Because the Dataset contained more of these challenging images, the Algorithm was better trained to handle various scenarios, and it became better prepared for training.

4.3.1.3 Image Size

Selecting the suitable image size was difficult since the Dataset contained a wide range of images from different locations. For example, some images were approximately 416x416, while others were as large as 1280x1280. Therefore, after selecting the optimal percentage of background images for the Dataset, the ideal image size to be used as input for the model was evaluated.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
720x720	29	0.884	0.825	0.878	0.507	99.167
640x640	24	0.857	0.813	0.879	0.532	94.126
512x512	29	0.905	0.806	0.878	0.542	103.082
416x416	29	0.916	0.787	0.863	0.528	102.148

Figure 4.8: Comparison of the Best weights of Datasets with different Image Sizes

Based on these findings, a figure size of 512 is the most effective for this particular YOLOv5 Dataset. This outcome is unsurprising, given that the majority of the images in the Dataset are relatively small and closer in size to 416x416. Therefore, by utilizing a figure size of 512x512, we can compensate for the larger image sizes, such as 1280x1280, while minimizing the loss of image features.

4.3.1.4 Batch Size

Once the optimal values were established for the previously stated Hyperparameter, it was time to determine the optimal batch size. A series of trainings were conducted, where the batch size was altered while keeping the other parameters constant. The results are presented in the attached Figure labeled as 4.9. Notably, the results confirmed that a batch size of 16 is optimal for the situation under study, which is consistent with the commonly recommended values of 16, 32, or 64 for this parameter.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
128	28	0.893	0.809	0.876	0.521	93.464
96	22	0.913	0.784	0.868	0.534	89.588
80	27	0.9	0.795	0.872	0.54	99.968
64	22	0.917	0.769	0.859	0.512	91.357
40	29	0.91	0.801	0.877	0.517	86.743
32	19	0.909	0.761	0.857	0.525	87.998
20	27	0.89	0.798	0.87	0.516	95.936
16	29	0.905	0.806	0.878	0.542	103.082
8	29	0.893	0.795	0.863	0.519	95.187

Figure 4.9: Comparison of the Best weights of Datasets with different Batch Sizes

Increasing the batch size has several advantages, such as processing more data in

parallel, resulting in a more stable training process. This can reduce variance in the gradient estimates and prevent the optimization process from getting stuck in a local minimum. However, larger batch sizes require more GPU memory which can be a limitation depending on available resources. Nonetheless, larger batch sizes can significantly reduce the training time, allowing the training of the model with more epochs in less time.

4.3.1.5 Model Size

As we know, YOLOv5 has five different model sizes. Each of those models has a tradeoff between model speed and accuracy. Therefore, choosing a model that best suits the configurations that one has is very important because it will directly affect the models and therefore affect the inference speed and accuracy. These tradeoffs between speed and accuracy are inversely proportional, meaning that the faster models are generally less accurate and vice-versa. Nevertheless, the smaller models tend to fall on the first rule previously stated, while the bigger models tend to fall on the later rule stated.

In this work, this Hyperparameter is no exception to this rule. Therefore, it is very important to have this Hyperparameter well prorated. To do so, it is necessary to train each model size to know the best one.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
Extra Large	28	0.938	0.836	0.909	0.665	27.175
Large	29	0.926	0.83	0.907	0.634	50.29
Medium	29	0.924	0.839	0.911	0.626	70.851
Small	24	0.932	0.798	0.888	0.569	91.583
Nano	29	0.905	0.806	0.878	0.542	102.148

Figure 4.10: Comparison of the Best weights of Datasets with different Model Sizes

Table 4.11 shows that larger models generally have a higher mAP@0.5:0.95. This was expected as larger models have a deeper backbone and can learn more features from the Dataset. As we know, it could be a problem with the Velocity on which this XL model works. Nevertheless, the speed that it shows is not small enough to be a complication in the scenario where it will be used. Despite this, if the GPU processing power on which this model would be trained were to be smaller, the decision on which model size to use could easily differ because of the highly demanding memories that higher models tend to have, leading this way to a significant discrepancy on the inference times of each model.

4.3.1.6 Optimizer

As described in Section 3.4.1, the Optimizer's job is to minimize the loss function during training by updating the model's weight parameters. Choosing the best Optimizer may be why the model converges faster or slower on its metrics to a potentially high value. Therefore one must take extra care in deciding the best value for this Hyperparameter.

YOLOv5 lets its users use one of the following three Optimizers: SGD, Adam, and AdamW. To decide which could be the best value here, there was only one way to find out: training each of them one by one using the previously finetuned configurations.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
AdamW	27	0.901	0.82	0.898	0.581	27.571
Adam	28	0.906	0.823	0.899	0.537	28.02
SGD	28	0.938	0.836	0.909	0.665	27.175

Figure 4.11: Comparison of the Best weights of Datasets with different percentages of Background Pictures

As we can see from the table 4.11, the best Optimizer is SGD. Usually, as the author describes, the SGD is a great Optimizer in the long run, but even with this information, that could not be the case on the Dataset being used here. However, this Hyperparameter is significantly empirical. Therefore necessary to train it because in theory may not mean in practice.

4.3.2 Hyperparameter Size

As we have seen before, the Hyperparameter's Size commonly depends on the Model's Size. Usually, when we have Models that are either Small or Nano in size, this Hyperparameter should have a small size. So for the medium and large-sized models the same rule applies. With that said, one could understand that no finetuning should be made on this Hyperparameter. Nevertheless, it is essential to understand how greatly the Hyperparameter's values would affect the model's mAP@0.5:0.95 and other metrics. So for that exact reason, the trainings were done with Small, Medium, High and No Augmentations as Hyperparameter Sizes.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
No Augmentations	27	0.914	0.816	0.881	0.562	27.205
High	28	0.932	0.844	0.918	0.667	27.609
Medium	28	0.932	0.844	0.918	0.667	27.005
Small	28	0.938	0.836	0.909	0.665	27.175

Figure 4.12: Comparison of the Best weights of Datasets with different Hyperparameter Sizes

As we can see in the table 4.12, the Hyperparameter Sizes generally did not affect the Algorithm well enough on 30 epochs to understand if it would be better to use either the Medium or the High Sized Hyperparameter Sizes.

Nonetheless, the decision upon what Hyperparameters Size ended up falling upon what is recommended by the original author of YOLOv5, making the Highest Sized

Hyperparameter File the one that would fit the best the Algorithm.

4.3.3 Dropout

As we have seen in Section 3.5.2, some neurons may start having higher attention given to them than others on the Neural Network. To avoid this undesirable situation, sometimes Dropout may be helpful because it ignores a certain predefined amount of the neurons on the Neural Network, avoiding bias in the System. However, this amount should be a reasonably small value so that the System does not ignore too many neurons that may analyze critical features in the data. For this same reason, it was decided that a Dropout value of 0.1, meaning that 10% of the neurons would be randomly dropped out.

Model	Epochs	Precision	Recall	mAP@0.5	mAP@0.5:0.95	FPS
Dropout	29	0.945	0.825	0.92	0.645	27.745
No Dropout	28	0.932	0.844	0.918	0.667	27.609

Figure 4.13: Comparison of the Best weights of Models trained with and without Dropout

As we can see from table 4.13, implementing Dropout on the Algorithm proves to be a bad solution because the mAP@0.5:0.95 and other metrics become much smaller. That happens because some key features are being dropped out, and therefore the model is not learning as well as it should be.

4.4 Faster-RCNN

Just like YOLOv5, the training of this Algorithm was also arranged hierarchically, discarding the Hyperparameter values that were not the best fit for the Algorithm.

However, the training funneling was done differently due to different implementations on YOLOv5 and Faster-RCNN. Some Hyperparameters could be finetuned on YOLOv5 that could not be finetuned on Faster-RCNN because of the coding and possible drawbacks and problems it could generate.

Nevertheless, different Hyperparameters were changed to obtain the best mAP@0.5:0.95 for the reasons mentioned above. Those Hyperparameters were namely:

1. Model Size.
2. Image Size.
3. Batch Size.
4. Learning Rate.
5. Momentum.

This Faster-RCNN implementation, unlike YOLOv5, discarded every image with no labeled object on it. For this same reason, no different amount of Background Images was trained and compared. Also, because it had already been proved that Noise is not a good Data Augmentation to be used on the problem being studied to save a higher amount of computer processing power for other pieces of training, this step was skipped.

Faster-RCNN was implemented with Detectron2. This Platform has the advantage of providing many different object detection and image segmentation algorithms in one place. However, it leads to a higher amount of code to be written and more variables that can affect the training process.

This implementation of Faster-RCNN was created by a different company than the one that developed YOLOv5. Therefore some parameters not only had a different name but also had a different implementation. A noticeable case was the number of epochs. In contrast to YOLOv5, the number of epochs was not a Hyperparameter for this, but rather the number of iterations. In order then to train the Algorithm similarly, the number of Iterations was calculated according to the Equation described on :

$$Iterations = \frac{Total\ number\ of\ Images \times Number\ of\ Epochs}{Batch\ Size} \tag{4.1}$$

Just like YOLOv5, in order to discard the values that would be best for each parameter, there was at first a need for a starting point where the following Hyperparameters were then implemented:

- Batch = 128. Unlike YOLOv5, this batch was enough to assure that the model could have the computer processing power to train the other Hyperparameters without crashing due to a lack of processing power.
- Epochs = 30. To optimize computer resources, the number of epochs chosen for this model was based on finding a value sufficient for the metrics to converge or come close to converging. This would provide an idea of what the plateau would be. This approach is similar to the one used in the YOLOv5 section.
- Size = 512x512. This image size was strongly based on the value YOLOv5 had of image size and also on the normal average size of the Dataset by looking at the images.

4.4.1 Model Size

As discussed in Section 4.3.1.5, the model size is a Hyperparameter of great importance because it can be one of the main reasons a model might perform better or worst in terms of inference time as well as mAP@0.5:0.95 quality. On Faster-RCNN, the reasons previously stated are no exception. Therefore, this parameter must be chosen meticulously.

At the beginning of the Faster-RCNN pieces of training, good values were observed for the mAP@0.5:0.95, and impressive Precision was offered by the Algorithm. However, it

was also possible to verify a significant drawback. This Algorithm's speed was significantly low, which was one of the first problems that had to be solved. Because the Model Size significantly affects the model's speed, that was the first Hyperparameter change.

The training process involved the largest and smallest algorithms to determine which would be the most effective and to compare the differences in inference times between the two.

Model	Iterations	AP@0.5:0.95	AP@0.5	FPS
R-50-FPN	652	0.34	0.774	3.559
X_101_FPN	652	0.418	0.796	2.234

Figure 4.14: Comparison of the Best weights of Datasets with different Model Sizes

As we can observe from table 4.14, the Algorithm that had as Backbone the Resnet-X-101 was better in terms of mAP@0.5:0.95. However, its inference speed was so low that it could be dangerous in a real-life scenario. However, the Resnet-FPN, on the upper hand, revealed a fine mAP@0.5:0.95, as we can see in table 4.14, and it had almost twice the number of FPS that the Resnet-X-101 Backbone was providing. Knowing this, it was obvious that the Backbone that would better fit the job being studied was the Resnet-FPN.

4.4.2 Image Size

As we know, every CV Deep Learning Algorithm has a constraint named Image Size. However, on Faster-RCNN, unlike YOLOv5, this constraint does not appear as a configuration that can be altered directly on the Platform. Therefore, this important value was changed on the Roboflow platform instead to understand how the image's amount of pixels could affect this Algorithm.

Just like on YOLOv5, here, different image-sized Datasets were fed to the Algorithm. These different image sizes were 416x416,512x512,640x640. However, unlike YOLOv5, on this Algorithm, only three different image sizes were trained because of the limits in computer resources that Roboflow provided.

Model	Iterations	AP@0.5:0.95	AP@0.5	FPS
640	651	0.302	0.721	3.773
512	652	0.34	0.774	3.559
416	652	0.357	0.764	3.582

Figure 4.15: Comparison of the Best weights of Datasets with different Image Sizes

As we can see in table 4.15, the results proved that the best Image Size to feed the Algorithm is 416x416.

4.4.3 Batch Size

As it has already been discussed on YOLOv5 and in Section 3.4.7, this Hyperparameter has greater importance both in terms of training time as well as quality of the model. Therefore, it is imperative to be meticulous when choosing this value because it will affect the training results.

Like on YOLOv5, this Hyperparameter was trained on the Algorithm's training under nine different values: 128,96,80,64,40,32,20,16 and 8. By doing so, there was a higher spectrum of tests upon the training of the Algorithm, leading to a better-sustained decision on choosing its value.

Model	Iterations	AP@0.5:0.95	AP@0.5	FPS
128	652	0.357	0.764	3.582
96	869	0.351	0.761	3.64
80	1043	0.314	0.73	3.672
64	1304	0.395	0.769	3.699
40	2086	0.309	0.762	3.673
32	2608	0.397	0.788	3.516
20	4173	0.377	0.792	3.597
16	5216	0.329	0.79	3.531
8	10432	0.334	0.776	3.037

Figure 4.16: Comparison of the Best weights of Datasets with different Batch Sizes

As we can see from the results obtained in table 4.16, the best value for Batch Size is 32. This value does not come as a surprise because, as stated before, the value for Batch Size should usually fall on one of the three values: 16, 32 or 64.

4.4.4 Learning Rate and Momentum

The Learning Rate and the Momentum values directly affect the weights update after each Iteration. Usually, one should test these values as pair on the training of the CV Algorithm. This way, it is easier to understand the best values to employ to obtain the best possible results.

The Learning Rate will decide what percentage of the total change proposed by the gradient descent should be employed upon the weights of the Algorithm. On the upper hand, the Momentum will act as inertia to the direction changes of the gradient descent so that it does not do a very abrupt change.

This way, it is possible to understand why both values should be tested together rather than separately. In order to do so, it was decided that the training should be done using each of these lr0 values:(0.001,0.0001,0.00001) for each of these Momentum Values (0.9,0.99,0.999).

Unfortunately, because the training of Faster-RCNN with Momentum = 0.999 and lr0 = 0.1 was constantly crashing, it was not possible to obtain the metrics for this same training.

Model	Iterations	AP@0.5:0.95	AP@0.5	FPS
Momentum = 0.99	2608	0.16	0.507	2.799
Momentum = 0.9	2608	0.371	0.783	2.802

(a) lr0 = 0.001

Model	Iterations	AP@0.5:0.95	AP@0.5	FPS
Momentum = 0.999	2608	0.097	0.273	3.365
Momentum = 0.99	2608	0.353	0.779	2.647
Momentum = 0.9	2608	0.305	0.712	3.07

(b) lr0 = 0.0001

Model	Iterations	AP@0.5:0.95	AP@0.5	FPS
Momentum = 0.999	2608	0.326	0.745	3.505
Momentum = 0.99	2608	0.331	0.732	3.288
Momentum = 0.9	2608	0.159	0.421	3.167

(c) lr0 = 0.00001

Figure 4.17: Different Trainings done with various lr0 and Momentum values on Faster-RCNN

As we can see from Table 4.17, using Momentum and lr0 values that are higher than the predefined values does not bring any positive change to the value of mAP@0.5:0.95 that was obtained on the last training. Herefore, it is possible to understand then that the standard values, which are Momentum = 0.9 and lr0 = 0.001, are the best values among all values presented in this section to train the Faster-RCNN Algorithm.

4.5 Comparing Results

As we understand by now, the results obtained from all of the training done with the YOLOv5 and the Faster-RCNN algorithms proved some critical differences between them.

The first noticeable key difference between these Algorithms was the FPS or, in other words, the number of pictures that both Algorithms could process in one second. From

the start of the training processes of both algorithms, we could understand that the YOLOv5 had an inference time that was much superior to the one that Faster-RCNN showed. When on the best Algorithm that was reached on each Algorithm, it was more straightforward and clear that YOLOv5 has an FPS value about 8 times superior to the one that Faster-RCNN had.

Depending on the real-life scenario that the training is being conducted for, this value can affect more or less the decision on if that is a deciding factor or not to discard one Algorithm over the other. For the PL Tower detection, it is clear that the FPS is an essential factor to consider. A small UAV usually flies at a velocity smaller than 15Km/h, and a bigger UAV flies at a velocity that can be up to 100Km/h[89]. This means that at each second, the environment that the UAV is checking can change at a very high rate.

Using the Faster-RCNN under these circumstances means that if the UAV is flying at a speed of 15 Km/h, only one frame will be processed every 5 meters that he is flying on the upper hand if the YOLOv5 is used to complete the task at study on this document, this Algorithm instead will be capable of detecting its surrounding every 0.5 meters.

This way we can understand that a drone that is mounted is an algorithm that is capable of processing a higher amount of frames per second is going to be safer because, in the eventuality of an unexpected object or animal crossing the path of this UAV, it will be capable of responding to that adversity better and in a smother and less dangerous way than an algorithm that processes a smaller amount of FPS. For this reason, after every test during the training of both Faster-RCNN and YOLOv5 Algorithm, it became clear that YOLOv5 would be a better option on a scene where it needed a higher and faster response to the world's possible adversities.

After looking at the results of both Algorithms under study in this document, one can understand that the Faster-RCNN revealed better Precision results when compared to the ones that YOLOv5 provided. At first, it would seem like a great advantage because it would better recognize the Object under study, but Precision is not the only important metric. To determine the best Algorithm for the task being investigated in this document, the mAP0.5:0.95 metric was selected as the main factor so that a lower rate of false negatives and false positives would be achieved. This way, the model would be exact in its detection. After this metric, metrics such as Precision also had great importance for obvious reasons.

Therefore, after a deeper look at the metrics that Faster-RCNN had outputted on its validation during training, one can understand that the high Precision that this Algorithm proved to have had a significant drawback on mAP@0.5:0.95 when compared to YOLOv5. For this same reason, it is possible to understand why YOLO mistakes himself fewer times on detecting the Object of interest when compared to Faster-RCNN. This became clearer with the video used to calculate the FPS that each Algorithm outputted. Therefore, the YOLOv5 is a better Algorithm for detecting PL Towers.

5

Conclusions and Future Work

In this document, the focus was on the Object Detection of PL Towers. This was possible by using Deep Learning CV Algorithms.

To achieve the goal described in this document, it was imperative that a Dataset would have enough pictures to capture the event where the Algorithm would be later deployed. By training and taking a closer look at the metrics outputted by the Algorithm, it was possible to understand the importance of using a Dataset that represents every possible nuance and little detail that would represent reality.

Although great results were obtained for the job under investigation, if the number of computer units used were to be bigger, it would be easier to train these Deep Learning Algorithms with more pictures of PLs.

This way, more images would make the Algorithm better prepared for all the situations where the Object under study could be found. However, adding images that are harmful to the Algorithm, such as images with low lighting conditions, overexposed or underexposed, images with occlusions of other objects, and images that are taken from misleading angles, can be misleading on its learning. Therefore, properly gathering the Algorithm images should always be meticulous and carefully done so that the Algorithm can learn to Detect the Object of Interest in the best way possible.

Not only the images might be a problem, but also the BBs and the annotations that come with them can also lead the Algorithm to mistake himself. Incorrect annotations and BBs can negatively impact the CV Algorithm's ability to get accurate predictions because the Algorithm might get confused about what can be the Objects of Interest.

Nevertheless, using a greater Dataset can also lead to a higher training time. With the increase of images in the Dataset, the number of iterations needed for the Dataset to do each Epoch of training will be bigger because of the amount of data that needs to be processed. Therefore, a longer training time will be needed.

Only one class was trained during the training so that the UAV would first focus on one task at a time. By doing so, the UAV has first to detect a PL Tower and only then it focus itself on detecting possible malfunctioning parts of the Grid, such as Insulators, Cables, and other possible damaged components of the PL Grid. Therefore, only the PL

Towers detection was performed in this document. The aim was to expand knowledge on how object detection tasks can be developed and refined to ensure their reliability and safety in real-life scenarios.

As Future work, the UAV must get closer to the PL Towers according to the data gathered from the Deep Learning Object Detection Algorithm. Then, when the device is at a certain predefined distance, it should gather information about the different components of the Tower using another algorithm trained to detect and output the conditions of the components and decide if any of those components have any fault that should be addressed and fixed. After doing this task on one Tower of the Grid, the UAV could either, with the help of a GPS or with the help of another CV Algorithm, proceed to the next Tower to redo the same task of detecting the possible faults on it until it would reach the end of the Grid under study.

Also, it would be essential to stabilize the outputted Object Detections in videos. For example, when a drone is flying near the PLs, the images captured of the PL Towers do not change much between frames of the obtained video. However, the CV Algorithms, unlike people, do not understand when they draw their detected BBs around the Object leading to the BBs having apparent differences between frames of the video where the changes of the images were not significant. By doing this update to the proposed tool, the detection of the Objects would be smoother and better by having a greater consistency in detecting the PL Towers between video frames.

Also, every training of both the YOLOv5 and Faster-RCNN was done using pre-trained weights instead of doing Transfer Learning. In Future work, it would be helpful for these algorithms to perform all the pieces of training using this technique so that other results could be gathered. However, using this technique could lead one to do each piece of training with a higher amount of Epochs because the saved weights obtained from each training could be harder to compare due to very close results between them, deciding what value for each Hyperparameter to take more challenging to do.

Other interesting future work could be changing the Detectron2 Faster-RCNN weights saved from the last to the best ones. By doing so, better weights could be obtained with this Algorithm leading to possible higher values in mAP@0.5:0.95 and FPS.

Another significant improvement would be using different activation functions during training. This improvement would effectively improve a neural network's performance. Activation functions determine the output of each neuron based on its input and can impact the accuracy and computational costs of the network. While there was no exploration of different activation functions in this document, they are essential for achieving better results in detecting PL Towers. Therefore, the choice of activation function should be carefully considered based on the specific problem and network architecture.

Due to limited computer resources, the training of the algorithms could not be completed as planned. However, a single training on YOLOv5 to evaluate the benefits of detecting PL Towers using a mighty GPU was done. This training was conducted using a Tesla A-100 GPU, which achieved a speed of 60 FPS in detecting the Object of Interest,

proving how game-changing that can be for this task.

Also, it is essential to point out that only a Tesla T4 GPU was used for the rest of the trainings. Therefore, for future work, it would be interesting to train the Algorithm using the computer's CPU or a graphics card embedded in a UAV to perform PL Towers detection.

After everything, it is possible to conclude that this implementation could be a novel and game-changing approach to PL Detection where a safer, faster, and cheaper solution would guarantee any Electric Company that its Grid has everything working as it should. Of course, there would still be many implementations to be studied and done, but with the resource of the study presented in this document, we are one step closer to achieving that same objective.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[2] S. J. Mills et al. "Evaluation of Aerial Remote Sensing Techniques for Vegetation Management in Power-Line Corridors". In: *IEEE Transactions on Geoscience and Remote Sensing* 48.9 (2010-09), pp. 3379–3390. ISSN: 0196-2892, 1558-0644. DOI: 10.1109/TGRS.2010.2046905. URL: http://ieeexplore.ieee.org/document/5466249/ (cit. on p. 1).

[3] H. Zhang et al. "Detecting Power Lines in UAV Images with Convolutional Features and Structured Constraints". en. In: *Remote Sensing* 11.11 (2019-06), p. 1342. ISSN: 2072-4292. DOI: 10.3390/rs11111342. URL: https://www.mdpi.com/2072-4292/11/11/1342 (cit. on pp. 1, 6, 17).

[4] S. Reif-Acherman. "Ernst Werner Von Siemens and the Early Evolution and Diffusion of Electric Telegraphy [Scanning Our Past]". In: *Proceedings of the IEEE* 105.11 (2017-11), pp. 2274–2284. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2017.2752583. URL: http://ieeexplore.ieee.org/document/8074557/ (cit. on p. 2).

[5] M. Guarnieri. "The Beginning of Electric Energy Transmission: Part One [Historical]". In: *IEEE Industrial Electronics Magazine* 7.1 (2013-03), pp. 50–52. ISSN: 1932-4529. DOI: 10.1109/MIE.2012.2236484. URL: http://ieeexplore.ieee.org/document/6482228/ (cit. on p. 2).

[6] A. Gómez Expósito, A. J. Conejo, and C. Cañizares, eds. *Electric energy systems: analysis and operation*. The electric power engineering series. OCLC: ocn213765933. Boca Raton: CRC Press, 2009. ISBN: 978-0-8493-7365-7 (cit. on pp. 2–4).

[7] M. H. Brown and R. P. Sedano. *Electricity transmission: a primer*. OCLC: ocm56480735. Denver, Colo.: National Council on Electric[ity] Policy, 2004. ISBN: 978-1-58024-352-0 (cit. on pp. 2–4).

[8] T. Kishore and S. Singal. "Optimal economic planning of power transmission lines: A review". en. In: *Renewable and Sustainable Energy Reviews* 39 (2014-11), pp. 949–974. ISSN: 13640321. DOI: 10.1016/j.rser.2014.07.125. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364032114005772 (cit. on p. 3).

[9] IAN Symbols. *Energy: nuclear power station*. [Online; accessed June 22, 2022]. 2022. URL: https://vecta.io/symbols/310/human-development-infrastructure/31/energy-nuclear-power-station (cit. on p. 4).

[10] Anton Tokarev. *Ligne à haute tension aérienne tour de transmission*. [Online; accessed June 17, 2022]. 2022. URL: https://fr.dreamstime.com/ligne-%C3%A0-haute-tension-a%C3%A9rienne-tour-transmission-image130241130 (cit. on p. 4).

[11] rastudio. *Factory hand drawn outline doodle icon. air pollution by smoke coming out of factory chimneys vector sketch illustration for print, web, mobile and infographics isolated on white background.* [Online; accessed June 22, 2022]. 2022. URL: https://www.freepik.com/premium-vector/factory-hand-drawn-outline-doodle-icon-air-pollution-by-smoke-coming-out-factory-chimneys-vector-sketch-illustration-print-web-mobile-infographics-isolated-white-background_16378600.htm#page=3&query=steam%20coming%20out&position=18&from_view=keyword (cit. on p. 4).

[12] Global Film Locations. *Drawing UP House (In 30 Steps)*. [Online; accessed June 22, 2022]. 2022. URL: https://globalfilmlocations.net/2017/05/25/drawing-up-house-in-30-steps/ (cit. on p. 4).

[13] Vector Home. *Ícone bonde do transformador - ilustração do vetor*. [Online; accessed June 17, 2022]. 2022. URL: https://pt.dreamstime.com/%C3%ADcone-bonde-do-transformador-ilustra%C3%A7%C3%A3o-vetor-image112130561 (cit. on p. 4).

[14] N. Mbuli et al. "A literature review on capacity uprate of transmission lines: 2008 to 2018". en. In: *Electric Power Systems Research* 170 (2019-05), pp. 215–221. ISSN: 03787796. DOI: 10.1016/j.epsr.2019.01.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378779619300124 (cit. on p. 4).

[15] M. Khanna and N. D. Rao. "Supply and Demand of Electricity in the Developing World". en. In: *Annual Review of Resource Economics* 1.1 (2009-10), pp. 567–596. ISSN: 1941-1340, 1941-1359. DOI: 10.1146/annurev.resource.050708.144230. URL: https://www.annualreviews.org/doi/10.1146/annurev.resource.050708.144230 (cit. on p. 4).

[16] T. Santos et al. "PLineD: Vision-based power lines detection for Unmanned Aerial Vehicles". In: *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. Coimbra, Portugal: IEEE, 2017-04, pp. 253–259. ISBN: 978-1-5090-6234-8. DOI: 10.1109/ICARSC.2017.7964084. URL: http://ieeexplore.ieee.org/document/7964084/ (cit. on pp. 6, 17–19, 24).

[17] J. Wang et al. "Cooperative Transmission Tower Inspection with a Vehicle and a UAV in Urban Areas". en. In: *Energies* 13.2 (2020-01), p. 326. ISSN: 1996-1073. DOI: 10.3390/en13020326. URL: https://www.mdpi.com/1996-1073/13/2/326 (cit. on pp. 6, 18).

[18] B. Han and X. Wang. "Detection for Power line Inspection". In: *MATEC Web of Conferences* 100 (2017). Ed. by L. Zhao et al., p. 03010. ISSN: 2261-236X. DOI: 10.1051/matecconf/201710003010. URL: http://www.matec-conferences.org/10.1051/matecconf/201710003010 (cit. on pp. 6, 18).

[19] F. Azevedo et al. "LiDAR-Based Real-Time Detection and Modeling of Power Lines for Unmanned Aerial Vehicles". en. In: *Sensors* 19.8 (2019-04), p. 1812. ISSN: 1424-8220. DOI: 10.3390/s19081812. URL: https://www.mdpi.com/1424-8220/19/8/1812 (cit. on pp. 6, 17–19).

[20] M. Gazzea et al. "Automated Power Lines Vegetation Monitoring Using High-Resolution Satellite Imagery". In: *IEEE Transactions on Power Delivery* 37.1 (2022-02), pp. 308–316. ISSN: 0885-8977, 1937-4208. DOI: 10.1109/TPWRD.2021.3059307. URL: https://ieeexplore.ieee.org/document/9354535/ (cit. on pp. 8, 10).

[21] K. Hafeez et al. "Risk management of overhead electric power lines". In: *2018 1st International Conference on Power, Energy and Smart Grid (ICPESG)*. Mirpur Azad Kashmir: IEEE, 2018-04, pp. 1–5. ISBN: 978-1-5386-5482-8. DOI: 10.1109/ICPESG.2018.8384504. URL: https://ieeexplore.ieee.org/document/8384504/ (cit. on pp. 8, 9).

[22] L. D. Biasotto and A. Kindel. "Power lines and impacts on biodiversity: A systematic review". en. In: *Environmental Impact Assessment Review* 71 (2018-07), pp. 110–119. ISSN: 01959255. DOI: 10.1016/j.eiar.2018.04.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0195925517304432 (cit. on pp. 8, 10).

[23] D. Pylarinos. "Overhead Transmission Line Easement and Right-of-Way Cases in Crete, Greece: A Statistical Analysis of 1220 Cases from 1974 to 2019". In: *Engineering, Technology & Applied Science Research* 10.3 (2020-06), pp. 5581–5589. ISSN: 1792-8036, 2241-4487. DOI: 10.48084/etasr.3591. URL: https://etasr.com/index.php/ETASR/article/view/3591 (cit. on p. 8).

[24] L. Matikainen et al. "Remote sensing methods for power line corridor surveys". en. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 119 (2016-09), pp. 10–31. ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2016.04.011. URL: https://linkinghub.elsevier.com/retrieve/pii/S0924271616300697 (cit. on pp. 8, 13, 14).

[25] S. Guggenmoos. "Effects of Tree Mortality on Power Line Security". In: *Arboriculture & Urban Forestry* 29.4 (2003-07), pp. 181–196. ISSN: 19355297, 21550778. DOI: 10.48044/jauf.2003.022. URL: https://joa.isa-arbor.com/article_detail.asp?JournalID=1&VolumeID=29&IssueID=4&ArticleID=93 (cit. on p. 8).

[26] I. Ituen, G. Sohn, and A. Jenkins. "A CASE STUDY: WORKFLOW ANALYSIS OF POWERLINE SYSTEMS FOR RISK MANAGEMENT". In: (2008-01) (cit. on p. 8).

[27] Agdekon Media Visuals. *Electric Tower Vector Art*. [Online; accessed June 10, 2022]. 2022. URL: https://www.istockphoto.com/pt/vetorial/electricity-power-lines-silhouette-on-white-gm1390256986-447280923?irgwc=1&cid=IS&utm_medium=affiliate&utm_source=Eezy (cit. on p. 9).

[28] B. Weedy. "Environmental aspects of route selection for overhead lines in the U.S.A." en. In: *Electric Power Systems Research* 16.3 (1989-05), pp. 217–226. ISSN: 03787796. DOI: 10.1016/0378-7796(89)90014-X. URL: https://linkinghub.elsevier.com/retrieve/pii/037877968990014X (cit. on pp. 9, 10).

[29] D. Jones. "Power line inspection - a UAV concept". en. In: *IEE Forum on: Autonomous Systems*. Vol. 2005. London, UK: IEE, 2005, pp. 6–6. ISBN: 978-0-86341-586-9. DOI: 10.1049/ic:20050472. URL: https://digital-library.theiet.org/content/conferences/10.1049/ic_20050472 (cit. on pp. 12–14).

[30] C. Martinez et al. "The Power Line Inspection Software (PoLIS): A versatile system for automating power line inspection". en. In: *Engineering Applications of Artificial Intelligence* 71 (2018-05), pp. 293–314. ISSN: 09521976. DOI: 10.1016/j.engappai.2018.02.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S0952197618300290 (cit. on pp. 12–15, 19, 21, 24).

[31] J. Katrasnik, F. Pernus, and B. Likar. "A Survey of Mobile Robots for Distribution Power Line Inspection". In: *IEEE Transactions on Power Delivery* 25.1 (2010-01), pp. 485–493. ISSN: 0885-8977, 1937-4208. DOI: 10.1109/TPWRD.2009.2035427. URL: http://ieeexplore.ieee.org/document/5345712/ (cit. on pp. 12–15).

[32] S. Rong et al. "Intelligent Detection of Vegetation Encroachment of Power Lines With Advanced Stereovision". In: *IEEE Transactions on Power Delivery* 36.6 (2021-12), pp. 3477–3485. ISSN: 0885-8977, 1937-4208. DOI: 10.1109/TPWRD.2020.3043433. URL: https://ieeexplore.ieee.org/document/9301267/ (cit. on pp. 12, 13).

[33] D. Jones et al. "Aerial video inspection of overhead power lines". en. In: *Power Engineering Journal* 15.1 (2001-02), pp. 25–32. ISSN: 0950-3366. DOI: 10.1049/pe:20010103. URL: https://digital-library.theiet.org/content/journals/10.1049/pe_20010103 (cit. on pp. 12–14).

[34] M. Nayyerloo et al. "Cable-Climbing Robots for Power Transmission Lines Inspection". en. In: *Mobile Robots - State of the Art in Land, Sea, Air, and Collaborative Missions*. Ed. by X. Chen, Y. Chen, and J. Chase. InTech, 2009-05. ISBN: 978-953-307-001-8. DOI: 10.5772/6989. URL: http://www.intechopen.com/books/mobile-robots-state-of-the-art-in-land-sea-air-and-collaborative-missions/cable-climbing-robots-for-power-transmission-lines-inspection (cit. on pp. 13–15).

[35] L. Yang et al. "A Review on State-of-the-Art Power Line Inspection Techniques". In: *IEEE Transactions on Instrumentation and Measurement* 69.12 (2020-12), pp. 9350–9365. ISSN: 0018-9456, 1557-9662. DOI: 10.1109/TIM.2020.3031194. URL: https://ieeexplore.ieee.org/document/9225728/ (cit. on pp. 13–17).

[36] INMR. *Practical Issues for Effective Aerial Patrols of Overhead Lines*. [Online; accessed July 1, 2022]. 2022. URL: https://www.inmr.com/practical-issues-effective-aerial-patrols-overhead-lines-2/ (cit. on p. 13).

[37] L. F. Luque-Vega et al. "Power line inspection via an unmanned aerial system based on the quadrotor helicopter". In: *MELECON 2014 - 2014 17th IEEE Mediterranean Electrotechnical Conference*. Beirut, Lebanon: IEEE, 2014-04, pp. 393–397. ISBN: 978-1-4799-2337-3. DOI: 10.1109/MELCON.2014.6820566. URL: https://ieeexplore.ieee.org/document/6820566 (cit. on pp. 14, 15, 18).

[38] S. Hrabar, T. Merz, and D. Frousheger. "Development of an autonomous helicopter for aerial powerline inspections". In: *2010 1st International Conference on Applied Robotics for the Power Industry (CARPI 2010)*. Montreal, QC, Canada: IEEE, 2010-10, pp. 1–6. ISBN: 978-1-4244-6633-7. DOI: 10.1109/CARPI.2010.5624432. URL: http://ieeexplore.ieee.org/document/5624432/ (cit. on pp. 14, 17, 18).

[39] B. Wang et al. "Power line inspection with a flying robot". In: *2010 1st International Conference on Applied Robotics for the Power Industry (CARPI 2010)*. Montreal, QC, Canada: IEEE, 2010-10, pp. 1–6. ISBN: 978-1-4244-6633-7. DOI: 10.1109/CARPI.2010.5624430. URL: http://ieeexplore.ieee.org/document/5624430/ (cit. on pp. 14, 18).

[40] J. Bian et al. "A monocular vision–based perception approach for unmanned aerial vehicle close proximity transmission tower inspection". en. In: *International Journal of Advanced Robotic Systems* 16.1 (2019-01), p. 172988141882022. ISSN: 1729-8814, 1729-8814. DOI: 10.1177/1729881418820227. URL: http://journals.sagepub.com/doi/10.1177/1729881418820227 (cit. on pp. 14, 15).

[41] HiBot. *Inspection of high-voltage power lines is costly, difficult, and dangerous. It's the perfect job for a robot*. [Online; accessed July 4, 2022]. 2022. URL: https://spectrum.ieee.org/expliner-robot-inspects-high-voltage-lines (cit. on p. 15).

[42] L. Zheng and R. Yi. "Fault diagnosis system for the inspection robot in power transmission lines maintenance". In: ed. by T. Yoshizawa, P. Wei, and J. Zheng. Shanghai, China, 2009-11, 75130E. DOI: 10.1117/12.837984. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.837984 (cit. on p. 16).

[43] G. Yang et al. "Modeling and control of a bi-brachiate inspection robot for power transmission lines". In: *2010 IEEE International Conference on Mechatronics and*

Automation. Xi'an, China: IEEE, 2010-08, pp. 1036–1041. ISBN: 978-1-4244-5140-1. DOI: 10.1109/ICMA.2010.5588110. URL: http://ieeexplore.ieee.org/document/5588110/ (cit. on p. 16).

[44] X. Huang et al. "Design and Application of Insulator Detection Robot System for UHVDC Transmission Line". en. In: *International Journal of Mechanical Engineering and Applications* 7.4 (2019), p. 106. ISSN: 2330-023X. DOI: 10.11648/j.ijmea.20190704.13. URL: http://www.sciencepublishinggroup.com/journal/paperinfo?journalid=220&doi=10.11648/j.ijmea.20190704.13 (cit. on p. 16).

[45] X. Li et al. "Unmanned Aerial Vehicle for Transmission Line Inspection: Status, Standardization, and Perspectives". In: *Frontiers in Energy Research* 9 (2021-07), p. 713634. ISSN: 2296-598X. DOI: 10.3389/fenrg.2021.713634. URL: https://www.frontiersin.org/articles/10.3389/fenrg.2021.713634/full (cit. on pp. 17, 18).

[46] Y. Zhang et al. "Automatic Power Line Inspection Using UAV Images". en. In: *Remote Sensing* 9.8 (2017-08), p. 824. ISSN: 2072-4292. DOI: 10.3390/rs9080824. URL: http://www.mdpi.com/2072-4292/9/8/824 (cit. on pp. 17, 18, 22, 23).

[47] Z. Li et al. "Advances in vegetation management for power line corridor monitoring using aerial remote sensing techniques". In: *2010 1st International Conference on Applied Robotics for the Power Industry (CARPI 2010)*. Montreal, QC, Canada: IEEE, 2010-10, pp. 1–6. ISBN: 978-1-4244-6633-7. DOI: 10.1109/CARPI.2010.5624431. URL: http://ieeexplore.ieee.org/document/5624431/ (cit. on p. 17).

[48] W. Liu et al. "UAV Inspection Path Planning Based on Transmission Line Technology". In: *Journal of Physics: Conference Series* 1648.4 (2020-10), p. 042083. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1648/4/042083. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1648/4/042083 (cit. on pp. 17, 18).

[49] DJI. *M200 SERIES: A NEW TOOL FOR POWER-LINE INSPECTIONS*. [Online; accessed July 7, 2022]. 2022. URL: https://enterprise.dji.com/news/detail/m200-power-line-inspection-tool (cit. on p. 17).

[50] A. Pagnano, M. Höpf, and R. Teti. "A Roadmap for Automated Power Line Inspection. Maintenance and Repair". en. In: *Procedia CIRP* 12 (2013), pp. 234–239. ISSN: 22128271. DOI: 10.1016/j.procir.2013.09.041. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212827113006823 (cit. on pp. 18, 21).

[51] R. M. Haralick. "Computer vision theory: The lack thereof". en. In: *Computer Vision, Graphics, and Image Processing* 36.2-3 (1986-11), pp. 372–386. ISSN: 0734189X. DOI: 10.1016/0734-189X(86)90082-4. URL: https://linkinghub.elsevier.com/retrieve/pii/0734189X86900824 (cit. on p. 20).

[52] V. Wiley and T. Lucas. "Computer Vision and Image Processing: A Paper Review". In: *International Journal of Artificial Intelligence Research* 2.1 (2018-06), p. 22. ISSN: 2579-7298. DOI: 10.29099/ijair.v2i1.42. URL: http://ijair.id/index.php/ijair/article/view/42 (cit. on p. 20).

[53] J. Chai et al. "Deep learning in computer vision: A critical review of emerging techniques and application scenarios". In: *Machine Learning with Applications* 6 (2021-12), p. 100134. DOI: 10.1016/j.mlwa.2021.100134. URL: https://doi.org/10.1016/j.mlwa.2021.100134 (cit. on pp. 20, 37, 39, 41).

[54] Ò. Lorente, I. Riera, and A. Rana. *Image Classification with Classic and Deep Learning Techniques*. 2021. DOI: 10.48550/ARXIV.2105.04895. URL: https://arxiv.org/abs/2105.04895 (cit. on p. 20).

[55] L. Zhu et al. *Weakly Supervised Object Localization as Domain Adaption*. 2022. DOI: 10.48550/ARXIV.2203.01714. URL: https://arxiv.org/abs/2203.01714 (cit. on p. 20).

[56] Z.-Q. Zhao et al. *Object Detection with Deep Learning: A Review*. 2018. DOI: 10.48550/ARXIV.1807.05511. URL: https://arxiv.org/abs/1807.05511 (cit. on pp. 20, 35, 36, 41).

[57] S. Minaee et al. *Image Segmentation Using Deep Learning: A Survey*. 2020. DOI: 10.48550/ARXIV.2001.05566. URL: https://arxiv.org/abs/2001.05566 (cit. on p. 20).

[58] R. Aslanzadeh, K. Qazanfari, and M. Rahmati. *An Efficient Evolutionary Based Method For Image Segmentation*. 2017. DOI: 10.48550/ARXIV.1709.04393. URL: https://arxiv.org/abs/1709.04393 (cit. on p. 20).

[59] F. Miralles, N. Pouliot, and S. Montambault. "State-of-the-art review of computer vision for the management of power transmission lines". In: *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*. Foz do Iguassu, Brazil: IEEE, 2014-10, pp. 1–6. ISBN: 978-1-4799-6422-2 978-1-4799-6420-8. DOI: 10.1109/CARPI.2014.7030068. URL: http://ieeexplore.ieee.org/document/7030068/ (cit. on pp. 21, 22).

[60] G. Yan et al. "Automatic Extraction of Power Lines From Aerial Images". In: *IEEE Geoscience and Remote Sensing Letters* 4.3 (2007-07), pp. 387–391. ISSN: 1545-598X. DOI: 10.1109/LGRS.2007.895714. URL: http://ieeexplore.ieee.org/document/4271471/ (cit. on p. 23).

[61] P. Jiang et al. "A Review of Yolo Algorithm Developments". In: *Procedia Computer Science* 199 (2022). The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 2021): Developing Global Digital Economy after COVID-19, pp. 1066–1073. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.01.135. URL: https://www.sciencedirect.com/science/article/pii/S1877050922001363 (cit. on pp. 26, 32, 33).

[62] J. Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640 (cit. on pp. 26–30, 32).

[63] S. Gothane. "A Practice for Object Detection Using YOLO Algorithm". In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (2021-04), pp. 268–272. DOI: 10.32628/CSEIT217249 (cit. on p. 26).

[64] M. Karthi et al. "Evolution of YOLO-V5 Algorithm for Object Detection: Automated Detection of Library Books and Performace validation of Dataset". In: *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*. 2021, pp. 1–6. DOI: 10.1109/ICSES52305.2021.9633834 (cit. on p. 26).

[65] T. Diwan, G. Anirudh, and J. V. Tembhurne. "Object detection using YOLO: challenges, architectural successors, datasets and applications". In: *Multimedia Tools and Applications* (2022-08). DOI: 10.1007/s11042-022-13644-y. URL: https://doi.org/10.1007/s11042-022-13644-y (cit. on pp. 26–30, 32, 35, 37, 39, 41).

[66] U. Handalage and L. Kuganandamurthy. "Real-Time Object Detection Using YOLO: A Review". In: (2021-05). DOI: 10.13140/RG.2.2.24367.66723 (cit. on pp. 26, 27, 32).

[67] Z. Zou et al. *Object Detection in 20 Years: A Survey*. 2019. DOI: 10.48550/ARXIV.1905.05055. URL: https://arxiv.org/abs/1905.05055 (cit. on pp. 27, 35).

[68] D. Thuan. "EVOLUTION OF YOLO ALGORITHM AND YOLOV5: THE STATE-OF-THE-ART OBJECT DETECTION ALGORITHM". In: 2021 (cit. on pp. 27–29, 32–34).

[69] K. Bagla, A. Diwan, and K. Agarwal. "DarthYOLO: Using YOLO for Real-Time Image Segmentation". In: 2022-11. ISBN: 9781643683362. DOI: 10.3233/ATDE220794 (cit. on p. 32).

[70] B. Benjdira et al. "Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3". In: (2018). DOI: 10.48550/ARXIV.1812.10968. URL: https://arxiv.org/abs/1812.10968 (cit. on pp. 32, 33).

[71] J. Redmon and A. Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/ARXIV.1612.08242. URL: https://arxiv.org/abs/1612.08242 (cit. on p. 32).

[72] J. Redmon and A. Farhadi. *YOLOv3: An Incremental Improvement*. 2018. DOI: 10.48550/ARXIV.1804.02767. URL: https://arxiv.org/abs/1804.02767 (cit. on p. 33).

[73] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Hong. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *arXiv preprint arXiv:2004.10934* (2020) (cit. on p. 33).

[74] C. Li et al. *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. 2022. DOI: 10.48550/ARXIV.2209.02976. URL: https://arxiv.org/abs/2209.02976 (cit. on p. 34).

[75] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. DOI: 10.48550/ARXIV.2207.02696. URL: https://arxiv.org/abs/2207.02696 (cit. on p. 34).

[76] R. Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. DOI: 10.48550/ARXIV.1311.2524. URL: https://arxiv.org/abs/1311.2524 (cit. on pp. 35, 36).

[77] J. Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (2013-09), pp. 154–171. DOI: 10.1007/s11263-013-0620-5 (cit. on p. 36).

[78] R. Girshick. *Fast R-CNN*. 2015. DOI: 10.48550/ARXIV.1504.08083. URL: https://arxiv.org/abs/1504.08083 (cit. on pp. 36, 37).

[79] C. Eggert et al. "A closer look: Small object detection in faster R-CNN". In: 2017-07, pp. 421–426. DOI: 10.1109/ICME.2017.8019550 (cit. on pp. 36, 38–40).

[80] O. Hmidani and E. M. Ismaili Alaoui. "A comprehensive survey of the R-CNN family for object detection". In: *2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet)*. 2022, pp. 1–6. DOI: 10.1109/CommNet56067.2022.9993862 (cit. on pp. 36–38, 40).

[81] S. Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015) (cit. on pp. 38–40).

[82] C. C. Nguyen et al. "Towards Real-Time Smile Detection Based on Faster Region Convolutional Neural Network". In: *2018 1st International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*. 2018, pp. 1–6. DOI: 10.1109/MAPR.2018.8337524 (cit. on p. 40).

[83] R. Padilla et al. "A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit". In: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279 (cit. on pp. 42–44, 46–50).

[84] I. S. Isa et al. "Optimizing the Hyperparameter Tuning of YOLOv5 for Underwater Detection". In: *IEEE Access* 10 (2022), pp. 52818–52831. DOI: 10.1109/ACCESS.2022.3174583 (cit. on pp. 44, 46, 48, 50).

[85] H.-K. Jung and G.-S. Choi. "Improved YOLOv5: Efficient Object Detection Using Drone Images under Various Conditions". In: *Applied Sciences* 12.14 (2022). ISSN: 2076-3417. DOI: 10.3390/app12147255. URL: https://www.mdpi.com/2076-3417/12/14/7255 (cit. on pp. 44, 46).

[86] G. Jocher. *Tips for Best Training Results*. URL: https://docs.ultralytics.com/yolov5/tips_for_best_training_results/#dataset (cit. on p. 55).

[87] A. Mumuni and F. Mumuni. "Data augmentation: A comprehensive survey of modern approaches". In: *Array* 16 (2022), p. 100258. ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2022.100258. URL: https://www.sciencedirect.com/science/article/pii/S2590005622000911 (cit. on pp. 59, 63).

[88] C. Shorten and T. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (2019-07). DOI: 10.1186/s40537-019-0197-0 (cit. on pp. 63, 68).

[89] S. A. H. Mohsan et al. "Towards the Unmanned Aerial Vehicles (UAVs): A Comprehensive Review". In: *Drones* 6.6 (2022). ISSN: 2504-446X. DOI: 10.3390/drones6060147. URL: https://www.mdpi.com/2504-446X/6/6/147 (cit. on p. 82).