POLITECNICO DI TORINO

Master degree in Communications and Computer Networks Engineering

Master Degree Thesis

A drone system for assisted radio site diagnostic

Supervisors:
Prof. Roberto Garello (PoliTo)
Prof. Luca Reggiani (PoliMi)
Prof. Gerald Q. Maguire Jr. (KTH)

Candidate:
Filippo Vannella

December 2018

Abstract

Within the context of radio site maintenance, there is a requirement to perform periodical inspections and troubleshooting of radio sites in order to ensure the correct functioning of the site equipment and of the systems depending on it. The main problems related to traditional base stations' inspections are related to the high costs and risks involved in the operation. This is principally because the radio site's equipment is usually positioned at high altitude and this makes the manned inspection process particularly complex and dangerous. Also, the high level of heterogeneity and complexity of the cell tower topology and lack of knowledge about the inspected radio site equipment can make the inspection task difficult and require multiple visits in order to solve the problems that are found.

This thesis will design, develop, and evaluate a system where an Unmanned Aerial Vehicle (UAV) assists field engineers performing the inspection of radio sites by detecting and tracking base-station equipment in real-time while collecting aerial data about the hardware status. The solution relies on the integration of a state-of-the-art Convolutional Neural Network (CNN) object detection algorithm, providing high detection accuracy (in terms of mean Average Precision (mAP)), with an object tracker running at high Frames Per Second (FPS). The implemented algorithm is computationally lightweight and suited for power-constrained computing platforms.

Ideally, this solution works independently of the radio site architecture, by automatically detecting and tracking the base station hardware component to inspect. This system will help operators cope with the large architectural and structural variability of the radio sites and provides a further step towards the autonomization of base station inspection practice.

Keywords: Radio site inspection, cell tower inspection, telecommunications tower inspection, UAV, Unmanned Aircaft System (UAS), drone, computer vision, CNN, object detection, object tracking.

Sommario

Nell'ambito della manutenzione di base stations, si presenta la necessità di effettuare ispezioni periodiche per assicurare il corretto funzionamento delle apparecchiature hardware e dei sistemi che da esso dipendono. I principali problemi legati alla tradizionale manutenzione delle torri di telecomunicazioni sono relativi agli alti costi e rishi che l'operazione comporta. Ciò è dovuto principalmente al fatto che l'equipaggiamento delle base stations è solitamente posizionato ad altezze elevate, e questo rende l'ispezione fisica particolarmente complessa e rischiosa. Inoltre, a rendere complicata l'ispezione può concorrere l'alto grado di eterogeneità strutturale delle torri di telecomunicazioni, nonché la scarsa conoscenza relativa al loro equipaggiamento; in questo caso, infatti, potrebbero rendersi necessarie diverse visite, prima di risolvere i problemi rilevati.

L'obiettivo del presente lavoro è quello di progettare, sviluppare e valutare un sistema UAV, che possa coadiuvare il lavoro di ispezione di base stations di tecnici e ingegneri specializzati; un sistema che, nello specifico, sia in grado di individuare (object detection) e tracciare (object tracking), in tempo reale, apparecchiature installate nella base station, durante la raccolta di dati relativi allo stato dell'equipaggiamento hardware.

Il sistema è basato sull'integrazione di un algorithmo di object detection che utilizza reti neurali convoluzionali (CNN) con un algoritmo di object tracking. L'algoritmo implementato è leggero dal punto di vista computazionale (in termini di Frames Per Second (FPS) ed ha un elevata accuratezza (in termini di mean Average Precision (mAP)). Per questi motivi, l'algoritmo può essere eseguito su dispositivi con una potenza di calcolo limitata, come ad esempio uno smartphone.

Il sistema ideato è in grado di funzionare indipendetemente dalla architettura della base station, inidividuando e tracciando le componenti hardware da ispezionare. Questo sistema può essere di aiuto agli operatori di telecomunicazioni per gestire la variabilità strutturale nei siti radio durante le ispezioni. Il suddetto sistema, inoltre, rappresenta un ulteriore passo verso l'automazione delle pratiche di ispezione di base stations.

Keywords: Radio site inspection, cell tower inspection, telecommunications tower inspection, UAV, UAS, drone, computer vision, CNN, object detection, object tracking.

Contents

List of Figures

List of Tables

Acronyms

ACF	Aggregated Channel Features
AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
BLOS	Beyond Line Of Sight
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network library
CV	Computer Vision
DoD	Department of Defense
EASA	European Aviation Safety Agency
FC	Fully Connected
FPS	Frames Per Second
FSO	Field Service Operations
GCC	GNU Compiler Collection
GCS	Ground Control Station
GPU	Graphical Processing Unit
GUI	Graphical User Interface
ICAO	International Civil Aviation Organization
IDE	Integrated Development Environment
IoU	Intersection Over Union
KCF	Kernelized Correlation Filters
KLT	Kanade-Lucas-Tomasi
LOS	Line of Sight
MAdd	Multiply-Add
mAP	mean Average Precision

MAV	Micro Aerial Vehicle
ML	Machine Learning
MP	Megapixel
NDK	Native Development Kit
OS	Operating System
PPV	positive predictive value
RAM	Random Access Memory
RC	Radio Controller
ReLU	Rectified Linear Unit function
RF	Radio Frequency
RPAS	Remotely Piloted Aererial System
RRU	Remote Radio Unit
RSA	Radio Site Assistant
SDK	Software Development Kit
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SSD	Single Shot Detector
SSD	Sum of Squared Differences
TF	TensorFlow
TFL	TensorFlow Lite
TFM	TensorFlow Mobile
TPR	True Positive Rate
UAS	Unmanned Aircaft System
UAV	Unmanned Aerial Vehicle
VOC	Visual Objects Classes
WRC	World Radiocommunication Conference
YOLO	You Only Look Once

Acknowledgements

This thesis work took place as part of an Erasmus+ programme carried out at Ericsson Research and KTH Royal Institute of Technology in Stockholm, Sweden. Also this thesis is set in the context of the Alta Scuola Politecnica Double Degree programme between Politecnico di Torino and Politecnico di Milano. There are a number of people that helped and supported me during my work on this thesis and I would like to acknowledge.

First of all, I would like to thank my supervisor at Politecnico di Torino Prof. Roberto Garello for his great support and understanding during my thesis work and for his valuable teaching during these years at Politecnico. Also, I want to thank Prof. Andrea Carena at Politecnico di Torino for his support during my exchange programme. I want to thank Prof. Luca Reggiani at Politecnico di Milano for his precious feedbacks and suggestions.

I thank my supervisors at Ericsson Athanasios Karapantelakis and Maxim Teslenko for the support and their precious experience. Also, I appreciate the support received from Volodya Grancharov and his research team at Ericsson. I thank my examiner at KTH Prof. Gerald Q. Maguire Jr. for his feedbacks on the thesis.

Finally, a special thanks to my family and friends for their love and support during the work of the thesis.

Chapter 1

Introduction

This chapter presents general concepts concerning the area of interest in which the thesis project is framed, defines the purpose and objectives of the project, formulates the problem and the research questions, and discusses contributions and limitations.

1.1 Overview

The traditional manned inspection practice for radio sites involves skilled technicians visiting the site and climbing the cellular tower to reach the hardware, typically located at a great height. This inspection methodology may require multiple visits to the site before solving all of the issues, especially when there is the need for additional tools or equipment to fix the problem.

During recent years the drone sector has experienced astonishing growth and this trend is expected to continue into the future [15], despite the restrictions of international drone regulations [16]. There exists a huge variety of case-studies involving Unmanned Aerial Vehicle[†] (UAV) technology being applied in different applications. The sector on which this work focuses is visual-based infrastructure inspection. A study conducted by Goldman Sachs [17] predicts that within the period 2016-2020, US $1.1 billions will be spent on UAV systems for infrastructure inspection and that the commercial segment would be the fastest-growing market segment within this application area.

There are many benefits related to employing the UAV technology for radio site inspection. Firstly, drones can collect imagery in dangerous places, hence they can substantially improve safety, and this represents a major issue affecting manned cell tower inspections. In fact, according to a recent study by Ryan Knutson and Liz Day [18], between 2003 and 2011 only in the United States, 50 climbers died working on cell sites, and in more recent years the statistics concerning cell-towers fatalities do not seem to have improved.

[†]In this thesis, the term drone is used interchangeably with UAV.

The statistics concerning the number of tower fatalities in the U.S. from 2003 to 2017 are shown in Figure 1.1.

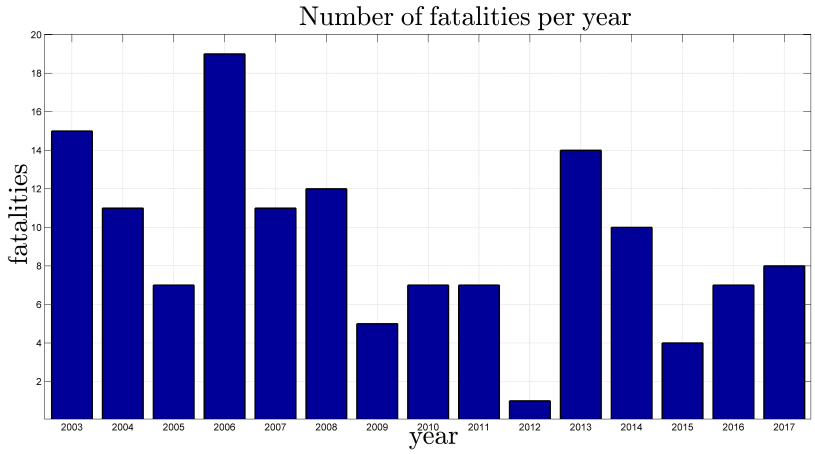

Figure 1.1: Tower fatalities by year - data from [1]

The use of UAVs for inspecting a radio site also offers a large cost reduction in the maintenance of the site. J. Karpowicz has estimated that the cost for a manned inspection can be reduced by up to 50% by taking advantage of unmanned inspection technology [19]. This cost reduction could lead to an increased frequency of inspection of each site, allowing a better understanding of the conditions of the structures and systems present at the radio site. Additionally, UAV systems can collect data in a much more effective way than an operator could ever do since nowadays there exist UAV systems which can acquire high-resolution multimedia content and process them in real-time thanks to the recent advance in Artificial Intelligence (AI) and increased computational capabilities of computing hardware platforms. These beneficial factors led to the adoption of the UAV inspection technology by large telecommunications companies. For example, the world's largest telecommunications company AT&T [20], is employing drone technology to inspect its 65,000 cell towers using AI to automate the radio site inspection [21].

However, there are many problems that field engineers face during the cell tower inspections. Currently, UAV systems can only acquire data that might be used to detect a fault, but fixing the fault requires human presence at the site since the technology for remote repair is not yet mature. Mazin Gilbert, vice president of the advanced technology at AT&T labs, affirmed: "The ultimate goal is full automation, but we are not there yet. With regulatory changes and further research, we hope that automated inspections will be possible" [21]. The development of technologies aiming at assisting and automating the base-station inspection practice is very valuable since employing a drone system saves a lot of resources and reduces dangers related to the inspection when compared to manned inspections.

The prototype developed during this thesis project aims to assist radio site engineers during the unmanned inspection by employing real-time object detection and object tracking algorithms. The knowledge about which radio site equipment to inspect is of paramount importance for the efficacy of each inspection. The distance between the physical operator represents a challenging obstacle to the identification and correct classification of the equipment. Object detection techniques can overcome this obstacle by directing the control of the physical operator towards the correct hardware. Also, identifying the equipment to inspect is essential for the development of future autonomous UAV radio site systems able to carry out an autonomous inspection mission and for this reason it can be considered an important milestone towards the autonomization of the system.

1.2 Purpose

The main purpose of this work is to reduce the high costs and risks for the current base-station inspection practice by facilitating unmanned inspections by employing an assisted drone system controlled by technicians on the ground. Also, the potential to develop a novel inspection system for the detection of hardware equipment in radio towers and the prospect of finding adopters in the industrial base-station inspection's sector are additional motivations for this project. Finally, the possibility to expand the object detection system in the future for control tasks based upon the object detection algorithm is also considered a good reason to implement the proposed system.

1.3 Problem statement

Although unmanned inspection is becoming more popular because of the benefits it provides, there exist different problems that field service engineers still face during UAV inspection activity. One of the main challenges is represented by the cell tower heterogeneity (see Figure 1.2) at different levels:

- **Architecture heterogeneity** concerns the physical structure of the base-station and the different positions in which the equipment is mounted.
- **Equipment heterogeneity** relates to the presence of different components on the radio site such as Remote Radio Unit (RRU), antennas, feeder cables, etc.
- **Model heterogeneity** involves the presence of different models and operators' brands of the same hardware equipment within the cellular tower.

This high level of diversity usually hinders the unmanned inspection task as the radio site technicians do not have sufficient information on which radio equipment to inspect.

Figure 1.2: Examples of base-station heterogeneity. Images credits: public domain work by Wikimedia Commons users [2–5]

Also, the pilots carrying out unmanned visual inspections have limited time, are constrained by different factors such as the UAV battery life or the number of radio tower to be inspected in a limited amount of time, and need to be aware of the equipment to be inspected in order to efficiently collect efficiently useful visual inspection data.

1.4 Research question

The principal research question that this thesis addresses is:

Which technologies can be used to implement an UAS assisting field engineers to detect radio site hardware equipment during unmanned inspections?

This main question can be divided into three sub-questions that will be addressed within this thesis and will lead to the implementation of a prototype UAV system. These sub-questions are:

1. Which capabilities and characteristics should the system have to assist field engineers to effectively inspect the radio site equipment?

2. Which unmanned hardware platform is best suited to provide these capabilities?

3. Which software platform and algorithms can be used to provide the needed object detection and tracking functionalities and how to effectively integrate this with the hardware platform?

1.5 Limitations

This thesis involves the development of an autonomous UAV cell tower inspection system. This ultimate goal goes beyond the scope of this project and the technology to develop a fully-autonomous UAV system is yet immature. For these reasons, this work limits its scope, focusing mainly on the design, development and evaluation of a drone vision algorithm for the detection and tracking of radio site equipment. In particular, it applies a cutting-edge object detection algorithm based on CNNs combined with an object tracking algorithm to the field of base-station hardware inspection. This result represents a milestone and a novelty in the field of base-station inspection and constitutes a further development towards fully autonomous of the radio site inspection.

1.6 Contributions

The contributions of this thesis project are threefold:

1. It explores technologies and UAS platforms suitable for base-station unmanned inspections by reviewing scientific literature in the area and similar applications.
2. It proposes a real-time architecture for detecting and tracking base-station hardware integrating it with a low cost UAV platform on a power-constrained platform.
3. It develops an Android application to realize the proposed architecture and test it.

1.7 Structure of the Thesis

- **Chapter 2** outlines the main theoretical concepts involved in this project and provides the reader with a general understanding of the framework within which this thesis is developed. It also reviews the current literature in UAV inspection systems and explores the employed object detection and tracking technology.
- **Chapter 3** presents the overall system architecture and the methodology used to design, implement, and test the proposed system.
- **Chapter 4** discusses the realization of the architecture of the implemented prototype and analyzes its constitutive part. It also describes implementation technicalities about the algorithms and its integration with the UAV system.
- **Chapter 5** performs an analysis of the collected experimental data and presents them.
- **Chapter 6** draws conclusions on the basis of the results presented in Chapter 5 and suggests future directions to explore based upon the results of this project and describes limitations.

Chapter 2

Background and related works

This chapter establishes the theoretical framework used in this thesis project by presenting the key concepts on which the proposed system will be developed. Also, it reviews the current literature in the area and presents the state of the art of similar UAV inspection systems employing object detection and tracking that are relevant for this thesis.

2.1 Unmanned Aircraft System

UAS, also referred to as Remotely Piloted Aererial System (RPAS), is a term coined in 2005 by the U.S. Department of Defense (DoD) [22] and consists of three components [23], namely:

1. A **UAV** defined as "a pilotless aircraft [...] which is flown without a pilot-in-command on-board and is either remotely and fully controlled from another place (ground, another aircraft, space) or programmed and fully autonomous" [24].

2. A **Ground Control Station (GCS)**, that is an autonomous or human-operated, stationary or transportable hardware/software system to monitor and command the unmanned aircraft.

3. A **communication system** that links the UAV to the GCS.

2.1.1 UAS classification

UASs can be categorized in a variety of ways based on different attributes. A detailed UAS classification can be found in [25]. Since the level of detail of such classification goes beyond the scope of this thesis, it will not be reported in the following and only the most relevant classifications useful for this project will be reported in the following.

2.1.1.1 Aircraft type

Based on the aircraft structure, a UAV has a twofold classification:

- **Rotary wing** UAVs are made of a central body and multiple rotors that power propellers to control the UAV flight. The number of rotors may vary from one, two, three, and four (quadcopter) to six (hexacopter), eight (octocopter) or even more. However, the most common category is the quad-rotor aircraft. The control of the aircraft's motion is realized by varying propeller's blade angular speed.
- **Fixed wing** UAVs consist of platforms with a rigid wing that makes flight capable by generating lift generated by the UAV's forward airspeed generated by a forward thrust.

Figure 2.1: Left: example of rotary wing UAV. Right: example of fixed wing UAV. Images credits: public domain work by Flickr Users [6, 7]

2.1.1.2 Aircraft size

A possible classification based on the aircraft's size is proposed by Fahlstrom and Gleason [26], in which UAVs are grouped into four categories, namely:

- **very small UAV** having dimensions in the order of 30-50 cm,
- **small UAV** having dimensions up to 2 m,
- **medium UAV** having dimensions up to 10 m, and
- **large UAV** having dimensions larger than 10 m.

2.1.1.3 System autonomy

The classification of UAS autonomy takes into account the level of human involvement. Dalamagkidis [27] proposes the following classification:

- **Remotely piloted** UAS (non-autonomous) that are remotely controlled by a pilot either within Line of Sight (LOS) or with feedback from the UAV sensors such as a camera.
- **Remotely operated** UAS (semi-autonomous) in which high-level commands are given to the system (waypoints, objects to track, etc.), and the UAS performance is monitored by an operator in the field. In this case, the flying is performed by the UAS itself, but all the decision-making is delegated to a human operator.
- **Fully autonomous** UAS in which general tasks are given to the aircraft and the system is capable of determining how to accomplish them, even at the face of unforeseen events. Since human presence the site is not strictly required for this class of UAVs, they can operate in Beyond Line Of Sight (BLOS) mode.

Multirotor platform is by far the most diffused type of UAV platforms for commercial applications. For example, a study conducted by DroneDeploy [28] shows that 97% of the commercial drone sector for mapping is occupied by multirotor platforms, while fixed wing UAVs have only the 3% market share for commercial UAV mapping applications. Also, concerning applications similar to infrastructure inspection and maintenance, rotor crafts are more suitable due to their hover capabilities and agile maneuvering, while fixed wing vehicles better fit aerial surveillance and mapping of large areas from greater heights [29]. Usual sizes for UAS infrastructure inspection applications fall in the categories of very small and small UAVs [30–32]. This is mainly due to their ability to fly in hard-to-reach environments and their ease of use and transportation. Different levels of autonomy have been experimented within UAS for inspection applications but for the application concerned in this thesis, the focus is on remotely piloted and remotely operated UAS.

2.1.2 UAS regulation

There exist different international regulations, such as the ones issued by the International Civil Aviation Organization (ICAO) [24] and the European Aviation Safety Agency (EASA) [33]. The use of UAVs is generally more strictly regulated by the national aviation authority of each country who supersede these international directives via national laws. For this reason, in the following text, the focus is on national drone regulation in Sweden, as this is the state in which this project was performed. On February 1, 2018, the Swedish Transport Agency (Swedish: Transportstyrelsen) issued the new regulation TSFS 2017:110 [34] on unmanned aircraft.

All drones weighing less than 150 kg are covered by these rules with no differentiation between private and commercial flight. Other regulations applying to UAVs in Sweden are: the European Union's Regulation EU 923/2012, the Swedish Transport Agency's Regulation TSFS 2014:71, the general guidelines for aviation TSFS 2014:94, along with the Swedish Aviation Act 2010:500, and Aviation Ordinance 2010:770. The most important behaviours to be observed according to the above rules when flying a UAV are:

(i) The UAV shall be marked with the operator's name and telephone number.

(ii) The UAV's maximum flying altitude must be below 120 m from ground in uncontrolled air.

(iii) License and permissions are needed for drones weighing more than 7 kg and flying in BLOS conditions.

(iv) Fly at least 5 kilometers from any airport runway. It is still possible to fly within the control zone of the airport without any special permission provided that the flight altitude is below 50 meters, the speed lower than 90 km/h.

For UAVs weighting more than 150 kg, other regulations are in place but they will not be given here as are outside the scope of this thesis project. Regarding the regulations for the communications bands assigned for UAV applications, in 2012 at the World Radiocommunication Conference (WRC) the 5030-5091 MHz band was allocated for UAV Radio Frequency (RF) communication systems. Also, many UAS in the commercial category uses Wi-Fi technology for communication (typically in bands around 2.4 GHz).

2.2 UAS vision

UAS inherently require sensing the surrounding environment. Among the different possible types of sensors, cameras represent one of the cheapest and most effective data sources. For this reason Computer Vision* (CV) plays a central role in UAS automation and the literature is rich with methods, algorithms, and use-cases concerning UAS visual inspection applications.

The concept of *feature* is a key element in CV. Features are representative points with certain invariance characteristics such as invariance to rotation, resizing, change in brightness, viewpoints, and other image transformations. Based on this concept, computer vision techniques can be classified in two macro-categories, namely:

*Computer Vision is a multidisciplinary field that studies and reproduces computational models of the human visual system and build autonomous systems to perform tasks which the human visual system can perform [35]

1. **Hand-crafted features algorithms** that are based on the extraction of a set of features from the image, apply a series of image-processing operations performed on the extracted feature image [36] and represents the traditional approach to computer vision developed since the 1960s.

2. **Machine learnt features algorithms**, based on the use of Machine Learning (ML) techniques and CNNs, process the image through different layers of neurons that are able to extract features from the image. This type of approaches have been developing since 2012 when the first CNN algorithm for object detection, AlexNet, [37] outperformed all hand-crafted feature methods for object detection developed until then.

The following presents the major contributions and describes CV algorithms useful for this project. In particular, the discussion will focus on hand-crafted feature approaches for *object tracking*, while focusing on the machine learnt feature approach for *object detection*. The former is a CV function that deals with detecting instances of objects of certain classes (e.g. people, towers, cars, etc). The latter detects motion of salient points, thus allowing tracking of the temporal object movement in consecutive video frames. These two tasks constitute a crucial function for UAS-based visual infrastructure inspection applications. Moreover, these tasks are particularly challenging due to drone motion and consequent noise and blurring in the image. Also, another problem currently facing UAS inspection operations is performing the detection and tracking tasks in real-time in order to receive an immediate feedback or utilize visual control algorithms. This is especially challenging on embedded hardware platforms with constrained computational capabilities such as smartphones.

2.2.1 Hand-crafted feature algorithms

The following presents a theoretical introduction on the techniques to implement the object tracking technique used within this project. There exist two main classes of feature tracking methods based on handcrafted feature: the first approach uses the KLT tracker [38] which uses spatial intensity information to search for a feature match. The second approach involves the use of descriptors, such as Scale Invariant Feature Transform (SIFT) [39], for features extracted from subsequent frames in the video sequence and then the matching of these representative points. KLT is the preferred method when computational resources are limited since extracting and matching feature descriptors is significantly more computationally expensive. Since the application developed for this thesis project involves the development of the algorithm in an embedded device context the background knowledge to describe the KLT tracking algorithm will be presented, while the complete algorithm will be presented in Section 3.7 together with the choice of the object detection algorithm.

2.2.1.1 Harris corner detector

The Harris corner detector is a particular type of *feature detector* algorithm introduced by Chris Harris and Mike Stephens in 1988 [40]. Feature detection is the image-processing operation that refers to detecting a set of representative features inside the image. The most representative features of an image are *edges* and *corners* because they delineate the boundaries of objects and have good invariance characteristics. The primary origin of *edges* is the presence of a discontinuity, that is an abrupt change in the pixel intensity in the image data. *Corners* are constituted by the intersection of edges and have discontinuities in two orthogonal directions.

Let $I(x, y)$ be a grayscale image, $w(x, y)$ a 2-D window function that selects a patch of pixels, and a displacement vector (u, v). The basic idea behind the Harris corner detector is that corner points have strong differences in appearance when compared to all regions in its neighborhood (see Figure 2.2).

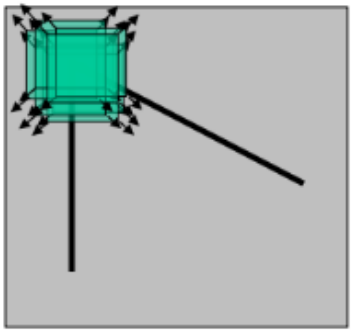

Figure 2.2: Multiple patch windows on a corner point. Image credits Aseel M. [8]

For this reason, the Harris detector imposes the maximization of the Sum of Squared Differences (SSD) between the pixels' intensity of the image and the replica of the image displaced by (u, v) in all directions:

$$E(u, v) = \sum_{(x,y)} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{2.1}$$

This is because the Harris detector looks for points that have big changes in all directions. By applying Taylor series expansion, and stopping at the first order term, it is possible to write:

$$\begin{aligned} E(u, v) &\approx \sum_{x,y} w(x, y)(u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2) = \\ &(u, v) \left(\sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{pmatrix} u \\ v \end{pmatrix} = \\ &(u, v)M \begin{pmatrix} u \\ v \end{pmatrix} \end{aligned} \tag{2.2}$$

where I_x and I_y denote the partial derivatives of the Image pixel intensity with respect to the x and y directions. The Matrix M is a structure tensor and characterizes the pixels within the windows. It can be proven that the information of the presence of the corner is contained in the eigenvalues (λ_1, λ_2) of M [41]:

- if $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, then this pixel (x, y) has no features of interest.
- if $\lambda_1 \approx 0$ and λ_2 has some large positive value, then an edge is found.
- if λ_1 and λ_2 have large positive values, then a corner is detected.

In order to avoid computing the eigenvalues which are computationally expensive, a score R is computed and used to determine the presence of the corner:

$$R = det(M) - k(trace(M))^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{2.3}$$

where $k \in [0.04, 0.06]$ is a constant determined empirically. J. Shi and C. Tomasi, in their paper "*Good Features to Track*" [42], proposed a new score function which exhibits better results

$$R = min(\lambda_1, \lambda_2) \tag{2.4}$$

In the end, the value of R is compared to a threshold λ_{min} which represents the minimum value of R in order to be considered as a corner.

2.2.1.2 Optical flow

Optical flow is defined as the relative motion between a camera and the scene between consecutive frames in a video [43]. Optical flow outputs a set of vectors representing the motion of a point from the first frame to the second. Assumptions have to be made in order to ensure good performances of optical flow algorithms, these are [44]:

1. Brightness and feature constancy: only the movement of the object with respect to the camera can cause local changes in image intensity
2. Spatial smoothness: the motion is uniform over a small neighborhood of pixels
3. Small motion: the sampling frequency of the video is fast enough to represent the image motion gradually over time

By denoting the intensity of the (x, y) pixel of the t^{th} frame in grayscale with $f(x, y, t)$, it is possible to express the first hypothesis as

$$f(x, y, t) = f(x + dx, y + dy, t + dt) \tag{2.5}$$

assuming that in the next dt time, the pixel moves by (dx, dy) the intensity remains constant in consecutive frames. Expanding the right-hand side of the equation using a Taylor series, it is possible to write (to the first order)

$$f(x, y, t) = f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt \tag{2.6}$$

Dividing both sides of the equation by dt it is possible to write the *Optical Flow equation*:

$$f_x u + f_y v + f_t = 0 \tag{2.7}$$

Unfortunately this equation has two unknowns (u, v) and this problem is underdetermined. One of the most widely used methods to solve this equation is presented in the upcoming section.

2.2.1.3 Lucas-Kanade Method

The Lucas-Kanade method overcomes the problem of the optical equation being underdetermined by assuming that pixels in a 3×3 neighborhood have the same optical flow. In this way, it is possible to expand the optical flow equation (Equation 2.7) to all these pixels:

$$\begin{aligned} f_{x1}u + f_{y1}v + f_{t1} &= 0 \\ &\vdots \\ f_{x9}u + f_{y9}v + f_{t9} &= 0 \end{aligned} \tag{2.8}$$

The previous equation can be rewritten in matricial form as

$$\begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ f_{x9} & f_{y9} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -f_{t1} \\ \vdots \\ -f_{t9} \end{bmatrix} \iff A \cdot u = b \tag{2.9}$$

A is a 9×2 matrix, u is a 2×1 column vector, and b is a 9×1 column vector. Ideally, to solve the system, it is sufficient to multiply by A^{-1} on both sides of the equation. However, since A is not a square matrix, the pseudo-inverse of A is considered:

$$(A^T A)u = A^T b \tag{2.10}$$

Since $(A^T A)$ is a square matrix, it is possible to take its inverse:

$$u = (A^T A)^{-1} A^T b \iff \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{xi}^2 & \sum_i f_{xi} f_{yi} \\ \sum_i f_{xi} f_{yi} & \sum_i f_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{xi} f_{ti} \\ -\sum_i f_{yi} f_{ti} \end{bmatrix} \tag{2.11}$$

This can be simplified by taking the inverse as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\sum_i f_{xi}^2 \sum_i f_{yi}^2 - (\sum_i f_{xi} f_{yi})^2} \begin{bmatrix} \sum_i f_{yi}^2 & \sum_i f_{xi} f_{yi} \\ \sum_i f_{xi} f_{yi} & \sum_i f_{xi}^2 \end{bmatrix} \begin{bmatrix} -\sum_i f_{xi} f_{ti} \\ -\sum_i f_{yi} f_{ti} \end{bmatrix} \tag{2.12}$$

Solving for u and v:

$$\begin{aligned} u &= \frac{\sum_i f_{xi} f_{yi} \sum_i f_{yi} f_{ti} - \sum_i f_{yi}^2 \sum_i f_{xi} f_{ti}}{\sum_i f_{xi}^2 \sum_i f_{yi}^2 - (\sum_i f_{xi} f_{yi})^2} \\ v &= \frac{\sum_i f_{xi} f_{yi} \sum_i f_{xi} f_{ti} - \sum_i f_{xi}^2 \sum_i f_{yi} f_{ti}}{\sum_i f_{xi}^2 \sum_i f_{yi}^2 - (\sum_i f_{xi} f_{yi})^2} \end{aligned} \tag{2.13}$$

The values for all the f_{xi}, f_{yi}, and f_{ti} are known at this point and it is possible to find the values of u and v for each pixel.

2.2.2 Machine learnt features algorithm

This section introduces concepts used to implement the object detection algorithm used within this thesis project. In particular the output of the object detection consists in the classification and localization of the object to be recognized, resulting in multiple bounding boxes around the detected objects. Recently, increased computational capabilities and developments in AI have allowed the development of new approaches based on CNNs. Today CNN approaches represent the state of the art methods in object detection models and outperforms the hand-crafted object detection techniques by enhancing the accuracy of image classification and extracting image features in a more efficient and complete way [45–48]. However, CNNs are very computationally expensive and require a lot of data in the training phase with respect to traditional feature-based approaches and there is the need to adopt appropriate techniques to overcome such limitations. In this section, CNNs are described in the context of this thesis following an introductory section on ANN, on which CNNs are based.

2.2.2.1 Artificial Neural Networks

An ANN is a computational processing system inspired by the way biological nervous systems operate. The fundamental elements in ANNs are interconnected artificial neurons and connections between them. These units are organized in a multi-layered structure with D input units $(x_1, ..., x_D)$, C output units $(y_1^{(L+1)}, ..., y_C^{(L+1)})$, and L hidden layers (Figure 2.3). Each artificial neuron has connections with the previous and the next layer and $w_{i,k}^{(l)}$ is the weight of the connection between k^{th} neuron at layer $(l-1)$ and i^{th}neuron at layer (l).

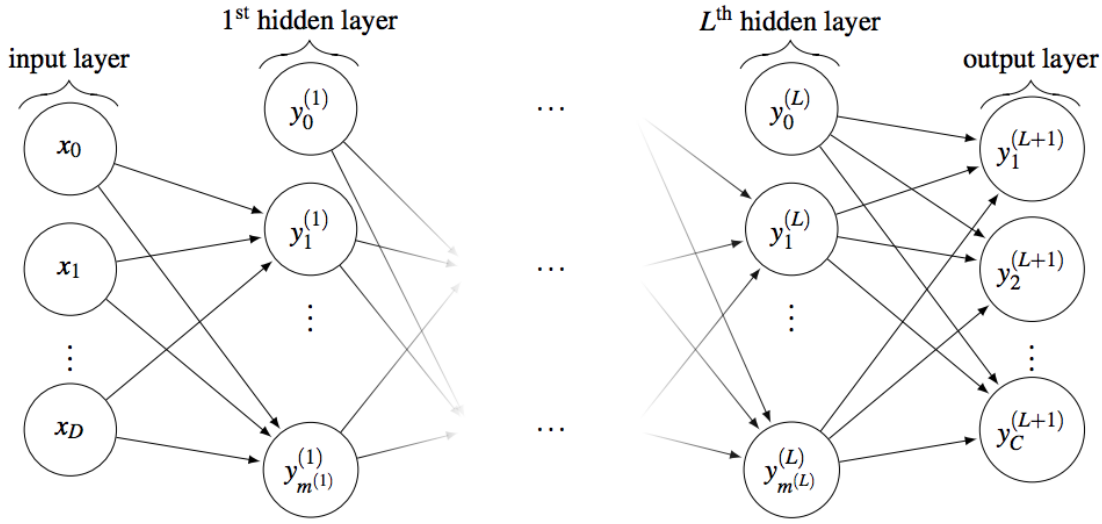

Figure 2.3: Topological structure of an ANN

The i^{th} artificial neuron at layer (l) is also characterized by a bias, $b_i^{(l)}$ that describes how easy it is to get the neuron to *fire*, and an activation function $g(\cdot)$ that generates a non-linear mapping between its input and output value. Early choices for the activation function were inspired by biological neuron activation processes such as the Sigmoid function (Figure 2.4 left) or the Heaviside function (Figure 2.4 center). However, using these activation functions made the learning process very slow and new functions such as the Rectified Linear Unit function (ReLU) were proposed (Figure 2.4 right).

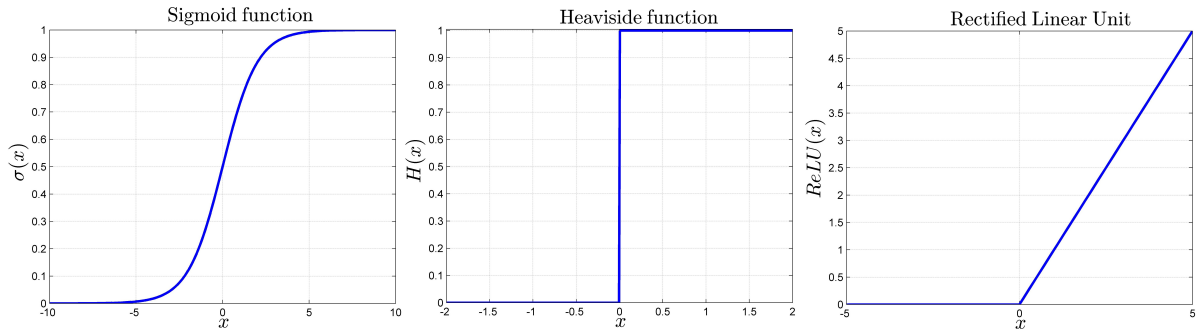

Figure 2.4: Examples of activation functions. Left: Sigmoid function. Center: Heaviside function. Right: ReLU function

The output of the i^{th} artificial neuron at layer (l) is computed as

$$y_i^{(l)} = g\left(\sum_{k=1}^{m^{(l-1)}} w_{i,k}^{(l)} y_k^{(l-1)} + b_i^{(l)}\right) = g\left(\sum_{k=1}^{m^{(l-1)}} z_i^{(l)}\right) \tag{2.14}$$

By defining the following quantities:

1. Layer weight matrix $\mathbf{w}^{(l)} : [\mathbf{w}^{(l)}]_{i,k} = w_{i,k}^{(l)}$ collect all the weights at layer (l).

2. Layer bias vector $\mathbf{b}^{(l)} = \begin{bmatrix} b_0^{(l)} & b_1^{(l)} & \cdots & b_{m^{(l)}}^{(l)} \end{bmatrix}^T$ collect all the biases at layer (l).

3. Layer output vector $\mathbf{y}^{(l)} = \begin{bmatrix} y_0^{(l)} & y_1^{(l)} & \cdots & y_{m^{(l)}}^{(l)} \end{bmatrix}^T$ collect all the outputs at layer (l).

4. Weighted input vector at the (l)-th layer $\mathbf{z}^{(l)} = \mathbf{w}^{(l)}\mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}$

it is possible to write the Equation 2.14 in vector form:

$$\mathbf{y}^{(l)} = g\left(\mathbf{w}^{(l)}\mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}\right) = g\left(\mathbf{z}^{(l)}\right) \tag{2.15}$$

where the vector function $g(\cdot)$ acts on the vectors elementwise.

2.2.2.2 Learning process

The weights and biases of an ANN can be tuned by a process called *learning* which is governed by a learning algorithm. The aim of the learning algorithm is to find weights $\mathbf{w}$ and biases $\mathbf{b}$ that minimize a certain cost function that represents a measure of the distance between the desired output and the actual network output. By defining the vectorization* of the weight tensor as $\vec{\mathbf{w}} = vec(\mathbf{w})$ and the vector containing both weights and biases as $\mathbf{u} = [\vec{\mathbf{w}}\ \mathbf{b}^T]$, the cost function will depends only on this vector $C(\mathbf{u})$. There exist two major learning paradigms in ANN in the object detection context:

- **Supervised learning** paradigm aims to find network weights and biases to approximate a specific target mapping ψ, through a training set $T_S \triangleq \{(x_n, t_n) : 1 \leq n \leq N\}$ includes both input values x_n and desired output values $t_n = \psi(x_n)$.

- **Unsupervised learning** paradigm finds similarities and regularities within the data by itself and the training set does not include any labels or target value, but only the input values $T_U :\triangleq \{x_n : 1 \leq n \leq N\}$

This thesis project focuses on **supervised learning** because most object detection tasks usually depend on classification using supervised learning [49]. The strategies used to tackle the problem of minimizing the cost function are mainly based on two techniques: **gradient descent** and **backpropagation**.

*The vectorization of an n_d-dimensional tensor $\mathbf{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ is defined as:
$\text{vec}(\mathbf{A}) = \begin{bmatrix} a_{1,1,\ldots,1} & a_{2,1,\ldots,1} & \cdots & a_{n_1,1,\ldots,1} & a_{1,2,1,\ldots,1} & \cdots & a_{n_1,n_2,\ldots,n_d} \end{bmatrix}$

2.2.2.3 Gradient descent

A closed-form solution of the high-dimensional minimization problem of the cost function arising in the learning process is very difficult to find and different iterative approaches have been proposed to solve this issue. Gradient descent is an iterative algorithm that aims to minimize the cost function $C(\mathbf{u})$ starting from a configuration of weights and biases $\mathbf{u}[0]$ and progressively updating this configuration through a variation of weights and biases $\mathbf{u}[n+1] = \mathbf{u}[n] + \Delta\mathbf{u}[n]$, where $\mathbf{u}[n]$ denote the weight vector at the n-th iteration. This will generate a variation in the cost function $\Delta C(\mathbf{u})$. The gradient descent algorithm updates the configuration in the direction of the steepest descent (i.e. the one that causes the maximum decrease of the cost function), that is expressed mathematically by the direction of the negative gradient $\nabla C(\mathbf{u})$

$$\Delta\mathbf{u}[n] = -\gamma \nabla C(\mathbf{u}) \tag{2.16}$$

The parameter γ is named **learning rate**. It is an hyper-parameter in the design of the ANN and it is chosen such that the algorithm converges to a minimum in a reasonable time.

2.2.2.4 Mini-batch stochastic gradient descent

The gradient descent procedure presented in the last subsection, also named *batch gradient descent*, computes the gradient of the loss function for all training samples N before executing a single weight update. Since the number of training samples can be relatively big, the standard gradient descent can take a long time to converge to a solution, and is unusable when the number of training samples does not fit in memory.

Stochastic mini-batch approximates the gradient descent by executing a parameter update after a number of training samples equal to the mini-batch size $\mathcal{B}$. Mini-batch SGD reduces the variance of the parameter updates, which can lead to more stable convergence. Stochastic Gradient Descent (SGD) is the case where the mini-batch size $\mathcal{B} = 1$, named also *online learning* because the parameters are updated for each training sample.

2.2.2.5 Backpropagation

What is missing in the procedure presented in the previous sections, is a way to compute the gradient of the cost function. Backpropagation computes the gradient $\nabla C(\mathbf{u}) = \left[\frac{\partial C}{\partial w} \; \frac{\partial C}{\partial b}\right]$ with respect to any weight w and bias b in the network. The total cost is computed by averaging the cost functions of N training examples

$$C(\mathbf{u}) = \frac{1}{N} \sum_{n=1}^{N} C_n(\mathbf{u}) \tag{2.17}$$

where C_n represents the cost for the n-th training example. Without loss of generality, from now on, it is considered a single training example $C_n \triangleq C$. In order to compute partial derivatives $\frac{\partial C}{\partial w_{jk}^{(l)}}$ and $\frac{\partial C}{\partial b_j^{(l)}}$ through backpropagation, an auxiliary quantity is introduced $\delta_j^l \triangleq \frac{\partial C}{\partial z_j^{(l)}}$ defined as the error in the j^{th} neuron at the l^{th} layer. The backpropagation algorithm is composed of the following steps:

1. Propagate the input value x_n through the network to get the actual input and output of each unit.
2. Calculate the so called errors $\delta_j^{(L+1)}$ for the output units $\delta_j^{(L+1)} = \frac{\partial C}{\partial y_j^{(L+1)}} \cdot g'(z_j^{(L+1)})$
3. Determine $\delta_j^{(l)}$ for all hidden layers l by using error backpropagation $\delta_j^{(l)} = g'(z_j^{(l)}) \sum_{k=1}^{m^{(l+1)}} w_{j,k}^{(l+1)} \delta_k^{(l+1)}$
4. Calculate the required derivatives $\frac{\partial C}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} y_k^{(l+1)}$

2.2.2.6 Convolutional Neural Networks

CNNs are a particular subclass of ANN used to solve image-driven pattern recognition tasks. Traditional ANNs are not employed for image recognition due to the fact that full connectivity between layers of artificial neurons do not scale well to higher resolution images and requires a lot of computational power. A typical CNN structure is presented in Figure 2.5.

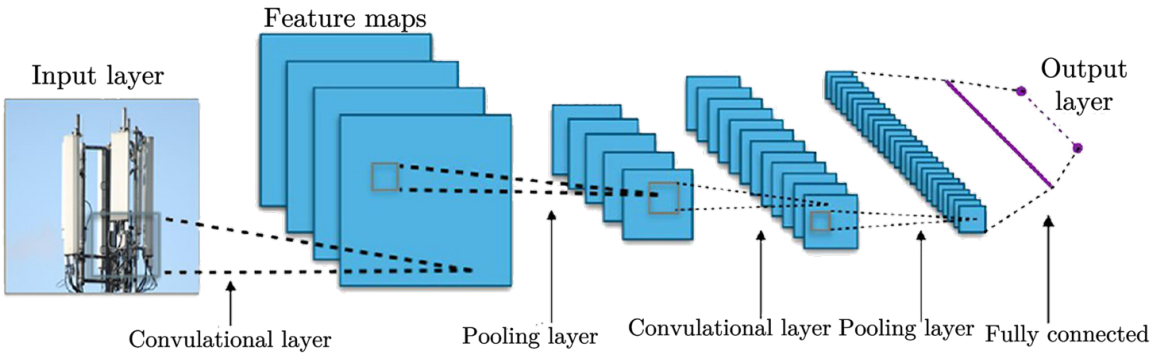

Figure 2.5: CNN typical architecture. Image credit: public domain work by Wikimedia Commons user [9]

.

The data entering the l^{th} layer of the CNN are order 3 tensors of the type: $\mathbf{x}^{(l)} \in \mathbb{R}^{H^l \times W^l \times C^l}$ that represent images with H column pixels, W row pixels, and C channels. Typically $C = 3$ and represents components of a color model (e.g. RGB, YUV, etc.).

Each image pixel is identified by a triplet (i^l, j^l, c^l) that refers to the j^{th} column and i^{th} row of the c^{th} color channel at the l^{th} layer. The input image is processed by a CNN through several layers of artificial neurons, where each layer represents a processing step that transforms an input tensor $\mathbf{x}$ to an output tensor $\mathbf{y}$.

The CNN takes as input a tensorial image representation that goes through different types of layers, responsible for making certain types of transformations on the input. The network output is a tensor of predictions that expresses the confidence with which each class of object has been predicted by the CNN. The following discusses the function that each CNN layer has in the object detection task and the main concepts behind CNNs.

2.2.2.7 Input layer

The input layer simply presents the raw pixel values of the image to the CNN and has the same dimension as the input image tensor $\mathbf{x}^{(0)}$.

2.2.2.8 ReLU layer

The ReLU layer returns a tensor $\mathbf{y}$ of the same dimension of the input $\mathbf{x}$ where each element of the vector is transformed according to a ReLU transformation (represented in Figure 2.4) of the type:

$$y_{i,j,c} = \max(x_{i,j,c}, 0) \tag{2.18}$$

There are no parameters inside the ReLU layer, hence no need for parameters learning in this layer. This layer is used to increase the non-linearity of the CNN. There is the need to increase non-linearities in the detection network because there exists a semantic gap between the digital representation of the image and the image's semantic information that is created by a non-linear mapping between these two representations. For this reason, there is the need for a non-linear mapping from the CNN's input to its output.

2.2.2.9 Convolutional layer

The convolutional layers play a central role in the CNN's architecture. In these layers, artificial neurons consist of tensors that represent learnable filters acting as feature extractors. The CNN learns the feature representations of the input image by tuning the filter's weights, i.e. their coefficients. These learnable kernel filters usually are of reduced dimension with respect to width W and height H dimension and extends in the depth dimension for the whole depth C^l of the input tensor. Assuming that D kernels are used at each layer and each kernel is of spatial span $H \times W$, the fourth-order kernels' tensor is denoted by $\mathbf{f} \in \mathbb{R}^{H \times W \times C^l \times D}$.

Intuitively, the learning algorithm change the filter's weights such that the filters activate when they receive as input pieces of images that contain some type of visual features, such as oriented edges or patterns. Stacking many convolutional layers creates non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then assembles from them representations of larger areas detecting features at a higher level of abstraction.

Since images are usually high-dimensional inputs it is impractical to have full connectivity between layers of neurons as is done for standard ANNs. Instead, in CNNs each neuron is connected only to a local region of the previous layer. The spatial extent of this connectivity is a hyper-parameter named the neuron's **receptive field** and is equivalent to the neuron's kernel filter size. Another characteristic of convolutional layers is the fact that the learnt filter's weights and biases are shared by the whole layer and form what it is called a **feature map**. In each feature map (convolutional layer) all the artificial neurons respond to the same feature within their specific receptive field (portion of the image). This allows detection of the same features regardless of its position in the visual field (translation invariance) and greatly reduce the number of learnable parameters in each layer.

Besides the receptive field size F and the depth D introduced before, CNNs have other important hyper-parameters that influence the performance. D represents the number of filters used in the convolutional layer, to capture different types of features in the input image at the same scale. The **stride** hyper-parameter S represents the discrete step size with which the kernel filter is slid across the image. For example, when the stride is 1 the filter is moved one pixel at a time, while when the stride is 2 then the filter jumps 2 pixels at a time along each axis. This will produce a smaller output tensor. The **zero padding** hyper-parameter P is used to reduce the discontinuity effect near the picture's border and to control the spatial size of the output tensor. In many cases, this parameter is used to preserve the spatial size of the input volume so the input and output width and height are the same. Formally, the operation that is performed on the input tensor by the convolutional layer is a tensorial convolution with the neurons' kernel filters.

$$y_{i,j,c} = \sum_{h=0}^{H} \sum_{w=0}^{W} \sum_{k=0}^{C} = x_{h,w,k} \cdot f_{i-h,j-w,c-k} \tag{2.19}$$

This convolution operation is repeated for all the D kernel filters of the l^{th} layer. In order to give a better understanding of this operator and the meaning of the convolution's hyper-parameters, a 2-D convolution example (i.e. with $C = D = 1$ corresponding to the case of grayscale image convolution) is shown in Figure 2.6. The example shows the convolution of a 4×4 matrix with a 3×3 kernel filter. In this example the stride is

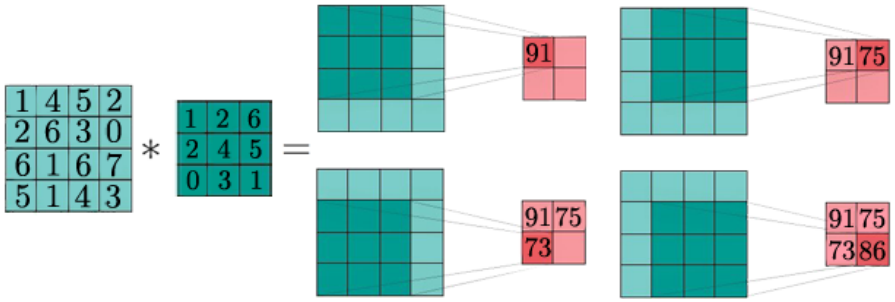

Figure 2.6: Example of grayscale image convolution. Image credit: public domain work by Wikimedia Commons user [10]

$S = 1$, the depth is $D = 1$, and zero-padding is not used (hence $P = 0$). The output is a 2×2 matrix, where each entry is given by the sum of the products of the input matrix and kernel's entries progressively shifted with respect to the entries of the input matrix.

2.2.2.10 Pooling layer

The purpose of the pooling layers is to reduce the spatial resolution of the feature maps by downsampling the input tensor. This allows the CNN to achieve spatial invariance with respect to input distortions and translations. It is also used to reduce the number of parameters and computation in the CNN. The depth dimension of the input tensor D remains unchanged after the pooling operation. There exist different ways to do pooling, but max pooling is the most widely used type of pooling operator. In max pooling, the pooling operator maps a sub-region to its maximum value, while the average pooling maps a sub-region to its average value. Formally, max pooling selects the largest element within each receptive field such that:

$$y_{i,j} = \max_{0 \leq i \leq W, 0 \leq j \leq H} x_{i,j} \tag{2.20}$$

For example, a pooling layer with filters of size 2×2 applied with a stride of 2, downsamples every depth slice of the input tensor by a factor of 2 along both width and height. In this case, every maximum operation would be computed over a receptive field of size 4 (see Figure 2.7).

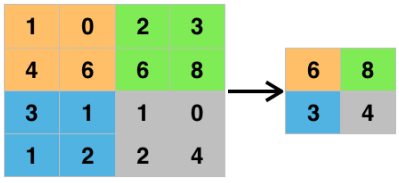

Figure 2.7: Max pooling example. Image credit: public domain work by Wikimedia Commons user [11].

Note that this layer has no parameters to be trained since it computes a fixed function of the input, hence there is no need for training in pooling layers.

2.2.2.11 Fully Connected layer

The Fully Connected (FC) layer consists in a layer in which neurons have a full connection to all activation in the previous layer, as seen in a regular ANN, shown in Section 2.2.2.1. Their activation can be computed with a matrix multiplication followed by a bias offset. The fully connected layer in analogy with the brain interprets the feature representations constructed in the layer before and performs the function of high-level reasoning [50].

2.2.2.12 Output layer

The output layer is the CNN's final layer and assigns to each input image a probability value that represents the class detection confidence.

2.2.2.13 Dataset

In the object detection context, the dataset is a collection of images used to train, test, and validate the CNN model. More in particular, the training set is composed by target values for the classes of each element in the training set and the coordinates of the localization for the bounding box around the image (ground truth box). Choosing a proper dataset for a given detection task is of paramount importance. In fact, the model's performance is highly dependent on the quality of the underlying dataset. For any supervised machine learning task, it is common to split the entire dataset into three subsets, namely:

1. **Training set** contains data used for learning, i.e. to find the CNN parameters (weights and biases) during the backpropagation process. It usually contains the majority of examples contained in the dataset.

2. **Validation set** contains data used to tune the hyper-parameters of the CNN model and should follow the same probability distribution as the training dataset. Using only the training set to train the model would lead to *overfitting*, i.e. to produce a model that corresponds too closely to training set but one that may fail to reliably detect future generalized data.

3. **Test set** contains data used to assess the performance of the trained model and is not used during training. The test set is usually employed only once the model's parameters and hyper-parameters are fully specified in order to assess the model's generalized performance. The test set is independent of the training dataset, but is assumed to follow its same probability distribution.

2.2.2.14 Validation-based early stopping

Early stopping [51] is a *regularization* technique used during training to decide the optimal point to stop training. It is used to avoid *overfitting* and to minimize the *generalization error*. The *generalization error* is a measure of how accurately the CNN model is able to predict outcome values for previously unseen data. It may happen that the best result in terms of results is not achieved by the last training epoch. The error on the validation set is used as an indicator for the generalization error in determining when overfitting has begun.

2.2.2.15 Hyper-parameters

Hyper-parameters choice is key in the design of neural networks. These are parameters that are not optimized through the learning process and needs to be tuned independently of the model's parameters. They are typically initialized before the learning process begins and empirically adjusted using the validation set according to heuristics and experimentation. By contrast, the values of other parameters are derived via training (weights and biases). The following provide an overview of the most important hyper-parameters for CNNs trained with backpropagation and SGD:

1. **Learning rate** is a constant used to control the rate of learning. If the learning rate is too large, the optimization procedure to diverge, while if it is too small, the learning process might take much longer to train the network and reach the same minimum it could have reached with a higher learning rate in less time.

2. **Learning rate decay** refers to the rate of annihilation of the learning rate over time. If this parameter is reduced too quickly this can lead to sub-optimality in computational time while bouncing around the desired minimum, while reducing it too aggressively causes the system to stop before reaching that minimum.

3. **Momentum** is the parameter that controls learning rate improvement. Momentum helps to determine the direction of the next step with the knowledge of the previous steps. It helps to prevent oscillations. A typical choice of momentum is between 0.5 to 0.9.

4. **Batch size** refers to a number of training examples in one forward/backward pass of the CNN. The impact of the batch size is mostly of a computational nature since modern hardware can parallelize matrix multiplications and convolutions.

5. **Epoch numbers**, that is defined as a single forward/backward step of the entire dataset through the neural network, is the number of times the whole training data is shown to the network while training. Since the family of gradient descent algorithms

are iterative, multiple forward/backward pass of the algorithm in the training process are needed to minimize the cost function.

6. **Detection threshold** regulates the confidence level with which a given instance of an object is considered as correctly detected.

7. **Kernel, stride, and padding size** discussed in Section 2.2.2.9

8. **Input network dimension** that corresponds to the size of the input image that will be processed by the CNN

9. **Activation function** (e.g. Sigmoid, Heavyside, ReLU) as discussed in Section 2.2.2.1

10. **Cost function** as discussed in Section 2.2.2.3

2.2.2.16 Transfer Learning

Within the context of object detection, transfer learning is a ML method in which a CNN pre-trained model, developed for a given task, is reused as the starting point for a model on a second task. This reduces the computational time and enables to training the network with a smaller training dataset. Quoting Yosinski et al. [52]: *In transfer learning, we first train a base network on a base dataset and task, and then we repropose the learned features, or transfer them, to a second target network to be trained on a target dataset and task.* Different CNNs trained on natural images exhibit a common phenomenon: on the first layer, they learn non-specific and similar low-level features to recognize edges, corners and other low level characteristics of the image. For this reason, the weights and biases learned in these first layers are reusable for many. Instead of training the entire CNN from scratch, it is common to start from a CNN model that has been pretrained on a very large image dataset (e.g. ImageNet dataset [53], which contains 14 million images of 20,000 categories), and then re-use the pretrained CNN model applying transfer learning. The two major transfer learning scenarios are [54]:

1. **Fixed feature extractor**: starting from a pretrained CNN model, the last fully-connected layer is removed and retrained, while the rest of the CNN is treated as a fixed feature extractor for the new dataset with the same pretrained weights.

2. **CNN fine-tuning**: the weights and biases of the pretrained CNN are fine-tuned by continuing the backpropagation. It is possible either to fine-tune all the layers of the CNN, or to keep some of the earlier layers and only fine-tune some higher-level portion of the network since the earlier features of a CNN contain more generic features (e.g. edges, corners, blobs, etc.), but later layers of the CNN becomes progressively more specific to the details of the classes contained in the original dataset.

Three main benefits are related to the use of transfer learning [55]. First, the initial performance achievable in the target task is higher using only the transferred knowledge, before any further learning is done, as compared to the initial performance of a non-pretrained model. Second, the amount of time it takes to fully learn the target task given the transferred knowledge compared to the amount of time to learn it from scratch is much lower. Third, the final performance level achievable in the target task compared to the final level without transfer is higher.

2.3 Related works

The current state of the art in UAS object detection and tracking offers a plethora of case studies related to very different applications [56–61]. Surveillance, inspection, and reconnaissance are among the most common UAV-based operations. In these applications, object search, detection, and classification are key tasks, which usually are still manually performed. Automatic object detection and classification offers benefits both during the drone operations and during the posterior data analysis. At the time of this writing, there are no contributions in the scientific literature specifically addressing cell site inspections or cell site equipment detection, the main areas of interest for this thesis. However, techniques used in the general sector of UAS object detection and tracking can be adapted and reused in the framework of cell tower inspection.

2.3.1 Kanade-Lucas-Tomasi object tracker

Tanathong and Lee [62] performed a preliminary experiment to measure the performance of KLT for real-time feature tracking from UAV image sequences. They concluded that the tracker can identify corresponding features in consecutive images with greater than 80% accuracy if the displacement is small.

Jabar, Fraokhi, and Sheikh [63] proposed a semi-automatic object tracking method by combining SIFT and KLT tracker for UAV-based applications. In their approach, the user selects the region of interest and KLT is then used to track the specified object in consecutive frames. To overcome rapid changes in appearance, occlusion, or disappearance from the camera view, they employ forward-backward error compensation. Through experimental results, they showed that their proposed method exhibits higher performance when compared to similar algorithm employed in the field.

Michael and Vani [64] attempt to find the efficiency of SIFT and SURF features in a semiautomatic object tracking method based on KLT for UAV applications. In this approach, the region of interest is specified by the user and the interest points are found. Two promising approaches were used to detect the robustness; The outcome is applied to KLT for further video tracking. SIFT provides a higher matching rate than SURF when applied to a KLT tracker.

2.3.2 YOLO-based object detection

Various CNN real-time object detection applications employing UAV have been developed, such as the ones presented in [65–67]. Many of these applications are based on the YOLO [12] ("You Only Look Once") algorithm running on the darknet CNN [68].

Tijtgat et al. [67] compared the YOLOv2 [69] algorithm, the successor of YOLO, with a hand-crafted feature detector named Aggregated Channel Features (ACF) in the context of a UAV Warning System. Accuracy and speed of both detectors were tested and YOLOv2 outperformed ACF for both inference time and detection accuracy. They also presented a method to choose the optimal object detection configuration given specific frame rate and detection accuracy requirements.

Radovic, Adarkwa, and Wang [70] employed a CNN algorithm based on YOLO to detect, classify, and track objects from real-time video feeds supplied by UAVs. They presented in details a procedure to choose the hyperparameters used to train the CNN on a set of aerial images in order to realize efficient and automated object recognition. They showed how CNNs accuracy depends on the selection of operational hyperparameters. They detect and classify objects with a high level of accuracy (97.5%) and high computational efficiency (150 FPS). Their method allows UAVs to perform real-time and complex object detection tasks in an efficient and accurate manner.

Maher et al. [71] proposed a real-time human gesture identification scheme for controlling a Micro Aerial Vehicle (MAV) by exploiting cutting edge CNN real-time object detection algorithms based on the state-of-art YOLOv2. They also built a new dataset to train or fine-tune the deep neural networks for human gesture detection and proposed to infer the gesture from detection results, in which each interpreted gesture is equivalent to a UAV flying command.

2.3.3 Mobilenet-based object detection

Anar et al. [72] implemented a general purpose object detection module on a DJI phantom 3 UAV with a TensorFlow (TF) MobileNet implementation running on an Android device. Also, they used the InceptionV3 model for classification and reported accuracy and time performance results for 10 classes of objects such as cats, laptops, cars, etc. They claim to reach an inference time of 851 ms with the highest accuracy obtained equal to 28.74% on a Samsung Galaxy S2 tablet.

Bonatti et al. [73] present a novel algorithm for actor detection. They tested SSD Mobilenet, Faster R-CNN [74], and YOLOv2 and conclude that MobileNet [75] is suited for their application due to its low memory usage and fast inference speed (around 300 ms) even though Faster R-CNN exhibits a higher accuracy. Finally for tracking, they used a KCF tracker [76] due to its real-time performance.

Budiharto et al. [77] presented a UAV object detection application to deliver medical aids for patients in emergency situations. They exploited the drone for transporting items efficiently, detecting the object target through an object detection module that combines a MobileNet [75] architecture with the SSD [78] framework. They run the object detection algorithm on the Parrot AR drone 2.0, a UAV with low specification computing on-board hardware, achieving a processing speed of about 14 FPS. The MobileNet SSD was first trained on the COCO dataset [79] and then fine-tuned on PASCAL VOC [80] on 20 classes reaching 72.7% mAP.

2.3.4 CNN transfer learning

CNNs perform well for analyzing large datasets, but the performance may become worse when dealing with data sets of smaller dimension. For these scenarios, it is useful to employ CNN transfer learning doing only fine-tuning on the custom dataset, given the pre-trained CNN model, thus reducing both the needed data set size and training time. Some of the most significant to the development of this project are described below.

Marmanis et al. [81] used a pretrained CNN to generate an initial feature representation of the remote sensing images being analyzed. They reshaped the outputs of the last two fully connected layers into 2-D arrays and trained a new CNN composed of two convolutions and two fully connected layers followed by a softmax classifier. They improved the overall accuracy from 83.1% up to 92.4%.

Kucuksubasi and Sorguc [82] investigated the problem of transfer learning in the context of an autonomous UAV system for surface crack detection. They implemented a fine-tuning of the InceptionV3 model [83] that had been pretrained on the Imagenet dataset [53]. They fine-tuned the CNN with a custom dataset containing 582 images with cracks (positive samples), and 458 images without cracks (negative samples). After 20 epochs of training, they obtained an accuracy of 97.382%.

Finally, Othman et al. [84] generated an initial feature representation of the scenes being analyzed from a CNN that had been pre-trained on a large amount of labeled data from an auxiliary domain. Then, the convolutional features are fed as input to a sparse autoencoder to learn a new feature representation. After this pre-training phase, they build a classification system adding a softmax layer on the top of the autoencoding encoding layer and then fine-tune the resulting network in a supervised manner using the target training images available at hand.

Chapter 3

Methodology

This chapter starts with the description of the high level system architecture in Section 3.1. Subsequently, it describes the procedure used for the dataset collection (Section 3.2) and dataset augmentation (Section 3.2), the experimental methodology (Section 3.3), the model training (Section 3.5), and the evaluation metrics (Section 3.6). Finally, in Section 3.7 the choice of the object detection and tracking algorithms is discussed.

3.1 System architecture

The high level system architecture is shown in Figure 3.1.

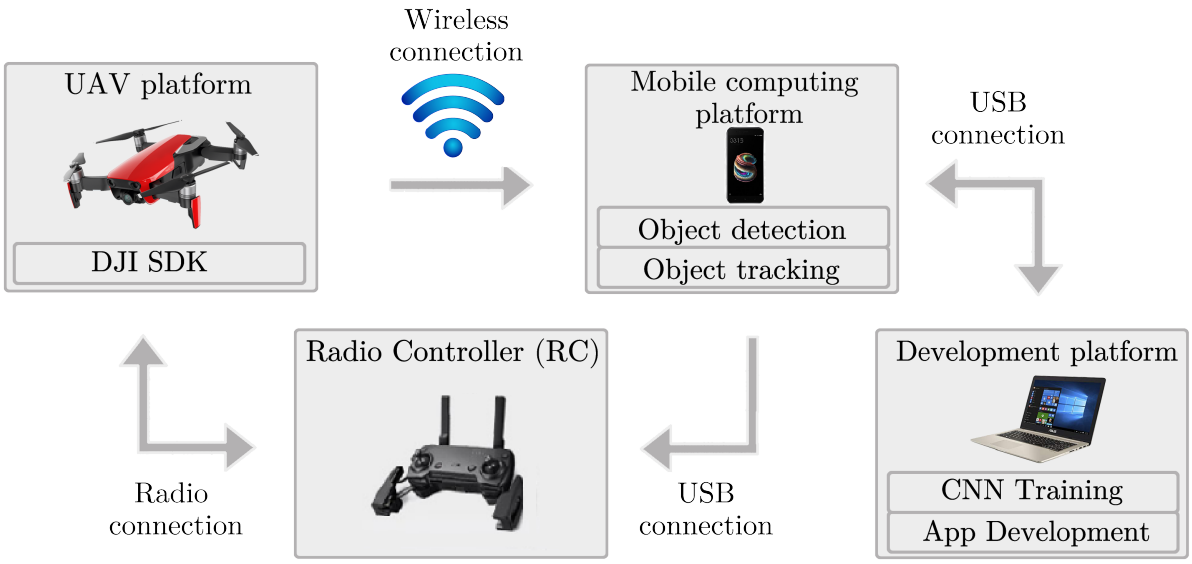

Figure 3.1: System architecture

The fundamental system components are:

1. The **UAV platform**,
2. The **mobile computing platform**,
3. The **RC**, and
4. The **development platform**.

The purpose of the UAV system is to assist with the aid of object detection and tracking modules the radio-site engineers during unmanned radio site inspections. These modules are designed as support to cope with the inherent heterogeneity of base stations (see Section 1.3). As described earlier, this heterogeneity makes the identification of the target hardware equipment to be inspected hard and hinders the inspection itself. More specifically, the UAV platform, controlled by a technician from the ground, is flown near the equipment to be inspected through commands given via RC. The DJI SDK allows the connection between the embedded mobile and the drone platform and to send the video stream via a Wi-Fi link. The mobile device will run the object detection and tracking algorithm in real-time in order to discriminate which is the equipment of interest for the inspection. The result of the detection consists of bounding boxes around the equipment of interest denoted with an object label and a confidence level. These results are displayed on a mobile platform to the site technician, allowing them to make real-time decisions regarding which radio equipment to inspect. The development platform is used for training the CNN model and developing of the mobile detection/tracking application. The proposed architecture is the result of multiple design decisions, described in the following subsections.

3.1.1 UAV platform

In this subsection, the choice of the UAV hardware platform is discussed. The choice was based on the following design constraints:

1. The **dimensions** constraint has been set while considering two main aspects. First, the system's dimensions should allow the radio tower technicians to easily transport the system to the site without any special transport equipment. Second, the UAV dimensions should allow indoor flight for laboratory tests in an area of a few square meters and the aircraft should be able to fly in hard to reach environments with very limited operating space. Keeping in mind the fact that other tools will be carried to the site along with the aircraft and looking at standard toolbox manufacturers' dimension (e.g. $40,64 \times 29,21 \times 19,05$ cm [85]), and the dimension of the testing environment, the longest dimension requirement is fixed to be **less than 30 cm**. Thus, the dimension requirement restricts the potential choice of aircraft to the **very small UAV** category (see Section 2.1.1.2).

2. The **weight** is set according to the ease of transport criteria. By looking at the recommendations released by the department of public and environmental safety at Tufts University [86], i.e. the maximum load carried must be lower of 15% of body weight. For example, an 80 kg person should limit their carry weight to 12 kg. Considering the weight of other equipment and tools, the maximum weight is fixed to **2 kg**.

3. The **cost** requirement is set by Ericsson,as the main stakeholder and funder of the project. The maximum cost was set to **7000 kr**. This requirement further restricts the field of choice to very low-cost UAV platforms. A particular focus is put on quad-rotor platforms because they are the cheapest in the multi-rotor category.

4. The **flight time** should be long enough to allow for a complete inspection of the radio site. This requirement was set by consulting Field Service Operations (FSO) engineers at Ericsson who are doing drone site surveys in U.K. They have stated that the average time for a UAV site inspection is around 10 min. Also they stated that the longest UAV inspection they experienced lasted around 20 minutes. For this reason the the flight time requirement is fixed at **20 minutes**.

5. The **flight range** should be such that the communication system is able to support connectivity for the duration of the inspection. It is determined primarily by the vertical range, i.e. the cell cite's height. According to a study [87], with a database of approximately 285.000 cell cite location in the U.S., the maximum height of a base-station is around 1000 ft that is **300 m** circa. The horizontal range is not considered as a constraint because it usually has a smaller extent than the vertical range.

6. The **camera** equipment is set by considering the required object detection and tracking tasks that this project aims to implement. The camera sensor specifications were chosen based on the camera payload of similar studies presented in Section 2.3 and other consideration on algorithms such as [88]. Based upon these studies, the minimum resolution is set to **8 Megapixel (MP)**, the aspect ratio to **1280×720** and the frame rate at **24 FPS**.

7. The **software developing environment** must ensure the availability of appropriate Application Programming Interface (API) and Software Development Kit (SDK) in order to integrate the proposed object detection algorithm with the UAV flight and control modules.

These requirements narrow down the field of choice for the UAV. A further restriction, is the fact that Ericsson is interested in a DJI™ product because FSO engineering are already carrying out unmanned inspections using DJI UAVs and the reusability and integrability of the new solution with their base-station inspection framework is a key requirement. These factors led to the choice of **DJI Mavic Air™*** as flight platform employed in this project, a low-end and ready-to-fly customizable commercial quadrotor UAV. The UAV's specifications are listed in Table 3.1. As can be observed from the specifications table, the UAS platform satisfies all the project's requirements.

Table 3.1: UAS platform specifications

Characteristic	Value
Takeoff weight	430 g
Dimensions (unfolded)	$16.8 \times 18.4 \times 6.4$ cm
Price	6699 kr (Swedish krona)
Max flight time	21 minutes
Max flight distance	10 km
Max speed	68.4 kph
operating frequency range	2.400-2.4835 GHz / 5.725-5.850 GHz
Camera pixels	12 MP
Frame rate	24/25/30/48/50/60/120 FPS
Minimum resolution	1280×720
Camera gimbal range	$-90°$ to $+17°$
Software development environment	Mobile SDK

3.1.2 Mobile computing platform

The embedded mobile platform must be able to receive the video stream from the UAV platform and support the computational cost for the real-time object detection and the control of the UAV. The prototype is based on the Android development framework [89] and the SDK provided by DJI [90]. The Android testing platform is a Xiaomi Mi A1. According to the Android Benchmarks performed by PassMark [91], the Xiaomi Mi A1 is a mid-range device, ranked at the 347^{th} position in computational performances out of the 4687 evaluated devices. It is worth noting that the limited computational power of the embedded device is the most challenging aspect for the design of a real-time object detection system based upon CNNs and it is the main reason that makes necessary the integration of the object detection module with a real-time object tracking module. On the other hand, the fact that the testing device is not a top of the line device reduces costs and allow for easier adoption of the solution as there are more potential target platforms that could be used.

*https://www.dji.com/mavic-air

The computational specifications of interest for this project are reported in Table 3.2.

Table 3.2: Mobile platform specifications

Characteristic	Value
Operating System	Android 8.1.0 (Oreo)
CPU	Octa-core Qualcomm Snapdragon 625 at 2.0GHz
GPU	Adreno 506 at 650 MHz
RAM	4GB

3.1.2.1 Android

Android is the most popular Operating System (OS) for mobile devices. Android was developed by Google [92]. It is based on a Linux kernel [93] and a Java [94] based framework. The open source Android SDK provides a set of tools for application development and testing. The Android Native Development Kit (NDK) [95] allows modules of an application to be implemented in native code, using languages such as C and C++ [96, 97]. Hence NDK will be used to create the object detection module. Details of the Android OS and its SDK are outside the scope of this thesis. However, the details needed for the development of the prototype system will be discussed as needed in the implementation described in Chapter 4.

3.1.2.2 DJI SDK

The DJI mobile SDK offers a framework to create customized mobile applications by managing different functions of the UAV, such as high and low level flight control, telemetry and sensor data, camera and gimbal control, live camera video feed, and support for basic communication and control functionality. The SDK simplifies the application development process by taking care of lower level functionality such as flight stabilization, battery management, signal transmission, and communication. In order to allow the developer to focus on the application the DJI product is integrated without considering the low level control robotics or embedded system details. Specifically, in the SDK includes:

1. A set of libraries that can be imported into an Android project to access the UAV and its module,

2. A UAV simulator and visualization tool, and

3. SDK and API documentation.

3.1.2.3 TensorFlow

TF [98] is a popular software library for ML developed by Google [92]. TF is able to run on heterogeneous systems, including mobile devices. As ML tasks are computationally expensive, model optimization is used to improve performance when running on mobile embedded devices by using Tensorflow's mobile optimized mobile version: TensorFlow Mobile (TFM). The minimum hardware requirements of TFM in terms of Random Access Memory (RAM) size and Central Processing Unit (CPU) speed are low, hence the primary bottleneck is the calculation speed as the desired latency for mobile applications is low. TensorFlow Lite (TFL) is an evolution of TFM, which supports deployment on mobile and embedded devices. TFL was born from the necessity to further optimizing TFM for lightweight mobile use as there is a clear trend to incorporate ML in mobile applications. Some of the optimizations included in TFL are hardware acceleration through the silicon layer and frameworks such as the Android Neural Network API. TF-trained models are converted to the TFL model format by TF.

3.1.2.4 OpenCV

OpenCV [99] is open source CV and ML software library designed for computational efficiency and with a strong focus on real-time applications. OpenCV was launched in 1999 by Intel. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. It was originally written in C/C++ [96, 97] with bindings in Python [100] and Java [94] languages and supports Windows, Linux, Mac OS, iOS, and Android. OpenCV is widely adopted and has more than 47 thousand people in its user community and the estimated number of downloads exceeds 14 million.

3.1.3 Radio controller (RC)

The RC has two main functions. Firstly, the RC is used to physically pilot the UAV platform to the radio equipment to be inspected through a radio connection to RC. Secondly, the Android device is connected to the RC by a USB connection in order to register the device and to the SDK platform made available by DJI. This will allow to establish a Wi-Fi connection between the aircraft and the mobile platform and receive the video stream from the UAV camera that will be processed by the android application to display the detected hardware.

3.1.4 Development platform

The main requirement of the development platform is the ability to train the proposed CNN in a limited amount of time. The use of a GPU in the training process significantly reduces the amount of time needed for this operation, therefore this is considered to be an essential requirement. The computing platform also implements the Android application that will run the detection algorithm during the inference.
The computing platform used as development environment is an ASUS VivoBook Pro 15 N580VD whose specifications are listed in Table 3.3.

Table 3.3: Development platform specifications

Characteristic	Value
OS	Ubuntu 16.04.5 LTS (Xenial)
CPU	Intel Core i7-7700HQ at 3.80 GHz
GPU	NVidia GeForce GTX 1050 at 1.345 MHz
RAM	16GB

3.2 Dataset

The total dataset is composed 1024 images containing two models of Ericsson RRU, namely **Radio 4415** and the **Radio 2219** (Figure 3.2).

Figure 3.2: Left: RRU 4415. Right: RRU 2219

This baseline dataset has been acquired at Ericsson Studio (Grönlandsgatan 8, 16440 Kista, Sweden), where different demo base-stations are installed. Since a single image may contain more than 1 RRU, the total number of RRUs present in these images amounts to 1969. Figure 3.3 shows the histograms associated to the dimensions (width and height) of the RRUs in the scenes.

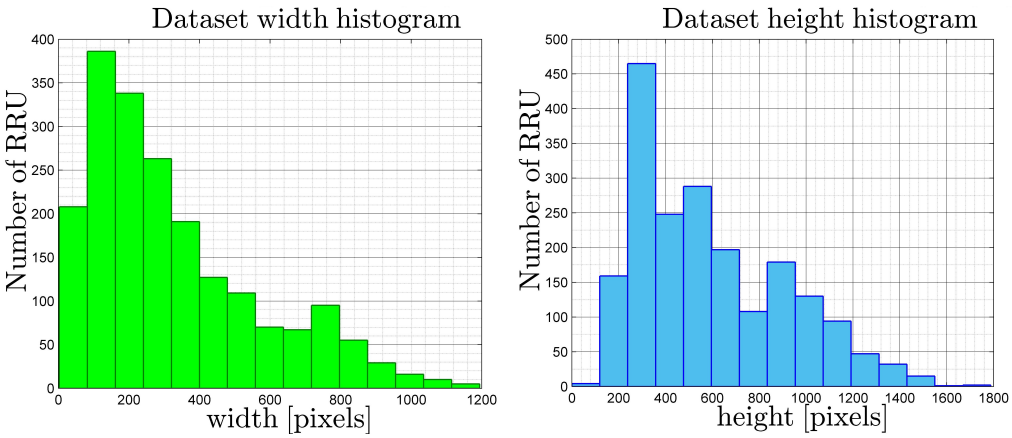

Figure 3.3: Dataset dimensions histograms

The camera used for the acquisition has a resolution of 12 MP camera with regular wide-angle with 26mm f/2.2 lens and 1.25 μm pixels. This camera is different from the drone camera that will be used at inference time. The reason behind this choice lies in the fact that using an hand-held camera (i.e. not mounted on a drone platform) to acquire the dataset facilitates the acquisition greatly and provides a scalable acquisition methodology to other types of radio site equipment. However, other works focusing on UAV object detection applications [101–103] highlight the pitfalls in using hand-held camera images for training versus those captured by a drone during the operational detection phase. Difficulties in detecting objects in data from a drone may arise due to the positioning of the camera compared to images taken by a human with an hand-held camera. Due to the specificity of the object classes to be detected, it was impossible to use publicly and privately accessible databases to compose the dataset. This factor imposes a huge limitation on the dataset size since the number of images that can be acquired is limited and there are no previously established datasets containing the radio equipment of interest. However, transfer learning and data augmentation (see Section 3.3) overcomes these limitations by reducing the dataset size needed to properly train the model and enhancing the data with realistic transformations. The method with which the dataset is prepared included the following steps:

1. Videos are acquired at 1080×1920 and 24 FPS of different lengths and with different backgrounds, scaling, and point of views.

2. Video frames are extracted from the recorded videos at 6 FPS.

3. Redundant and blurred video frames are manually discarded.

4. Each video frame is labeled by manually drawing bounding boxes around the objects of interest.

The extraction rate (6 FPS) is chosen empirically, finding a good trade-off between difference in frames appearance in terms of orientation, luminosity, and size of the radio site equipment. Finally, 70% of the complete dataset is reserved for use as the training set, while the remaining 30% is used as test set and validation set. The implementation details and the complete procedure to build the dataset is presented in Section 4.1.

3.3 Data Augmentation

Data augmentation is a regularization technique that allow to increase the dataset variety and size by applying transformations on the original data. In general, the larger and more diverse the dataset is, the better generalization performance are achieved by the model. In the context of image detection, the augmented dataset is obtained through image processing operations that alter the content of the image data. In order to be effective, the resulted augmented dataset must remain realistic and the set of transformation applied to the image should be selected in a way to not create unrealistic images. In order to determine what type of transformations should be used on the RRU dataset, some considerations on the current use-case need to be made. The major challenges that may hinder detection during the UAV inspection activity are:

1. **Brightness change**: the video feed may be too bright or dark based on the UAV camera exposition to sources of light (e.g. camera facing the sun). This is overcomed by using a *brightness modifications* augmentation technique. This consists in increasing or decreasing the brightness of the image by changing each pixel image according to the formula
$$x_{\text{aug}}(x, y) = x_{\text{base}}(x, y) \cdot \xi \tag{3.1}$$
where ξ represents the brightness change factor.

2. **Occlusion**: there may be obstacles between the UAV camera and the objects of interests that partially or totally hide them. The techniques used to overcome occlusion is to insert zero padding areas in the frames in order to partially hide (occluding) objects of interests.

3. **Blurring**: fast motion of the UAV or the excessive proximity of the UAV with respect to the equipment to inspect may cause blurring and out of focus images. The blurring effect is faithfully reproduced by the Gaussian Blur, that is a filtering (convolution operation for each channel) with a Gaussian kernel:
$$x_{\text{aug}}(x, y) = x_{\text{base}}(x, y) * \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2}$$

4. **Corrupted data**: it may happen that some of the frames received by the mobile computing platform are corrupted and distorted due to transmission errors. A possible model for this corruption is given the addition of white Gaussian noise $n \sim \mathcal{N}(\mu, \sigma)$

$$x_{\text{aug}}(x, y) = x_{\text{base}}(x, y) + n \tag{3.3}$$

Along with these transformations, other images are generated starting from standard RRU sample images that are resized and rotated on different backgrounds images to further increase the dataset. The complete image augmentation pipeline is shown in Figure 3.4.

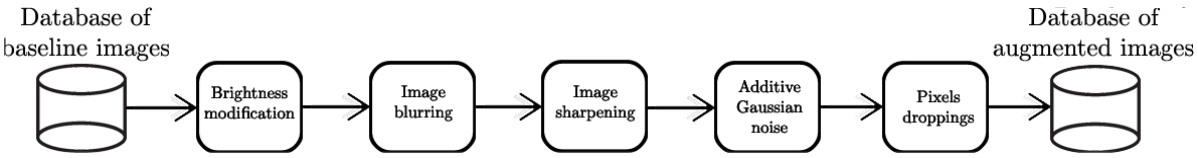

Figure 3.4: Image augmentation pipeline

Some examples of augmented images are shown in Figure 3.5.

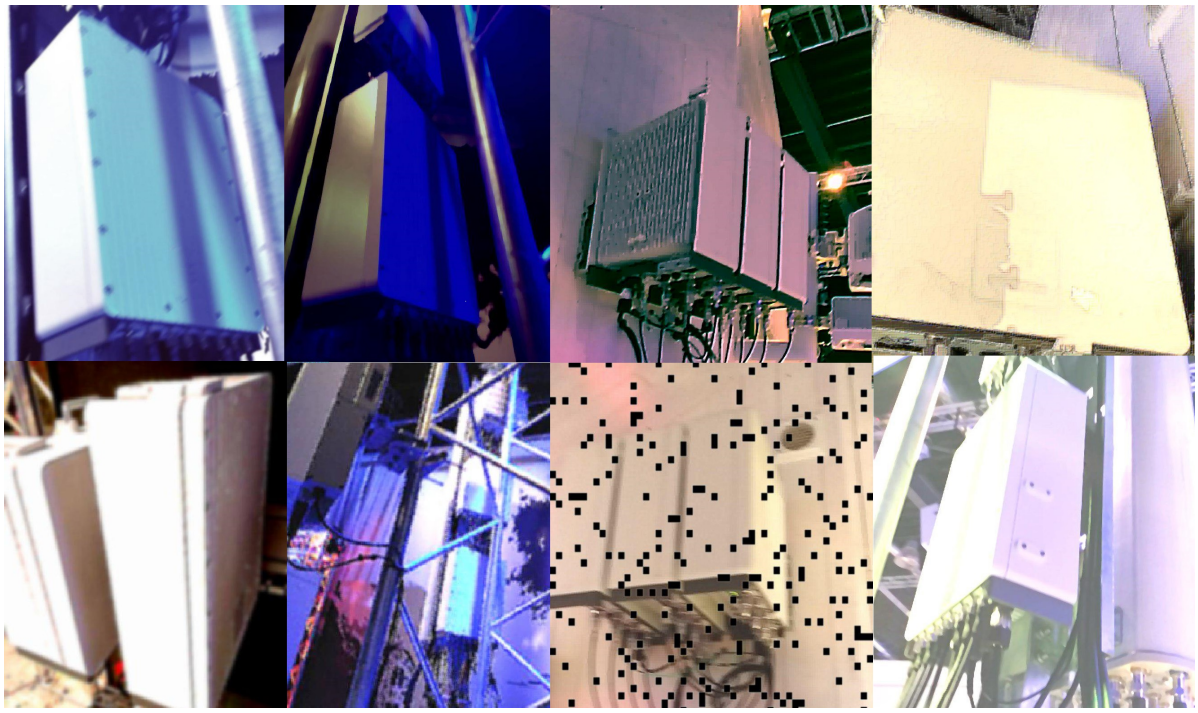

Figure 3.5: Sample images from the augmented dataset

3.4 Experimental methodology

The main purposes of experiments are to assess the performance of the object detection algorithms and to evaluate their viability for execution environments with limited computational capabilities (e.g. smartphones) in terms of execution speed. The prototype system is tested through two main experiments:

1. **Single-class detection**: the CNN model is trained on both the model of Ericsson RRUs present in the dataset, and there is no distinction between Radio models, to which the semantic label 'RRU' is assigned. This experiment consists in a binary classification task, i.e. the result gives information only about the presence or the absence of an object of interest in the scene. It does not give any information about the presence of classes of objects in the scene different from the one on which the model has been trained. In this way, the misdetection probability is reduced to zero and it is possible to isolate the model's performance from misleading classifications.

2. **Multi-class detection**: the CNN model is trained on two models of Ericsson radio with separate semantic labels, namely 'RRU_4415' and 'RRU_2219'. This experiment takes into consideration the classification of different models of the same piece of equipment present in the scene. It is a multi-classification problem and the misdetection probability is different from zero, since there is the possibility to assign an incorrect label to an object. This experiment investigates the detection granularity at which is possible to distinguish different models of the same piece of hardware.

Due to the limited computational capability of the device on which the algorithm will run, the time performances of the detection and tracking algorithms running on a mobile computing platform will be measured. Also, when the object detection model is re-trained on the augmented dataset, a gain in terms of accuracy is expected. This gain is measured by repeating the same experiments on the same test-set and measuring the accuracy gain with respect to the training on the baseline dataset.

3.5 CNN model training

The tiny YOLOv2 model is trained using the open-source library Darknet [68]. The fine-tuning transfer learning approach is employed (see Section 2.2.2.16). In particular, the tiny YOLOv2 base model, originally trained on the Imagenet dataset [53], is fine-tuned by loading the pre-trained model weights and retraining the network with the custom dataset described in Section 3.2. The darknet framework has a corresponding Python framework named Darkflow*, that is compatible with the TF environment. Darkflow

*https://github.com/thtrieu/darkflow

allows export of the Darknet model to TF for deployment on mobile devices, which was necessary to implement the proposed solution. The MobileNetV2 model is trained using the TF Object Detection API [104], an open source framework built on top of TF that allow to construct, train and deploy object detection models. Fine tuning is executed on the MobileNetV2 weights pre-trained on the COCO [79] dataset, from the TF API's model zoo. The training is conducted on the development platform described in Section 3.1.4, while the actual detection will run on the mobile platform described in Section 3.1.2. The implementative details of the training procedure are further discussed in Section 4.2.

3.6 Evaluation methodology

Each model is evaluated by its performance over the test set using different statistics (accuracy, precision, recall, etc.). For this project, the metrics to evaluate the detection performances are primarily detection accuracy and inference time (computed in FPS at which the algorithm is running on the mobile computing platform). The most common metrics used in the context of object detection and classification, based on the PASCAL VOC competition [80], will be presented in the following text.

3.6.1 Intersection over Union IoU

The IoU is a measure based on Jaccard's Index [105] that evaluates the overlap between two bounding boxes. It requires a ground truth bounding box B_g (i.e. a hand labeled bounding box that specifies where the target object is located in the training set image) and a predicted bounding box B_p (i.e. the bounding box as predicted by the object detection model).

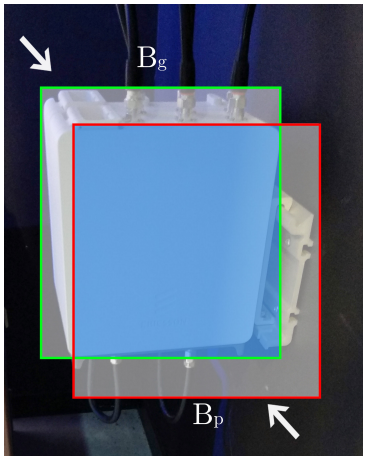

Figure 3.6: IoU on a dataset sample

In Figure 3.6 the predicted box (in red) and the ground truth box (in green) are shown on a sample image from the project's dataset with the union of the boxes indicated in light blue and the intersection in white. Formally, the IoU is defined as the ratio between the area of the intersection and the area of the union of the ground truth bounding box and the predicted bounding box:

$$\text{IoU} = \frac{B_g \cap B_p}{B_g \cup B_p} \tag{3.4}$$

3.6.2 Detection classification

Each detection event is classified according to four categories, namely: *true positive*, *true negative*, *false positive*, and *false negative*. These categories are used to compare the results of the object detector under test with trusted external judgments. The terms positive and negative refer to the actual detection (expectation), and the terms true and false refer to whether that detection corresponds to the external judgment (observation). In the context of this project, the following definitions will be used:

- **True Positive** (TP) represents a correct detection event, i.e. a detection of an instance that is actually present in the image.
- **False Positive** (FP) represents an incorrect detection event, i.e. a detection of an instance that is not present in the image.
- **False Negative** (FN) represents a missed detection event, i.e. the instance of an object is present in the image but is not detected.
- **True Negative** (TN) is inapplicable in this context. It would represent a correct misdetection. In the object detection task there are many possible bounding boxes that should not be detected within an image. Thus, TN would represent all possible bounding boxes that were correctly not detected (so many possible boxes within an image). For this reason, this category is not used in this context.

In order to determine whether a prediction is a TP, FP or FN a threshold t_{IoU} is set on the IoU to identify if the object is actually detected or not. This means that if the IoU $\geq t_{\text{IoU}}$ the detection is considered detected, otherwise it is considered as non detected. Different threshold values have been used in the literature: the PASCAL VOC evaluation metric [106] suggests a threshold value $t_{\text{IoU}} = 0.5$ while the COCO evaluation metric [79] recommends measurement across various IoU thresholds ranging from 0.05 to 0.95. In this project the PASCAL VOC approach is employed. Another factor affecting performance is the confidence level of each detection $\mathbb{P}[\text{det}]$ and the detection confidence threshold t_{det}. All detections with a confidence level higher than this threshold $\mathbb{P}[\text{det}] > t_{\text{det}}$ are considered as positive detection, otherwise they are considered as false detection.

3.6.3 Precision and Recall

Precision, also named positive predictive value (PPV), measures how accurate the predictions are, i.e. the percentage of positive predictions that are correct. Precision is defined mathematically as the ratio between the number of true positives $\#(TP)$ and the number of total detected instances $\#(TP) + \#(FP)$

$$\text{Precision} = \frac{\#(TP)}{\#(TP) + \#(FP)} = \frac{\#(TP)}{\text{all detections}} \tag{3.5}$$

The main limitation of this metric is the fact that does not give any information about how many actually existing objects are *undetected* in the scene.

Recall, also named True Positive Rate (TPR) or sensitivity, measures how many relevant objects are detected in the scene. Recall is defined mathematically as:

$$\text{Recall} = \frac{\#(TP)}{\#(TP) + \#(FN)} = \frac{\#(TP)}{\text{all ground truths}} \tag{3.6}$$

The main limitation of this metric is the fact that does not give any information about the actual number of TPs that are correctly detected.

Precision and recall metrics are related by an inversely proportional relationship and in order to overcome their respective limitations, they are discussed together. It is desirable to maximize both precision and recall, so that few instances are incorrectly classified while at the same time few instances that should have been classified are left out. For these reasons, the precision versus recall curve is a good way to evaluate the performance of an object detector. An object detector is considered good if it can identify only relevant objects ($\#(FP) = 0$ means high precision), finding all ground truth objects ($\#(FN) = 0$ means high recall). A bad object detector needs to increase the number of detected objects (increasing ($\#(FP)$ to lower the precision) in order to retrieve all ground truth objects and obtain high recall. This is the reason why the Precision-Recall curve starts with high precision values, decreasing as recall increases.

3.6.4 mean Average Precision mAP

The mAP is a good performance metric for the object detection task because it summarizes the Precision and Recall metrics in a single scalar value to evaluate the detection capabilities of the model. The mAP metric was first introduced in the context of the PASCAL VOC recognition challenge [80] and has become the de facto standard for the evaluation of object detection algorithms. This metric is computed as a mean across all the classes C of the AP

$$\text{mAP} = \sum_{i=1}^{C} \text{AP}_i \tag{3.7}$$

The AP summarizes the shape of the precision-recall function $p(r)$ that is computed for each class at eleven recall levels $\mathcal{L} = \{0, 0.1, ..., 1\}$ and is defined as:

$$\text{AP} = \frac{1}{|\mathcal{L}|} \sum_{r \in \mathcal{L}} \text{AP}_r = \frac{1}{|\mathcal{L}|} \sum_{r \in \mathcal{L}} p_{\text{interp}}(r) \tag{3.8}$$

The precision at each recall level r is interpolated by taking the maximum precision measured for which the corresponding recall exceeds r on the precision-recall curve.

$$p_{\text{interp}}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{3.9}$$

AP is related to the area under a precision-recall curve. It is desirable for this area to be large in order for both precision and recall to be maximized.

3.6.5 Model inference time

The inference time T_{inf} is the time taken to execute the detection task on a single frame of the video sequence. It is one of the key parameters to evaluate the model's performance, especially for embedded device applications with limited computational resources. It is usually expressed in terms of statistical mean μ and standard deviation σ by considering a large number of measurements of the inference time.

Ideally, to have a smooth real-time experience, the object detection algorithm should run at the same speed at which the human visual system perceives the sequence of frames as a continuum, thus it should occur at the FPS rate at which the video is recorded. For example, if a video is recorded at 24 FPS the inference time should be $T_{\text{inf}} = \frac{1}{\text{FPS}} = 41.6\text{ms}$. Reaching such a speed in a typical embedded device is considered unfeasible. For this reason, the integration of the object detector with the object tracker is necessary. As shown in the results in Chapter 6, the chosen tracker algorithm is able to run in real-time and thus compensate for the object detection inference time.

3.7 Choice of Object Detection and Tracking algorithms

In this section the choice of the object detection and tracking algorithms is discussed on the basis of the system functionalities that the final system must provide and of the platform limitation and constraints. Also the proposed algorithms are discussed in detail in the next subsections. In order to effectively assist the radio-site engineers to detect the correct hardware, the object detection and tracking algorithms have to satisfy two essential requirements:

1. The detection system needs to run in real-time to support the cell-site engineers' decision making process of which hardware equipment they wish to inspect.

2. The algorithm has to run on a low power embedded mobile system (Android device) and hence must be computationally lightweight.

The main challenge consists in realizing a system satisfying these two conflicting requirements, while achieving an acceptable level of detection accuracy. Considering these requirements and looking at the use-cases for similar systems (as reviewed in Section 2.3), two object detection algorithms are tested, namely Tiny YOLOv2 [69] and SSD-MobileNetV2 [13]. Although the proposed object detection models are considered lightweight and fast with respect to other CNNs, such as region-based algorithms (R-CNNs) [48, 58], they are still unable to run in real-time on a low-powered embedded device. For this reason, the proposed solution integrates the object detection module with a real-time KLT tracker [38], that is able to run at a much higher FPS with respect to the object detector.

Figure 3.7 shows the processing pipeline for the detection and tracking of the radio-site hardware and the high-level interaction between tracker and detector.

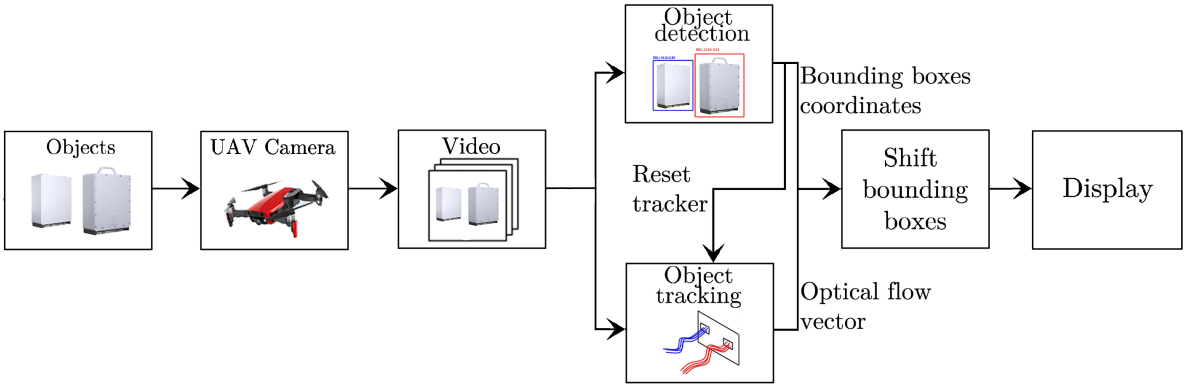

Figure 3.7: Hardware inspection processing pipeline

The object detection and tracking algorithms run in parallel on separate threads. As soon as a detection instance is successfully processed, the results in terms of bounding boxes and confidence level it is displayed. The object tracking role is to compute the optical flow vector generated by the relative movement between the objects in the scene and the UAV, during the computation of the detection instance. Since the employed tracking algorithm is able to run in real-time (see Section 5.2), the overall algorithm will also run in real-time because the time performance will be determined by the speed of the tracking algorithm. The detector is assisted by the tracker that shifts the previous bounding boxes related to the previous detection by the optical flow vector computed by the object tracking algorithm. The tracker is reset each time a detection is executed and also the optical flow vector is reset to zero. This will allow the tracker to achieve better accuracy since new updated feature points are re-computed each new detection on the same frame where detection is executed.

3.7.1 YOLO and YOLOv2

The YOLO algorithm was developed in 2015 by J. Redmon and A. Farhadi [12]. It was able to detect bounding boxes and associate class probabilities to each bounding box using a single CNN directly from full images in one evaluation. The original YOLO architecture is shown in Figure 3.8

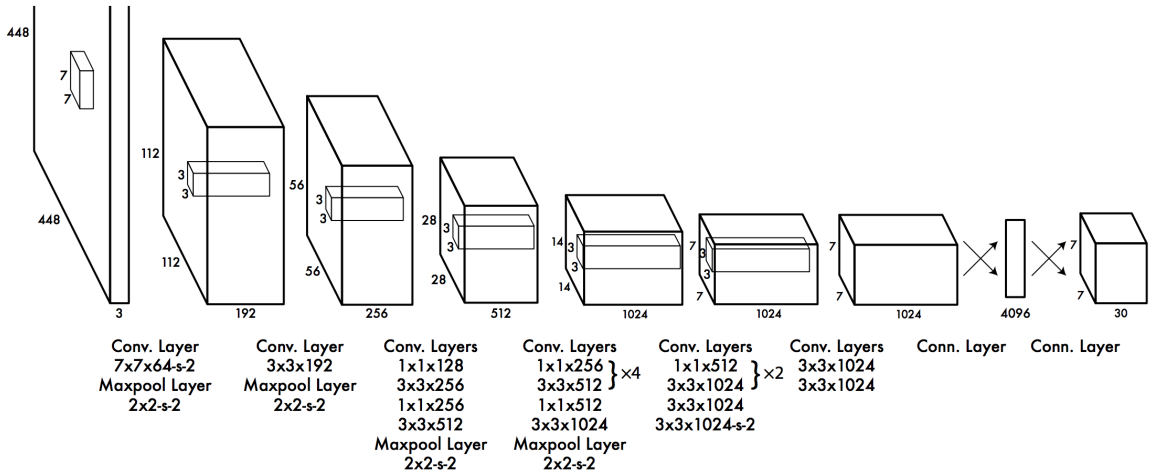

Figure 3.8: YOLO architecture (Image courtesy of Joseph Redmond [12])

The original architecture has 24 Convolutional layers followed by 2 connected layers and was designed for use in the Pascal VOC dataset [80] with $C = 20$ objects' classes. The YOLO CNN takes as input a $448 \times 448 \times 3$ tensor representing the color image and outputs a $7 \times 7 \times 30$ prediction tensor. The algorithm divides the input image into a $S \times S$ grid ($S = 7$ in the original architecture), if a grid cell contains the center of an object, then the grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes ($B = 2$ in the original architecture) and computes the box confidence scores Ψ for all the boxes. The output has dimensions $S \times S \times (5 + C)$. The confidence scores reflect how confident the model is that a given box contains the object and is expressed by the following equation:

$$\Psi = \mathbb{P}\{\text{Object}\} \cdot \text{IoU} \tag{3.10}$$

The first term on the right-hand side is the probability that the box contains an object, while the second term is named Intersection Over Union (IoU) and it is used as a metric for evaluating the algorithm's performance. From each bounding box the YOLO algorithm outputs 5 predictions: $(x, y, w, h, p(c))$, where (x, y) represent the center of the bounding box, (h, w) are the width and height from the center coordinates' and $p(c)$ is the class confidence. Given the number of classes C, the conditional class probability given that the object is detected per each grid cell is computed $\mathbb{P}\{\text{Class}_i|\text{Object}\}$.

Then YOLO computes the class confidence score for each prediction box $p(c)$ as:

$$p(c) = \mathbb{P}\{\text{Class}_i|\text{Object}\} \cdot \Psi = \mathbb{P}\{\text{Class}_i|\text{Object}\} \cdot \mathbb{P}\{\text{Object}\} \cdot \text{IoU} \tag{3.11}$$

The activation function used in each layer is the leaky ReLU, defined as:

$$g(x) = \begin{cases} x & \text{if } x > 0 \\ 0.1x & \text{otherwise} \end{cases} \tag{3.12}$$

and the final layer uses a linear activation function ($g(x) = x \quad \forall x$). The cost function $\mathbb{C}$ used in the YOLO algorithm is the following:

$$\begin{aligned} \mathbb{C} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 - (y_i - \hat{y}_i)^2 \right] + \\ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{\omega}_i - \sqrt{\hat{\omega}_i}\right)^2 - \left(\sqrt{h}_i - \sqrt{\hat{h}_i}\right)^2 \right] + \\ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 + \\ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i\right)^2 + \\ \sum_{i=0}^{S^2} \mathbb{1}_{ij}^{noobj} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2 \end{aligned} \tag{3.13}$$

It is composed of different parts:

1. The first term computes the loss related to the predicted bounding box position $(\hat{x}, \hat{y})$ with respect to the true positions (x, y)

2. The second term computes the loss related to the predicted bounding box width and height $(\hat{w}, \hat{h})$ with respect to the true positions (w, h). The square root is used because the error metric should reflect that small deviations in large boxes matter less than in small boxes as explained in the paper [12]

3. The third and fourth terms represent the loss associated with the confidence score for each bounding box predictor. C is the confidence score and $\hat{C}$ is the IoU of the predicted bounding box with the ground truth box provided in the labeled examples

4. The last term is the classification loss. It looks similar to a normal sum-squared error for classification, except for the $\mathbb{1}_{ij}^{obj}$ term that is 1 when there is an object in cell i and 0 elsewhere and is used to avoid penalizing classification error when there is no object in the cell.

The terms $\mathbb{1}_{ij}^{obj} = 1$ if an object is present in the i^{th} grid cell and the j^{th} bounding box predictor is responsible for that prediction, otherwise $\mathbb{1}_{ij}^{obj} = 0$ and $\mathbb{1}_{ij}^{noobj}$ is the exact opposite. The λ hyper-parameters represent penalties given to each term and are necessary to increase model stability. YOLO is able to obtain an accuracy of 63.4% on a variety of objects but exhibits several limitations: YOLOv2 is build on top of the concepts introduced for the YOLO algorithm. However a set of modifications were made to the original architecture to overcome some of YOLO's limitations and further improve performances. These modifications are:

- **Batch normalization** [107] is added at the input of each convolutional layers. This batch normalization normalizes each dimension of the input tensor before it is processed by the convolutional layer. Batch normalization allows use of a higher learning rate during training and makes the system less sensitive to the weights used at initialization.

- **Multi-Scale Training** performs training on various input dimensions in order to achieve a good trade-off between accuracy and speed depending on the requirements of the application.

- **Convolution with anchor boxes** predicts the coordinates of bounding boxes through anchor boxes, in contrast to YOLO where this was done through a fully connected layer and the initial boxes dimensions were randomly initialized. In YOLOv2, 5 anchor boxes are defined and offsets with respect to the dimensions of these standard boxes are predicted. This increases the variability of the bounding boxes and allows to detect smaller objects in the scene.

- **k-means clustering** [108] exploits correlation between bounding boxes. This allows the network to learn and adjust the anchor box's dimension appropriately according to the detected objects' dimensions.

As YOLOv2 requires large amounts of computational power for inference, an architecture developed and optimised for use on embedded systems and mobile devices is employed, specifically the Tiny YOLOv2 network [69]. Tiny YOLOv2 network is inferior to the full YOLO networks in terms of accuracy but runs at significantly higher FPS. As computational power is a limiting factor for mobile devices, the Tiny YOLOv2 network is a good candidate for the application this thesis aims to realize The proposed Tiny YOLOv2 CNN structure is presented in Table 3.4. The input images are resized from their original size to a dimension of $416 \times 416 \times 3$, as that is the input tensorial size for the YOLOv2 model, and the total number of layers is reduced from 24 to 15. This allows faster detection and reduces the computational complexity of the networks in terms of bias and weights to optimize during the training process.

Table 3.4: Tiny YOLOv2 CNN architecture

Layer name	Kernel size	Output size	Layer name	Kernel size	Output size
Conv0	3x3x16	416x416x16	Conv8	3x3x256	26x26x256
MaxPool1	2x2x16	208x208x16	MaxPool9	2x2x256	13x13x256
Conv2	3x3x32	208x208x32	Conv10	3x3x512	13x13x512
MaxPool3	2x2x32	104x104x32	MaxPool11	2x2x512	13x13x512
Conv4	3x3x64	104x104x64	Conv12	3x3x1024	13x13x1024
Max5	2x2x64	52x52x64	Conv13	3x3x1024	13x13x1024
Conv6	3x3x128	52x52x128	Conv14	1x1x30	13x13x30
MaxPool7	2x2x128	26x26x128	Detection (FC)	/	/

3.7.2 SSD MobileNet

SSD-Mobilenet is an efficient CNN architecture specifically designed for object detection on mobile applications. It is based on the combination of two sub-networks: MobileNet [75], used as feature extractor for classification of the objects present in the image, and SSD [78], used for localization of the classified objects. The combination of classification and localization provides object detection.

MobileNets are a class of CNNs developed by Howard et al. [13, 75] at Google. The key idea behind the efficiency of this network architectures is to reduce the computational cost of the convolution operation, that is considered the most expensive operation during inference. The authors of MobileNets achieved this by proposing a new form of convolution named *depthwise separable convolution*. The depthwise separable convolution factorizes a standard convolution into a depthwise convolution and a 1×1 pointwise convolution. The depthwise convolution applies a single filter to each input channel (input depth). The 1×1 pointwise convolution is then used to create a linear combination of the output of the depthwise layer. This factorization has the effect of drastically reducing computational cost and model size.

Considering the input feature map $\mathbf{x} \in \mathbb{R}^{W \times W \times C}$ and assuming the use of square kernel filters $\mathbf{f} \in \mathbb{R}^{S \times S \times C \times D}$, the standard tensorial convolution presented in Equation 2.19 has a computational complexity equal to:

$$\gamma_1 \triangleq S \cdot S \cdot C \cdot D \cdot W \cdot W \tag{3.14}$$

The depthwise separable convolution is expressed by the following formula

$$\hat{y}_{k,l,c} = \sum_{i=1}^{S} \sum_{j=1}^{S} \hat{f}_{i,j,c} \cdot x_{k+i-1,l+j-1,c} \tag{3.15}$$

where $\hat{f}$ is the depthwise convolutional kernel of size $S \times S \times C$ where the c^{th} filter in $\hat{f}$ is applied to the c^{th} channel in x to produce the c^{th} channel of the filtered output feature

map $\hat{y}$. The computational cost for this operation is $S \cdot S \cdot C \cdot W \cdot W$. However, this operation only filters input channels and does not combine them to create new features. The additional layer that computes a linear combination of the output of depthwise convolution via 1×1 convolution is needed in order to generate these new features and the total computational cost will be:

$$\gamma_2 \triangleq S \cdot S \cdot C \cdot W \cdot W + C \cdot D \cdot W \cdot W \tag{3.16}$$

The reduction in computation is given by:

$$\gamma \triangleq \frac{\gamma_2}{\gamma_1} = \frac{1}{D} + \frac{1}{S^2} \tag{3.17}$$

Since MobileNet uses 3×3 kernel filters, the reduction in terms of computation is between 8 and 9 times. Also, batch normalization [107] and ReLU6 activation function (same as ReLU but saturates at the value 6) are used after both depthwise and pointwise layers.

Their model uses two global hyper-parameters, *width multiplier* α and the *resolution multiplier* ρ, in order to tune the design requirements for mobile and embedded vision applications. The former has the goal of thinning the network uniformly at each layer by a factor α; the latter parameter concerns the choice of the input resolution and is implicitly defined by setting the input resolution to a certain value. Different combinations of these hyper-parameters were experimented by the authors of MobileNets, and offer compromises between computational speed and detection accuracy. The original MobileNet architecture has an input size of $224 \times 224 \times 3$ and is composed of 28 layers, counting pointwise and depthwise as separate layers, and achieves 70.6% of detection accuracy on the ImageNet dataset (with $\alpha = 1$) with approximately 569 million Multiply-Add (MAdd) operations and 4.2 million parameters. MobileNetV2 improves the performance of MobileNet by introducing two new features to the architecture: *inverted bottleneck blocks* and *residual connections*. The combination of the two is named *inverted residual blocks*. Standard residual blocks [109] connect the beginning and end of a convolutional layer with a skip connection, by following a wide-narrow-wide approach while inverted residual blocks follow the opposite approach i.e. narrow-wide-narrow (see Figure 3.9).

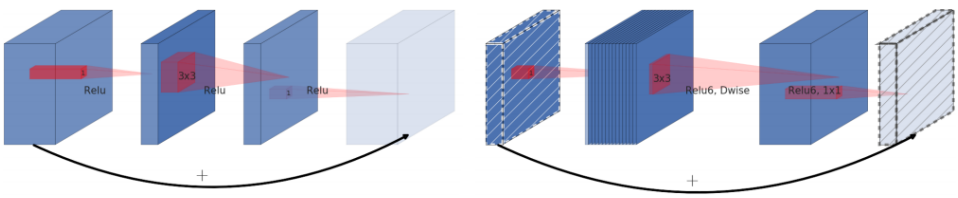

Figure 3.9: Left: Residual blocks. Right: inverted residual blocks [13]).

Inverted bottleneck blocks are composed by three convolutional layers represented in Figure 3.10: the expansion layer (1×1), the depthwise layer (3×3), and the pointwise convolution layer (1×1).

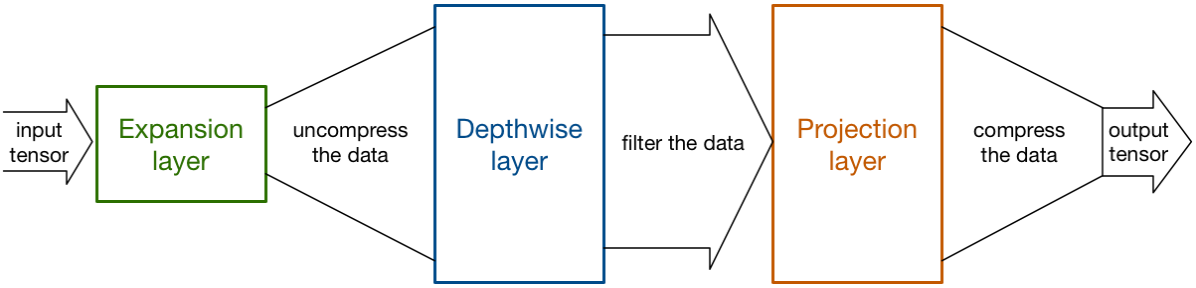

Figure 3.10: Inverted bottleneck blocks. (Image courtesy: Matthijs Hollemans [14])

The last two blocks are the same that are found in the MobileNetV1 architecture. In the MobilenetV2 case, the pointwise convolution makes the number of channels smaller and is also known as projection layer because projects higher dimensional data structures number into tensors with a lower number of dimensions. The purpose of the first block is to expand the number of channels before the data go into the depthwise convolution, i.e. has the opposite role with respect to the projection layer.

Residual connections take inspiration from the ResNet architecture [109] and are only active when the number of channels at the input of the inverted residual block is the same as the number of the output channels. These connections have proved to be a good technique to increase the depth of the network: similarly to traditional residual connections, shortcuts enable faster training and better accuracy. The MobileNetV2 architecture is shown in Table 3.5

In table 3.5, the parameter t stands for expansion rate of the channels (factor of 6), c represents the number of input channels, n how many times the block is repeated, and S indicates the stride of the first repetition of a block (all the others use $S = 1$). Also, all the convolution kernels have 3×3 dimension.

SSD Multibox detector [78] is a CNN designed to be independent of the base network and is able to run on top of MobileNet as detector. SSD MultiBox loss function is composed by two main components:

$$L_{tot} = \frac{1}{N} L_{conf} + \alpha L_{loc} \tag{3.18}$$

where L_{loc} is the localization loss that takes into account the positions ground truth and the predicted box, and the confidence loss L_{conf} this measures of the objectness of the computed bounding box. The alpha term is an hyper-parameter that balances the contribution of these two losses. The author of MobileNet developed a mobile

Table 3.5: MobileNetV2 CNN architecture

Layer name	Input size	t	c	n	S
Conv	$224 \times 224 \times 3$	/	32	1	2
Bottleneck	$112 \times 112 \times 32$	1	16	1	1
Bottleneck	$112 \times 112 \times 16$	6	24	2	2
Bottleneck	$56 \times 56 \times 24$	6	32	3	2
Bottleneck	$28 \times 28 \times 32$	6	64	4	2
Bottleneck	$14 \times 14 \times 64$	6	96	3	1
Bottleneck	$14 \times 14 \times 96$	6	160	3	2
Bottleneck	$7 \times 7 \times 160$	6	320	1	1
Conv	$7 \times 7 \times 320$	/	1280	1	1
Avg Pool	$7 \times 7 \times 320$	/	/	1	/
Conv	$1 \times 1 \times 1280$	/	k	1	/

friendly variant of regular SSD named SSDLite that replaced all the regular convolutions with depthwise separable convolutions in the SSD prediction layers. In this way they dramatically reduces both parameter count and computational cost from 1.25 billions of MAdd to 0.35 billions of MAdd for the SSDLite. In order to integrate the two architectures, the first layer of SSDLite is attached to the expansion of layer 15.

3.7.3 KLT tracking algorithm

The KLT tracker implements an optical flow to track points detected through Harris corner detection algorithms in consecutive frames of a video, producing an optical flow vector for each detected corner point. The steps of the algorithm are the following:

1. Detect Harris corners (Section 2.2.1.1) in the first frame of the video. This step is used to reduce the amount of processed data for motion tracking, since the number of corners found through the Harris detector is only a small percentage of the image.

2. For each detected Harris corner, compute the optical flow between consecutive frames using the Kanade-Lucas method (Section 2.2.1.3). The optical flow vectors are computed for each detected Harris corner point, and then all the motion vectors are averaged to obtain the mean motion vector.

3. Link the motion vectors from frame to frame to track for all the corners.

4. Generate new Harris corners after a specific number of frames to compensate for new points entering the scene or to discard the ones going out of the scene. This step is executed every time a new detection is available.

5. Track the new Harris points.

Chapter 4

Implementation

This chapter presents the implementation of the system architecture introduced in Section 3.1 and describes the detailed procedure for the dataset preparation (Section 4.1), the object detection model training (Section 4.2), and the application development process (Section 4.3). The implementation procedure is exemplified on a specific type of hardware equipment, namely the Ericsson Radio models 4415 and 2219, but aims to be a general procedure that could be extended to different objects typologies.

4.1 Dataset preparation

The dataset preparation is a crucial step in the development of the object detector employed in the project solution. The dataset is the primary source of information for the detector and allow to train, validate and test the developed CNN model. The main steps for the preparation of this dataset are reported in this section.

4.1.1 Frames extraction

The raw dataset consists of videos with an aspect ratio of 1080×1920 at 24 FPS in MP4 format. Since the training process takes as input training samples of image data, individual frames need to be extracted from these videos. In order to extract these frames an open source software library named FFmpeg [110] (used to handle multimedia files streams) was used to extract the frames from the video named video_name.mp4. The form of the command is the following:

```
ffmpeg -i video_name.mp4 -vf fps=6 frame%04d.jpg -hide_banner
```

This command produces as output all of the frames of the video ranked in ascending numerical order as individual files numbered with four digits. The speed at which the frames have been extracted from the stream (6 FPS) has been determined empirically

by trying different values of FPS extracted and trading off low redundancy across frames and large number of dataset samples. A further selection is made of the frames before being labeled, in order to discard blurred and redundant frames.

4.1.2 Dataset labeling

The labeling operation consists in delimiting object instances of the classes to be detected with bounding boxes and then assigning a semantic label to each of these boxes. The number of labels is equal to the number of possible object classes to detect. In the present case, two class of objects are considered for detection the semantic labels assigned to these classes are 'RRU_4415' and 'RRU_2219', from based upon Ericsson's radio hardware names. The boxes generated at this step are the ground truth boxes used for the computation of the IoU (see Section 3.7.1). The labeling process has been executed with Ericsson proprietary software provided by Volodya Grancharov and his research team. The tool creates a bounding box around the detected object and stores the labeling result either in the standard *txt* YOLO format (see Equation 4.1) or in the MobileNetV2 label format, that is a *xml* file, whose structure is presented in Listing 4.1.

$$[\text{class number}] \left[\frac{\text{x center}}{\text{image width}}\right] \left[\frac{\text{y center}}{\text{image height}}\right] \left[\frac{\text{box width}}{\text{image width}}\right] \left[\frac{\text{box height}}{\text{image height}}\right] \quad (4.1)$$

Listing 4.1: MobileNetv2 label format

```
<annotation>
    <folder>folder_name</folder>
    <filename>file_name</filename>
    <path>path_to_file</path>
    <size>
        <width>width_size</width>
        <height>heigth_size</height>
        <depth>depth_size</depth>
    </size>
    <object>
        <name>object_name</name>
        <bndbox>
            <xmin>\textit{x_min_bounding_box}</xmin>
            <ymin>y_min_bounding_box</ymin>
            <xmax>x_max_bounding_box</xmax>
            <ymax>y_max_bounding_box</ymax>
        </bndbox>
    </object>
</annotation
```

The labeling tool provided by Ericsson speeded up the labeling procedure by means of a tracking algorithm that allowed to label objects in consecutive video frames semi-automatically. The first frame was manually labeled by drawing bounding boxes around the objects of interest. Subsequently, the expected position of the box in the next frames were computed. When the detection of a given object was lost, it should be inserted again through the manual procedure. Figure 4.1 presents examples of labeled images used in this project.

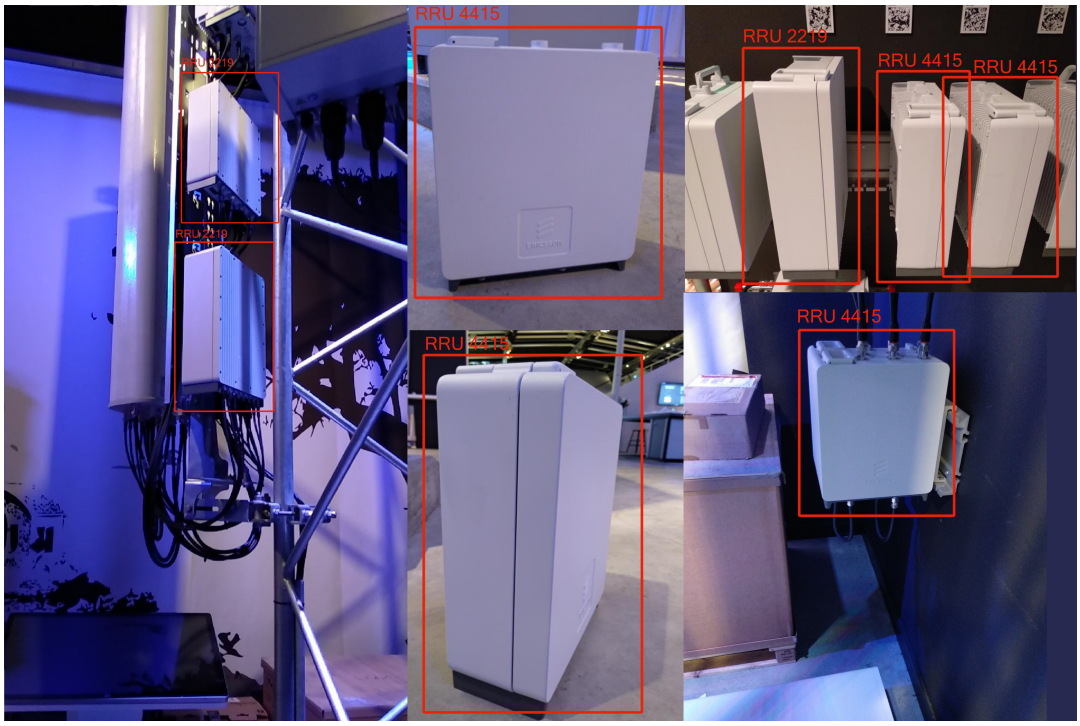

Figure 4.1: Examples of labeled images

Once all the frames are labeled, the dataset should be split into *training set*, *test set*, and *validation set*. An *ad hoc* python script was developed to randomly take 70% of the dataset samples of training, 15% for validation, and 15% for test. This is

4.1.3 Dataset augmentation

In order to make the object detection algorithm more robust against different situations in which the UAV may operate the available images are manipulated as presented in Section 3.3 to extend the dataset. An open-source Python library for image augmentation named imgaug* is used in for this purpose. The library functions used to augment the dataset for the current case are the following:

*https://github.com/aleju/imgaug

- *Multiply*($\xi_{min} = 0.5$, $\xi_{max} = 1.5$) is used to randomly increase or decrease the brightness of the image between 50% and 150% of the original value.
- *AdditiveGaussianNoise*($L = 0$, $S = 33.15$, $\mathbb{P}[ch] = 0.7$) adds white Gaussian noise from a normal distribution $n \sim \mathcal{N}(L, S)$ pixelwise to the baseline image. $\mathbb{P}[ch]$ is the probability that the sampled noise values is different for each image channel.
- *Sharpen*($\alpha = [0.3,\ 1]$,$L = [0.75,\ 1.5]$) and *Emboss*($\alpha = [0,\ 1]$,$S = [0.3,\ 2]$) runs a sharpening or embossing kernel with lightness L or with strength S. Mixes the result with the original image using α.
- *GaussianBlur*($\sigma = [1.3, 3.3]$) blurs the input image by applying a Gaussian kernel with variance σ, that randomly varies between the specified values for each image.
- *CoarseDropout*($\mathbb{P}[drop] = (0.03, 0.15)$,$S = (0.02, 0.05)$) sets pixels to zero with probability $\mathbb{P}[drop]$. Samples the locations of pixels that are to be set to zero randomly on the image with area is between 2% and 5% of the whole image.

4.2 Training

The training is the key process that enables the CNN model to *learn* to detect objects in the video data feed produced by the UAV. This section presents the steps to carry out the CNN model training procedure. It describes the configuration of the training parameters and present the output of the training procedure .

4.2.1 YOLO setup

The first step to train the YOLOv2 CNN model is to download and install *Darknet*, the YOLO CNN framework [69] used for training and detection, on the computing platform to be used for training. Darknet is cloned from the AlexeyAB's GitHub repository*. The setup instructions, together with the dependencies and requirements to be satisfied are extensively described at the repository's website. In particular, for this project the requirements listed in Table 4.1 have been satisfied.

Table 4.1: Darknet requirements

Requirement	Version
Linux GNU Compiler Collection (GCC) [111]	5.4.0
Compute Unified Device Architecture (CUDA)	9.0.176
CUDA Deep Neural Network library (cuDNN)	7

*https://github.com/AlexeyAB/darknet

4.2.2 MobileNet setup

The MobileNet training and testing environment is based on the TF Object Detection API*, an open source framework built on top of TF. First, the TF requirements are satisfied (see Table 4.2) and the setup instructions were performed as described in the repository.

Table 4.2: TensorFlow requirements

Requirement	Version
Tensorflow-GPU	1.9
Python	3.6.6
Protocol buffer	3.0.0

4.2.3 YOLO training

The YOLO framework makes available different types of CNN architectures, which differ in their choice of hyper-parameters (such as number of layers, stride, padding, number of filters per layer, etc.). As stated earlier, pre-trained weights from the training of the Tiny YOLOv2 algorithm on the Imagenet dataset [53] are used. These weights are downloaded from the official YOLO website by the following command:

```
$ git wget https://pjreddie.com/media/files/darknet19_448.conv.23
```

These weights are based upon the training of a modified version of GoogleNet [112] and will be the starting point for fine-tuning the model on the actual training dataset. The configuration files needed for this training are:

1. The **cfg** file contains the CNN's structure and the network's hyper-parameters (the complete *cfg* file is provided in Appendix **??**).

```
batch=64
subdivisions=2
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
learning_rate=0.001
scales=.1,10,.1,.1
...
```

*https://github.com/tensorflow/models/blob/master/research/object_detection

The major changes that are done to this configuration file are the following: the number of classes is set to the actual number (2 classes for the multi-label case, 1 for the case of single-class training), the filters variable of the last convolutional layer is set to $filters = (C + 5) \cdot 5$ in order to resize the network coherently with the number of classes.

2. The **data** file contains information about the number of classes, training dataset, validation dataset, class labels, and output directory to save the results of the training (in the backup variable).

```
classes=2
train=/path_to_training_label_file
valid=/path_to_validation_label_file
names=/path_to_label_name_file
backup=/path_to_training_weights_backup
```

3. The **names** file contains the objects' names, i.e. the label associated with each bounding box.

```
RRU_4415
RRU_2219
```

The input size is 416×416 and any image is resized to this dimension before being processed by the CNN. The YOLO architecture is able to efficiently and accurately perform the object detection task. However, the correct choice of hyper-parameters needs to be done in order to train the CNN effectively. *Batch learning* (see Section 2.2.2.4) is used as the learning method. Since the computing platform can run out of memory during the training process, the batch size is further split in subdivisions. These subdivisions are used to divide the work and limit memory usage. In general, if the batch size is $\mathcal{B}$ and the subdivision is $\mathcal{S}$, then the GPU will process $\frac{\mathcal{B}}{\mathcal{S}}$ images at a time but will update weights only after S batches.

The hyper-parameters tuning process for the training of the YOLO model has been executed mostly on the batch, subdivision, and learning rate parameters. When the batch size is reduced, worse performance are achieved in terms of accuracy of the model. Increasing the subdivision size, leads to faster training, as more images are processed at once. However, by using the computing platform presented in 3.3 for training, batch with sizes greater than 8 lead to out of memory error. Different values of learning rate are also tested ($\eta = [0.0005 : 0.0005 : 0.005]$) and tests are run for the first 2000 steps, choosing the learning rate that shown the higher decrease of the cost function. After these trials the final configuration chosen for the considered hyper-parameters is $\mathcal{B} = 64$, $\mathcal{S} = 8$, and $\eta = 0.001$.

The command used to start the training process is:

```
./darknet detector train cfg/obj.data cfg/obj.cfg \
cfg/darknet19_448.conv.23 >> training.log
```

In this way the training log will be saved in the file *training.log*. The training is split according to the values of the parameters batch and subdivisions. Each row of the log file contains both a line for each batch (Figure 4.2) and for each subdivision (Figure 4.3).

```
12358: 0.084428, 0.088587 avg loss, 0.001000 rate, 1.007266 seconds, 790912 images Loaded: 0.297869seconds
```

Figure 4.2: YOLO batch log line

In the batch log line are reported the current training iteration (12358), the instantaneous loss (0.084428), the average loss (0.088587), the learning rate (0.001000), the iteration processing time (1.007266), and the number of processed image (equal to the training iteration number multiplied by the batch size $\mathcal{B}$ (in the figure this is 790912)).

```
Region Avg IOU: 0.829022, Class: 0.999219, Obj: 0.827850, No Obj: 0.014415, Avg Recall: 1.000000,  count: 12
```

Figure 4.3: YOLO subdivision log line

For each batch line, there are S subdivision lines containing the following information:

Region Avg IoU: average of the IoU of every image in the subdivision

Class: average of the probabilities that the class of the detected objects in this subdivision belong to their correct classes

Obj: confidence that the bounding boxes that must contain an object (ground truth) actually detected an object in them. The higher the better.

No Obj: same idea as *Obj*, but for locations where there must be no object.

Avg recall: computed as presented in Section 3.6

Count: number of objects detected in the current subdivision of images.

The parameters that are considered for training and evaluation are the batch and subdivision size, the number of iterations, the average IoU, and the detection threshold (that regulates the number of True Positives). The structure of the CNN is the same as described in Table 3.4. The training process is stopped when the loss reaches a given threshold and an *early stopping* technique was adopted for the validation set.

4.2.4 MobileNet training

The MobileNetV2 training procedure has been executed through the TF APIs. The pre-trained weights for MobileNetV2 are downloaded from the official TF API model zoo repository*. This collection contains different models pre-trained on different popular datasets. The one chosen for this project is the *ssd_mobilenet_v2_coco* pre-trained on the COCO [79] dataset. The model is supposed to achieve $mAP = 22\%$ and inference speed $T_{inf} = 31\ ms$ on 600×600 input image using a Nvidia GeForce GTX TITAN X GPU [75]. The input files for the training are *record* files, generated from the *xml* file described in Listing 4.1. Finally, the object detection training pipeline must be configured. This file defines which model and what parameters will be used for training. Two record files are created one for the training set (*train.record*) and another one for the test set (*test.record*). For training, a *.pbtxt* file is also created that contains the mapping of class names to class ID numbers. The file contains the following lines:

```
item {
  id: 1
  name: 'RRU_4415'
}
item {
  id: 2
  name: 'RRU_2219'
}
```

Finally, the configuration of the the object detection training *pipeline* file is executed (reported in appendix. This pipeline file defines which model and what parameters will be used for training, similarly to the YOLO *cfg* file (the complete *cfg* file is provided in Appendix **??**). The following changes are made to the original pipeline file in order to fine-tune the network on the custom dataset:

1. Change the variable *num_classes* to 2.
2. Specify in the variable *fine_tune_checkpoint* the *.ckpt* file downloaded from the TF model zoo.
3. In the *train_input_reader* section, change input_path and label_map_path variables to the path to the *.pbtxt* file created for the label map and the *train.record* file.
4. In the *eval_input_reader* section, change input_path and label_map_path variables to the path to the *.pbtxt* file created for the label map and the *test.record* file.

*https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Training is started by running the following command

```
python train.py --logtostderr --train_dir=training/ \
    --pipeline_config_path=training/ssd_mobilenet_v2.config
```

It is possible to check the progress of the training process by using *TensorBoard*. To do this, it is sufficient to enter the following command inside the TF object detection directory:

```
tensorboard --logdir=training
```

This will create a web page displayed on the local server on TCP port 6006. This page provides information and graphs that show how the training is progressing. The hyper-parameters considered for the training of the MobileNet architecture are mainly batch size $\mathcal{B} = 24$ and learning rate $= 0.004$ after some trial resulted to achieve the best training loss.

4.3 Android Application

The Android application has been developed using the Android Studio (version 23.0) Integrated Development Environment (IDE). Information about the Android requirements that are satisfied for this project are reported in Table 4.3.

Table 4.3: Android requirements

Requirement	Version
Android API	Android 8.1 API level 28
Android NDK	17.1.4828580
Android SDK	26.1.1

The application developed is named *Radio Site Assistant* (RSA) and this section provides a modular description of this application.

4.3.1 Registration/Connection module

This module is responsible for both the DJI SDK registration and the connection with the DJI product. These are essential steps that enable the use of the SDK in the Android application and allow to connect the application to the UAV. This module is implemented by the *ConnectionActivity* Java Class, whose GUI is shown in Figure 4.4. This activity is launched when the RSA application starts. As soon as the Android device is connected to the RC, the activity status changes from "No Product Connected" to "DJI Aircraft Connected" displaying the model and ID number of the connected UAV.

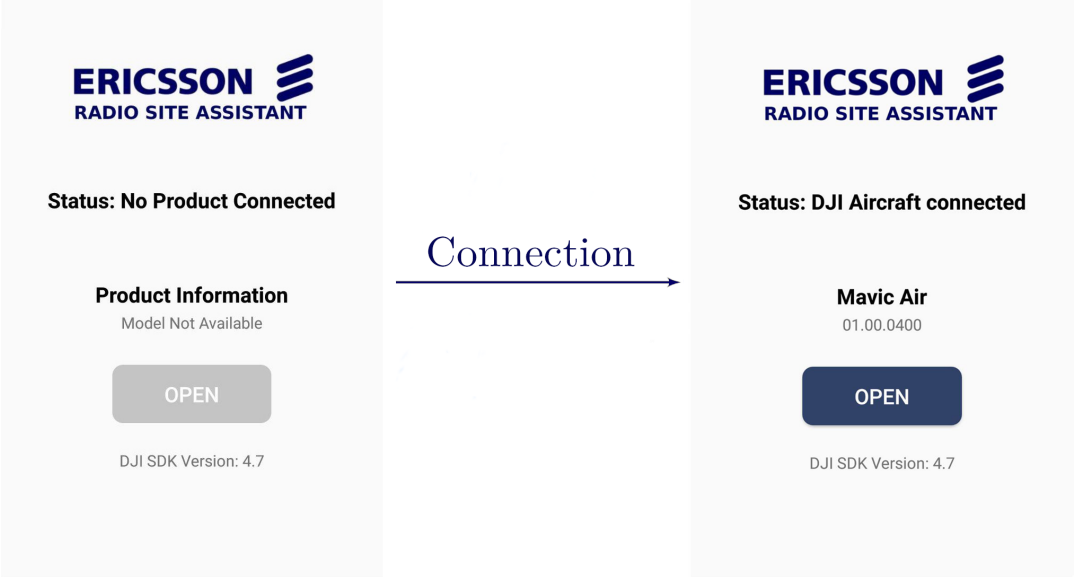

Figure 4.4: Connection Activity GUI

Also, when the connection is established it is possible to launch the *MainActivity* (described in the following sections) by clicking the open *Button* graphical element. The implementation of the registration is done by creating a new application on the DJI's developers website*. This will generate a 24-digit SDK activation key that is entered in the meta-data of the Android application manifest, inside the *android:value* field as shown in Listing 4.2.

Listing 4.2: DJI SDK application code

```
<meta-data
    android:name="com.dji.sdk.API_KEY"
    android:value="Enter_App_Key_here"
/>
```

4.3.2 Streaming acquisition module

The streaming acquisition module receives the raw YUV video stream from the UAV camera, converts it to RGB and passes it to a *TextureView* object to display the stream on screen of the Android device. After streaming is initialized in the method *initStreaming*, the acquisition and display occur by overriding the DJI callback method named *onSurfaceTextureAvailable* that is updated each time a frame is received.

*https://developer.dji.com/mobile-sdk/

4.3.3 Streaming pre-processing module

The streaming pre-processing module manages the video stream to create a suitable object for both the detection and tracking modules. A frame of streaming video is extracted directly from the *TextureView* object, transformed into a *Bitmap* object and resized to the object detection model input size ($S \times S$). There exists two main ways in which a *Bitmap* element can be resized to the input dimension:

1. *Crop mode*: the input frame, having an input dimension of $W \times H$, is resized by taking a square portion of the window with dimensions $\min(W, H) \times \min(W, H)$ and scaled to the input detector size ($S \times S$).

2. *Pad mode*: In order to avoid losing information contained in the resized frame, zero padding is used at the borders of the shortest frame dimension. For an input frame size of $W \times H$, the $\min(W, H)$ a zero padding of $P = \frac{\max(W,H)-\min(W,H)}{2}$ pixels per border is executed in such a way the resulting frame has a dimension of $\max(W, H) \times \max(W, H) - \min(W, H)$ and is subsequently resized to $S \times S$.

Unfortunately the padding method, creates spurious information content at the border and reduces the size of the object to be detected, achieving worse performance than with crop mode. While the crop method achieves better precision accuracy, it reduces the visual region on which detection is executed. The processing pipeline is shown in Figure 4.5

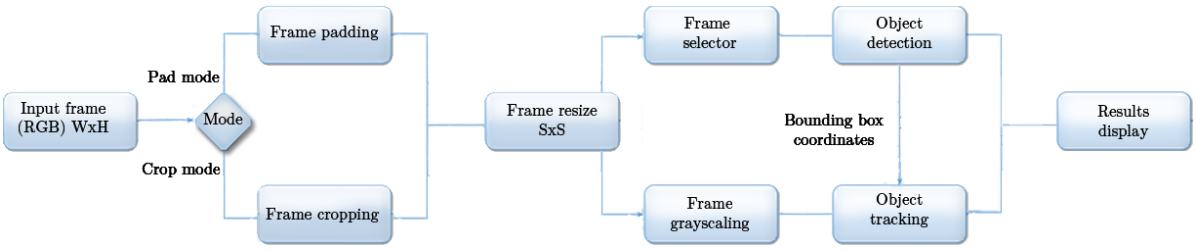

Figure 4.5: Application processing pipeline

The input RGB frame, after padding or cropping, is resized to the object detection input dimensions and used for both detection and tracking. The frame selector discards all the frames arriving at the detector when the detector is already working on a frame, while the tracking module, turns the *Bitmap element* into a grayscale image before being processed (this is done because the KLT algorithm works on one-channel images). The tracker takes as input the bounding box coordinates and tracks the boxes during the new detection inference. Finally, the results are displayed on the *TextureView* superimposed on the UAV video stream.

4.3.4 Detection module

The detection module is responsible for detecting and displaying the detection results, superimposed on the same *Surface* Object where the original stream is displayed. This module is implemented through the TFL library as an *ad hoc* thread, and is partly adapted from the official TF Android example*. The YOLO detector is enclosed in the *TensorflowYoloDetetor* Class, while the MobileNetV2 detector is implemented by the *TensorFlowObjectDetectionAPIModel* Class within the Android project. The method *recognizeImage* is used to run the object detection CNN on a *Bitmap* Object and returns a List of *Recognition* Objects. Each recognition object has the following parameters:

id: unique identifier for what has been recognized.

title: name for the recognition.

confidence: score for how good the recognition is.

location: location within the source image for the location of the recognized object.

Next, the detection results are filtered based on the detection threshold, scaled to the correct size, and then displayed on the application *Surface* Object.

The model, trained as explained in Section 4.2, is imported into the assets' folder of the Android project. After the TFL object detector is initialized, it starts the detection process on the first available frame. The frame is pre-processed as presented in Section 4.3.3 and the detectors can run in both *padding mode* or *cropping mode*. An example of a detection instance is presented in Figure 5.5

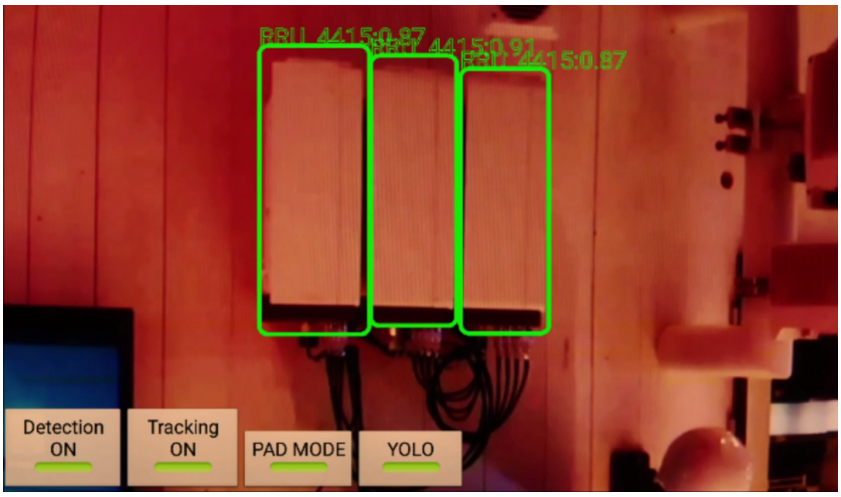

Figure 4.6: Mobile object detection inference example

*https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android

4.3.5 Tracking module

The KLT tracking module is responsible for following the tracked hardware equipment detected with the object detection module. The tracking module is always active during the *mainActivity* and computes the relative motion between the UAV camera and the scene. The tracker is implemented through the OpenCV3.4.3 javacv library [99]. The core functions of this module are the openCV functions *goodFeaturesToTrack* that implements the Shi-Tomasi detector [42] and the *calcOpticalFlowPyrLK* that contains the pyramidal implementation of the KLT tracker [113]. It is important to mention the arguments of the *GoodFeaturesToTrack* function, because by adjusting these parameters different performance of the tracker can achieved. These arguments are:

image	*Mat* Object containing the grayscale input frame on which the Shi-Tomasi corner detector will be run.
corners	output vector of detected corners.
maxCorner	maximum number of corners to return. If there are more corners that are found, the strongest of them are returned.
qualityLevel	parameter characterizing the minimal accepted image corners quality. This parameter determines the number of tracked points.
minDistance	minimum possible Euclidean distance between the tracked corners.

The most decisive parameter is the *qualityLevel* because once this parameter is fixed, increasing the parameter *maxCorners* or reducing the *minDistance* parameter will have no effect on the number of tracked points. Figure 4.7 shows an example of the optical flow computation for different values of the parameter $qualityLevel = [0.5, 0.05, 0.005]$ for a fixed choice of the parameters $maxCorners = 1000$ and $minDistance = 0.01$.

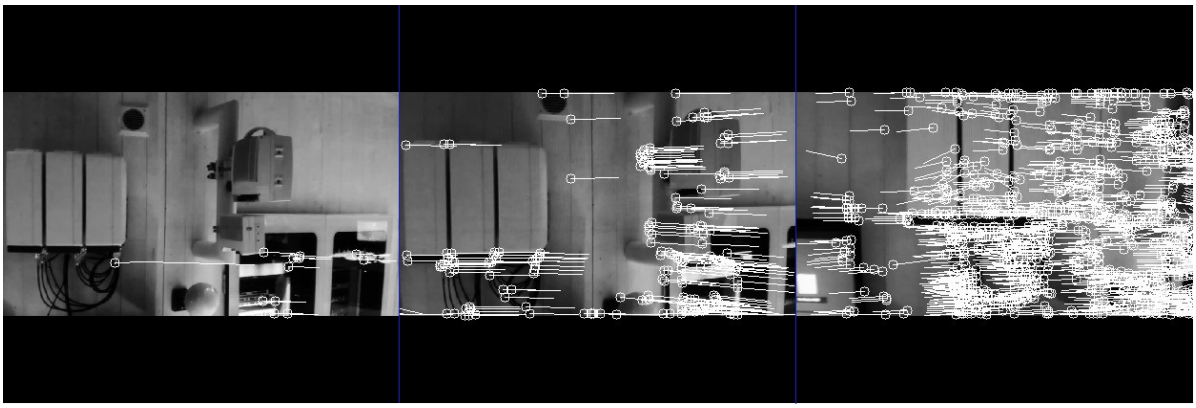

Figure 4.7: KLT sparse optical flow example. Left: $qualityLevel = 0.5$. Center: $qualityLevel = 0.05$. Right: $qualityLevel = 0.005$

Increasing the number of tracked points leads to dense optical flow and is more computationally expensive (because there are more points on which the Kanade-Lukas method will be applied) and vice-versa. However, decreasing *qualityLevel*, may cause incorrect computation of motion vectors since it is more difficult to track points. The *calcOpticalFlowPyrLK* parameters are:

prevImg	first input image on which the optical flow is computed.
nextImg	second input image corresponding to the next frame with respect to *prevImg*.
prevPts	vector of 2-D points for which the flow needs to be found.
nextPts	output vector of 2-D points containing the calculated new positions of input features in *nextImg*.
status	output status vector where each element of the vector is set to 1 if the flow for the corresponding features has been found, otherwise, it is set to 0.
err:	output vector of errors where each element of the vector is set to an error value for the corresponding feature. Type of the error measure can be set in the flags' parameter.

The tracker module tracks the object during the detection time interval for which the detection is not ready to be displayed and displays the shifted box. Also, it computes the shift for the detected bounding boxes from the frame on which the detection has been run to the current frame in order to shift the box once the new detection is ready. Once the motion of each point is computed, the computed motion vectors $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ are averaged, and the mean motion vector v

$$v = \sum_{i=1}^{N} v_i \tag{4.2}$$

is used to translate the bounding boxes associated to the Recognition Objects detected by the detection module (see Section 4.3.4). This is done in order to increase the algorithm's robustness to image noise and misleading motion vectors. In fact, if the motion does not satisfy the optical flow assumptions presented in Section 2.2.1.2 there could be motion vectors that do not correspond to the actual relative motion between the UAV's camera and the scene. By averaging over the whole set of motion vectors $\mathcal{V}$, the effect of these misleading motions are smoothed and the overall motion estimate is still acceptable.

Chapter 5

Results

This chapter presents the evaluation of the performance of the proposed object detection/tracking system and discusses the tests' results. In particular, the experiments are carried out according to the experimental methodology discussed in Section 3.4, and the metrics presented in Section 3.6 are considered for the evaluation. Section 5.1 presents the training loss of the object detectors together with the choice done during the training process. Section 5.2 shows the results about time performances of the object detector and tracker algorithm, while Section 5.3 discusses the mAP results.

5.1 Training loss

Training loss is the most important parameter to monitor when training a CNN model. The value of this parameter is considered indicative of when to decide to stop the training process. It also gives hints about the quality of the trained model and the correctness of the training procedure. The following subsections report the results of the training process in terms of training loss for both the YOLO and the MobileNetV2 models, in the case of multi-class and single class training.

5.1.1 YOLO training loss

The YOLO training loss is computed by averaging the loss function (Equation 3.13) at each training step. The YOLO authors suggest to train the model for at least $2,000$ iterations for each class and more generally to stop training when the average loss no longer decreases. Thus, according to them, the training should be executed for a minimum of $2,000$ iterations for the single-class training case and for $4,000$ iterations in the multi-class training case. However, from Figure 5.1, it is possible to observe that the training loss for both multi-class and single-class cases continues to decrease after this number of iterations, hence the training was executed for 30,000 iterations.

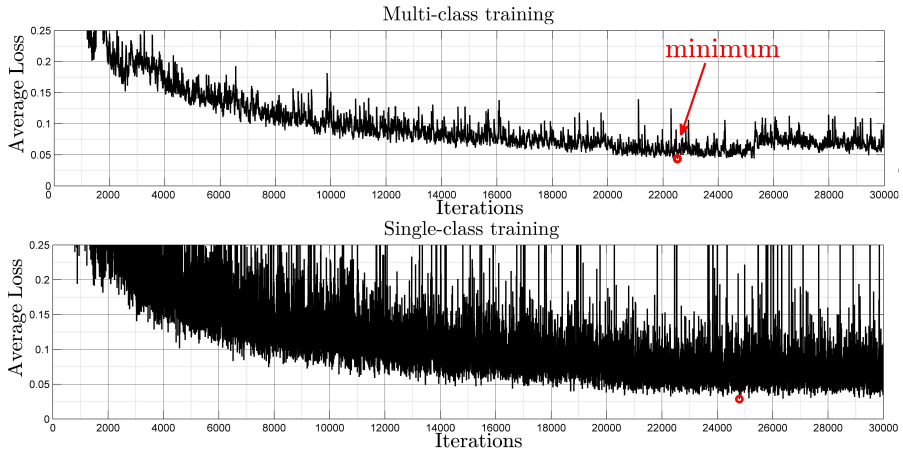

Figure 5.1: YOLOv2 training loss. Top: multi-class training loss. Bottom: single-class training loss

The overall loss curve exhibits a decreasing behaviour, as expected. The single-class case presents higher variance in the oscillations with respect to the multi-class. Figure 5.1 shows that after approximately $20,000^{th}$ iterations the value of the average loss stops decreasing, for both single-class and multi-class training, showing even a small increase for the latter case. The minimum value of the average loss is achieved at the point $p_{multi-class} = (22520, 0.043851)$ for the multi-class training and at $p_{single-class} = (24883, 0.257462)$ for the single-class training. Once the training is stopped, the decision as to which model weights to use is done by adopting the *validation-based early stopping* technique (see Section 2.2.2.14). In particular mAP and IoU are evaluated on the training set every 1000 iterations. The results are presented in Figure 5.2

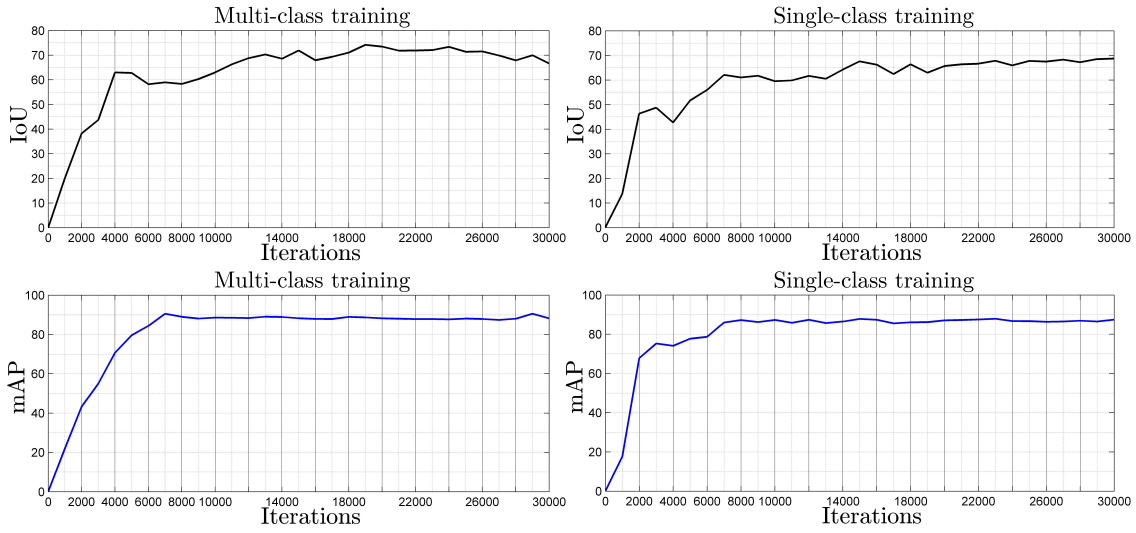

Figure 5.2: YOLOv2 mAP and IoU on the validation set. Left: multi-class training. Right: single-class training

It is insufficient to look only at the minimum of the average loss to select the model with the best performances. According to the early-stopping technique, it is necessary to look at the results on the validation to avoid overfitting the data and achieve performance generalization. By looking at the data from the validation set (Figure 5.2), the joint maximization of the mAP and IoU is achieved at the $19,000^{th}$ iteration for the multi-class training and at at the $23,000^{th}$ iteration for the single class training, and thus these weight models are selected for further evaluation on the test set (see Section 5.3).

5.1.2 MobileNetV2 training loss

The results for the MobileNet loss are shown in Figure 5.3 It is possible to make some

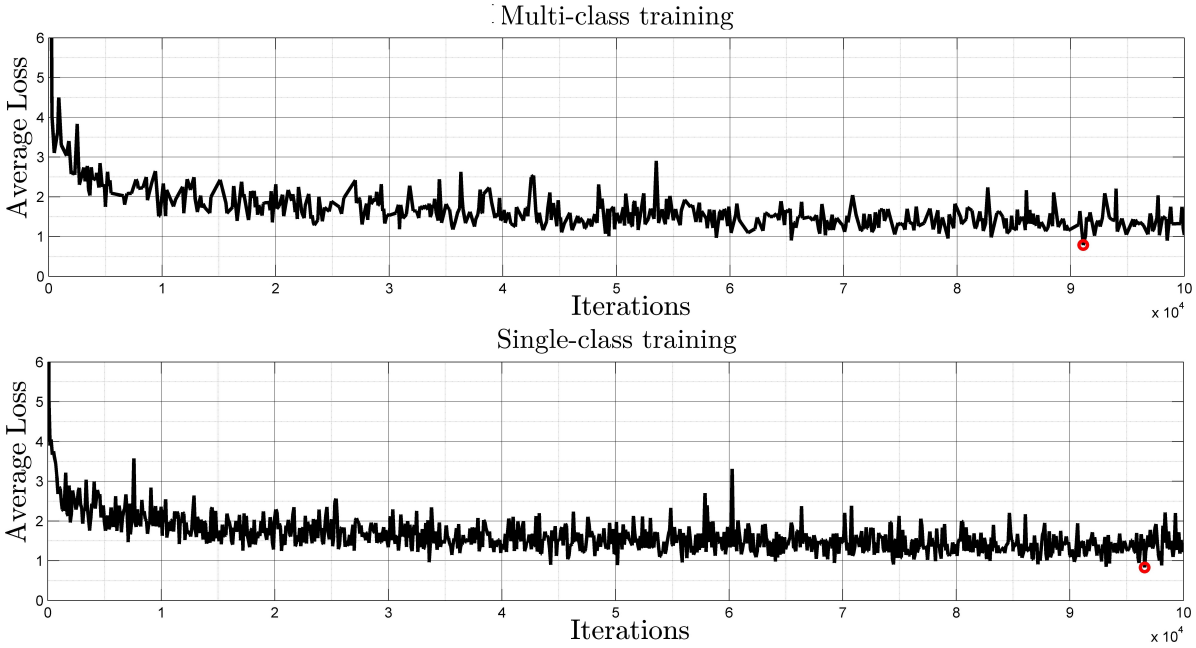

Figure 5.3: MobileNetV2 training loss

observation based on these two plots of the loss function. First of all, the loss for the MobileNetV2 does not converge to zero but and keeps its value higher with respect to the YOLO loss. The training is executed for 10^5 steps, i.e. a greater number of iterations with respect to the YOLO-based model in order to check that the average loss has converged. The loss for this model is composed by the classification and localization loss and the total loss is given by $TotalLoss = ClassificationLoss + LocalizationLoss$ and Tensorboard shows that the dominant loss contribution is due to the classification loss term. The minimum for the two losses are located at $p_{multi-class} = (91748, 0.74625)$ and $p_{single-class} = (96845, 0.75154)$, indicated with the red circle in Figure 5.3. The evaluation on the validation dataset is shown in Figure 5.4.

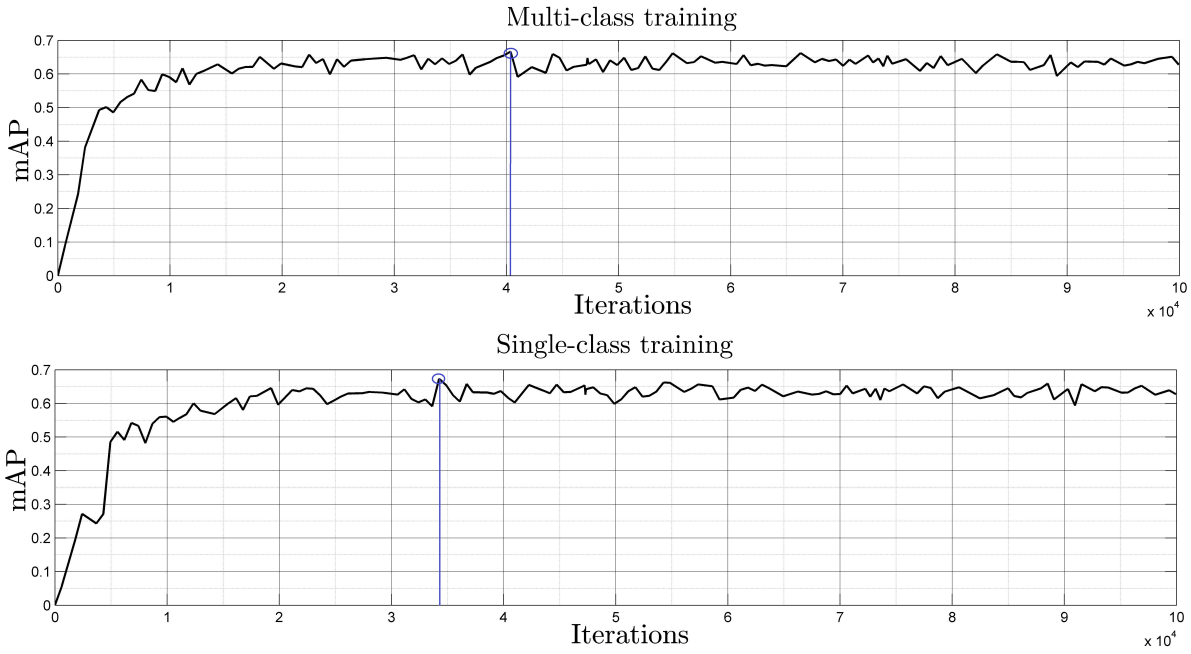

Figure 5.4: MobileNetV2 mAP on the validation set. Top: multi-class training. Bottom: single-class training

The best models are selected based on the best performance on the test set that is achieved at step 40366 and 24257 (denoted in blue in Figure 5.4) respectively for multi-class and single-class training.

5.2 Time performance

The time performance greatly determines the algorithms' quality, especially in embedded device applications with limited computational resources were real-time execution is required. The trained models were evaluated based on their execution time, inherently determining the FPS. The object detection and tracking tasks were executed in a separate thread on the mobile computing platform. This data was later analyzed and assessed to evaluate the viability of the model in terms of FPS performance. During the execution of these algorithms, the inference time was measured in ms and logged in a text file (stored in the memory of the mobile device). It is clear that the time performance is invariant with respect to the number of classes on which the model is trained. For this reason, the time performance measurements will be performed only on the multi-class model.

5.2.1 Object detection inference time

In order to measure the object detection time performance on the Android mobile platform, 10^4 detection instances are run for both the YOLOv2 and the MobileNetV2 models. The results of these measurements are shown in Figure 5.5 for both the YOLOv2 model and the MobilenetV2 model (for the sake of clarity only 3000 detection instances are shown in the figure).

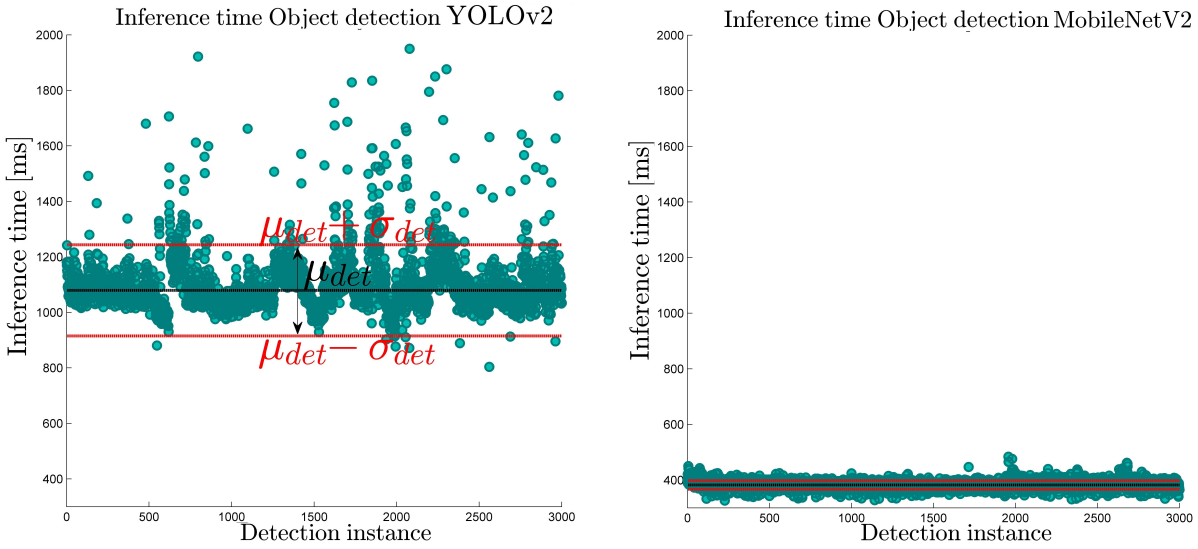

Figure 5.5: Left: inference time YOLOv2 model. Right: inference time MobilenetV2 model.

The statistics in terms of mean and standard deviation of the inference time for the two models are reported in Table 5.1.

Table 5.1: Inference time statistics

CNN model	μ_{det}	σ_{det}	FPS
YOLOv2	1079.31 ms	164.21 ms	1.08
MobileNetV2	382.23 ms	15.67 ms	2.61

As can be observed, The MobileNetV2 model outperforms the YOLOv2 with respect to inference time. By considering the average inference time μ_{det}, the average FPS at which the detection algorithm is able to run is equal to $\overline{FPS_{YOLO}} = 1.08$ FPS, while the MobileNetV2 model has an average FPS equal to $\overline{FPS_{MobileNet}} = 2.61$ FPS. However, none of the two algorithms is able to run in real-time on the mobile computing platform, since the input video stream runs at 24 FPS.

5.2.2 Object tracking time performance

The time performance of the KLT tracker was determined by measuring the time needed to execute the tracking task while running on a separate thread with respect to the object detector. In order to obtain the tracking statistics, almost 10^5 tracking instances are recorded and the results are shown in Figure 5.6. By considering the average tracking execution time μ_{track}, the average FPS at which the tracking algorithm is able to run is equal to $\overline{FPS}_{KLT} = 89.04$ FPS. Thus the tracking algorithm is widely able to run in real-time (because its greater than 24 FPS), assisting the object detection algorithm during the execution of the detection inference.

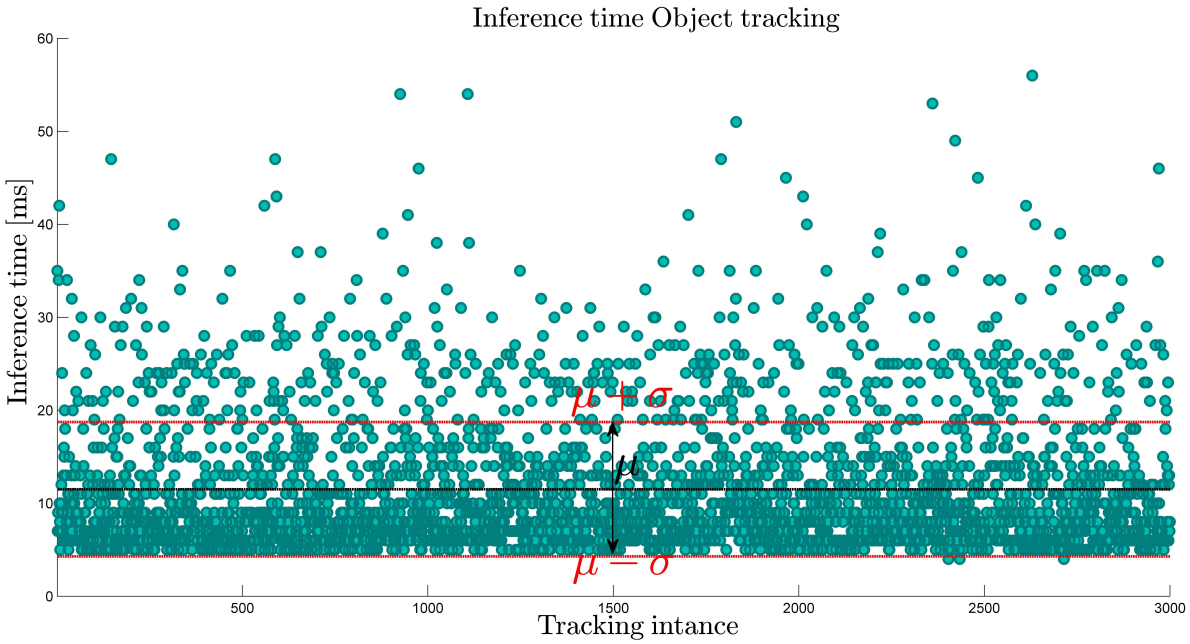

Figure 5.6: KLT time performance

Note that all the samples are below 70 ms. The tracking time statistics are reported in Table 5.2.

Table 5.2: Tracking time statistics

Tracking algorithm	μ_{track}	σ_{track}	FPS
KLT Tracker	11.23 ms	8.23 ms	89.94

.

The KLT thus allows for real-time tracking of the detected object on the proposed computing platform, since it is able to run at an higher frame rate with respect to the input video FPS. Thus the combination of object detection and object tracking algorithm satisfy the real-time requirement.

5.3 mean Average Precision (mAP)

The mAP is the most important indicator for quantifying the performance of the object detection model's precision. The AP for each class is reported, together with the number of true and false predictions, as computed on the test set. The test set contains only baseline dataset (no augmented images) for a total of 153. Around 50% of this test set is acquired from drone flight within the Ericsson studio, and the remaining 50% is acquired with the hand-held camera described in Section 3.2.

5.3.1 YOLOv2 mAP

The model at training iteration $19,000^{th}$ for the multi-class training and at the $23,000^{th}$ iteration for the single-class training are considered for these experiment, as described in Section 5.1.1. For the YOLOv2 model at these training step, the precision/recall curve per class are computed and the results are shown in Figure 5.7.

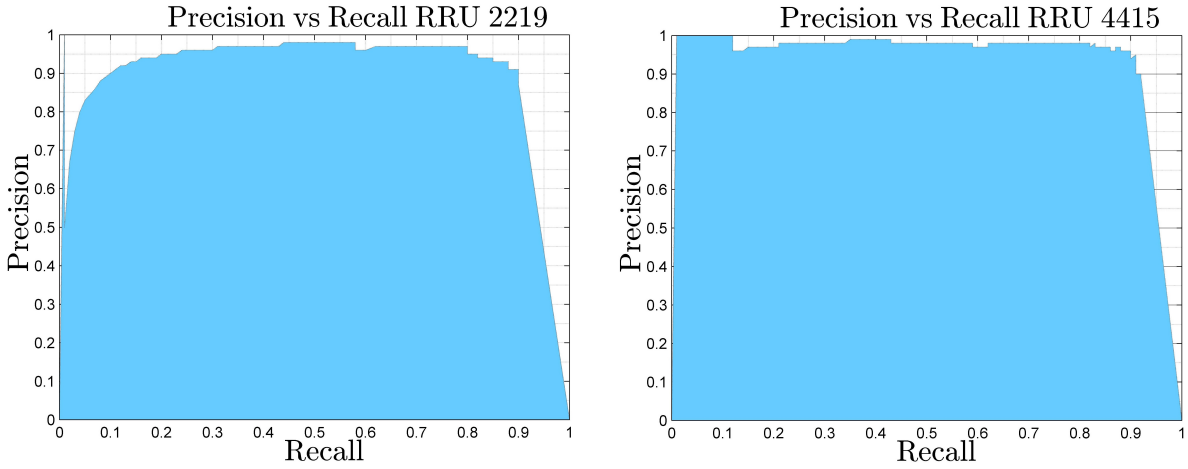

Figure 5.7: Precision-Recall curves for the multi-class case

The area under these curves represent the AP for both classes, and the results on the test are reported in Table 5.3 in terms of AP, True Predictions (True Positives) and False predictions (True Negatives+False Positives)

Table 5.3: YOLOv2 multi-class AP

Class name	AP	True predictions	False predictions
RRU 2219	87.79%	174	21
RRU 4415	90.10%	82	12

Considering these values, the model's is mAP= 88.96%. The highest precision is achieved by the RRU 4415 class; a possible reason for this result could be that this is the class that has most examples in the dataset and thus achieves better performances.

The single-class mAP is computed similarly to the multi-class case by computing the area under the precision-recall curve (Figure 5.8) and in this case the AP= mAP= 85.49%, as only one class is considered.

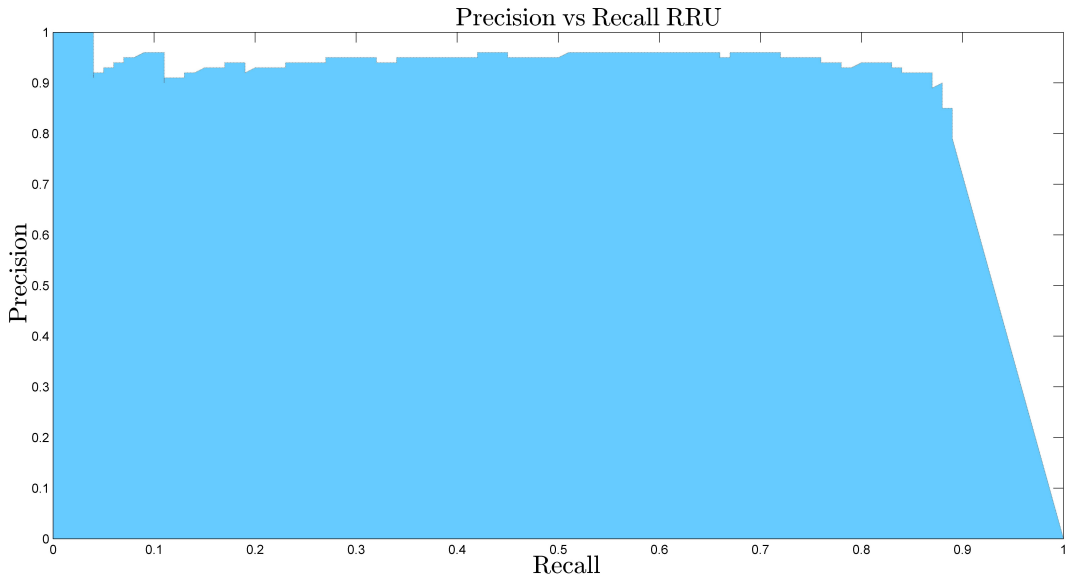

Figure 5.8: YOLOv2 multi-class AP

In Table 5.4 the number of false and positive predictions is reported.

Table 5.4: YOLOv2 single-class AP

Class name	mAP	True predictions	False predictions
RRU	85.49%	286	60

Comparing this result with the multi-class case, it is possible to affirm that there no improvement in terms of mAP is obtained from the experiment in the single-class training case. Thus, in this case, the multi-class training is preferred because allow for a greater precision and a finer classification of the hardware equipment based on the model and not only on the hardware typology.

5.3.2 MobileNetV2 mAP

The mAP for the MobileNetV2 is analyzed in a similar way as done with YOLOv2 model. The model at training iteration $40,366^{th}$ for the multi-class training and at the $24,257^{th}$ iteration for the single-class training are considered for these experiment, as described in Section 5.1.1. The mAP is computed real-time during training, varying the iteration number. The mAP is shown in Figure 5.9.

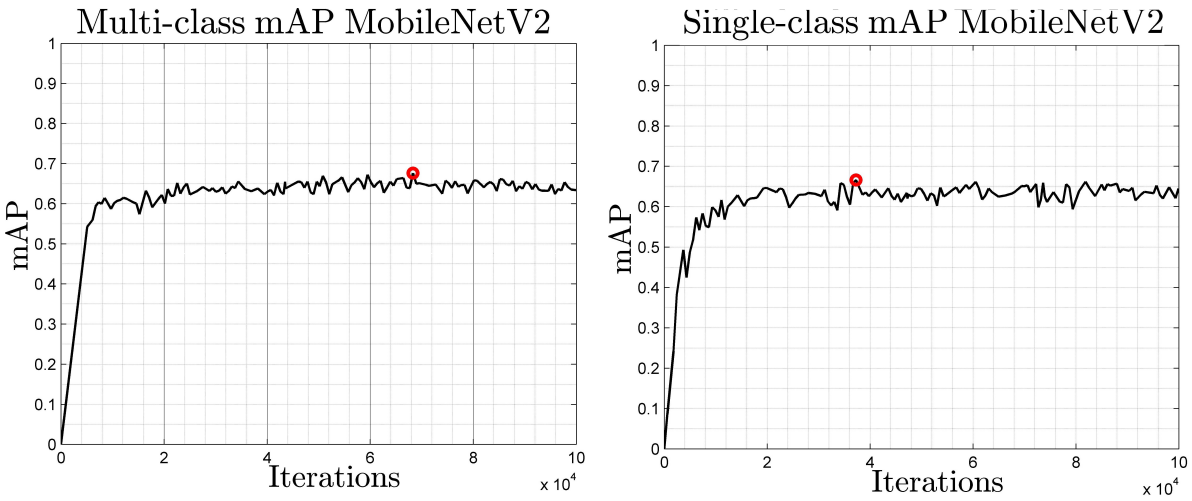

Figure 5.9: MobileNetV2 mAP varying training step. Left: multi-class case. Right: single-class case

As expected, the mAP values are much lower than the ones achieved for the YOLOv2 model for both single-class and multi-class case. This result confirms the existence of a trade-off between computational inference time and precision of the model. The values of mAP for the chosen models are reported in Table 5.5

Table 5.5: MobileNetV2 mAP results

Case	mAP	True predictions	False predictions
multi-class	66.87%	242	120
single-class	65.24%	240	123

Similar results for the YOLOv2 model are obtained for the single-case versus multi-case training. In particular, the best performances are achieved for the multi-case model confirming that also for the MobileNetV2 the results suggest to use a finer granularity when labeling the dataset.

5.3.3 Augmented training mAP

This section present the results in terms of mAP obtained on the augmented dataset (see Section 3.3). Given the results obtained in the previous experiments, the combination YOLOv2 and multi-class case is chosen since the best results in terms of mAP are obtained for this case. The training loss presents a very similar behaviour to the baseline dataset, but the mAP increases with respect to the baseline training. For the augmented dataset an analogous training procedure to the one followed for the base-line dataset is employed (Section 5.1.1).

The model weights are chosen according to the early stopping technique, by selecting the model weights at step number 26,000. The results in terms of Precision-Recall curves are shown in Figure 5.10.

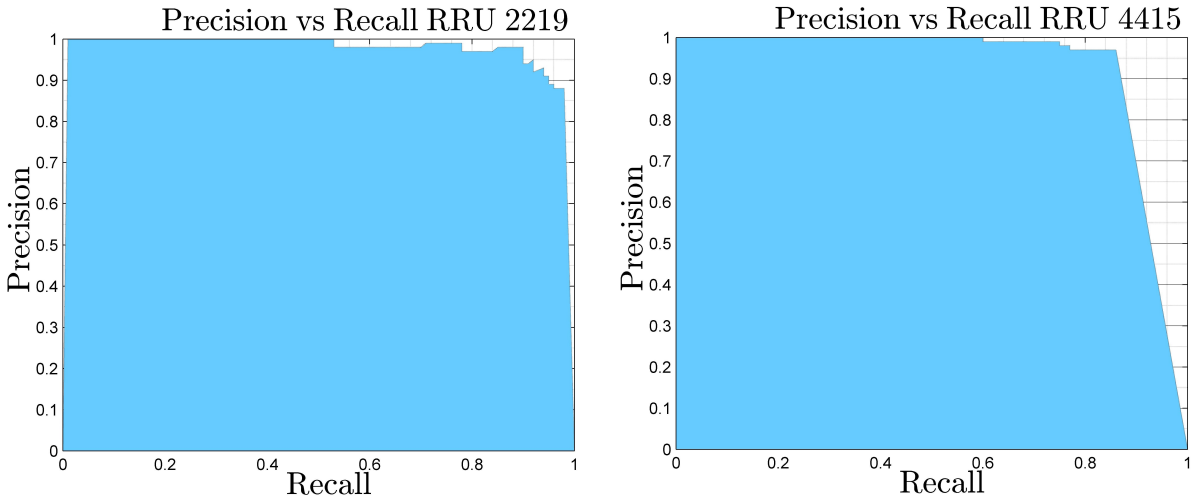

Figure 5.10: Precision-Recall curves for augmented dataset YOLOv2 multi-class training

The results of the tests are reported in Table 5.6 in terms of AP and number of true and false predictions. By averaging out these values, a mAP = 91.08% is obtained.

Table 5.6: YOLOv2 multi-class AP augmented dataset

Class name	AP	True predictions	False predictions
RRU 2219	96.56%	196	7
RRU 4415	85.61%	91	12

The test set used in this experiment, is the same as the baseline dataset case and thus the mAP results for the augmented dataset are comparable. The highest precision is achieved by the RRU 2219 class showing an inverse trend with respect to the non-augmented dataset.

Chapter 6

Conclusions and future works

This chapter draws conclusions on the basis of the conducted experiments (Section 6.1), gives final remarks about the limitations of the developed solution (Section 6.2), indicates future research directions and further development opportunities (Section 6.3), and discusses ethical and sustainability issues (Section 6.4).

6.1 Final outcomes

This thesis presented an investigation of object detection and tracking algorithms suited for UAV applications on power-constrained mobile computing platforms. In particular, it took into consideration the case of hardware detection and tracking applied to UAV-based telecommunication tower inspections. Even though the proposed system and methodology has been exemplified on a particular area of interest (base-station inspection), and on a particular object category (RRUs), it can be adapted more in general to any application where detecting and tracking objects can be useful with few modifications. The final outcomes concern both the scientific and technical aspects:

- From the scientific point of view, this work explored the combination of two object detection algorithms based on CNNs with the KLT tracker based on traditional hand-crafted feature approach. Among the possible combinations, the preferable algorithm is the one that maximizes the accuracy in terms of mAP while satisfying the real-time constraint. Also, this thesis tested and evaluated (i) the detection granularity at which the model is able to run by taking into consideration a single-class versus multi-class detection evaluating the possibility to categorize similar object categories with different labels, (ii) the possibility to increase the mAP through augmentation techniques, applying realistic transformations to the baseline dataset, in order to increase the dataset size and achieve better generalization performances.

- From the technical and practical point of view, it provided an operational methodology to acquire and augment a dataset, training two different types of CNN-based object detection algorithms (YOLOv2 and SSD MobileNetV2). This methodology will allow to expand the prototype by re-training the CNN in the future as needed (e.g. different type of equipment). It also developed an Android application with a modular structure that is able to detect and track a particular category of hardware equipment in real-time. The modularity of the solution allows for future expansions and integration with other modules, or the substitution of the tracking and detection modules with the future state of the art algorithms.

From the results, no substantial differences, in terms of mAP, arise for what concerns the single-class and multi-class training cases. A slightly lower performance is achieved by training both hardware models with the same label (single-class). For this reason, the results suggest that a better choice is to train the CNN with a multi-class approach. Since the performance in terms of running FPS is set by the tracking algorithm that runs at very high speed the most accurate object detection model (YOLOv2) is employed in the proposed algorithm. In fact, even if YOLO needs more time for inference, the KLT algorithm is able to overcome this pitfall by tracking the previously detected boxes.

The chosen solution is able to detect and track different models of the target equipment (multi-class detection) at about 90 FPS with a mAP of 91.07% on the evaluation dataset (augmented training case).

6.2 Limitations

The main limitation of the prototype concerns primarily the image input size of the object detector, which poses a big limitation in the field of view of the UAV. In order to avoid distortion or spurious contents in the input image (padding areas), which greatly degrades the performances, the area on which the UAV is able to successfully detect and track the targeted object is reduced to a square image having a size equal to the shortest dimension of the image when operating in pad mode.

Also, another limitation is due to the fact that the results found on the training granularity have been tested only on two models of hardware. Tests on a greater number of categories would lead to generalized results that can confirm or confute the results found in this work.

6.3 Future works

The current proposed implemented system is successfully able to detect and track equipments in real-time during base station inspections. Based upon the results of this work, possible future research directions are the following:

1. Apply the proposed algorithm to detect anomalies in radio towers such as uncompliant cable bendings, loose or unsupported jumpers (e.g. check whether jumpers are weatherproofed), structural abnormalities (e.g. rust, subsidence), loose or missing hardware, etc (see Figure 6.1).

2. Build upon the information gained through the hardware identification, design control-based algorithms to navigate the base-station autonomously, without the need of a technician on the ground to execute the inspection.

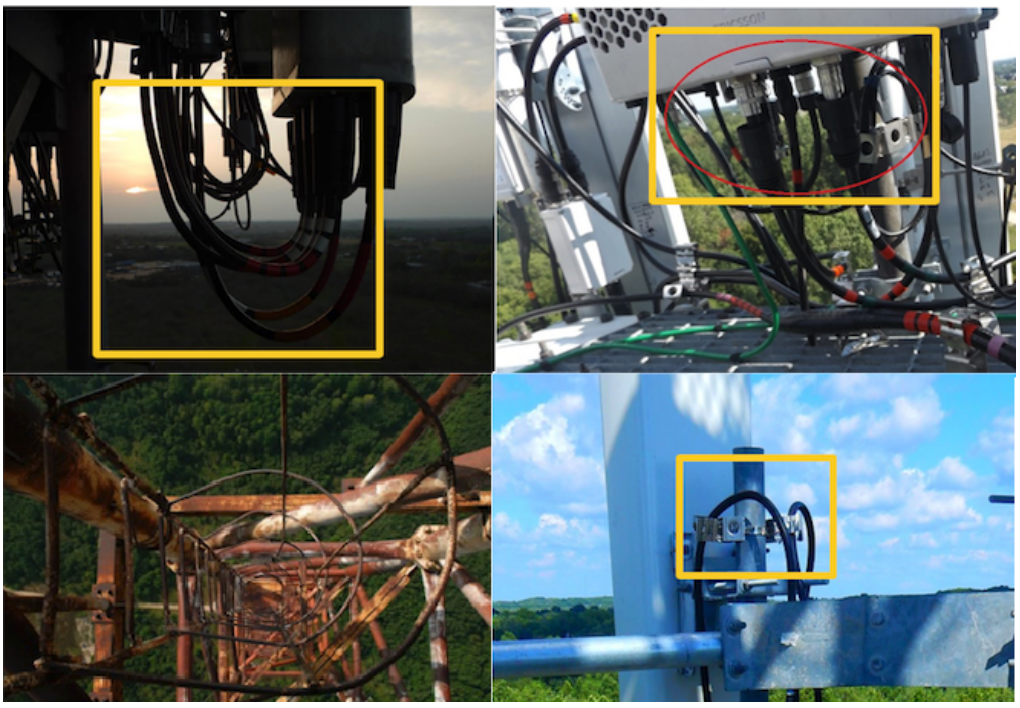

Figure 6.1: Future applications of the proposed algorithm in the field of base-stations inspection

6.4 Ethics and sustainability

The use of AI-powered UAS for commercial inspection applications is questionable because of the issues it raises for what concerns privacy, ethics, and safety. Considering the impact on the employment, the wide adoption of UAV and AI technologies might cause a higher rate of unemployment as drones allow to operate in a more fast and efficient way with respect to the human counterpart, and many jobs could be replaced by drones operating instead of humans. The Federal Aviation Administration estimates that only in the U.S. there will be more than 450.000 commercial drones by 2018 [114]. However, it is likely that other jobs will be created by this technological evolution.

Also, from the privacy point of view, drone-based inspections may raise issues for what concerns the acquisition of sensible third-party data (e.g. private citizens), even if that is unintentional. Specifically, for the current inspection use case, privacy issue may arise in the case of base-stations placed nearby or inside private areas (e.g. at the top of private buildings). In these cases, there is the need to carefully asses privacy issues that could arise from the drone operations before carrying them out. This is necessary to be sure that any operation that involves the collection of personal data must comply with legal obligations and restrictions related to the specific cases.

From the safety perspective, the use of drones represents a huge leap in the reduction of accidents related to dangerous inspection as the statistics in Section 2.1.1.1 point out. However, there may still be issues related to the misuse of the UAV or to unpredictable events that may happen during the inspection.

Bibliography

[1] Wireless Estimator. U.S. tower structure related fatalities, 2018. URL http://wirelessestimator.com/content/fatalities.

[2] Kreuzschnabel (Wikimedia Commons user). Transmitter station for mobile communication in a pasture, germany [digital image] (CC by-sa 3.0), 8 August 2014. URL https://commons.wikimedia.org/wiki/File:2014_mobile_transmitter_station.jpg. [Online; accessed 24 July 2018].

[3] BAmundsen (Wikimedia Commons user). Mobile base station at vikafjell [digital image] (CC by-sa 3.0), 8 October 2010. URL https://commons.wikimedia.org/wiki/File:Telenor_base_stations_Vikafjellet_BA_01.JPG. [Online; accessed 24 July 2018].

[4] Bjoertvedt (Wikimedia Commons user). Lindesnes fyr. norway [digital image] (CC by-sa 3.0), 8 April 2009. URL https://commons.wikimedia.org/wiki/File:Mobile_base_station_Lindesnes_fyr_080420091330.jpg. [Online; accessed 24 July 2018].

[5] Bjoertvedt (Wikimedia Commons user). Hotellneset base station near longyearbyen/airport, svalbard [digital image] (CC by-sa 3.0), 11 September 2011. URL https://commons.wikimedia.org/wiki/File:Telenor_base_stations_Hotellneset_IMG_2400.JPG. [Online; accessed 24 July 2018].

[6] Lee (Flickr user). Ar drone 2.0 carbon [digital image] (CC by-nc-nd 2.0), 12 August 2013. URL https://www.flickr.com/photos/myfrozenlife/9506286079/in/photolist-fu3cE8-VYEzLm-22Sz5QY-he3ka6-M32BFT-28fojv3-TBJz6C-aaXJ6A-ZbAHG7-WAXKxi-MDcyZG-VgmQ9L-PEgru4-WxCs48-hDXEZ9-aWQvM2-8XMjeD-rWvtTH-275FgnE-ft7ysx-26ytx3B-21dE9Jo-fqQvJJ-Wxe9Y2-29P8ehH-agWuzR-28HA7eo-nwrmT-27C9TKG-VWSTTS-Gjz5U4-VN9Te7-pr8ftX-M34LLc-Vaj49C-SjzpLZ-J45HBo-pPC6Ha-r8XFRF-dZzsgX-eggyqJ-VXgHY5-MiQjxC-fvhUwe-oy7uXb-rWmMPQ-Wx9Gji-fvhUe4-28GMgSw-WqHwtd. [Online; accessed 5 October 2018].

[7] Jonathan Cutrer (Flickr user). CBP predator b drone san angelo regional airport, border surveillance drone operated by u.s. customs border patrol [digital image]

(CC by-sa 2.0), 2 October 2018. URL https://www.flickr.com/photos/joncutrer/45065114421/in/photolist-2bEfutp-VkbuDD-27mPkXw-SnoqMt-TpzKmZ-Sn9hje-VF9aFU-264ciaT-X16Qkt-VYEzLm-mJ581D-P75uka-Wvj26w-mWx3Uz-27Rjcbg-TxnSyf-VYvkCE-SnowxB-WeDtcd-SQ4zET-28U5HDm-9LxXRe-KV3Jng-TxDK3G-pabckr-2busfXo-WcBUvL-Toq8Vh-G2Jo73-28fW7BW-22b9Emg-TxDGQW-2b1rPYx-NKBWtK-dBdYuo-ejUxsK-ZBgumf-249JHKe-TBhHYV-coT173-24FhT26-bxsAxf-KyQMcD-bLnhJM-WvhEmN-Wj9vCU-27mPm5A-kerKSy-28XCVFf-T22btj/. [Online; accessed 5 October 2018].

[8] M.M. Aseel, E.G. Loay, and G.M. Faisal. The effect of using inter-frame coding with JPEG to improve the compression of satellite images. volume 58, pages 1970–1978. Iraqui Journal Of Science, 2017. URL https://ijos.iraqjournals.com/article_134035.html.

[9] Aphex34 (Wikimedia Commons user). Typical CNN architecture [digital image] (CC by-sa 4.0), 16 December 2015. URL https://commons.wikimedia.org/wiki/File:Typical_cnn.png. [Online; accessed 24 May 2018].

[10] Narges Khatami (Wikimedia Commons user). Valid padding convolution gif [digital image] (CC by-sa 4.0), 6 July 2018. URL https://commons.wikimedia.org/wiki/File:Valid-padding-convolution.gif. [Online; accessed 24 May 2018].

[11] Aphex34 (Wikimedia Commons user). Max pooling with 2x2 filter and stride 2 [digital image] (CC by-sa 4.0), 16 December 2015. URL https://commons.wikimedia.org/wiki/File:Max_pooling.png. [Online; accessed 24 May 2018].

[12] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. volume abs/1506.02640, 2016. doi: 10.1109/CVPR.2016.91. URL https://ieeexplore.ieee.org/document/7780460.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobilenetV2: Inverted residuals and linear bottlenecks. 2018. URL https://arxiv.org/abs/1801.04381v3.

[14] Matthijs Hollemans. Mobilenet version 2, 22 April 2018. URL http://machinethink.net/blog/object-detection/.

[15] B. de Miguel Molina and O. Marvival Segarra. The drone sector in europe. In *Ethics and Civil Drones*, pages 7–33. Springer, Cham, 2018. ISBN 978-3-319-71086-0. URL https://doi.org/10.1007/978-3-319-71087-7_2.

[16] T. Jones. International commercial drone regulation and drone delivery services. Rand Corporation, 2017. doi: 10.7249/RR1718.3.

URL https://www.rand.org/content/dam/rand/pubs/research_reports/RR1700/RR1718z3/RAND_RR1718z3.pdf.

[17] Goldman Sachs Research. Drones reporting for work, 2013. URL http://www.goldmansachs.com/our-thinking/technology-driving-innovation/drones/. [Online; accessed 18-May-2018].

[18] R. Knutson and L. Day. *In Race For Better Cell Service, Men Who Climb Towers Pay With Their Lives*. FRONTLINE, WGBH Educational Foundation, 2012. URL https://www.pbs.org/wgbh/frontline/article/in-race-for-better-cell-service-men-who-climb-towers-pay-with-their-lives/. [Online; accessed 27-July-2018].

[19] J. Karpowicz. Reducing cell tower inspection costs by up to 50% with drones. Commercial UAV News, March 29 2018. URL https://www.expouav.com/news/latest/drones-used-help-reduce-cell-tower-inspections-50/. [Online; accessed 29-July-2018].

[20] J. Ponciano. The world's largest telecom companies 2018: AT&T, verizon remain on top as sector struggles. Forbes, June 6 2018. URL https://www.forbes.com/sites/jonathanponciano/2018/06/06/worlds-largest-telecom-companies-2018/#20e78fd97d39. [Online; accessed 18-July-2018].

[21] M. Gilbert. Drones & AI: The next phase of automation, August 15 2017. URL http://about.att.com/innovationblog/drones_automation. [Online; accessed 31-July-2018].

[22] Department of Defense of United States of America. Unmanned Aircraft Systems roadmap, 2005-2030. 2005. URL https://fas.org/irp/program/collect/uav_roadmap2005.pdf.

[23] I. Colomina and P. Molina. Unmanned Aerial Systems for photogrammetry and remote sensing: A review. In *ISPRS Journal of Photogrammetry and Remote Sensing*, volume 92, pages 79–97, 2014. URL https://doi.org/10.1016/j.isprsjprs.2014.02.013.

[24] International Civil Aviation Organization. Circular 328, unmanned aircraft systems (UAS), 2011. URL https://www.icao.int/Meetings/UAS/Documents/Circular%20328_en.pdf. [Online; accessed 31-July-2018].

[25] A.G. Korchenko and O.S. Illyash. The generalized classification of Unmanned Air Vehicles. *IEEE 2nd International Conference Actual Problems of Unmanned Air Vehicles Developments Proceedings*, pages 28–34, 2013. doi: 10.1109/APUAVD.2013.6705275. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6705275&isnumber=6705260.

[26] P. G. Fahlstrom and T. J. Gleason. *Introduction to* UAV *Systems*. Wiley, 2012. ISBN 978-1-119-97866-4.

[27] K. Dalamagkidis. Classification of UAVs. pages 83–91. Springer, Dordrecht, 2015. URL https://doi-org.focus.lib.kth.se/10.1007/978-90-481-9707-1_94.

[28] DroneDeploy. Commercial drone industry trends, 2017. URL https://cdn2.hubspot.net/hubfs/530284/10M_Acre_Report_2017_.pdf?blog. [Online; accessed 31-July-2018].

[29] C. Kanellakis and G. Nikolakopoulos. Survey on computer vision for UAVs: Current developments and trends. volume 87, pages 141–168. Springer Netherland. URL https://doi.org/10.1007/s10846-017-0483-z.

[30] A. Al-Kaffemail, F.M. Moreno, L.J. San Josè, F. Garcia, D. Martin, M. de la Escalera, A. Nieva, and L.J.M. Garcea. vbii-uav.

[31] X. Hui, J. Bian, Y. Yu, X. Zhao, and M. Tan. A novel autonomous navigation approach for UAV power line inspection. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 634–639. IEEE, 2017. doi: 10.1109/ROBIO.2017.8324488. URL https://ieeexplore.ieee.org/document/8324488.

[32] S. Han, W. Shen, and Z. Liu. *Deep Drone: Object Detection and Tracking for Smart Drones on Embedded System*. Report for CS231a: Computer Vision, From 3D Reconstruction to Recognition, Kottayam, Kerala, India, June 7 2016. doi: 10.1109/ARTCom.2009.219. URL https://web.stanford.edu/class/cs231a/prev_projects_2016/deep-drone-object__2_.pdf.

[33] European Aviation Safety Agency. Introduction of a regulatory framework for the operation of drones. 2017. URL https://www.easa.europa.eu/sites/default/files/dfu/NPA%202017-05%20%28B%29.pdf.

[34] Transportstyrelsen. Flyger du dronare? URL https://www.transportstyrelsen.se/sv/luftfart/Luftfartyg-och-luftvardighet/Obemannade-luftfartyg-UAS/#159818. [Online; accessed 29-July-2018].

[35] S.T. Huang. Computer vision: Evolution and promise. pages 21–25. 19th CERN School of Computing, 1996. URL https://doi.org/10.5170/cern-1996-008.21.

[36] N. Senthilkumaran and R. Rajesh. Image segmentation - a survey of soft computing approaches. In *Proceedings of the 2009 International Conference on Advances in Recent Technologies in Communication and Computing*, ARTCOM '09, pages 844–846, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3845-7. URL https://doi.org/10.1109/ARTCom.2009.219.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

[38] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1623264.1623280.

[39] D. G. Lowe. Distinctive image features from scale-invariant keypoints. volume 60, pages 91–110, Hingham, MA, USA, November 2004. Kluwer Academic Publishers. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[40] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, Sept 1999. doi: 10.1109/ICCV.1999.790410. URL http://dl.acm.org/citation.cfm?id=850924.851523.

[41] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988. URL http://www.bmva.org/bmvc/1988/avc-88-023.pdf.

[42] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1994. doi: 10.1109/CVPR.1994.323794. URL https://ieeexplore.ieee.org/document/323794.

[43] P. Agrawal, A. Ratnoo, and D. Ghose. Inverse optical flow based guidance for UAV navigation through urban canyons. volume 68, pages 163–178. Aerospace Science and Technology, September 2017. doi: 10.1016/j.ast.2017.05.012. URL https://www.sciencedirect.com/science/article/pii/S127096381730826X.

[44] Z. Gosiewski, J. Ciesluk, and L. Ambroziak. Vision-based obstacle avoidance for unmanned aerial vehicles. *2011 4th International Congress on Image and Signal Processing*, 2011. doi: 10.1109/CISP.2011.6100621. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6100621&isnumber=6100527.

[45] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning*

Research, pages 647–655, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/donahue14.html.

[46] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1717–1724, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.222. URL https://doi.org/10.1109/CVPR.2014.222.

[47] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5118-5. URL https://doi.org/10.1109/CVPR.2014.81.

[48] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. volume 39, pages 1137–1149, Washington, DC, USA, June 2017. IEEE Computer Society. URL https://doi.org/10.1109/TPAMI.2016.2577031.

[49] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015. URL http://arxiv.org/abs/1511.08458.

[50] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Comput.*, 29(9):2352–2449, September 2017. ISSN 0899-7667. doi: 10.1162/neco_a_00990. URL https://doi.org/10.1162/neco_a_00990.

[51] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998. ISBN 978-3-642-35289-8. URL https://doi.org/10.1007/978-3-642-35289-8_5.

[52] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014. URL https://dl.acm.org/citation.cfm?id=2969197.

[53] Princeton University Computer Science Department. ImageNet, 2009. URL http://www.image-net.org/. [Online; accessed 20-May-2018].

[54] Transfer learning. In *CS231n Convolutional Neural Networks for Visual Recognition*. URL http://cs231n.github.io/transfer-learning/. [Online; accessed 28-May-2018].

[55] E.S. Olivas. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, volume 2. IGI Global, 2009.

[56] S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. volume abs/1504.06066, 2015. URL http://dx.doi.org/10.1109/TPAMI.2016.2601099.

[57] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-based convolutional networks for accurate object detection and segmentation. volume 38, pages 142–158, Jan 2016. doi: 10.1109/TPAMI.2015.2437384. URL https://ieeexplore.ieee.org/document/7112511.

[58] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. volume abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497. [Online; accessed 1-June-2018].

[59] W. Ouyang, X. Zeng, X. Wang, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, J. Yan, C. Loy, and X. Tang. DeepID-net: Object detection with deformable part based convolutional neural networks. volume 39, pages 1320–1334, July 2017. doi: 10.1109/TPAMI.2016.2587642. URL https://ieeexplore.ieee.org/document/7506134.

[60] A. Oualid and A. Nabil. Visual servoing of a quadrotor UAV for autonomous power lines inspection. pages 1418–1424, June 2014. doi: 10.1109/MED.2014.6961575. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6961575&isnumber=6961315.

[61] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart. A UAV system for inspection of industrial facilities. pages 1–8. IEEE, March 2013. doi: 10.1109/AERO.2013.6496959. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6496959&isnumber=6496810.

[62] S. Tanathong and I. Lee. The improvement of KLT for real-time feature tracking from UAV image sequence. 2, 01 2009. URL https://www.researchgate.net/publication/267692948_The_improvement_of_KLT_for_real-time_feature_tracking_from_UAV_image_sequence.

[63] F. Jabar, S. Farokhi, and U. Sheikh. Object tracking using sift and klt tracker for uav-based applications. In *Robotics and Intelligent Sensors (IRIS), 2015 IEEE International Symposium on*, pages 65–68. IEEE, 2015. doi: 10.1109/IRIS.2015.7451588. URL https://ieeexplore.ieee.org/document/7451588.

[64] A.A. Micheal and K. Vani. Comparative analysis of SIFT and surf on KLT tracker for UAV applications. In *2017 International Conference on Communication and*

Signal Processing (ICCSP), pages 1000–1003. IEEE, 2017. doi: 10.1109/ICCSP.2017.8286523. URL https://ieeexplore.ieee.org/document/8286523.

[65] C. Sheppard and M. Rahnemoonfar. Real-time scene understanding for UAV imagery based on deep convolutional neural networks. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 2243–2246, July 2017. doi: 10.1109/IGARSS.2017.8127435. URL https://ieeexplore.ieee.org/document/8127435.

[66] C. Sheppard and M. Rahnemoonfar. Real-time scene understanding for UAV imagery based on deep convolutional neural networks. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 2243–2246, July 2017. doi: 10.1109/IGARSS.2017.8127435. URL https://ieeexplore.ieee.org/document/8127435.

[67] N. Tijtgat, W.N. Ranst, B. Volckaert, T. Goedemé, and F.D. Turck. Embedded real-time object detection for a UAV warning system. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2110–2118, Oct 2017. doi: 10.1109/ICCVW.2017.247.

[68] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[69] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016. URL http://dx.doi.org/10.1109/CVPR.2017.690.

[70] M. Radovic, O. Adarkwa, and Q. Wang. Object recognition in aerial images using convolutional neural networks. volume 3, page 21, 2017. URL https://doi.org/10.3390/jimaging3020021.

[71] A. Maher, C. Li, H. Hu, and B. Zhang. Realtime human-UAV interaction using deep learning. pages 511–519. Springer, Cham, 2017. ISBN 978-3-319-69922-6. URL https://doi.org/10.1007/978-3-319-69923-3_55.

[72] A. Anar, E. Bostanci, and M. G. Serdar. Live target detection with deep learning neural network and unmanned aerial vehicle on android mobile device. volume abs/1803.07015, 2018. URL http://arxiv.org/abs/1803.07015.

[73] R. Bonatti, Y. Zhang, S. Choudhury, W. Wang, and S. Scherer. Autonomous drone cinematographer: Using artistic principles to create smooth, safe, occlusion-free trajectories for aerial filming. 2018. URL https://arxiv.org/abs/1808.09563.

[74] C. Feichtenhofer, A. Pinz, and A. Zisserman. Detect to track and track to detect. International Conference on Computer Vision (ICCV), 2017. URL https://arxiv.org/abs/1710.03958.

[75] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

[76] J.F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. volume 37, pages 583–596, 2015. doi: 10.1109/TPAMI.2014.2345390. URL https://arxiv.org/abs/1404.7584.

[77] W. Budiharto, A. Gunawan, J. S. Suroso, A. Chowanda, A. Patrik, and G. Utama. Fast object detection for quadcopter drone using deep learning. In *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*, pages 192–195. IEEE, 2018. doi: 10.1109/CCOMS.2018.8463284. URL https://ieeexplore.ieee.org/document/8463284.

[78] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. URL https://doi.org/10.1007/978-3-319-46448-0_2.

[79] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *European conference on computer vision*, volume abs/1405.0312, pages 740–755. Springer, 2014. URL http://arxiv.org/abs/1405.0312.

[80] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) challenge. volume 88, pages 303–338. Springer, Cham, 2010. URL https://doi.org/10.1007/s11263-009-0275-4.

[81] D. Marmanis, M. Datcu, T. Esch, and U. Stilla. Deep learning earth observation classification using imagenet pretrained networks. volume 13, pages 105–109. IEEE, 2016. doi: 10.1109/LGRS.2015.2499239. URL https://ieeexplore.ieee.org/document/7342907.

[82] F. Kucuksubasi and A. Sorguc. Transfer learning-based crack detection by autonomous UAVs. 2018. URL https://arxiv.org/abs/1807.11785.

[83] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308. URL https://ieeexplore.ieee.org/document/7780677.

[84] E. Othman, Y. Bazi, N. Alajlan, H. Alhichri, and F. Melgani. Using convolutional features and a sparse autoencoder for land-use scene classification. volume 37,

pages 2149–2167. Taylor & Francis, 2016. doi: 10.1080/01431161.2016.1171928. URL https://dl.acm.org/citation.cfm?id=2996664.

[85] Craftsman toolbag. URL https://www.craftsman.com/products/craftsman-large-mouth-tool-bag-16-inch?taxon_id=2230. [Online; accessed 1st July 2018].

[86] Tufts Enviromental Health and Safety. Carrying work home-painlessly!, October 2014. URL http://publicsafety.tufts.edu/ehs/files/October2014.VI2-Backpacks.pdf.

[87] K. Schmidt. Cell tower heights across the US, 2007. URL http://www.steelintheair.com/Blog/2007/09/cell-tower-heights-across-the-us.html. [Online; accessed 31-July-2018].

[88] J. Lee, J. Wang, D. Crandall, S. Sabanovic, and G. Fox. Real-time, cloud-based object detection for unmanned aerial vehicles. pages 36–43, 2017. doi: 10.1109/IRC.2017.77. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7926512&isnumber=7926477.

[89] Google Inc. Android, 2008. URL https://www.android.com/.

[90] Dà-Jiāng Innovations (DJI). Mobile SDK, 2018. URL https://developer.dji.com/mobile-sdk/.

[91] PassMark Software. Android benchmarks, performance comparison of android devices. URL https://www.androidbenchmark.net/phone.php?phone=Xiaomi+Mi+A1. [Online; accessed 28-September-2018].

[92] Google Inc., 1998. URL https://www.google.com/about/our-company/. [Online; accessed 8-July-2018].

[93] L. Torvalds. Linux. URL https://www.linuxfoundation.org/. [Online; accessed 12-July-2018].

[94] J. Gosling. Java. URL https://www.java.com/en/. Online; accessed 26-May-2018].

[95] Google Developers. Android NDK . URL https://developer.android.com/ndk/. [Online; accessed 4-July-2018].

[96] B. Kernighan and D. Ritchie. The C programming language. Englewood Cliffs, 1988. ISBN 0131103628.

[97] S. Prata. C++ primer plus. Addison-Wesley, 2013. ISBN 9780133432381.

[98] Google Brain. Tensorflow. URL https://www.tensorflow.org/. [[Online; accessed 28-September-2018].

[99] Intel. Opencv library. URL http://opencv.org/about.html. [[Online; accessed 24-September-2018].

[100] G. Van Rossum. Python, 1991. URL https://www.python.org/. [Online; accessed 10-September-2018].

[101] J.C. van Gemert, C.R. Verschoor, P. Mettes, K. Epema, L.P. Koh, and S. Wich. Nature conservation drones for automatic localization and counting of animals. In *Workshop at the European Conference on Computer Vision*, pages 255–270. Springer, Cham. ISBN 978-3-319-161,77-8. URL https://doi.org/10.1007/978-3-319-16178-5_17.

[102] P. Rudol and P. Doherty. Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery. In *Aerospace Conference, 2008 IEEE*, pages 1–8. IEEE, 2008. doi: 10.1109/AERO.2008.4526559. URL https://ieeexplore.ieee.org/abstract/document/4526559/.

[103] L. Grip. Vision based indoor object detection for a drone. 2017. URL http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1108625&dswid=1280.

[104] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and Murphy K. Speed/accuracy trade-offs for modern convolutional object detectors. 2017. URL https://arxiv.org/abs/1611.10012.

[105] R. Real and J.M. Vargas. The probabilistic basis of jaccard's index of similarity. volume 45, pages 380–385. JSTOR, 1996. URL https://doi.org/10.1093/sysbio/45.3.380.

[106] D. L. Olson and D. Delen. *Advanced Data Mining Techniques*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 3540769161, 9783540769163. doi: 10.1007/978-3-540-76917-0. URL https://www.researchgate.net/publication/220695151_Advanced_Data_Mining_Techniques.

[107] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045167.

[108] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press. URL https://projecteuclid.org/euclid.bsmsp/1200512992.

[109] H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian. Deep residual learning for image recognition. volume abs/1512.03385, 2015. doi: 10.1109/CVPR.2016.90. URL http://arxiv.org/abs/1512.03385.

[110] F. Bellard. FFmpeg. URL https://www.ffmpeg.org/. [Online; accessed 22-September-2018].

[111] GCC Team et al. GCC, The GNU compiler collection. 2013. URL http://gcc.gnu.org.

[112] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. doi: 10.1109/CVPR.2015.7298594. URL https://arxiv.org/abs/1409.4842.

[113] J. Bouguet. Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001. URL http://robots.stanford.edu/cs223b04/algo_tracking.pdf.

[114] Federal Aviation Administration (FAA). FAA aereospace forecast fiscal year 2018-2038. 2018. URL https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2018-38_FAA_Aerospace_Forecast.pdf.

Appendix A

YOLOv2 configuration file

```
[net]
batch=64
subdivisions=8
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation=1.5
exposure=1.5
hue=.1
learning_rate=0.001
policy=steps
scales=.1,10,.1,.1
[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=2
[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=2
[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=2
[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=2
[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=2
[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky
[maxpool]
size=2
stride=1
[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky
[convolutional]
size=1
stride=1
pad=1
filters=35
activation=linear
[region]
anchors=1.08,1.19,3.42,
4.41,6.63,11.38,9.42,
5.11,16.62,10.52
bias_match=1
classes=2
coords=4
num=5
softmax=1
jitter=.2
rescore=1
object_scale=5
thresh = .5
random=1
```

Appendix B

MobileNetv2 configuration file

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.99999989895e-05
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.0299999993294
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.999700009823
          center: true
          scale: true
          epsilon: 0.0010000000475
          train: true
        }
      }
      use_depthwise: true
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
```

```
  iou_similarity {
  }
}
box_predictor {
  convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.99999989895e-05
        }
      }
      initializer {
        truncated_normal_initializer {
          mean: 0.0
          stddev: 0.0299999993294
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.999700009823
        center: true
        scale: true
        epsilon: 0.0010000000475
        train: true
      }
    }
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 3
    box_code_size: 4
    apply_sigmoid_to_scores: false
  }
}
anchor_generator {
  ssd_anchor_generator {
    num_layers: 6
    min_scale: 0.20000000298
    max_scale: 0.949999988079
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 3.0
    aspect_ratios: 0.333299994469
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 0.300000011921
    iou_threshold: 0.600000023842
    max_detections_per_class: 100
    max_total_detections: 100
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
loss {
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_loss {
    weighted_sigmoid {
    }
  }
  hard_example_miner {
    num_hard_examples: 3000
    iou_threshold: 0.990000009537
    loss_type: CLASSIFICATION
    max_negatives_per_positive: 3
    min_negatives_per_image: 3
  }
  classification_weight: 1.0
  localization_weight: 1.0
}
}
train_config {
```

```
 batch_size: 24
 data_augmentation_options {
   random_horizontal_flip {
   }
 }
 data_augmentation_options {
   ssd_random_crop {
   }
 }
 optimizer {
   rms_prop_optimizer {
     learning_rate {
 exponential_decay_learning_rate {
       initial_learning_rate:0.004
       decay_steps: 800720
       decay_factor: 0.949
       }
     }
     momentum_optimizer_value: 0.899
     decay: 0.899
     epsilon: 1.0
   }
 }
 fine_tune_checkpoint:
"/object_detection/model.ckpt"
 num_steps: 200000
   fine_tune_checkpoint_type: "detection"
 }
 train_input_reader {
   label_map_path:
"/object_detection/training/labelmap.pbtxt"
   tf_record_input_reader {
     input_path:
  "/object_detection/train.record"
   }
 }
 eval_config {
   num_examples: 153
   max_evals: 153
 }
 eval_input_reader {
   label_map_path:
"/object_detection/training/labelmap.pbtxt"
   shuffle: true
   num_epochs: 1
   num_readers: 1
   tf_record_input_reader {
     input_path: "/object_detection/test.record"
   }
   sample_1_of_n_examples: 1
 }
```