POLITECNICO
MILANO 1863

# Machine Learning Techniques for Detection and Tracking of Space Objects in Optical Telescope Images

**Master Thesis of:**
Jason Calvi 927100

**Advisor:**
Prof. Pierluigi Di Lizia
**Co-Advisors:**
PhD candidate Riccardo Cipollone
PhD candidate Andrea De Vittori
PhD candidate Alessandro Panico

28th April 2021

# Abstract

Performing real time and robust space object detection by means of optical acquisitions is a challenging task in space surveillance, especially when the tracking involves uncatalogued objects. Conventionally, this engineering problem is addressed by means of traditional detection or segmentation, based on computer vision techniques, in order to identify the tracklets and extract their relative position with respect to the sensor position and attitude and the position of the stars in the field of view. Nevertheless, the high computational time required to perform this task represents the main limitation for real time applications. This thesis proposes two innovative tools based on machine learning techniques, respectively to detect and track the objects in real-time. The tracklet detection and localization tool was generated by training a Convolutional Neural Network (CNN) based on YOLOv5 architecture. The development of a machine learning system requires several steps, including dataset creation, pre-processing, training, and testing. To optimise network accuracy and execution time, the whole process was repeated for different datasets, based on synthetic and real telescope acquisitions, and for different combinations of the neural network hyper-parameters. The results obtained on the validation dataset showed an accuracy of 98% and a computational time of 0.5 seconds for the inference phase. Faster configurations were investigated as well, showing a limited degradation in terms of detection accuracy. However, the application of the method on a real-case scenario requires to consider the image pre-processing into the computational performance, showing that the end-to-end process requires about 7 seconds to be executed, being still suitable for real time applications as it is comparable to the characteristic time of a typical single telescope acquisition. The tracker estimates the object angular path with a linear regression performed on multiple detections in successive pictures. Therefore, by updating the telescope pointing angles with the predicted path, the number of target acquisitions is maximized, especially for uncatalogued space objects. Since trajectory estimation is based on bounding boxes, the network plays a crucial role in object identification because it must be as accurate as possible. In order to be a real-time software, this technique adopts a faster but less efficient image processing, followed by detection and finally a tracking script. The total time required by the tracker is about 1.5 seconds. The accuracy of this process is 91%, mainly because of time limitations for fast image processing. The algorithms are based on Python codes executed on a 2017 machine with an i7-7700HQ CPU, 16Gb of RAM and a GTX 1050 graphics card with 4Gb of VRAM.

The results based on the proposed tools showed that the accuracy achieved by the detector network in identifying tracks could represent a valid alternative to traditional techniques, and conventional telescope survey, based on predefined path, can be replaced by more efficient approaches that include artificial intelligence, real-time object detection, and tracking. In particular, the tracker achieved promising results during simulations.

# Sommario

Eseguire in tempo reale e in modo robusto il rilevamento di oggetti spaziali per mezzo di acquisizioni ottiche è un compito impegnativo nella sorveglianza spaziale, soprattutto quando il tracciamento coinvolge oggetti non catalogati. Convenzionalmente, questo problema ingegneristico viene affrontato per mezzo del rilevamento tradizionale o della segmentazione, basata su tecniche di computer vision, al fine di identificare le tracklet ed estrarre la loro posizione relativa rispetto alla posizione del sensore e alla posizione delle stelle nel campo visivo. Tuttavia, l'alto tempo di calcolo richiesto per eseguire questo compito rappresenta la principale limitazione per le applicazioni in tempo reale. Questa tesi propone due strumenti innovativi basati su tecniche di apprendimento automatico, rispettivamente per rilevare e tracciare gli oggetti in tempo reale. Lo strumento di rilevamento e localizzazione di tracklet è stato generato addestrando una CNN (Convolutional Neural Network) basata sull'architettura YOLOv5. Lo sviluppo di un sistema di apprendimento automatico richiede diverse fasi, tra cui la creazione di dataset, la pre-elaborazione, l'addestramento, il test e la post-elaborazione. L'intero processo è stato ripetuto per diversi dataset, basati su acquisizioni sintetiche e reali del telescopio, e per diverse combinazioni di iper-parametri della rete neurale per ottimizzare l'accuratezza della rete e il tempo di esecuzione. I risultati ottenuti sul dataset di validazione hanno mostrato una precisione del 98% e un tempo di calcolo di $0,5$ secondi per la fase di rilevamento. Sono state studiate anche configurazioni più veloci, mostrando una degradazione limitata in termini di accuratezza di rilevamento. Tuttavia, l'applicazione del metodo su uno scenario reale richiede di considerare la pre-elaborazione dell'immagine nelle prestazioni di calcolo, mostrando che il processo end-to-end richiede circa 7 secondi per essere eseguito, essendo ancora adatto per applicazioni in tempo reale in quanto è paragonabile al tempo caratteristico di una tipica acquisizione con singolo telescopio. Il tracker stima il percorso angolare dell'oggetto con una regressione lineare eseguita su rilevazioni multiple di immagini successive. Pertanto, aggiornando gli angoli di puntamento del telescopio con il percorso previsto, il numero di acquisizioni del bersaglio è massimizzato, soprattutto per gli oggetti spaziali non catalogati. La rete svolge un ruolo cruciale nell'identificazione delle tracce dato che la stima della traiettoria dipende dalle bounding box, perciò, essa deve essere più accurata possibile. Per lavorare in tempo reale, questa tecnica adotta un'elaborazione dell'immagine più veloce ma meno efficiente, seguita dal rilevamento e infine da uno script di inseguimento. Il tempo totale richiesto dal tracker è di circa $1,5$ secondi. La precisione di questo processo è del 91%, principalmente a causa delle limitazioni di tempo per l'elaborazione veloce delle immagini. Gli algoritmi sono basati su codici Python eseguiti su una macchina del 2017 con una CPU i7-7700HQ, 16 Gb di RAM e una scheda grafica GTX 1050 con 4 Gb di VRAM.

I risultati basati sugli algoritmi proposti hanno mostrato che l'accuratezza raggiunta dalla rete di detector nell'identificare le tracce potrebbe rappresentare una valida alternativa alle tecniche tradizionali, e il tracciamento convenzionale tramite telescopio, basato su una traiettoria predefinita, può essere sostituito da approcci più efficienti che includono l'intelligenza artificiale, il rilevamento di oggetti in tempo reale e il tracciamento. In particolare, il tracker ha ottenuto risultati promettenti durante le simulazioni.

# Contents

# List of Figures

# List of Tables

# Acknowledgments

The present thesis is the result of a collaboration in conjunction with DAER (Politecnico) and the italian Air Force. I would like to pay special regards to my supervisor Pierluigi Di Lizia, who flawlessly guided me throughout these months and drove the work to the best result. I wish to express my deepest gratitude to the PhD students Riccardo Cipollone and Andrea De Vittori proved essential to accomplish this work's results and because they offered the fundamental TIG software used as basis of the application developed in this project. Also, the assistance and advice provided by PhD student Alessandro Panico: your advice was essential for the success of this thesis.

I also wish to acknowledge the support and great love of my family, who always encouraged and believed in me. I would like to express my deepest gratitude to my father Fernando and my mother Emanuela for their sacrifices and for sharing their thoughts and discussions with me. To my sister Jennifer for having supported me in these years. A special thanks to my girlfriend Dafne, who has always made my days more beautiful, and because ever you are my lighthouse lit during stormy seas.

I must express my deep gratitude to my roommates Federico, Davide, Riccardo, and Riccardo for living with me during these years and sharing moments of wonderful growth. To my fellow students Alessandro, Daniele, Marco, Mattia, Matteo, Paolo, and Tommaso for having helped me during this painful but worthwhile path in PoliMi. I will never forget the moments that made this experience unique. Furthermore, I want to thank Andrea, Gaia, Hermione, Loris, Matteo, Marco, Stefano and all my friends from the village for saving me from the monotony of routine. Last but not least thanks to my flatmates, friends and everyone else who contributed to my path to this achievement.

## List of abbreviated terms

### Acronyms

- **AI**: Artificial Intelligence
- **AP**: Average Precision
- **API**: Application Programming Interface
- **ASTRiDe**: Automated Streak Detection for Astronomical Images
- **BB**: Bounding Box
- **BIRALES**: BIstatic RAdar for LEO Survey
- **CCDs**: Charged Coupled Devices
- **CMOS**: Complementary Metal-Oxide-Semiconductor
- **CNN**: Convolutional Neural Network
- **CONV**: Convolutional
- **CPU**: Central Processing Unit
- **CSP**: Cross Stage Partial network
- **DEB**: Debris
- **DNN**: Deep Neural Network
- **ESA**: European Space Agency
- **EUSST**: European Space Surveillance & Tracking
- **FCN**: Fully Connected Network
- **FITS**: Flexible Image Transport System
- **FLOPS**: FLoating-point OPeration
- **FoV**: Field of View
- **FN**: False Negative
- **FP**: False Positive
- **FPS**: Frame Per Second
- **GEO**: Geosynchronous Orbit
- **GIoU**: Generalized Intersection over Union
- **GPU**: Graphic Unit Card
- **GTX**: Giga Texel Shader eXtreme
- **IADC**: Inter-Agency Space Debris Committee
- **IoU**: Intersection over Union
- **LEO**: Low Earth Orbit
- **LOT**: Linear Orbit Tracker
- **LSTM**: Long Short Term Memory
- **mAP**: mean Average Precision
- **MEO**: Mid Earth Orbit
- **OD**: Object Detection
- **OT**: Object Tracking
- **R**: Recall
- **RAM**: Random Access Memory
- **ReLU**: Rectified Linear Unit
- **RID**: Real Time Detector
- **RMSE**: Root Mean Square Error
- **RMSprop**: Root Mean Square prop
- **ROLO**: Recurrent YOLO
- **RPN**: Region Proposal Network
- **RSO**: Resident Space Object
- **P**: Precision
- **SVM**: Support Vector Machine
- **SATCAT**: Satellite Catalog
- **SCOOP**: SpaceCraft and Objects Observation Planning
- **SGD**: Stochastic Gradient Descent
- **SNR**: Signal to Noise Ratio
- **SPICE**: Spacecraft Planet Instrument C-Matrix Events
- **SSA**: Space Situation Awareness
- **SSD**: Single Shot Detector

- **SST**: Space Surveillance & Tracking
- **SVM**: Support Vector Machine
- **TIG**: Tracklet Image Generator
- **TLE**: Two Line Elements
- **TN**: True Negative
- **TP**: True Positive
- **US SSN**: United States Space Surveillance Network
- **VRAM**: Video Random Access Memory
- **YOLO**: You Only Look Once

## Reference Symbols

- Alg.: Algorithm
- Chap.: Chapter
- Eq.: Equation
- Fig.: Figure
- Part.: Part
- Sec.: Section
- Tab.: Table

## Legend

Here are listed how scalars, vectors and matrices are mentioned through the document:

- a: scalar
- **a**: vector
- **A**: Matrix

# Chapter 1

# Introduction

## 1.1 Space debris surveillance

A space debris is defined as a man-made and non operational objects orbiting the earth. In most cases, they are generated due to tank explosions and impacts between detritus and satellites. Even small objects can represent a threat because of their high velocity and therefore high kinetic energy. In recent years the number of launches has drastically decreased [1], but this does not mean that space pollution has slowed down, because the mass transported is greater. Nowadays a launcher can carry several satellites and it is made up of several stages and detachable parts, which are released during the orbit insertion phase of the payload (examples: payload, protections, adapter rings, bolts, tool covers, etc). The increasing overpopulation of space objects could jeopardize the realization of future space missions, both in the short and long term.
The vast majority of space junk population is composed of small objects. Most are generated by the deterioration due to the hostile space environment, and by the waste products of solid propellant engines. The typical diameter of these particles is on the order of millimeters and sub-millimeters.

The in-orbit objects are identified by their type (according to SATCAT classification): PL (Payload), RB (Rocket Body), DEB (Debris), OTHERS (not catalogued). Recent evaluations have estimated that these objects are [2]:

- About 20000 with a size larger than 10 cm.
- About 300000 with a size between 1 and 10 cm (based on statistical models).
- Several tens of millions with a size between 1 mm and 1 cm (still based on statistical models).
- Much more with a size smaller than 1 mm.

They are mainly located in two orbital regimes [3]:

- Low Earth Orbit (LEO): all satellites orbiting the earth at an altitude ranging from 160 to 2000 kilometers. The rotation period of these types of satellites ranges from about 90 minutes to 120 minutes. Approximately 55% of satellites are in this category due to the high quality of ground observations, environmental studies, military, and meteorological objectives.
- Geostationary Orbit (GEO): satellites with an altitude of about 35,786 km. This guarantees an orbital period equal to Earth's rotational period. Around 35% of the satellites belong to this region, making it the second most populous range. GEO orbits are typically used for telecommunications, defense, and meteorology.

Whole objects (e.g. inactive satellites or upper stages of launchers) constitute about half of the debris in orbit, while the other half is composed of fragments of various shapes and sizes (resulting from explosions, collisions, and various erosions), objects lost during previous missions (cover, belt, etc.) and fuel ejected from engines and tanks (via vent valves).

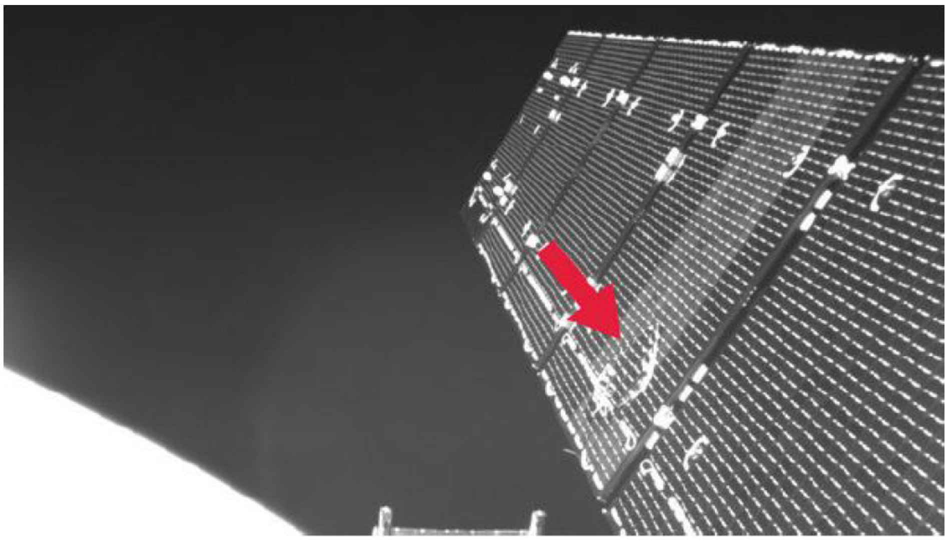


**Figure 1.1** Effect of debris impact on solar panels of SENTINEL-1A satellite [4].

This naturally leads to a risk of collision between satellites and/or debris and the increase of debris itself. Fig. 1.1 illustrates the impact of $5mm$ debris on the solar panels of the Sentinel-1A satellite [5], the damaged area is $40cm^2$ and caused a significant loss of power to the satellite.
Space Surveillance and Tracking (SST) covers a key role in monitoring and tracking objects in orbit. It defines the set of operations to be performed to keep space pollution under control and track population changes of man-made space debris. Some of its main objectives are to support safe operations of space activities, risk management, and liability assessment, to characterize the physical properties of space objects.
SST activities span from observations of RSOs (Resident Space Objects) by optical and radar networks, to launch information (date, country, payload, etc.), reentered objects, and owner/operator of each object. This range of operations can help mission planning and satellite maintenance reducing as much as possible collisions and failures. A catalogued and monitored environment can in fact support different steps of a space program [6]:

- launch and early operations, by confirming separation of the satellite from the launcher and providing information on initial orbit for tracking operations;
- contingencies, by tracking malfunctioning or passive satellites;
- collision warnings, by detecting conjunctions between satellites and other objects;
- search for released or lost objects;
- controlled and uncontrolled re-entry, by estimating trajectory, re-entry time and location, and risk to the ground;
- identification of new objects, detection, and characterization of in-orbit fragmentations.

Fig 1.2 represents a possible diagram of the SST activities.
The first block, colored in blue, is the observation block. Observation and measurement are fundamental activities for any SST network as they represent the only source of information to create a catalogue.

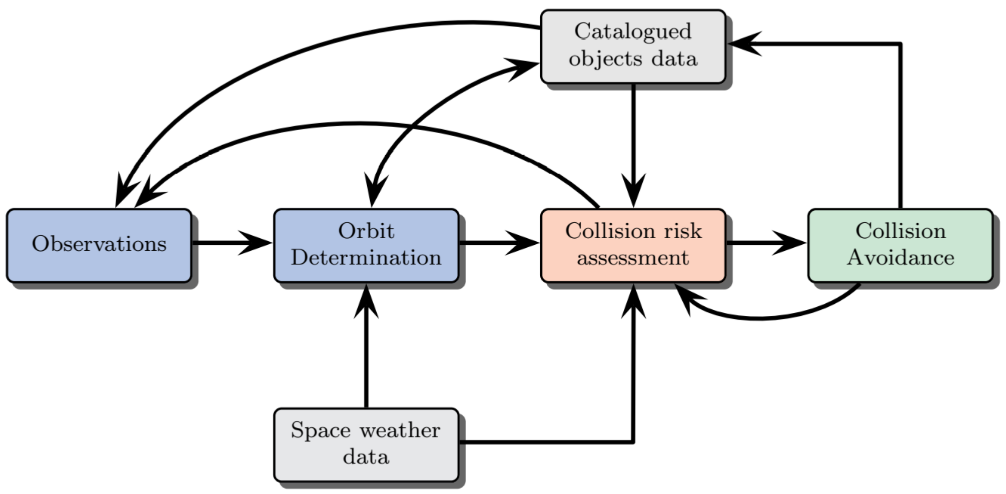


**Figure 1.2** The scheme shows, as a block diagram, the Space Surveillance and Tracking environment in which the track reconstruction algorithms (inside the orbit determination block) is placed [7].

Several international actors are involved in SST: US SST (Space Surveillance Network), a Russian surveillance network, and two European SST programs (Space Surveillance and Tracking). The last 2 just mentioned are ESST (European Space Surveillance and Tracking), managed by the European Commission, and SSA (Space Situation Awareness), controlled by the European Space Agency (ESA). The above programs use a large number of radars and electro-optical sensors placed all over the world with a detection threshold of a few centimeters in diameter for LEO objects and tens of centimeters in diameter for GEO objects [8]. Smaller sized debris can be detected in orbit, through active or passive instrumentation and subsequently cataloged. Through analysis of impacts on the outer surfaces of satellites, it is possible to characterize the density of small fragments. The sensors employed can be dedicated detectors, or on-orbit equipment meant for different tasks, such as the solar panels of the Hubble Space Telescope. The data obtained from observations (mainly radar and optical) is then processed to determine the orbital elements that are required to forecast the future trajectory of the object, through the algorithms contained in the orbit determination block. In addition to the six orbital parameters, also ballistic coefficients and area-to-mass ratios are estimated, which are needed to correctly model solar radiation pressure and air resistance.The information obtained is stored in a catalog, to be available when needed. The log is also useful to plan new observations and track known objects. This activity is especially critical for such elements whose orbit is greatly affected by perturbations, such as objects undergoing the reentry phase in the next few days or weeks. Moreover, the information contained in the catalogue could be used during the orbit determination phase. An object is defined as cataloged if it has received an international designation (COSPAR number), an orbit, and various characteristics and if these data are regularly updated.

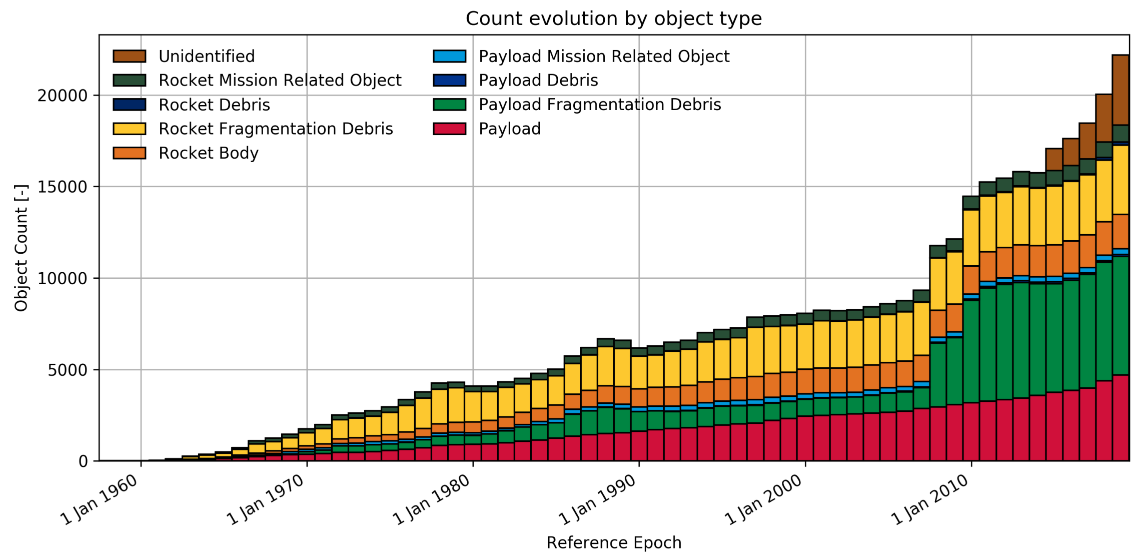


**Figure 1.3** Timeline of the number of space debris in orbit (plot available on the ESA website [9]

Fig. 1.3 reports the evolution of space debris population in orbit and the peaks resulting from previous collisions. Nowadays the regulation of orbital pollution is increasing more and more. The lifetime of space vehicles is reduced (25 years rule) by de-orbiting to the ground, or they can be placed in graveyard orbits. The probability of explosions is decreased by preventive systems, such as liquid and thermal passivation of the spacecraft at the end of its mission. In the future, the main source of space debris will likely be impacts between debris and operational systems [2]. The schedule of new observations must be as precise and reliable as possible to ensure sufficiently small uncertainties for accurate calculation of the collision risk assessment. Its operational management is in fact of maximum concern, although it only regards a very small number of objects in orbit (given that 5% of space objects are operational satellites). Indeed, fragmentation by collision with one of these objects can generate several additional thousands of debris [6].

## 1.2 State of the Art and proposed approach

### 1.2.1 Optical Telescope Observation

Space debris and satellites can be detected only and exclusively if the following conditions are met [10]:

- The object must be above the station's horizon;
- It must be illuminated by the sun;
- Its brightness must exceed that of the background sky by a certain margin.

There are two different techniques for optical debris tracking: staring (or survey) mode and chasing (or tasking) mode (see Fig. 1.4):

- Staring mode: the telescope is pointed at the sky and moves at sidereal speed, such that stars appear as dots in the background and debris as streaks, so the debris appears as a bright trail on the image. It sometimes happens that if more than one space object is in the telescope's FoV, multiple tracks are detected in a single observation. The light trail left by objects can be more or less long, depending on the orbit of the object.

- Chasing mode: as soon as the sensor detects an object, it starts moving at the same speed as the target, while acquiring the image. The object in the image appears as a single point, while other light sources appear as stripes. In this mode, there is no change in the brightness of the tracklets along the tracking.

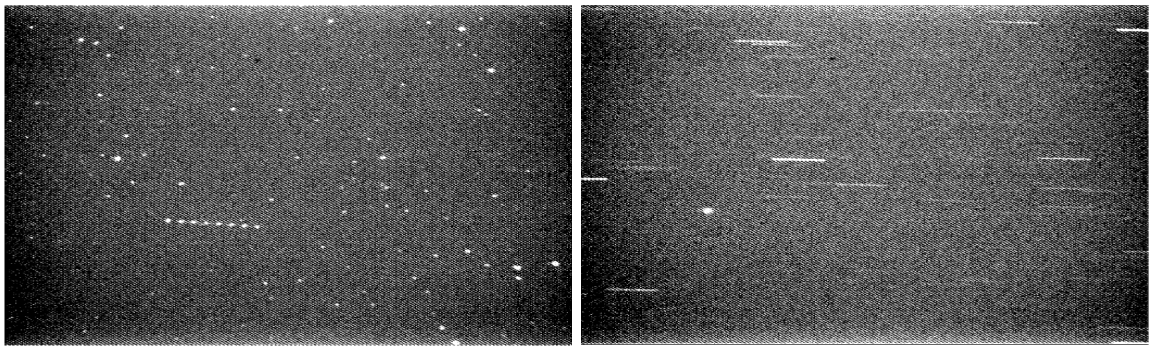


**Figure 1.4** In **(a)** staring, **(b)** chasing. Telescope observations of a GEO debris during an IADC debris campaign [11].

The absence of clear sky conditions hinders the detection of the satellite inside the FoV. Light and particle pollution are sources of disturbance in the vicinity of an observatory and should be as small as possible because they hamper the proper detection of tracks inside the FoV. It is worth noting that the debris does not emit its own light but must be illuminated by the sun or terrestrial light sources such as lasers, and its magnitude must be lower than that of the sky so that it is distinguishable and detectable from the background. An additional complication is the detection of objects with unknown characteristics (such as trajectory, distance from the ground, albedo, material, and size), in which case staring mode is adopted, as the observer is interested in detecting uncategorized objects. The speed at which an object passes can be problematic, as the telescope requires a certain amount of light to detect the bright signature. Then, if the object is too fast the instrument will not be able to identify the trace and the observation may fail. Focusing on tracking mode, if the object is moving too rapidly the telescope will not be able to spot it and the detection will fail.

### 1.2.2 Object Detection

One of the main applications of computer vision is object detection. Its methods generally fall into either neural network-based or non-neural approaches [12]. For non-neural approaches, it becomes necessary to first define targets features and then design an algorithm to detect them. Edge detection is an image processing technique for finding the boundaries of objects within images identifying discontinuities in brightness. ASTRiDE is one of the most used on astronomical images [13].
As explained by Zou et al. [14], neural network-based object detection consists in identifying instances of targets within an image and classifying them as belonging to a certain class (such as human, animal, or car). From the application point of view, it is possible to group object detection into two categories: "general object detection" and "detection applications" [15]. For the former, the goal is to investigate methods to detect different types of objects using a single framework, in order to simulate human vision and cognition. The latter refers to the recognition, under specific application scenarios, of objects of a certain class: this is the case of applications for pedestrian, face, or text detection. The currently developed models can be divided into two macro-categories: two-stage and one-stage detectors. The first case includes models that divide the task of object detection into several stages, following a "coarse-to-fine" policy. The

second one employs process attempts to complete the recognition in a single step using a single network. As shown in Fig. 1.5 several models exist for each category. The most prominent two-stage detector is Faster R-CNN, while the most popular one-stage detectors are SSD and YOLO.

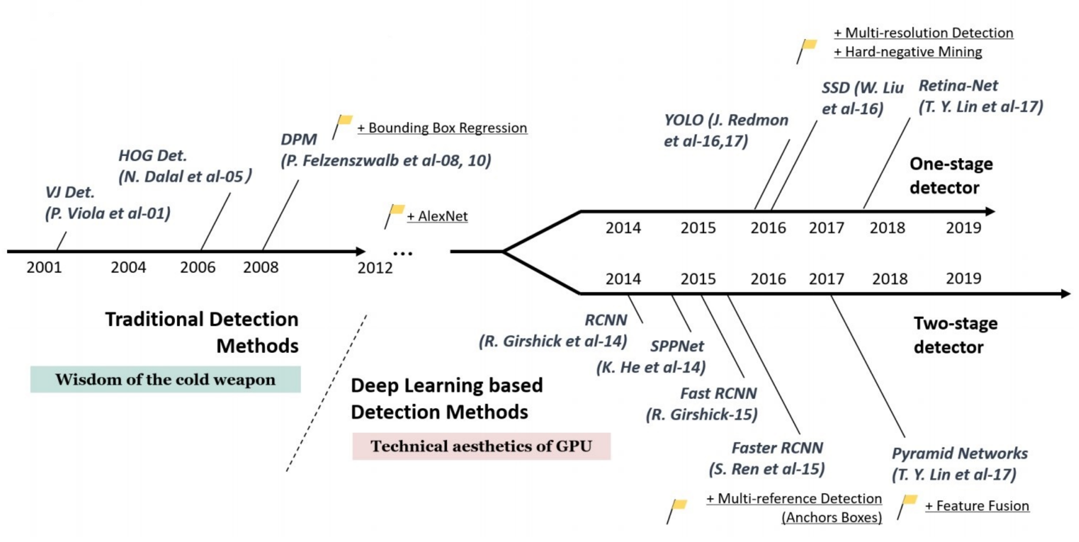

+ Multi-resolution Detection
+ Hard-negative Mining
SSD (W. Liu et al-16)
Retina-Net (T. Y. Lin et al-17)
YOLO (J. Redmon et al-16,17)
DPM (P. Felzenszwalb et al-08, 10)
+ Bounding Box Regression
HOG Det. (N. Dalal et al-05)
VJ Det. (P. Viola et al-01)
One-stage detector
+ AlexNet
2001
2004
2006
2008
2012
2014
2015
2016
2017
2018
2019
Traditional Detection Methods
Wisdom of the cold weapon
Deep Learning based Detection Methods
Technical aesthetics of GPU
RCNN (R. Girshick et al-14)
SPPNet (K. He et al-14)
Fast RCNN (R. Girshick-15)
Faster RCNN (S. Ren et al-15)
Pyramid Networks (T. Y. Lin et al-17)
Two-stage detector
+ Multi-reference Detection (Anchors Boxes)
+ Feature Fusion

**Figure 1.5** Timeline of the different Object Detection [16].

### Edge detection

One of the best known algorithms for border detection is ASTRiDE (Automated Streak Detection for Astronomical Images) [13]. It is a Python package that implements the functionality of streaks detection in astronomical images using the "border" for each object, i.e. "boundary-tracing" and their morphological parameters. Any streak-like traces in an astronomical image, caused by the passage of moving objects like satellites, space debris, or near-Earth objects, can be detected due to this improved method, capable of quantifying the shape of each border to determine whether or not it is a streak. The usual steps are:

1. Background removal: ASTRiDE first removes the background from the fits image. By default, It calculates background level and its standard deviation using sigma-clipped statistics. The background map is derived using Phoutils (an affiliated package of Astropy).

2. Contour map: using the scikit-image library, ASTRiDE derives the contour map of the fits image. The level of the contour is controlled by a threshold that is set automatically.

3. Streak determination: the algorithm recognises and removes stellar sources using morphological parameters derived from each boundary and keeps the tracklets to generate outputs.

However, this approach tends to fail if the brightness of the traces is not bright enough to stand out from the noise (clouds and light sources) and the background.

### Faster R-CNN

R-CNN series was developed by Ross Girshick et al. in 2014 improved with Fast R-CNN and Faster R-CNN. The idea behind RCNNs is relatively simple: they start by extracting a set of object proposals (object candidate boxes) using a selective search [17]. So, for each proposal, a fixed-size image is cropped and analysed by a trained CNN to extract its key features. Finally,

linear Support Vector Machine (SVM) classifiers are used to decide the presence of an object in each region and to recognize the categories of the found objects.
The main innovation introduced by Faster R-CNN is the Region Proposal Network (RPN) that implements a very efficient region proposal system, almost free of computational costs when compared to previous models.

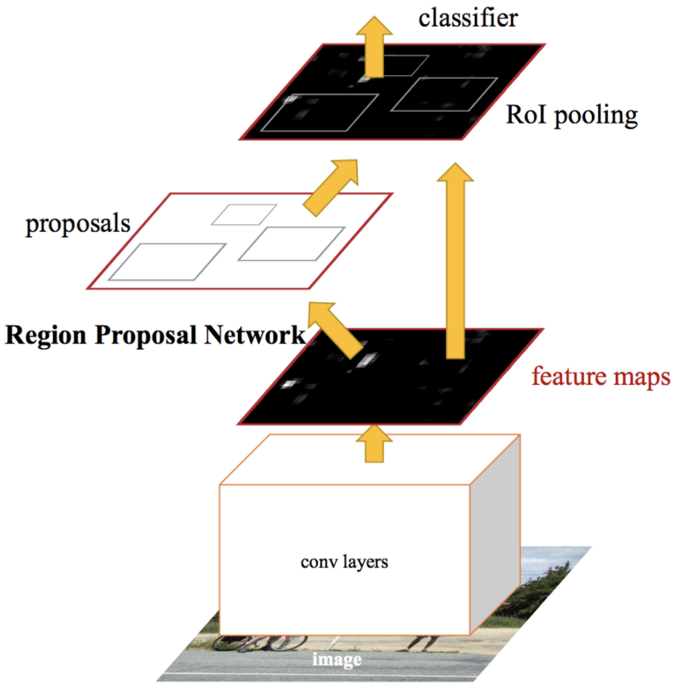


**Figure 1.6** Architecture of Faster R-CNN [18].

The RPN consists of a series of convolutional layers applied to the feature maps obtained from the initial layers of the Fast R-CNN network. To generate the region proposal, Ross Girshick et al. opt to integrate a small network that takes as input a feature map of dimension nxn and consists of three convolutional layers. A first shared level of dimension nxn and two "twin" levels entirely connected of dimension 1x1 (see Fig. 1.6): fixed the hyperparameter k, which denotes the maximum number of proposals to be advanced for each location, one of the two twin levels (reg) produces in the output the coordinates of k "bounding boxes"; the other twin level, (cls) returns, for each proposed bounding box, the probability that there is or is not an object in it.
Starting from the R-CNNs, to reach the Faster R-CNN models, the majority of the individual modules of an object recognition system have been gradually integrated into a single end-to-end framework. Although Faster R-CNN greatly exceeds the speed of Fast R-CNN, there is still redundancy in computation. A wide variety of network enhancements have been introduced subsequently, including RFCN and Light head RCNN.

### YOLO

YOLO algorithm was proposed by J. Redmon et al. in 2015. It was the first one-stage detector in the deep learning era. It is extremely fast: the fastest version runs at 155 fps with a mean Average Precision (mAP) equals to 52.7% on the test dataset VOC07, while a more accurate version runs at 45 fps with a mAP = 63.4% on VOC07. The name YOLO stands for "You Only Look Once". In this case, the approach differs from the previous algorithms since it applies a single model to the entire image. YOLO divides the image into regions, predicts the bounding boxes, and, for each of them, determines the probability of belonging to a certain class.

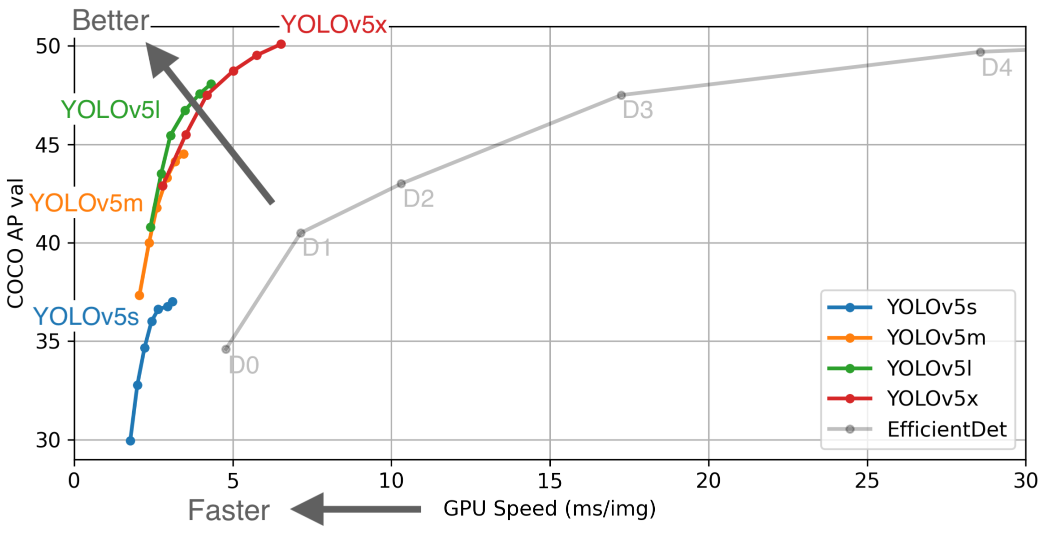


**Figure 1.7** YOLOv5 models' performance [19].

The latest official version of YOLO published in scientific papers is v4, but in June 2020 Gleen Rocher released a GitHub repository containing the code of a new version 5, which has sparked controversy in the scientific community. These two versions are based on different architectures, discussed in more detail in Chap. 3. The new releases have better recognition accuracy while maintaining very high execution speed. Fig. 1.7 shows the performances obtained by the several models of YOLOv5. However, despite the vast improvement in recognition times, YOLO suffers from reduced accuracy in object detection compared to two-stage detectors, especially for smaller objects.

## SSD

Single-Shot Detector, or SSD, was proposed by W. Liu et al. in 2015. It was the second one-stage detector in the era of deep learning. The main contribution of SSD was the change of perspective towards the bounding box generation: unlike previous models that were concerned with accurately predicting the location of an object within the image, SSD starts from a set of default bounding boxes. Starting from this set, SSD infere, for each of these default bounding boxes, a deviation. For each bounding box, translated by the predicted deviation, the model performs the classification. SSD achieves good prediction accuracy thanks to different filters, chosen according to the image proportions and the "multiple" feature maps, each obtained in a different point of the convolutional layers (see Fig. 1.8).

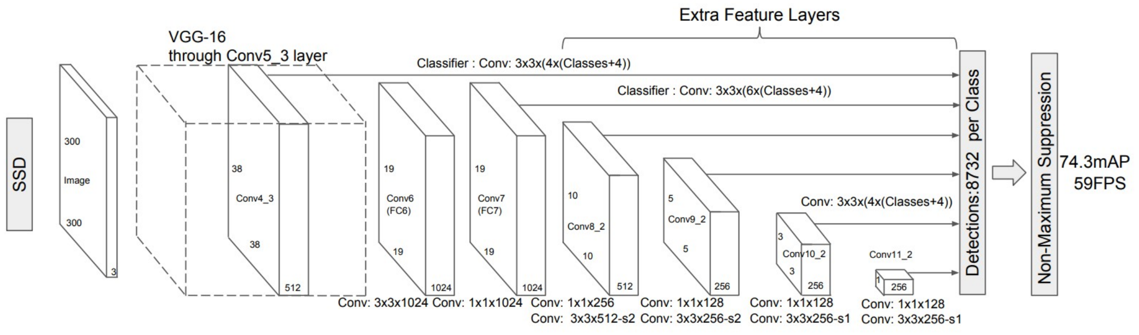

SSD
Image
VGG-16 through Conv5_3 layer
Conv4_3
Conv6 (FC6)
Conv7 (FC7)
Conv8_2
Conv9_2
Conv10_2
Conv11_2
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256 Conv: 3x3x512-s2
Conv: 1x1x128 Conv: 3x3x256-s2
Conv: 1x1x128 Conv: 3x3x256-s1
Conv: 1x1x128 Conv: 3x3x256-s1
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS

**Figure 1.8** SSD architecture [20].

Avoiding the bounding box prediction step saves the SSD model a significant amount of execution time. On the other hand, the use of different feature maps significantly improves the

accuracy of the model on low resolution images, especially when recognizing small objects. SSD has advantages in both speed and recognition accuracy, running at 59 fps using one of its fastest implementations.
The main difference between SSD models and previous networks is that it recognizes objects of different sizes and layers, whereas previously it was performed across the very last layers.

### 1.2.3 Object Tracking

Object tracking is the process of locating moving objects over time in videos. There are many types of trackers used for different applications, but all of them use the process of linking objects in different frames. They are able to identify an object that exits the FoV for a few photograms (even tens) and then re-enters it. A detector is not able to track by itself, even through multiple detections, because it cannot identify targets in different frames unless it is supported by a dedicated software. A moving object is contained in a frame sequence the visual appearance of the object is not clear. In all such cases, detection would fail while tracking succeeds as it also considers the motion model and history of the object [21]. A lot of traditional tracking algorithms are built into the OpenCV [22] tracking API. Most of these trackers are not very accurate compared to those that are based on machine learning. However, they can sometimes be useful to run in a resource-limited environment such as an embedded system. In terms of accuracy instead, deep learning based trackers are much more advanced than traditional trackers. The most widely used trackers are of three types: Offline Training Trackers, Online Training Trackers, LSTM + CNN video object trackers.

#### Conventional tracking methods

These types of algorithms are designed in order to be independent from the specific detector or tracker, coming from the CNN training and OpenCV [22] library respectively [23]. Basically, the pipeline performs a decision between applying a new detection or tracking the existing objects. The former algorithm looks for new objects and identifies very precise bounding boxes around the existing ones, but it is time consuming. The latter propagates the detected bounding boxes without searching for additional ones, but it saves computational time. After the first frame, in which the detector is always applied, the OpenCV [22] tracker follows the identified objects and generates new bounding boxes around them. At this point, for each couple of adjacent frames, the pipeline checks if the targets tracks are moving too fast or the targets shapes are changing significantly (by measuring the intersection of the corresponding bounding boxes coming from the adjacent frames). Then, the local and global image similarity are computed to assess if something changed in the scene. In any case, the pipeline takes advantage of an exit strategy based on a maximum frames interval for a new detection. If no relevant changes are detected in the scenario after a tunable number of frames, the algorithm performs a detection in order to refine the existing bounding boxes and verify if new objects appeared in the scene without triggering the similarity indexes [23].

#### CNN offline training trackers

These trackers were the first to use CNNs to detect objects in videos. The architecture is based on offline learning and consists of an offline convolutional neural network trained through many videos for general object tracking. These softwares can track objects that are not part of the training dataset. GOTURN is the most famous offline tracker and thanks to the analysis of several frames at the same time it is able to identify and track objects in the input video. Both frames pass through a bank of convolutional layers (see Fig. 1.9). The layers are simply the first five convolutional layers of the CaffeNet architecture (a deep learning framework). The outputs of these convolutional layers are concatenated into a single vector of length 4096 elements. This vector is the input for 3 fully connected layers. The last fully connected layer is finally connected to the output layer containing 4 nodes representing the top and bottom points of the bounding box. Due to the use of GPU this software has a speed of 100 fps with fairly good accuracy.

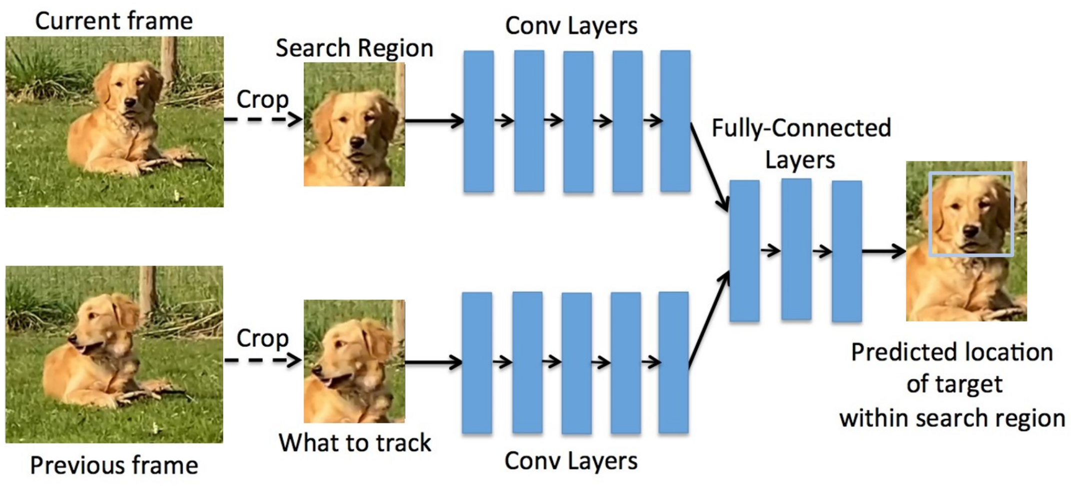


**Figure 1.9** GOTURN operating diagram [24].

### CNN online training trackers

These are online training trackers which use Convolutional neural networks. Due to the high computational cost associated to their training, small networks are used. As a consequence, their shallow structure prevents them from reaching high accuracy. One possible solution to this issue is to train a large network and use only the top layers to extract features from objects. In this way, the last layers can be trained online. The goal is to train a CNN that detects targets and background, but since the target of one video can be the background of another, the network is usually organized in two parts (see Fig. 1.10): the first section is shared, while the second one is independent for each domain (meaning an independent training video).

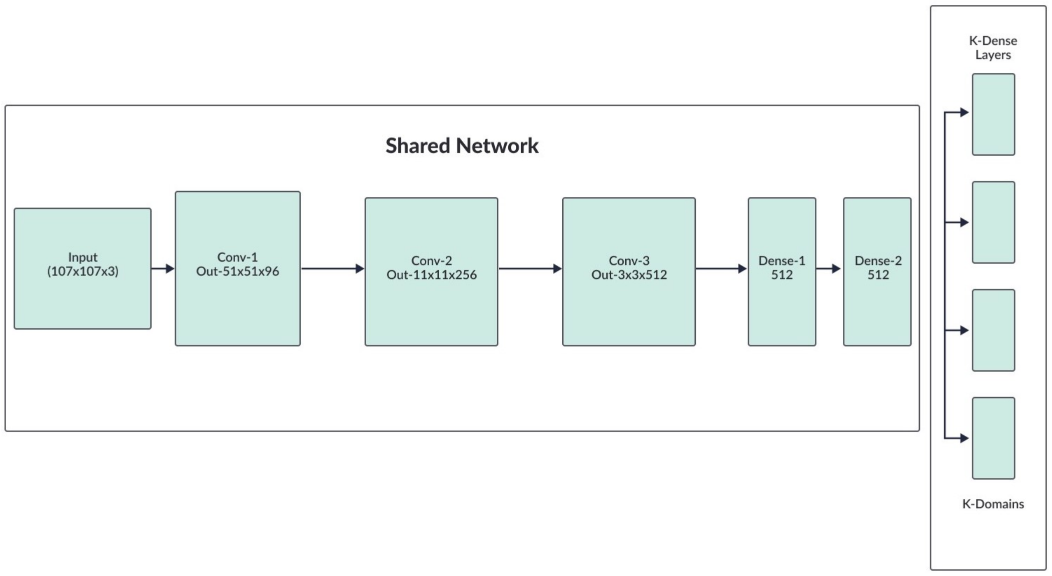


**Figure 1.10** Online training tracker architecture [21].

The training iteratively relies on K-domains that classify the target and the background. This helps to extract information from the videos to learn a better generic representation of the tracking task. After this step, the domain-specific binary layers are removed and a feature extractor (shared network) is obtained that can distinguish between any target object and the background in a generic way. During inference, the initial shared part is used as a feature extractor and the domain-specific layers are replaced with a binary classification layer. This layer is trained online. In each step, the region around the previous target state is searched for the object by random sampling. One of the most used trackers is DeepSORT, an extension to SORT (Simple Real time Tracker). It is based on multiple detections of the YOLO network (see chapter 3.3). Sequentially, it analyses frames of video tracks to detect and track objects giving them an identification code (ID). In order to work, it is necessary that, between the current frame and the previous one, the bounding boxes of the detected targets are overlapped (a threshold value must be set). Thanks to the ID and the overlapping, the framework tracks the various objects.

### LSTM + CNN detection video object tracker

Trackers based on Long Short Term Memory (LSTM) and CNN are becoming very popular in recent years because they use the advantages of both these methods. There are two reasons why LSTM with CNN are increasingly employed:

- LSTM networks are particularly good at learning historical patterns, so they are particularly well suited to visual object tracking.
- LSTM networks are not very computationally expensive, so it is possible to build very fast real-world trackers.

Recurrent YOLO (ROLO) is one such tracking algorithm based on the online detection of a single object. It uses the YOLO network to detect the object and an LSTM network to find the trajectory of the target object. As Fig. 1.11 shows: the input frames go through the YOLO network, from the YOLO network two different outputs (Image Features and bounding box coordinates) are taken, these two are given to the LSTM section, which elaborates the trajectories and the bounding box of the object to be tracked.

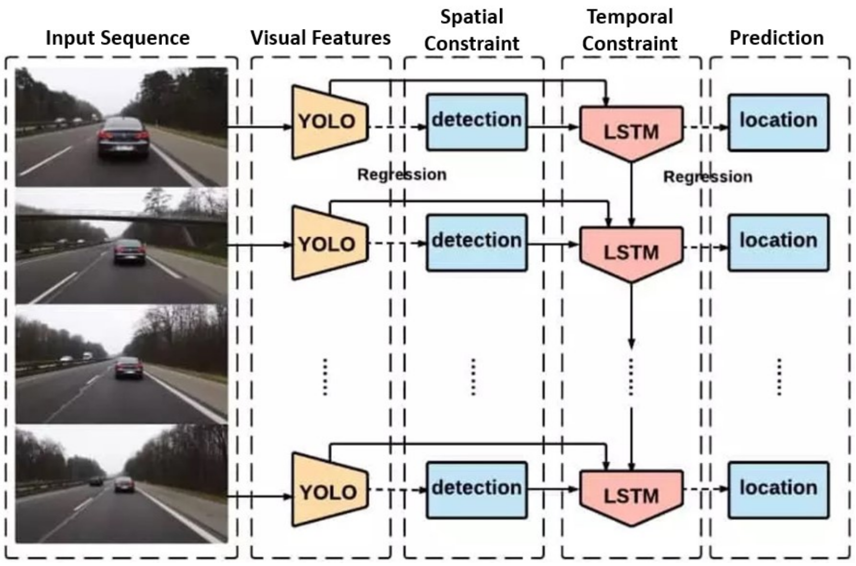


**Figure 1.11** ROLO operating diagram [21].

The preliminary location inference (from YOLO) helps the LSTM to pay attention to certain visual elements. Thus, ROLO explores the spatio-temporal history along with location history. Even when YOLO detection is flawed due to motion blur, ROLO tracking stays stable. Such trackers are less likely to fail when the target object is occluded.

**Criticalities**

Even with these valid methods, some criticalities remain unsolved:

- One-stage detectors are efficient and suitable for fast analysis of telescope observations but have difficulties with very small objects, e.g. GEO satellite tracks with short exposure times.
- Two-stage detectors, on the other hand, do not suffer from a drop in performance when detecting small objects but are much slower, so they cannot be used to design fast systems.
- Most open-source and commercial trackers require video as input, i.e. a sequence of images in which the bounding boxes of the targets overlap frame by frame. In case of real observations, this does not happen due to the lag of the telescopes.
- Most of today's track detection systems use a mapping of the sky, in particular maps, to check for extraneous light streaks and then detect passing tracks. This process is very computationally intensive.

### 1.2.4 Thesis purpose and workflow

This thesis aims to explore new approaches to the problem of spatial object detection and tracking by analysing their spatial trace using real images, as opposed to traditional methods. As far as object detection is concerned, it aims at reducing the computational time that usual telescope stations activity of processing and extracting traces present while maintaining high quality. For the processing and analysis of the observations, an algorithm was design to processes the images and then uses artificial intelligence to detect the targets. It is a complex task due to the variable brightness of the tracks in relation to noise and background, and the effort required to generate an efficient and accurate model. It could be a viable alternative to the heavy and often intricate algorithms used by observers. The LEO and MEO objects tracking part is the most challenging. There are many open source and commercial trackers that track the movement of targets by analysing the overlap of the bounding boxes of the various frames. This is not the case for real observations, as the telescope has a lag between shots. Therefore, a tracker was designed which is based on a statistical analysis of the objects' trajectory and makes decisions thanks to the machine learning model developed. The purposes of this algorithm are to improve the accuracy with which observed known objects are catalogued and to catalogue unknown objects automatically. It has been tested using simulations of real object passages. This tracker could be a valid alternative to address space surveillance.

The document is structured as follows: Chap. 2 presents the main notions of telescopes, Machine Learning (with a focus on fully connected and convolutional neural networks, Deep Learning, and YOLO), and finally, a focus on the tools used. Chap. 3 describes the datasets employed to create the artificial intelligence models needed to build algorithms. With chapters 4 and 5 the focus shifts to the design of two scripts that aims at improving the traditional techniques of detection and tracking of tracklets. In particular, Chap. 4 explains the design of a detector based on real observations and artificial intelligence for the detection and location of tracklets in the FoV (real images were provided by a military observatory at Pratica di Mare). Chap. 5 outlines the design of a tracker. It estimates the position of the object in the FoV by statistical analysis of the trajectory of orbiting objects in order to observe it again by moving the telescope automatically. Finally, conclusions and possible future developments are discussed.

# Chapter 2

# Fundamentals

## 2.1 Ground based telescopes

During astronomical observations, the main tasks of a telescope are [25]:

1. collect light from a large area, making it possible to study very faint sources;
2. increase the apparent angular diameter of the object and thus improve the resolution of the measurement;
3. measure the positions of objects in the sky.

Optical telescopes are divided into two categories according to the surface that collects light: lens telescopes or refractors if the surface is a lens, mirror telescopes or reflectors if the surface is a mirror (see Fig. 2.1). With ground based telescope the atmosphere affects observations in many ways. The air is never quite steady and there are layers with different temperatures and densities; this causes convection and turbulence [6]. When light from a star passes through the various layers of the atmosphere there are continuous refractive changes in different directions, so the amount of light reaching a detector varies constantly, which is why the star appears to scintillate. A telescope gathers light over a larger area, so the rapid changes are smoothed out and the scintillation diminishes. Instead, differences in refraction along different light paths through the atmosphere speckle the image, and point sources appear as vibrating specks. The phenomenon just described is called seeing, the size of the seeing disk can be less than a second of arc, up to several tens of seconds of arc. Therefore, when looking through a telescope, small details, such as a planet, are obscured due to seeing and scintillation.

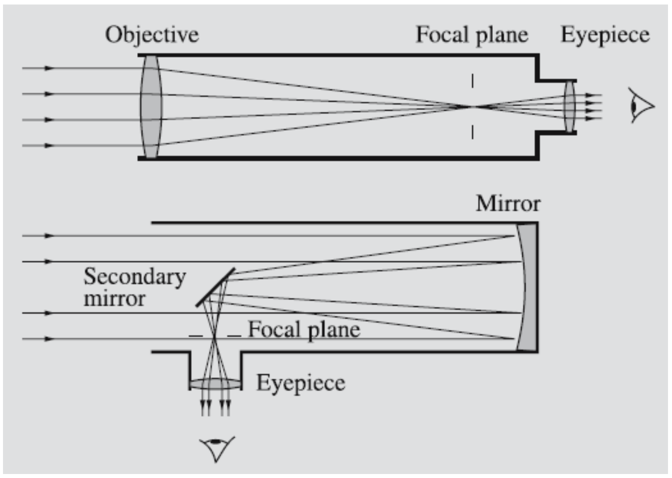


**Figure 2.1** Refractor and reflector telescope [25].

Most of the wavelengths in the electromagnetic spectrum are absorbed entirely by the atmosphere. A very important range of transparency exists between 300 and 800 nm, which coincides with the sensitivity region of the human eye (about 400-700 nm). To reduce light pollution as much as possible, ground-based telescopes are placed far from urban areas. They are also located in places with high altitudes and in dry regions, where the absorption of water vapor is reduced, in order to reduce the amount of light pollution.

### Refractor telescopes

Refractors consist of two types of lenses: the objective, which collects incoming light to form an image on the focal plane, and the eyepiece, which is a small magnifying glass that allows looking at the image. The lenses are mounted at opposite points on the cylinder, which is the structure of the telescope, and it can be directed to any desired point (see Fig. 2.2). The distance between the eyepiece and the focal plane can be adjusted to bring the image into focus, possibly using a sensor.

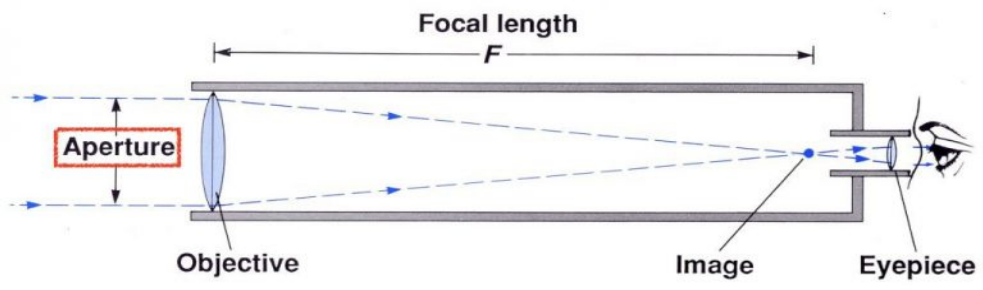


**Figure 2.2** A simplified scheme of a refractor telescope, highlighting the main components and geometrical quantities [26].

The focal length is effectively the length of the telescope. It is measured as the distance from the main optic to the point where the image is formed [6]. The diameter of the objective is named aperture. The ratio between aperture and focal length is called the aperture ratio and is used to characterize the light-gathering power of the telescope. If the aperture ratio is large, tending toward unity, you have a powerful and 'fast' telescope, i.e., photographs can be taken with short exposure times as the image is bright. Conversely, a small aperture ratio (so the focal length is much greater than the aperture) implies a 'slow' telescope. The inverse of the aperture ratio is the $f/N$ focal ratio.
The first refractors built acquired poor quality images, due to the chromatic aberration of the simple lenses installed [26]. The glass employed refracts different colors in different quantities, so the colors do not collide in a single focal point. In particular, the focal length increases as the wavelength of the received signal increases. In the 18th century, achromatic lenses composed of two parts were used to eliminate aberration. The color dependence of the focal length is much less than for a single lens, and at a certain wavelength $\lambda_0$ the focal length has an extreme (usually a minimum). Below this minimum, the variation of focal length with respect to wavelength is very small. Combining three or more lenses further improves chromatic aberration (such as apochromatic lenses).

### Reflector telescopes

Most telescopes employed for space missions are mirror or reflector telescopes (see Fig. 2.1) [25]. They are characterized by a mirror, usually parabolic in shape, covered by an aluminum layer that acts as a collecting surface. The advantage of the parabolic shape is that it converges all light rays, which enter parallel to the telescope axis, through a single point. Once shaped, the image can be visualized through an eyepiece or recorded through a detector.
The greatest advantage of reflectors is that they do not suffer from chromatic aberration due to the design of their lens. However, its architecture has the following limitations: first of all, the coma aberration makes point sources (stars) at the center of the image focused to a point appear as "comet-like" radial smudges, that get worse towards the edges of the image [6]. Secondly, the curvature of the field causes the image plane to be curved but this may not correspond to the shape of the detector and leads to focus error. This problem can be corrected by a field flattening lens. Another flaw is astigmatism, which is responsible for the azimuthal variation of the focus around the aperture generating images off-axis point sources to appear elliptical. This problem is usually experienced analysing large fields of view, while it does not influence narrow ones, as it varies quadratically with the angle of view. Lastly, distortion does not affect image quality (sharpness) but does affect object shapes. It is sometimes corrected by image processing.

### Optical sensor

Optical sensors work by gathering light in the visible and infrared range from an external source of interest (see Fig. 2.3).

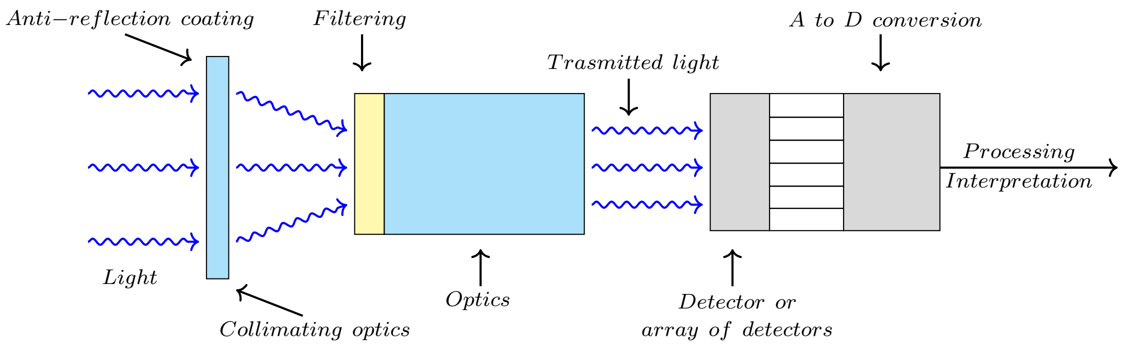


**Figure 2.3** Optical sensor scheme [6].

The static optic is the first component of the receiving chain, it is coated with an anti-reflective layer to capture all incident light [27]. The beams are collimated to fit subsequent lenses that use additional collimation optics and filters to select subsets of wavelengths. Finally, the light radiation is converged into a detector that converts the photons into voltage, followed by amplification, digitization, and then processing. CCD (charge coupled device) and CMOS (complementary metal oxide semiconductor) image sensors are the most widely used technologies for capturing images digitally [28]. Both types of imagers convert light into electric charge and process it into electronic signals. In a CCD sensor, every pixel's charge is transferred through a very limited number of output nodes (often just one) to be converted to voltage, buffered, and sent off-chip as an analog signal. All of the pixels can be devoted to light capture, and the output's uniformity (a key factor in image quality) is high. In a CMOS sensor, each pixel has its own charge-to-voltage conversion, and the sensor often also includes amplifiers, noise-correction, and digitization circuits, so that the chip outputs digital bits. These other functions increase the design complexity and reduce the area available for light capture. With each pixel doing its own conversion, uniformity is lower, but it is also massively parallel, allowing high total bandwidth for high speed.

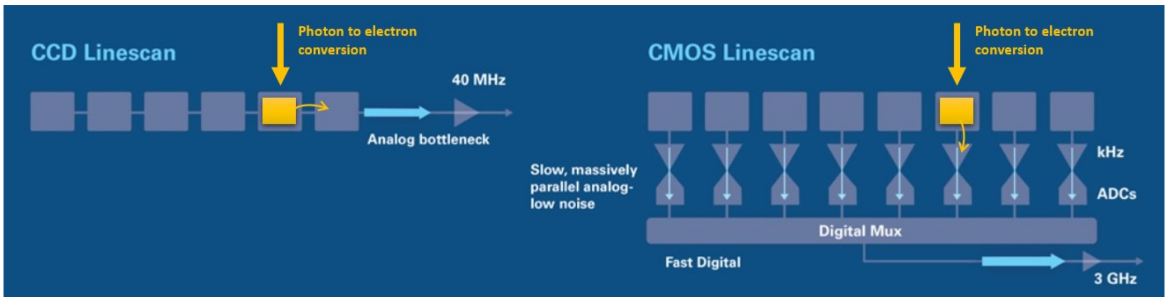

CCD Linescan
Photon to electron conversion
40 MHz
Analog bottleneck
CMOS Linescan
Photon to electron conversion
Slow, massively parallel analog- low noise
kHz
ADCs
Digital Mux
Fast Digital
3 GHz

**Figure 2.4** CCD and CMOS architecture scheme. [28].

CCD and CMOS imagers differ in the way that signals are converted from signal charge to an analog signal and finally to a digital signal (see Fig. 2.4). In CMOS area and line scan imagers, the front end of this data path is massively parallel. This allows each amplifier to have low bandwidth. By the time the signal reaches the data path bottleneck, which is normally the interface between the imager and the off-chip circuitry, CMOS data are firmly in the digital domain. In contrast, high speed CCDs have a large number of parallel fast output channels, but not as massively parallel as fast CMOS imagers. Hence, each CCD amplifier has higher bandwidth, which results in higher noise. Consequently, quick CMOS imagers can be designed to have much lower noise than high speed CCDs.

### Pratica di Mare ground station

ITAF (Italian Air Force) has recently installed an optical sensor for SST in Pratica di Mare AFB (Air Force Base) that is operated by personnel of Aero-Space System Engineering Group of Flight Test Wing [29]. PdM-MITE (Pratica di Mare Military Telescope) is a telescope properly designed for SST by GMSpazio and Officina Stellare and is able to observe the portion of space above it with coverage of 360°x90° in azimuth and elevation (see Fig. 2.5). It is built with an exclusive Riccardi-Honders flat field optical design, with a diameter of 350 mm and a focal ratio *f/2.8* (see the Officina stellare website for further information [30]) [6].

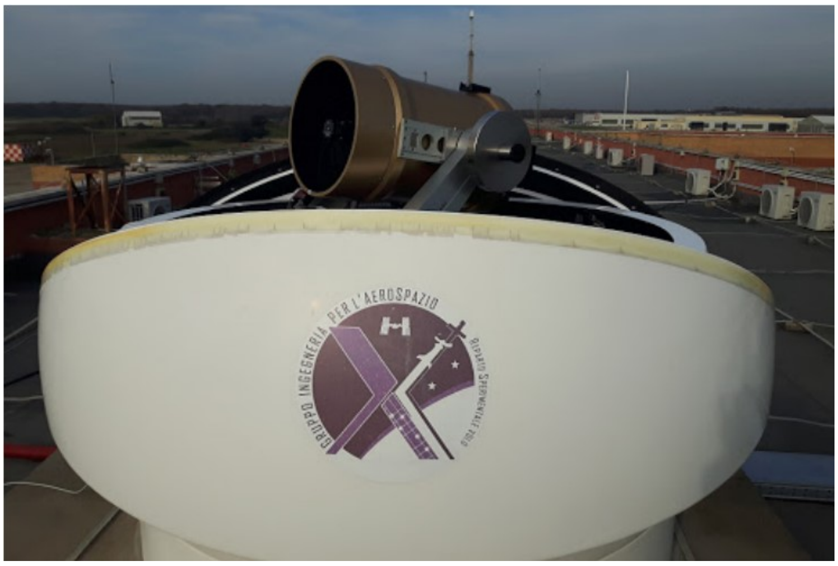


**Figure 2.5** The telescope for SST at PdM-MITE [31].

The telescope is equipped with two CCD sensors, one with a wide field of view used for surveillance and the second with a narrow FoV used for tracking specific objects. The second one ( from which the images used for this thesis were taken) has a resolution of 4096x4096 pixels, leading to 16$mpxl$ detector. It measures approximately 37$mm$ on both sides. The platform on which the telescope is mounted is an equatorial arrangement and is set up for high speed movement for tracking and pointing. It is also equipped with a remotely controlled filter wheel capable of supporting five standard filters allowing it to investigate the spectrum of light reflected from the target. The sensor has been designed for the following operating modes [6]:

- Survey: the sensor can scan a portion of the sky selected by the user in order to detect any objects in that area; as default, it can scan the area following spiral or linear movements, otherwise the operator can set some waypoints for a specific search path;
- Tracking: cued by an ephemeris file, the sensor can track an object passing in the sensor's field of view.

The acquired images are analyzed and processed for orbital parameter extraction, then for each image there is an automatic star matching process (based on known stars found in the stellar database). Time synchronization is performed by a GPS sensor, which allows knowing the exact time of the shot. The extraction of the coordinates of the first and last pixel is a choice still in testing phase that was made to increase the number of data available for orbit determination (see Fig. 2.6).

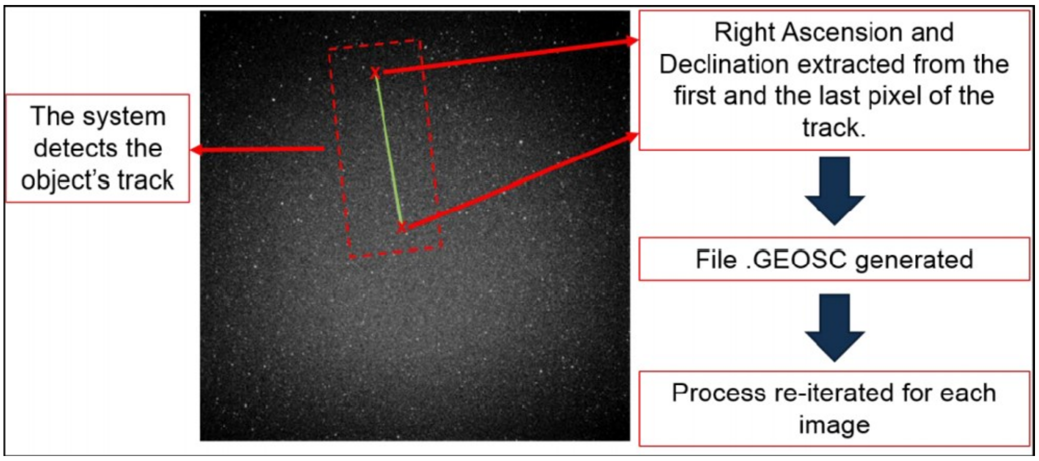

The system detects the object's track
Right Ascension and Declination extracted from the first and the last pixel of the track.
File .GEOSC generated
Process re-iterated for each image

**Figure 2.6** Tracklet extraction during an observation campaign at PdM-MITE [29].

Most observed satellites are LEO, MEO (Mid Earth Orbit), and GEO, so assuming favorable weather conditions and full-time sensor availability, the number of data available in different cases are [6]:

- for a LEO satellite, typical in-sight times are within 5-8 minutes. So, assuming a shot every 30 seconds, up to 16 images can be obtained from a single transit. Considering that a LEO satellite is in-sight for up to three passes per night, usually two, the number of images can significantly increase (40-50);
- for a MEO satellite, typical in-sight times are up to 6 hours so the number of images acquired per night can be up to several hundred;
- or a GEO satellite, in-sight times are all night long so the number of images acquired per night can be more than one thousand.

After analyzing the tracks and extracting the coordinates, the parser program generates a ".geosc" file containing all the data to feed the orbit determination software together with a FITS file containing the main information of the analysed passage. Once the orbital parameters have been retrieved from the images and the *.geosc* file has been created, the AGI software Orbit Determination Tool Kit (ODTK) is used to perform orbit determination. The data are processed by the program to extract the ephemeris of the tracked object, with the purpose of progressively refining the determination by performing day-by-day observations. The complicated ODTK software can be modified by adapting the numerical model to achieve accurate determination and prediction of the orbit of a recorded target. In particular, the more accurate the numerical model, the more accurate the orbit obtained will be.

## 2.2 Machine Learning

Machine learning is a subsystem of artificial intelligence that allows systems to automatically learn and improve their performance from experience without being explicitly programmed. The learning process relies on observations, data, or instructions in order to look for patterns in them to make decisions in the future based on the examples provided. The goal is to build a system that autonomously takes sensible decisions without regular human assistance and support [32]. There are three different categories based on the learning process [6]:

- Supervised learning: the machine receives examples coupled with their labels, i.e. the solution of the problem. Starting from the provided dataset, the system generates a correlation, therefore a set of rules, between input and output.
- Unsupervised learning: In this case, there are no labels attached to input data, so the machine elaborates models grouping them in clusters according to similarities.
- Reinforcement learning: the machine follows a goal-oriented algorithm to accomplish a complex objective. This method allows agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the "reinforcement signal".

The first case is the one used for this work.

### Supervised learning applications

There are two tasks in which Supervised Learning is commonly used [32]:

- Regression: the algorithm must predict the value of a continuous response variable. Generally, this type includes predicting sales of a new product, or the salary for a job based on its description.
- Classification: the software must establish discrete values. That is, predict the most likely class, category, or label for new examples. Applications of classification include spam detection, revenue prediction, sentiment analysis, dog breed detection, and so on.

In both cases, the algorithm is based on the minimization of a loss function, built on input data and weights. The latter are usually clustered in a $W$ matrix. To make the topic more useful to understand the linear regression case is firstly explained to introduce typical parameters and functions involved in neural networks since the classification problems are similar to the regression ones.

### Linear Regression

Simple linear regression is a type of regression analysis in which there is only one independent variable and there is a linear relationship between the independent variable ($x$) and the dependent variable ($y$) [33]. This relationship is formally represented as a mapping function $g$:

$$g(w, x) = wx + b \tag{2.1}$$

Where $W$ is a weighting coefficient and $b$ is a bias term (or intercept). The identity $y = g(w, x)$ must be satisfied to link the input and output of the regression problem (see Fig. 2.7). The formulation described is used for the scalar case, i.e., $x$ and $y$ are scalar quantities, but it can also be easily applied to the vector case where $x$ and $y$ are vectors:

$$\mathbf{g}(\mathbf{W}, \mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.2}$$

where $\mathbf{W}$ is the weighting matrix and $\mathbf{b}$ is the biases vector. In this case the input-output relationship is $\mathbf{y} = \mathbf{g}(\mathbf{W}, \mathbf{x})$. The statistical technique described attempts to compute the best values of $\mathbf{W}$ and $\mathbf{b}$ to better describe the regression passing through the input points.

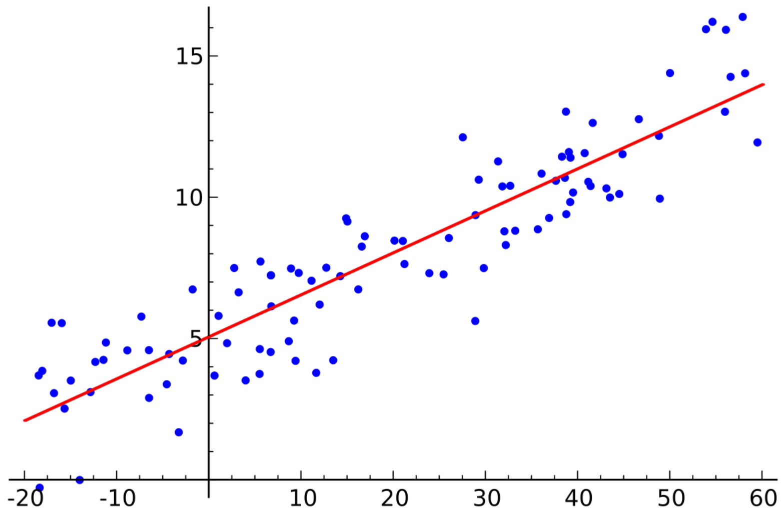


**Figure 2.7** An example of a scalar linear regression: the blue dots represent the input data points, while the red line is the function obtained as output [6].

As regards loss function definition, the scalar case is reported for the sake of simplicity. The loss (usually $J$) is used to better understand the values of $W$ and $b$ and thus to generate the best regression line. The search problem is converted into a minimization problem whose goal is to reduce the error between the predicted and true values as much as possible. The usual loss functions are MAE (Mean Absolute Error) and MSE (Mean Squared Error) [18]. Mean Absolute Error is defined as:

$$J_{MAE} = \frac{1}{n} \sum_i \mid y_{i,pred} - y_{i,exact} \mid \tag{2.3}$$

where n is the number of samples, $y_{i,pred} = Wx_i + b$ is the problem predicted solution, and $y_{i,exact}$ is the exact one. As for the Mean Squared Error [6]:

$$J_{MSE} = \frac{1}{n} \sum_i (y_{i,pred} - y_{i,exact})^2 \tag{2.4}$$

The idea is to start with some values for $w$ and $b$ and then to change these values iteratively to reduce the loss. Gradient descent helps to enhance the values.
The method is based on a progressive update of $w$ and $b$ using gradients from the loss function [6]. To find these gradients, partial derivatives of $J$ are computed concerning the parameters to be tuned. They are then multiplied by a scalar, also called learning rate ($\alpha$) to define the update step in the following way:

$$\begin{cases} W_{i+1} = W_i - \alpha \cdot \dfrac{\partial J}{\partial W} \\ b_{i+1} = b_i - \alpha \cdot \dfrac{\partial J}{\partial b} \end{cases} \tag{2.5}$$

Learning rate $\alpha$ is a hyperparameter, which means that it must be chosen a priori and tuned according to the performance of the algorithm. A smaller learning rate might approach the minima but take more time to reach them, a larger learning rate converges faster but there is a chance of overshooting the minima. Also, sometimes the cost function may be a nonconvex function where local minima increase the difficulty of convergence, but for linear regression, it is always a convex function.
More elaborate minimization procedures are adopted as the complexity of regression problems increases (see Sec. 2.2) [6].

### Linear Classification

Classification predictive modeling is the task of approximating a mapping function ($\mathbf{g}$) from input variables ($\mathbf{x}$) to discrete output variables ($\mathbf{y}$) [34].

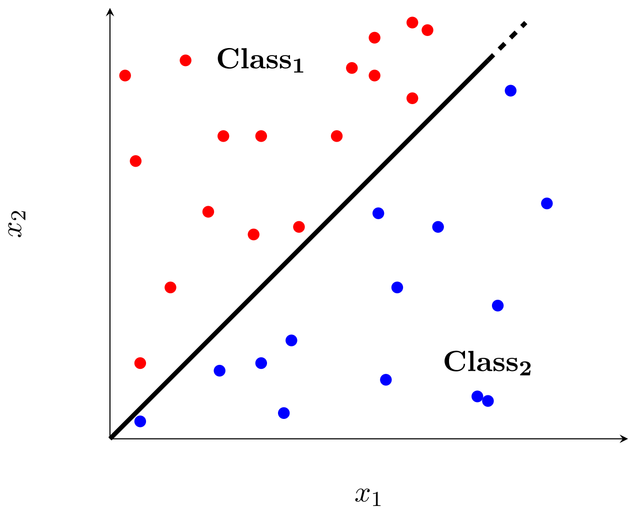


**Figure 2.8** The red points are associated to **Class$_1$**, the blue ones to **Class$_2$**. The two regions are split by a line defining a possible decision boundary [6].

The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation [6]. For the linear case shown in Fig. 2.8, the mapping function is a linear combination:

$$\mathbf{g}(\mathbf{W}, \mathbf{x}) = \mathbf{W}\,\mathbf{x} + \mathbf{b} \tag{2.6}$$

where $\mathbf{W}$ is a matrix of weights (consisting of 2 rows, one per class, and 2 columns, one per input item) and $\mathbf{b}$ is a bias vector (2 items, one per class). The output of the mapping function is a percentage score used as input for a loss function. The problem can be visualized as in Fig. 2.8: the blue and red dots correspond to two different classes, the black line was drawn to show this distinction between the two classes. Linear classification is used very frequently for the analysis of images, which are converted into 3D matrices if colored, or into 2D matrices if in grayscale. To simplify the concept, we consider an image as a 2D matrix of 4 pixels (see Fig. 2.9).

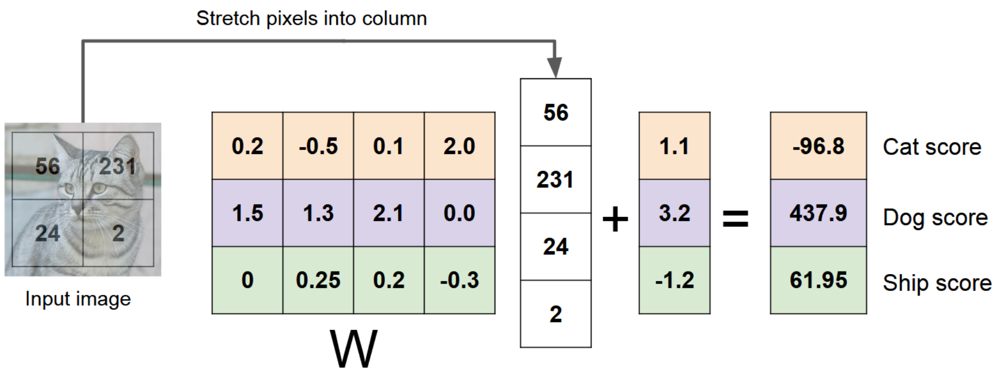


**Figure 2.9** Example of linear classification on a 4 pixels picture [18].

The first step is to deploy the image, hence the 2D matrix, into an input vector $\mathbf{x}$. From here on, the procedure is the same as in standard linear classification: a vector of scores is constructed for each sample and each of its elements is associated with a class. The loss function $J$ is useful to figure out the best possible values for $W$ and $b$ which would provide the right class attribution for each input $\mathbf{x}$. This search problem is again converted into a minimization of some particular function. A typical example of loss used in classification problems is the Multiclass SVM (Support Vector Machine) [18]:

$$J_i = \sum_{j \neq y_i} \begin{cases} 0 \quad if \quad s_{y_i} >= s_j + 1 \\ s_j - s_{y_i} + 1 \quad otherwise \end{cases} \tag{2.7}$$

Where $\mathbf{s} = \mathbf{g}(\mathbf{W}, \mathbf{x})$ is the vector of predicted scores, $J_i$ is the loss, and the subscripts i refer to its elements. Whereas $s_j$ are the scores that refer to the wrong classes (relative to the sample), and $s_{yi}$ are those associated with the right class. The safety margin chosen is one, which is the difference between correct and incorrect scores. If the score referred to a class increases, its loss value decreases, reaching zero once the safety margin reaches or exceeds the value of one. This loss function is also called hinge loss because of the characteristic shape of its graph (see Fig. 2.10).

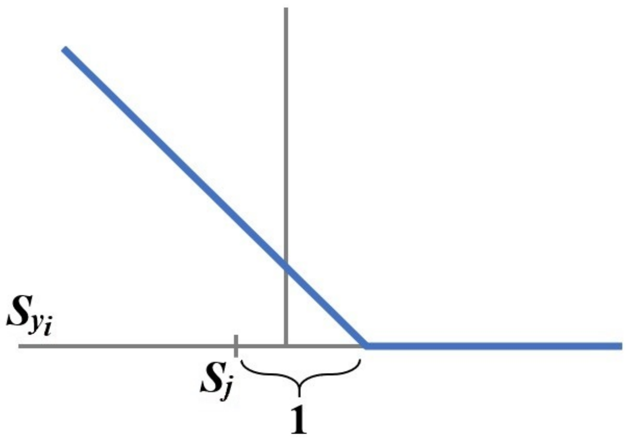


**Figure 2.10** Hinge Loss representation.

Finally, the average of $J_i$'s values constitutes the total $J$:

$$J = \frac{1}{N}\sum_i J_i(\mathbf{g}(\mathbf{x_i}, \mathbf{W}), \mathbf{y_i}) \tag{2.8}$$

The process starts with proper initialization of $W$ and $b$. They are updated at each iteration in a manner similar to linear regression, usually following a Gradient descent algorithm, or more sophisticated ones, if required.

**Neural Networks**

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input, so the network generates the best possible result without needing to redesign the output criteria [35].
Simplistically, a neuron [18]:

- takes input data;
- its dendrites perform complex computations;
- its synapses are a complex non linear dynamical system;
- its output is then transmitted to the following nodes.

A mathematical neuron instead is characterised by:

- an input $x$, coming from out of the network or from previous neurons, by means of connections;
- a weight matrix $W$ (it can be also a vector or a scalar, depending on input and output);
- a bias term $b$ (it can be a vector or a scalar, depending on the output);
- an activation function $f(\mathbf{W_j}\,\mathbf{x_j} + \mathbf{b_j})$, where $W_j$ is the $j-th$ row of a weighting matrix, $x_j$ and $b_j$ are the $j-th$ element of their respective vectors. This function "modulates" element-wise ($j$) every quantity exiting the neuron;
- connections with other neurons.

Once the input enters the neuron, it is linearly combined by the weight and bias terms. This quantity is then given to the activation function as input. Activation functions are used to determine the output of a neural network in terms of **yes/no** opposition [36]. They map the resulting values from a neuron between 0 and 1 or between $-1$ and 1 etc. (depending on the function). The information that travels from input to output changes accordingly. These functions are usually divided into two types:

- linear activation functions;
- non-linear activation functions.

The first category basically includes the identical activation function, which varies between -$\infty$ and $+\infty$. In this case, the output is not limited in any way [6]. On the other hand, the second category consists of several functions as shown in Fig. 2.11.

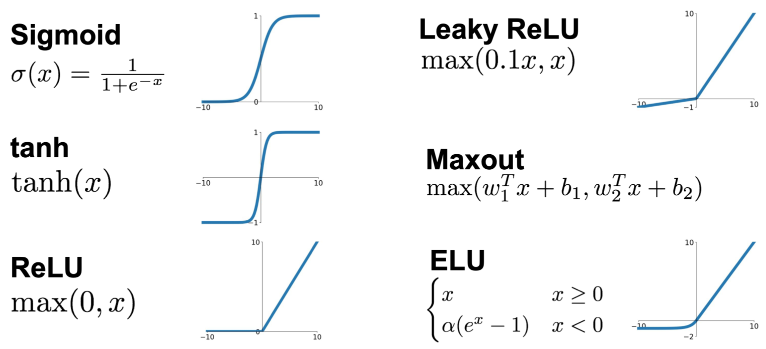


**Figure 2.11** The most used activation functions [18].

Neural networks are composed by groups of neurons interconnected with preceding and/or following neurons (depends on the location and type of the layer). There are generally three types of layers (see Fig. 2.12):

- Input layer: the elements of each sample are placed to act as input for the following layers.
- Hidden layer(s): it is made of a fixed number of neurons, taking as input processed information from neurons belonging to the previous layer. The output of each neuron, downstream of its activation function, becomes the input of the neurons belonging to the following layer.
- Output layer: the final layer, in which the data processed in the inner structure of the net are obtained.

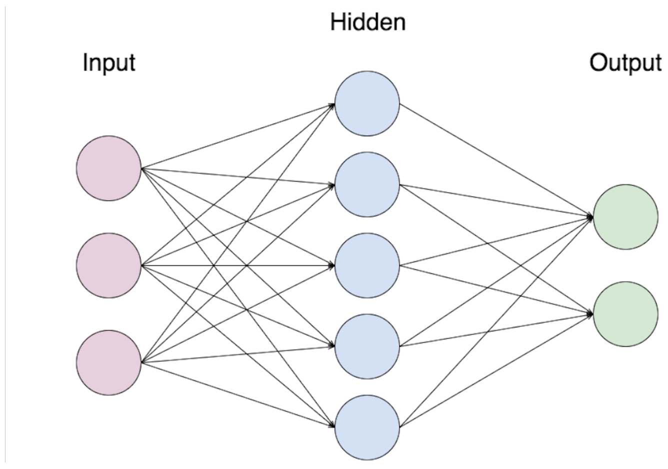


**Figure 2.12** An example of the typical structure of a Fully Connected Neural Network with a single hidden layer [37].

The simplest model of a neural network is the one illustrated in Fig. 2.12: a single-layer Fully Connected neural network, also known as Multi-Layer Perceptron. Its name highlights all its features: its architecture is characterized by connections that connect all the neurons of one layer to the previous and next. The adjective 'single layer' refers to the number of hidden layers (they do not include the first and last layers). By increasing the number of hidden layers the network becomes a Deep Neural Network (DNN), because the depth is increased. Generally, the more neurons are placed in the same hidden layer, the more degrees of freedom the loss function will have. In the same way, keeping a lower amount of neurons per hidden layer but increasing their total number gives the same result. Moreover, depth helps the network to generalize and to better cope with non-linearities of the regression function. Just like in simple linear regression and classification, each neuron is characterised by a weight term $\mathbf{W}$ and a bias term $\mathbf{b}$. The entire network aim is to find the best values of $\mathbf{W}$s and $\mathbf{b}$s to provide:

- a function that models the given input-output relationship the best, in the regression case;
- the right class attribution for each input, in the classification case.

Once again, the problem is turned into a minimization one by means of a loss function $J$: for every sample $i$, $J_i$ consists in a partial loss function, evaluated in the output of the neural network (prediction) [6].
The total loss instead is expressed as the average of each $J_i$ (see Eq.2.8). An additional feature of the loss function in this application is the regularization term. It is usually added to the formulation of the total loss in equation Eq.2.8 (modulated by a $\lambda$ parameter): its purpose is penalizing its complexity, building up sample after sample. There are a variety of regularization terms **[19]**:

- One of the most common is the $L_2$ regularization $R(\mathbf{W}) = \sum_k \sum_l \mathbf{W}_{k,l}^2$ (in this case it spreads the $\mathbf{W_i}$ across all the vector).
- L1 regularization, $R(\mathbf{W}) = \sum_k \sum_l |\mathbf{W}_{k,l}|$. It has a different conception of complexity (in this case the number of zeros in the weight matrix/vector).

To tune and refine the input-output relationship that is built into a neural network, it is necessary to use many samples and thus create a very large dataset: adding a new sample to those used to perform regression or classification adds a new loss term to the aforementioned $J_i$s average. In the case of regression, a sample is a pair of terms representing the values of the expected input and output of the regression function. In the case of classification, on the other hand, it consists of an input object (an image for example) and the class to which it belongs.
Due to the multidimensional nature of the total loss function of the neural network, a complex process is required to minimize it. This minimization process, exploiting the number of samples, is called neural network training. A general dataset employed to build a working neural network is divided into three different subsets: the training subset, the validation subset, and the testing subset. The first consists of most of the available dataset (usually 70%), the second comprises about 20% and the third includes the remainder. Training is a fundamental process of the algorithm, necessary to generate a model that can predict new cases with a certain level of accuracy. Validation is important because the model compares its learning with data that have solutions to increase the confidence of the detections and thus improve from epoch to epoch. The test consists in the evaluation of the accuracy: the obtained model is used to predict new cases (different from the training ones), whose solution is known. If the results of the test phase do not attest to a sufficient level of goodness of the model, the training phase is repeated. To obtain a well-trained model, there are two kinds of occurrences to avoid during training [38]:

- Overfitting: the function fits the data pattern too much. The absolute distances between the single data points and the fit are low but the fit function tries to fit each data point by one, while not recognizing the true function that describes the data distribution.
- Underfitting: it refers to a model that neither can fit the training data nor new data from the problem domain.

These two conditions are mainly due to the loss function complexity: if it is too high, the model fits every single training point, but loses its generalization ability, if it is too low, it is not able to fit training points due to a lack of degrees of freedom [6]. Regularization plays a key role in determining loss function complexity. Another parameter is the proportion followed to divide the total dataset in training and testing: too many samples in the training subset increase the probability of overfitting, on the contrary, too few can turn out to be an underfitted model. The starting point for the above mentioned minimization process is the initialization of the variables of interest $\mathbf{W}$ and $\mathbf{b}$. The initial value assigned to each weight can be chosen with different procedures [18]:

- All weights can be set to 0 and all the neurons will have the same behaviour. This, however, could turn out to be problematic.
- They can be set to small random numbers, produced following a Gaussian distribution with null mean and 0.01 standard deviation. This approach could generate issues with deep networks: if standard deviation collapses to 0, weights are updated with very small gradients, remaining practically unchanged.
- They can be fixed to small random numbers, inside a Gaussian bell with null mean and 1 standard deviation. The only flaw is the fact that by multiplying by high weights over and over, neurons could saturate and gradients could flatten to zero.
- Xavier initialization, scaling on the square root of the number of the inputs. It could have issues when working in a linear environment, but still, it is the most used initialization technique.

As regards input data $\mathbf{x}$ instead, they can be pre-processed to make the learning phase easier for the network. Batch normalization for example is a way to boost the speed, performance, and stability of neural networks. It is utilized to normalize the input layer by adjusting and scaling the entries. Consider a batch of them at some layer. To make each dimension unit Gaussian, the following formulation is applied [6]:

$$\hat{\mathbf{x}}^k = \frac{\mathbf{x}^k - E(\mathbf{x}^k)}{\sqrt{Var(\mathbf{x}^k)}} \tag{2.9}$$

Where $\hat{\mathbf{x}}^k$ is the normalized $k-th$ batch, while $\mathbf{x}^k$ is the non-normalized one. $E(\mathbf{x}^k)$ and $Var(\mathbf{x}^k)$ are respectively the expected value and the variance of the selected dataset batch [6]. The way forward consists of computing the empirical mean and variance independently for each dimension and then normalize. The positive aspects about batch normalization are:

- Improves gradient flow through the network.
- Allows higher learning rates.
- Reduces the strong dependence on initialization.
- Acts as a form of regularization, and slightly reduces the need for dropout (turning off some neurons to prevent overfitting).

Below are shown the various steps to create a normalization:

**Mini-batch mean**

$$\mu_b = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.10}$$

**Mini-batch variance**

$$\sigma_b = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_b)^2 \tag{2.11}$$

**Normalize**

$$\hat{x}_i = \frac{x_i - E(\mathbf{x})}{\sqrt{Var(\mathbf{x}^k)}} \tag{2.12}$$

**Scale and shift**

$$y_i = \gamma x_i + \beta = \mathbf{BN}(x_i)_{\gamma,\beta} \tag{2.13}$$

Inserting $\gamma$ and $\beta$ to re-scale input variables gives flexibility to the network training phase. Moreover, when sometimes taking the same normalization for each batch becomes ineffective, the original mapping is recovered and another normalization is adopted (or scaled using other learned parameters). Mean and standard deviation used at testing time are the same employed in the inputs scaling at training time, in order to feed the network with testing entries that are pre-processed in the same way as the original training ones [6]. Once both inputs and variables are configured, the minimization process can start. The Gradient Descent algorithm is the most used method to drive it (see Eq. 2.5). In addition to the standard one, some variants are more suitable for this case, in order to avoid convergence to local minima as much as possible [18]:

- Batch Gradient Descent: the total training set is used to update weight values.
- Minibatch Gradient Descent: the training set is divided into subsets called batches. These are iteratively used to tune the weights at each step.
- Stochastic Gradient Descent (SGD): it updates the parameters using only a single training batch at each iteration. The training batch is usually selected randomly. It is often preferable to optimize cost functions when there are hundreds of thousands of training instances or more, as it converges more quickly than batch gradient descent.

SGD is the most popular among these categories of Gradient Descent, although it has various drawbacks:

- Very slow progress along its shallow dimension, jittering along the steep one.
- Local minima can get SGD stuck due to gradients becoming 0. With saddle points the same occurs.

In order to cope with these flaws of the basic SGD, a momentum term $\rho$ can be added to:

- Build up velocity as a running mean of gradient
- Give friction

The formulations are the following:

**SGD**

$$x_{t+1} = x_t - \alpha \nabla f(x_t) \tag{2.14}$$

**SGD + momentum**

$$\begin{cases} v_{t+1} = \rho v_t + \nabla f(x_t) \\ x_{t+1} = x_t - \alpha v_{t+1} \end{cases} \tag{2.15}$$

Where $\alpha$ is the step-size (learning rate) of the algorithm, while $v$ is the velocity term composed of the gradient and $\rho$ is an additive momentum term. It usually ranges between 0.9 and 0.99, while initial velocity is set to 0.
Adagrad is the evolution of SGD + momentum method: it consists in keeping a running estimate or a running sum of all the squared gradients seen during previous training, throughout the course of the optimization process. Instead of having a velocity term, there is a squared gradient term. Weights are updated as follows:

$$\begin{cases} \nabla_{sq_{new}} = \nabla_{sq_{old}} + \nabla_W L(\mathbf{W_{old}})^T \cdot \nabla_W L(\mathbf{W_{old}}) \\ \\ \mathbf{W_{new}} = \mathbf{W_{old}} + \alpha \dfrac{\nabla_W L(\mathbf{W_{old}})}{\sqrt{\nabla_{sq_{new}}} + 10^{-7}} \end{cases} \tag{2.16}$$

Where $\nabla_{sq}$ is the squared gradient term, $\nabla_W L$ is the gradient, $\mathbf{W}$ is the weight term and $\alpha$ is the learning rate.
Using the following scaling in Adagrad, the step becomes smaller and smaller over time and slows down in the wiggling dimension. In the convex case, this is a good feature for the system: it is better to reduce the step size when it is close to the minimum. The problem occurs in the case of non-convex areas, such as saddle points, where the method may be get trapped without making any progress.
A further step in eliminating minimization drawbacks is the RMSprop model, based on Adagrad. With the introduction of a decay rate, it is able to avoid the difficulties at saddle points, unlike Adagrad [6]. The state of the art in loss function minimization is the Adam method, which realizes the advantages of both AdaGrad and RMSprop. It uses two variables: the learning rates of the parameters based on the average of the first moment (the mean) and the average of the second moments of the gradients (the uncentered variance) [39].
The optimization procedures described involve the learning rate hyperparameter. In the simplest configurations, it is set to a fixed value. On the contrary, when minimization complexity increases, it gets progressively updated during the training: it is usually set as a large value at the beginning and then decreased progressively (step decaying or decaying driven by a law) [6].
Differently from linear regression and classification, a neural network is composed of many neurons, all of them contributing to the network output (and consequently to the loss function value). The gradient employed in the minimization process is thus built taking into account all of the intermediate neurons lying between input and output, so that a direct gradient between loss and weights (and biases) can be obtained. The procedure followed to achieve this result is called Back-propagation. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch. Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization [40]. The starting point for calculating gradients is from the end of the network, moving backwards to the beginning of it. Following the example in Fig. 2.13, i.e., from the point of view of a node, the gradients are computed and multiplied to go back to the beginning. The red lines show the back-propagation workflow: entering the $z_2$ intermediate variable block, partial derivatives $\frac{\partial L}{\partial z_2}$ , $\frac{\partial z_2}{\partial b_2}$, $\frac{\partial z_2}{\partial h}$ and $\frac{\partial z_2}{\partial w_2}$ are computed. The first one is then multiplied by the remaining three to obtain the direct gradients between $L$ and $b_2$, $h$ and $w_2$ [6].

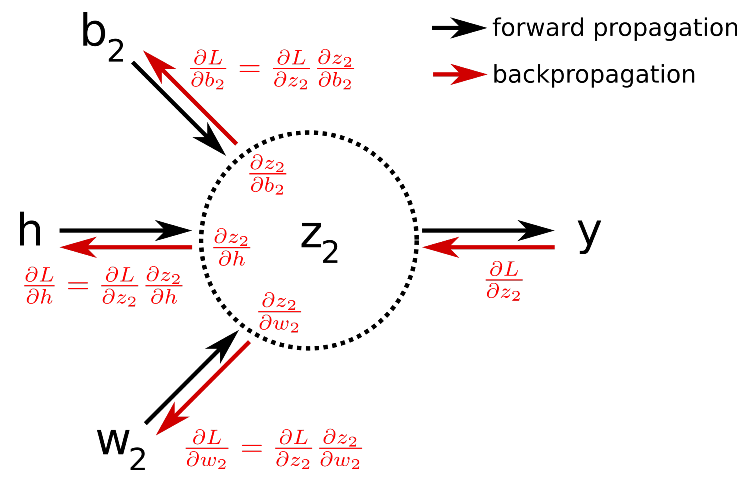

b2
$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial b_2}$
forward propagation
backpropagation
$\frac{\partial z_2}{\partial b_2}$
h
$\frac{\partial z_2}{\partial h}$
z2
y
$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial h}$
$\frac{\partial L}{\partial z_2}$
$\frac{\partial z_2}{\partial w_2}$
w2
$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial w_2}$

**Figure 2.13** Gradient computational graph: given the loss function $L$ and three generic variables $b_2$, $h$ and $w_2$ (in the case of a linear regressor they could be weight and input variable), the gradients are computed and multiplied to trace back to the beginning. The red lines show the back-propagation workflow: entering the $z_2$ intermediate variable block, partial derivatives $\frac{\partial L}{\partial z_2}$ , $\frac{\partial z_2}{\partial b_2}$, $\frac{\partial z_2}{\partial h}$ and $\frac{\partial z_2}{\partial w_2}$ are computed. The first one is then multiplied by the remaining three to obtain the direct gradients between $L$ and $b_2$, $h$ and $w_2$ [18].

The granulation of the computational graph is arbitrary, according to the wanted explicit calculation. The underlying idea is to get the gradient in a particular position of the graph.
Dealing with multidimensional problems, instead, attention must be paid to the tensors' relative dimensions during the backpropagation in the computational graph. All these passages in a computer environment are done by the processor unit.

## CNN

A Convolutional Neural Network (CNN or ConvNet) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms [41]. As seen before, a color image is processed by the system as a 3D matrix, where 2 dimensions correspond to the number of pixels given by the resolution, while the third is its color depth. A Fully Connected neural network decomposes the input matrix into a vector, so the information on the structure is lost, and therefore if the input image is too defined, the algorithm becomes too computationally heavy.
The ConvNet task is to reduce the images into a more streamlined form to be processed and analysed, without losing critical and useful data to perform a proper prediction. This is important not only for efficient and faster learning but also to be able to use massive datasets that would require a high computational time.
The architecture of the network is organized into several layers combined to achieve the best possible performance (see Fig. 2.14). The most important ones are:

- Convolutional
- Max pooling
- Fully Connected

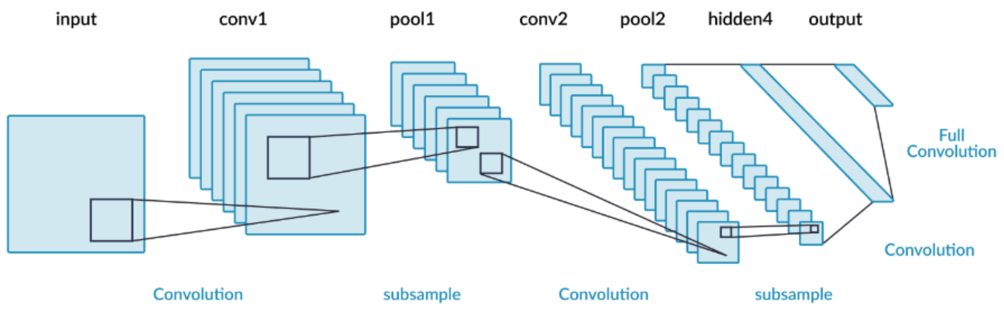


**Figure 2.14** A simple example of a CNN architecture [42].

### Convolutional layer

Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the simple application of a filter to input images that results in an activation. Repeated application of the same filter plus a bias term to an input results in a map of activations (see Fig. 2.15) [43].

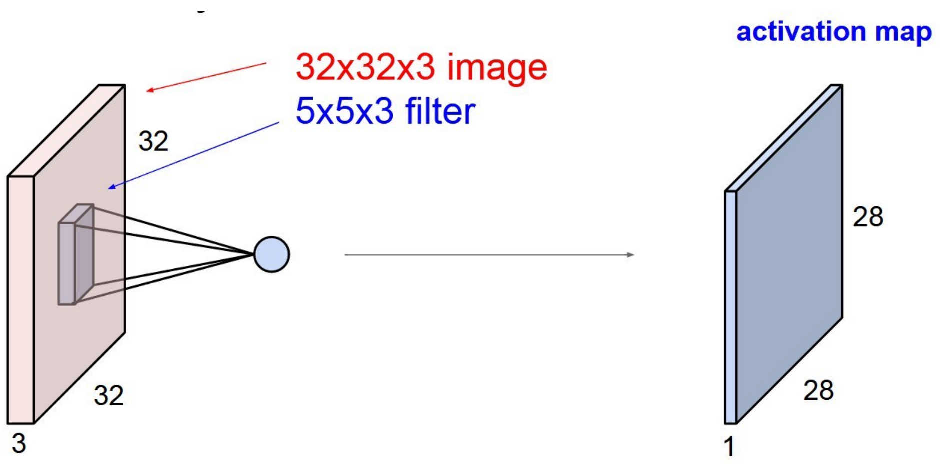


**Figure 2.15** The figure shows the product of the convolution, called activation map [18].

Different filters can be applied to an image to extract multiple features, forming different activation maps. They are the output of the convolutional layers and represent the input for the successive layer.
To represent the operation of the filter consider as input a matrix (7, 7) and a filter (3, 3) (see Fig. 2.16): starting from the upper left corner a term-by-term multiplication is done. Then, it is shifted horizontally by one value (in the case of an image it corresponds to one pixel) and the procedure is repeated. The output obtained is an activation map with reduced dimensions with respect to the starting input, in this case, the map has dimensions (5, 5).
Pixel stride can vary depending on the problem. The best results are obtained when the whole image is mapped.

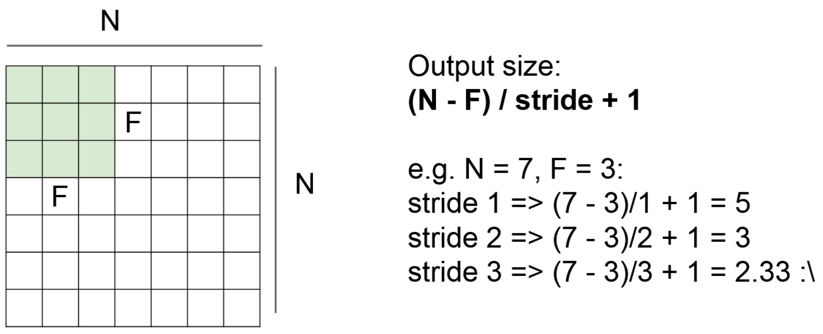

N
F
F
N
Output size:
(N - F) / stride + 1
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

**Figure 2.16** Sliding rule of a convolutional layer filter [18].

A way to understand whether the stride is appropriate is proposed by the following formula:

$$output_{size} = \frac{(N - F)}{stride} + 1 \tag{2.17}$$

where $N$ is the image size and $F$ is the filter size. The output can be an integer number or a fraction. In the second case, there are two solutions: change the value of the stride or by zero padding to the border, namely hemming the matrix with zeros. In general, is customary to use convolutional layers with stride 1, filters of size $(F, F)$, and zero padding with $\frac{F-1}{2}$.
The size of the activation maps is smaller than the input. The practice is to gradually reduce the size. Along the sequence of convolutional layers, the peculiarities that the network is able to recognize become increasingly sophisticated and complex.

### Max pooling layer

The purpose of these types of layers is to reduce the size of the data to optimize processing time by replacing the output with a maximum synthesis [44]. This allows us to determine the features that produce the greatest impact and reduce the risk of overfitting. Max pooling takes two hyperparameters: stride and size. The stride determines how far the value pools jump, while the size determines how large the value pools are in each jump. The procedure is shown in Fig. 2.17.

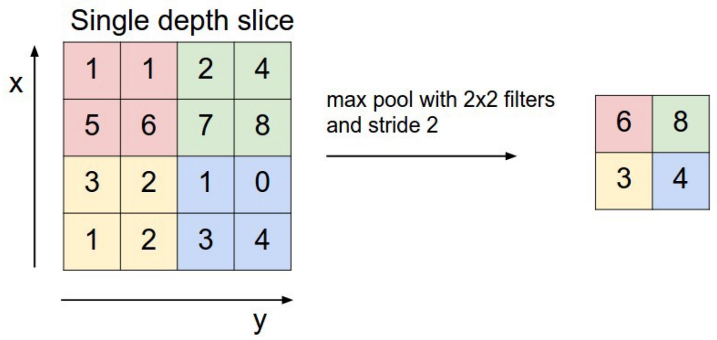


**Figure 2.17** The image shows how the Max pooling technique works [44].

### Fully Connected layer

This layer coincides with a single-layer Fully Connected neural network. Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space. The capability of this layer is to learn about possible nonlinear functions in the space in which it is placed. This network receives as input the rescaled image rendered as a column vector. The output is a vector of scores whose number of elements corresponds to the number of the classes of the model.
Every time that a training epoch (complete presentations of the dataset to be learned by a machine) is concluded the algorithm of backpropagation is applied. [45]. It is the practice of fine-tuning the weights of a neural net based on the error rate obtained in the previous iteration. Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization. Over a series of epochs, the model can distinguish between dominating and certain low-level features in images and classify them.

## Loss function

The training phase is also based on loss functions. It is a way to evaluate the learning ability of the network and if the prediction deviates too much from the actual results, the loss function returns large values. With the assistance of some optimization function, the loss function learns to reduce the prediction error iteration after iteration. [46].
No loss function satisfies every type of machine learning. Various factors influence the choice of a loss function for a specific problem such as the type of machine learning algorithm chosen, the ease of computing derivatives, and the type of data set. The most commonly used one for image segmentation is a pixel-wise cross entropy loss. This examines each pixel individually, comparing the class predictions (depth-wise pixel vector) to one-hot encoded target vector (see Fig. 2.18). Because the cross entropy loss evaluates the class predictions for each pixel vector individually and then averages over all pixels, it is essentially asserting equal learning to each pixel in the image [6].

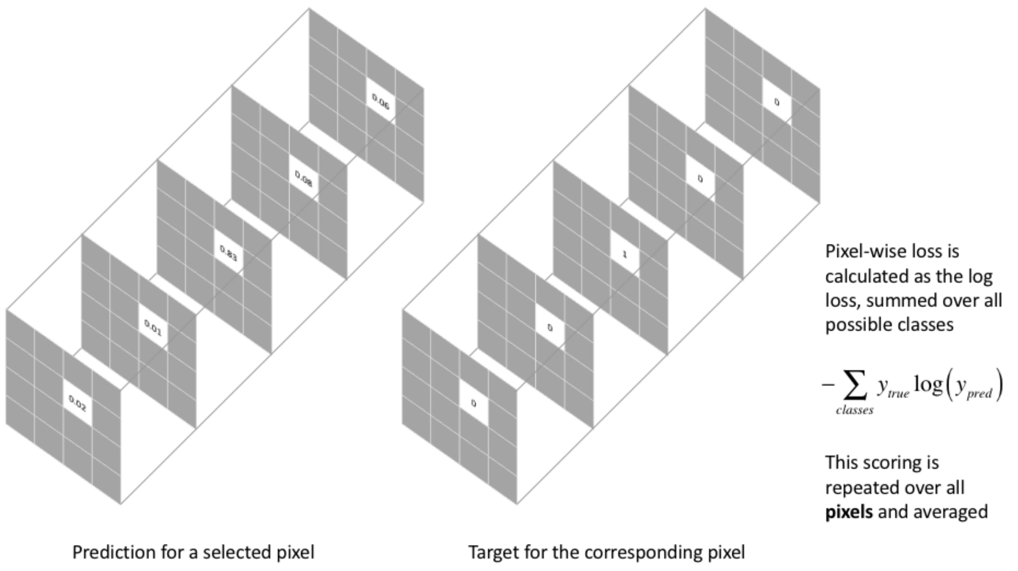

0.06
0.08
0.83
0.01
0.02
0
0
1
0
0
Pixel-wise loss is calculated as the log loss, summed over all possible classes
$-\sum_{classes} y_{true} \log\left(y_{pred}\right)$
This scoring is repeated over all **pixels** and averaged
Prediction for a selected pixel
Target for the corresponding pixel

**Figure 2.18** Logarithmic binary cross entropy [47].

## 2.3 YOLO

You Only Look Once (YOLO) is a real-time object detection framework, in which the image is only passed once through a Convolutional Neural Network (CNN). By means of a publication in 2015, Joseph Redmon announced the creation of YOLO. It is a family of compound-scaled object detection models trained on the COCO dataset, and includes simple functionality for Test Time Augmentation (TTA), model ensembling, hyperparameter evolution, and export to ONNX, CoreML and TFLite [19]. The models are famous for being highly performant yet incredibly small, making them ideal candidates for real time conditions and on-device deployment environments. The architecture of this framework (from version 1 to version 4) is based on Darknet, a flexible research framework used by few users, difficult to configure, and not very useful for production. In 2020 Glen Jocher created its most recent version with several differences and improvements, including the implementation of the famous mosaic augmentation technique. This updated version is based on PyTorch, which is an open source machine learning framework easy to configure and popular among the community. Compared to other object detection systems:

- it is extremely fast;
- it sees the whole image during training and testing, then implicitly encodes contextual information about classes and their appearance;
- it learns generalizable representations of objects so that when trained on natural images and tested on works of art, the algorithm outperforms other high-level detection methods.

YOLO is popular because it achieves high accuracy while still being able to work in real-time. The algorithm "looks only once" at the image, meaning that it requires only one pass of forwarding propagation through the neural network to make predictions.

### Architecture

An object detector is designed to create features from input images and then to feed these features through a prediction system to draw boxes around objects and predict their classes (see Fig. 2.19).

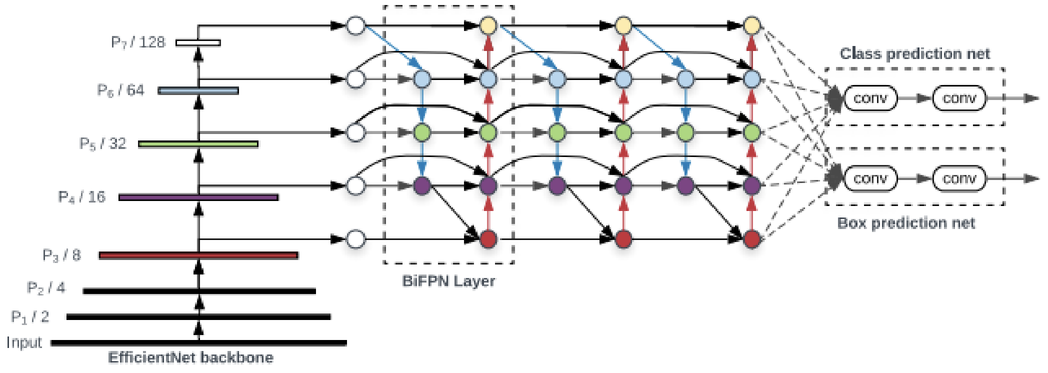


**Figure 2.19** Architecture of the layers of YOLO [19].

The YOLO models were the first object detectors to connect the procedure of predicting bounding boxes with class labels in an end to end differentiable network, as shown in figure [48]. The network is structured in three main components (see Fig. 2.20):

1. Backbone: is a convolutional neural network that aggregates and forms image features at different granularities.
2. Neck: is a series of layers to mix and combine image features to pass them forward to prediction.
3. Head: is composed of consumers' features from the neck, and it takes the box and class prediction steps.

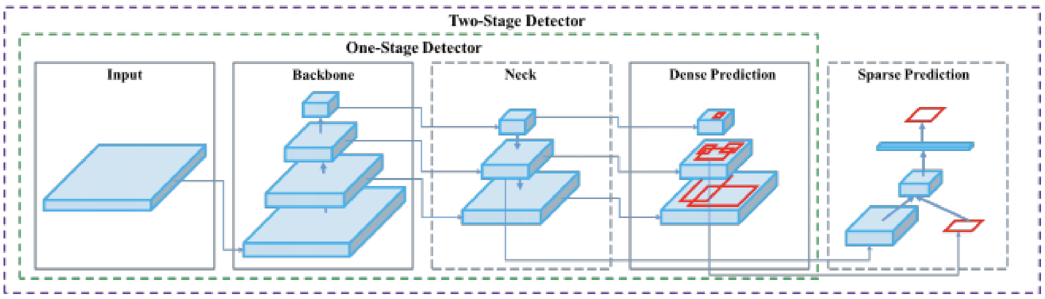


**Figure 2.20** Main components architecture of YOLO [19].

## Training procedure

The training dataset is composed of images associated with bounding boxes that highlight the desired solution. YOLO analyses the entire images and learns the characteristics of the selected classes and the background surrounding them. This makes the analysis very general and flexible. Moreover, it allows detecting multiple objects in a single image, returning as output the class, the characteristics of the bounding box, and the detection probability. The learning process is composed of two phases, which are repeated at each epoch:

- training part: commonly composed of about 70% of the complete dataset. This phase aims to find patterns and generalizations among the various classes. Hence, the model learns to distinguish features and make predictions based on the provided examples;
- validation part: commonly composed of about 20% of the complete dataset, it is the phase in which an accuracy analysis of the model happens and subsequently it provides weights modifications for effective learning.

Once the training is complete, the quality of the model can be checked through the testing phase. It uses the remaining 10% of the dataset to calculate Average Precision (AP), mean Average Precision (mAP), Precision (P), and Recall (R).
There are two fundamental procedures for successful learning:

1. Data Augmentation: makes transformation to the base training data to expose the model to a wider range of semantic variation than the training set in isolation.
2. Loss Calculations: YOLO calculates a total loss function from constituent loss functions, which are Generalized Intersection over Union (GIoU), objectness (obj), and class losses. These can be carefully constructed to maximize the objective of mean average precision [48].

### Data augmentation

With each training batch, YOLOv5 passes training data through a data loader, which augments data online. The data loader makes three kinds of augmentations: scaling, color space adjustments, and mosaic augmentation. The most novel of these is mosaic data augmentation combining four images into four tiles of random ratio. The Fig. 2.21 shows an example of an images mosaic:

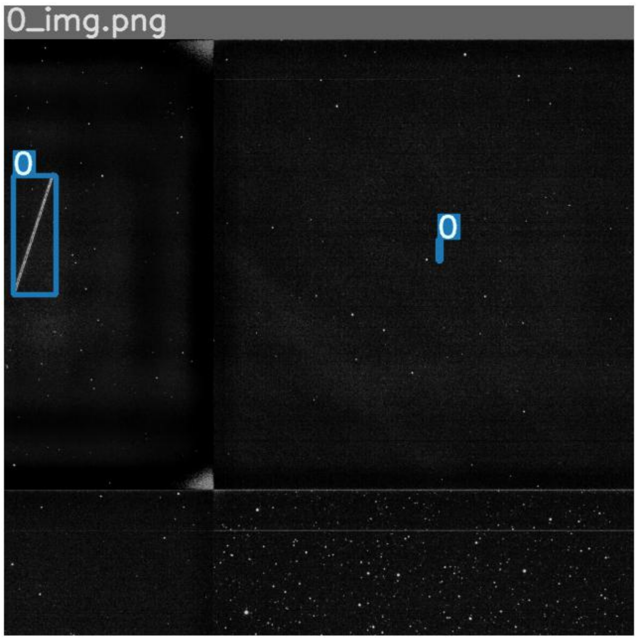


**Figure 2.21** Example of images mosaic for the YOLOv5 training phase.

Mosaic augmentation is especially useful to address the "small object problem", therefore where small objects are not as accurately detected as larger objects.

### Auto learning bounding box anchors

In order to make box predictions, the YOLO network predicts bounding boxes as deviations from a list of anchor box dimensions [48].
In the YOLOv3 PyTorch repo, Glenn Jocher introduced the idea of learning anchor boxes based on the distribution of bounding boxes in the custom dataset with K-means and genetic learning algorithms. This is very important for custom tasks, because the distribution of bounding box sizes and locations may be dramatically different from the preset bounding box anchors in the COCO dataset (which is the default).

The most extreme difference in anchor boxes can occur when the objects to be detected are very vertical and narrow or horizontal and fine, such as vertical or horizontal tracklets.

## CSP backbone

YOLOv5 implements the CSP Bottleneck (Cross Stage Partial) to formulate image features. The CSP addresses duplicate gradient problems in other larger ConvNet backbones resulting in fewer parameters and fewer FLOPS for comparable importance. This is extremely useful to the YOLO family, where inference speed and small model size are of utmost importance [48]. Fig. 2.22 shows the architecture of DenseNet, which is the network on which the CSP models are based. They were designed to connect layers in convolutional neural networks with the following motivations: to alleviate the vanishing gradient problem (it is hard to backprop loss signals through a very deep network), to bolster peculiarity propagation, encourage the network to reuse features, and reduce the number of network parameters.

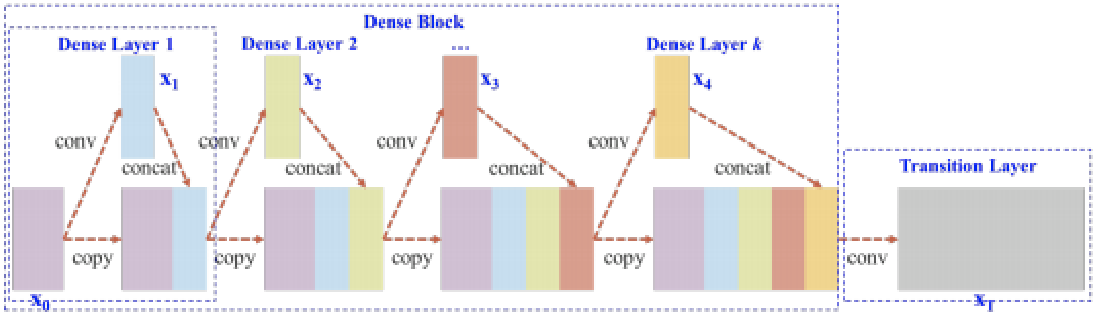


**Figure 2.22** CSP DenseNet layers [48].

Fig. 2.23 shows how the DenseNet used in YOLOv5 has been edited to separate the feature map of the base layer by copying it and sending one copy through the dense block and sending another straight on to the next stage. The idea is to remove computational bottlenecks in the DenseNet and improve learning by passing on an unedited version of the feature map.

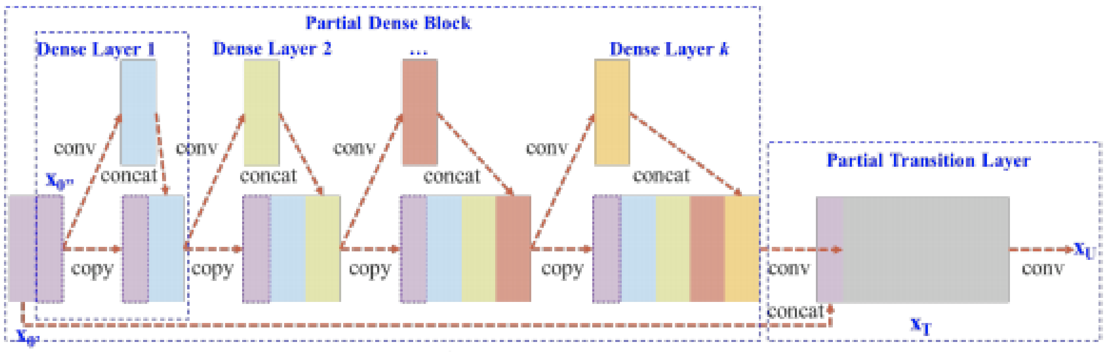


**Figure 2.23** CSP DenseNet employed in YOLOv5 [48].

## Performance analysis

Many object detection algorithms, such as Faster R-CNN, MobileNet SSD, and YOLO, use mAP to evaluate their models and publishing their research. Also in this thesis, this parameter is used as reference to measure the system performance and to compare different systems. For object detection tasks, Precision and Recall are calculated using the Intersection Over Union (IoU) value for a given IoU threshold. This parameter is calculated for each detected object and is defined as the overlap between the predicted bounding box and the ground truth bounding box (see Fig. 2.24) [49].

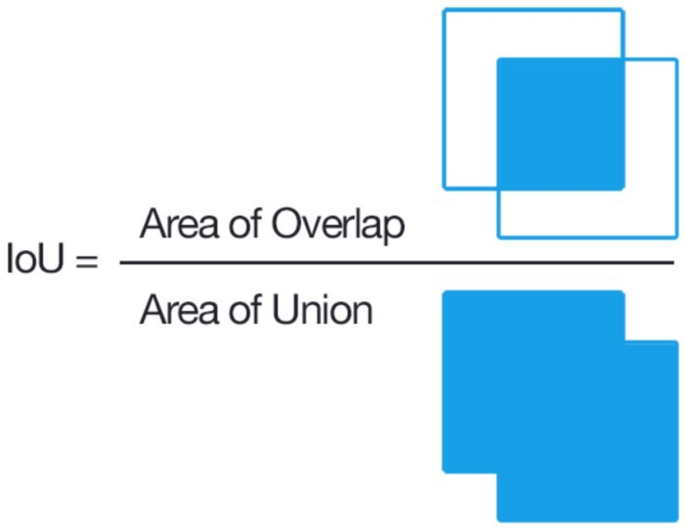


**Figure 2.24** Intersection over Union representation [50].

The typical value used for IoU threshold is 0.5, so:

- If the IoU value for a prediction is $> 0.5$, then the prediction is classified as True Positive (TP).
- If IoU is $< 0.5$, it is classified as False Positive (FP).
- If the element is not detected or if the IoU is $> 0.5$ but the classification is wrong, it is classified as False Negative (FN).

Precision and Recall are the most concise and useful metrics for understanding the behaviour of a network. They are defined as [51]:

- Precision: the ability of a model to identify only the relevant objects;
- Recall: the ability of a model to find all the relevant cases.

With the TP, FP and FN formally defined, the precision and recall of detection for a given class across the test set are calculated as:

$$\begin{gathered} Precision = \frac{TP}{TP + FP} \\ Recall = \frac{TP}{TP + FN} \end{gathered} \tag{2.18}$$

The AP is then calculated by taking the area under the Precision-Recall curve. The mAP is the average of the AP calculated for all the classes.

## 2.4 Instruments

Is worth noting that all the codes described from now on are written in Python programming language with notebook as editor (for further information see [52]).
Below there are brief descriptions of the tools used for the programming phase, and then a description of the TIG software used to generate synthetic images is provided.

### 2.4.1 Jupyter Notebook

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results [53]. The Jupyter notebook combines two components:

- A web application: a browser-based tool for interactive authoring of documents that combine explanatory text, mathematics, computations, and their rich media output.
- Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.

Unlike classic editors, notebooks present the following advantages [53]:

- In-browser editing for code, with automatic syntax highlighting, indentation, and tab completion/introspection.
- The ability to execute code from the browser, with the results of computations attached to the code which generated them.
- Displaying the result of computation using rich media representations, such as HTML, LaTeX, PNG, SVG, etc. For example, publication-quality figures rendered by the matplotlib library, can be included inline.
- In-browser editing for rich text using the Markdown markup language, which can provide commentary for the code, is not limited to plain text.
- The ability to easily include mathematical notation within markdown cells using LaTeX [54], and rendered natively by MathJax.

The normal workflow in a notebook is, then, quite similar to a standard IPython session, with the difference that you can edit cells in-place multiple times until you obtain the desired results, rather than having to rerun separate scripts with the "run" command [53]. Typically, a computational problem is divided into pieces, organizing related ideas into cells and moving forward once previous parts work correctly. This is much more convenient for interactive exploration than breaking up a computation into scripts that must be executed together, as was previously necessary, especially if parts of them take a long time to run.

### 2.4.2 Google Colaboratory

Any developer who has dealt with artificial intelligence and machine learning applications is aware of how much computing power can be required to implement sufficiently robust and efficient models. While relying on the local computer may be sufficient for initial testing, as the size of datasets increases, running complex training algorithms such as deep learning ones quickly becomes prohibitively expensive.
The problem can be solved by relying on cloud services that offer computing power, often for a fee and with various limitations. As an alternative to them, there is Google Colaboratory, an interesting platform that, albeit with some limitations, allows to run code directly on the cloud, taking advantage of the computing power provided by Google.
Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted

Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs (see Fig. 2.25) [55].

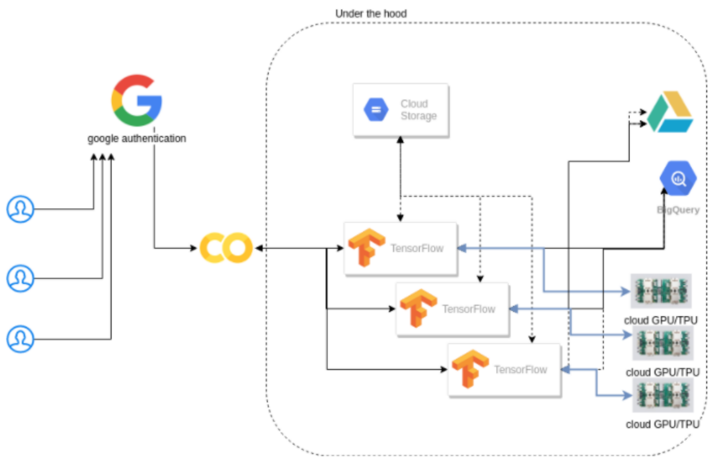


**Figure 2.25** Google Colaboratory architecture [55].

### 2.4.3 Tracklet Image Generator

Before diving into the description of the TIG algorithm architecture it is better to give concise descriptions of TLE files and SCOOP software.

#### TLE

Two-Line Orbital Element Sets are provided in an ASCII formatted text file and each file could contain TLEs for multiple objects [56]. A typical TLE entry can contain the name of the object on the first line (line 0), however this is not a requirement. The next two lines (lines 1 and 2) contain 69 characters which describe the orbit of the object in the NORAD Mean Element set (see Fig. 2.26) [6].

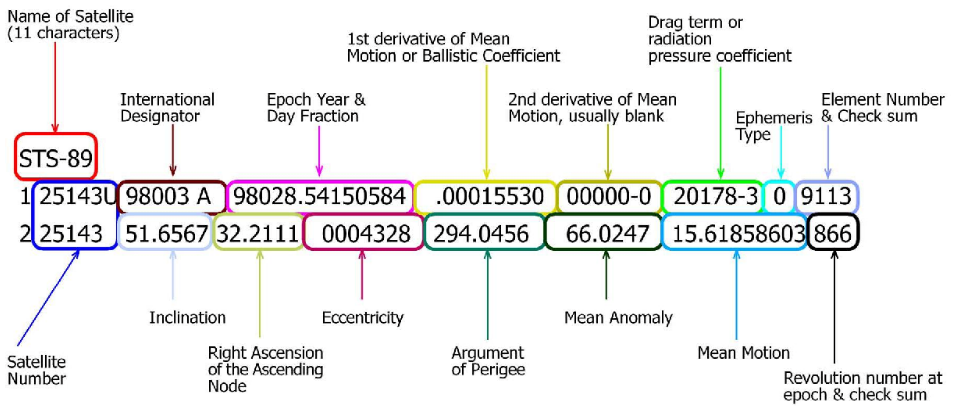

Name of Satellite (11 characters)
International Designator
Epoch Year & Day Fraction
1st derivative of Mean Motion or Ballistic Coefficient
2nd derivative of Mean Motion, usually blank
Drag term or radiation pressure coefficient
Ephemeris Type
Element Number & Check sum
STS-89
1 25143U 98003 A 98028.54150584 .00015530 00000-0 20178-3 0 9113
2 25143 51.6567 32.2111 0004328 294.0456 66.0247 15.61858603 866
Satellite Number
Inclination
Right Ascension of the Ascending Node
Eccentricity
Argument of Perigee
Mean Anomaly
Mean Motion
Revolution number at epoch & check sum

**Figure 2.26** Two Line Elements example [57].

#### SCOOP

TLEs alone are meaningless unless an intermediary program translates the input in azimuth and elevation angular coordinates linked to a desired location. SCOOP (namely SpaceCraft and Objects Observation Planning), an application internally developed in the Department of Aerospace Science and Technology, comes handy in this tedious process. In particular, it computes satellite passages, given their Two-Line Elements, visible from a set of ground stations [58].

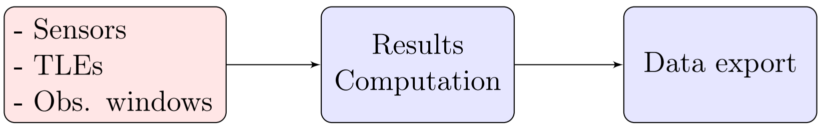
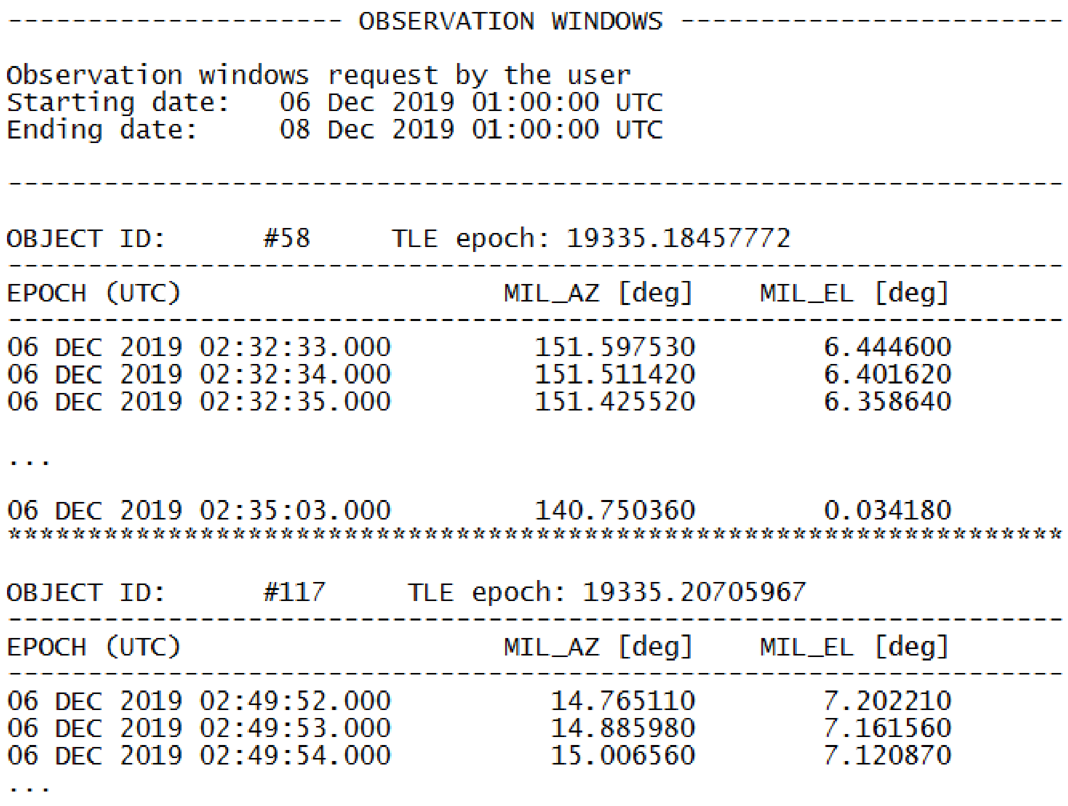


**Figure 2.27** SCOOP working principle [58].

The software requires to define the location of the sensor, which TLEs to use, and the observation windows. These data remain saved for later processing. The program's outputs are multiple:

1. The satellite state [Az, El], [R.A, Dec], [Lat, Lon], distance and radial velocity.
2. Illumination condition, phase angle, estimated magnitude.
3. Doppler shift, slant range.
4. Angular distance from the Sun and the Moon and the elevation of the Sun.

There is the possibility to select through a drop-down menu which output to store and then save the file. It is a text file that can be easily interpreted and read by today's programming (see Fig. 2.28).



```
--------------------- OBSERVATION WINDOWS ------------------------

Observation windows request by the user
Starting date:    06 Dec 2019 01:00:00 UTC
Ending date:      08 Dec 2019 01:00:00 UTC

-------------------------------------------------------------------

OBJECT ID:      #58     TLE epoch: 19335.18457772
-------------------------------------------------------------------
EPOCH (UTC)                     MIL_AZ [deg]    MIL_EL [deg]
-------------------------------------------------------------------
06 DEC 2019 02:32:33.000          151.597530        6.444600
06 DEC 2019 02:32:34.000          151.511420        6.401620
06 DEC 2019 02:32:35.000          151.425520        6.358640

...

06 DEC 2019 02:35:03.000          140.750360        0.034180
*******************************************************************

OBJECT ID:      #117     TLE epoch: 19335.20705967
-------------------------------------------------------------------
EPOCH (UTC)                     MIL_AZ [deg]    MIL_EL [deg]
-------------------------------------------------------------------
06 DEC 2019 02:49:52.000           14.765110        7.202210
06 DEC 2019 02:49:53.000           14.885980        7.161560
06 DEC 2019 02:49:54.000           15.006560        7.120870
...
```

**Figure 2.28** An example of a SCOOP text output file, displaying the relevant information of the selected passages.

In Fig. 2.28 it is worth noting that inessential fields are not displayed for the sake of clarity. Concerning the ground station, Milan is chosen for illustration purposes. The initial start observation time is set to 06 Dec 2019 01:00:00 UTC and the ending one, to 08 Dec 2019 01:00:00 UTC. The most important aspect is that time step between two following recordings of the same passage is one second. The thesis work, indeed, has the target of identifying trails in a short while, so the exposure time must be reduced to the minimum [6].

### Architecture

As already mentioned, the great power of machine learning systems lies in the fact that neural networks can autonomously learn the characteristics of the objects they need to detect and classify [49]. This process requires a large number of reference images to train the network. Moreover, the dataset must be as varied as possible to teach the network all possible scenarios that could occur in practice.
Finding such a large number of real images with these characteristics is a difficult task due to many reasons, including copyrights that the government and companies impose, and the cost to create a data collection system is very high. This problem is overcome by creating artificial images using special software, which can faithfully reproduce the original context.
In 2020, Cipollone De Vittori developed the Tracklet Image Generator (TIG) software that generates synthetic images. It is able to faithfully reproduce the necessary scenario, therefore images captured in staring mode with a background of stars and one or more transient objects. During the first phase of this thesis, the software generated thousands of images to build the first dataset.
The software is concerned with characterizing the elements within the image in such a way that they really seem captured from a real observation point. Its input is calculated in two steps [49]:

1. First step: it is necessary to select one or more TLEs and verify that the chosen observation window coincides with the operational life of the selected objects (otherwise SCOOP will not generate output). They can be found directly on SCOOP or in online databases.

2. Second step: Once the SCOOP inputs are selected, the program is launched. It will generate passes based on the settings made. The useful output to TIG is a text file in which the azimuth and elevation coordinates and their associated times are printed.

This information is combined with simulated images of the night sky and allows TIG to generate both the astronomical images containing the simulated streak and the relative mask. The files are saved in png and fits format and contain both the image produced and the astronomical information of the framed portion of the sky.
For some applications of this thesis, TIG has been modified. The modifications will be described in the interested chapters.

# Chapter 3

# Datasets Generation

Gathering data is one of the most important stages of machine learning workflows. The potential usefulness and accuracy of a trained network are closely related to the quality of the data. Therefore, dataset preparation is a fundamental step and a well trained network needs valuable data (this requires a lot of time and effort). This chapter outlines in detail everything concerning the datasets creation and the elaboration of the images with a briefly focus on *.fits* format.

## 3.1 Synthetic dataset

At the beginning of this thesis work, thousands of synthetic images were generated using TIG software due to the unavailability of real images. It allows randomness in length, thickness, position, and inclination of the tracklets and background noise.
With these it was possible to create a dataset comprising a wide variety of cases. The dataset consists of about 4000 images: 70% of them are used for the training phase, 20% for the validation phase, and the remaining 10% for the testing phase.

#### Preparing dataset

One of the most onerous operations when doing object detection is certainly the creation of bounding boxes, e.g. the identification of those areas of the image that contain the searched elements. This information is essential in the training phase to let the network learn the characteristics of the elements it will have to search for. Generally, this task cannot be automated, but it is necessary for an operator to manually label every single image and for large datasets, this requires considerable effort.
TIG outputs do not include tracklet labels, so the software was modified to provide text files with the labels automatically. This step was carefully designed to reduce the error concerning the bounding boxes position and dimensions estimate, thanks to the prior knowledge of the tracklets vertices location. These two information are stored as Cartesian coordinates line by line for each object in a text file as follows:

*[class ID] [object center in X] [object center in Y] [object width in X] [object width in Y]*

At the end of this routine, each image will be associated to a text file containing the coordinates of its bounding boxes, if the target is present, otherwise nothing.

## 3.2 Real images datasets

Real observations were processed and employed in two different networks. In this case, manual labeling was necessary. Luckily, this tedious operation has been made simpler thanks to the

CVAT open-source software, an interactive video and image annotation tool for computer vision. It was used to generate the text file with the bounding boxes [59]. The software inputs are the images to be processed, while the outputs are text files compatible with YOLO.
The datasets generated by processing *.fits* observations are the following

- Scikit-Image model-based: used for the detector as it achieves the highest performance at the expense of computational time. It consists of about 2100 images with attached labels.
- Mathplotlib.PyPlot model-based: used by the tracker due to the high conversion efficiency of the images which results in a reduction in detection quality. For this reason, the dataset consists of about 4000 images with associated labels. However, the performance is still lower than the skimage based model.

The training phase and the results of the two networks are described in Chap. 4 for skimage based dataset and Chap. 5 for PyPlot based dataset.

## 3.3 FITS format

Flexible Image Transport System (FITS) was initially developed by astronomers in the USA and Europe in the late 1970s to serve the interchange of data between observatories and was brought under the auspices of the International Astronomical Union in 1982 [60]. In 2012, FITS is still in widespread use as a data interchange and archiving format by astronomers. FITS is a file format designed to store, transmit, and manipulate scientific images and associated data. The term "image" in the standard's name is loosely applied and FITS files often contain only non-image data. FITS was designed to facilitate the unambiguous transmission of n-dimensional regularly spaced data arrays, an n-cube. These multi-dimensional arrays may be 1-D spectra, 2-D images or data cubes of three or more dimensions. Two-dimensional tables containing rows and columns of data can also be stored in a FITS file. Therefore, FITS is categorized primarily as a dataset format, with use for image data as secondary.

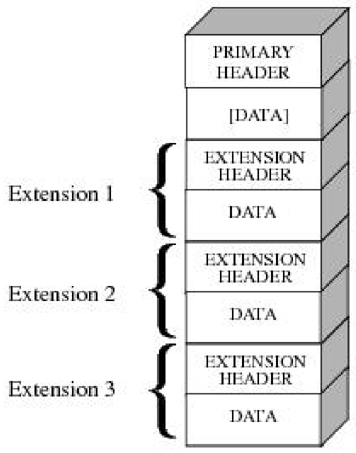


**Figure 3.1** FITS format structure [61].

The FITS images provided for this work by the Pratica di Mare observatory are two-dimensional ($X$, $Y$). They are based on a 16 bit integer architecture, therefore, a pixel color depth ranging from 0 (absolute black) to 65536 (absolute white). An additional a header describing the characteristics of the shot, including sensor name, location, and other parameters (see Fig. 3.1). To view this format file a free software ESO/ESA/NASA FITS Liberator [62] was used. It allows to perfectly adjust the content viewing and editing options. It is not thought to directly interact with the python code, hence custom libraries are employed to bring the trail out.

## 3.4 Image processing

The images produced by the telescope cannot be used directly for the analysis, it is necessary to apply a series of transformations, in order to highlight the moving object, reduce noise, and vignetting. Moreover, they are not compatible with YOLO, due to the 16-bit architecture. Two different image elaboration processes based on two distinct libraries have been used to create the datasets for the detector and the tracker:

- First dataset: employs the Scikit-Image Python library (shortened Skimage). It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy [63].
- Second dataset: it is based on the Python library Matplotlib.pyplot (shortened PyPlot). It is a collection of native MATLAB functions translated to the Python environment. It can edit and interact with figures, for example by adding lines and points or displaying images with a certain scale of values. Its advantage is the optimization of transformations using CPU, even if they are limited in number.

Skimage transformations aim to obtain the highest possible quality for improving the output prediction in the only detection case. PyPlot processing converts images in a shorter time without penalizing quality too much in order to create an efficient and responsive tracker.

### 3.4.1 Scikit-Image

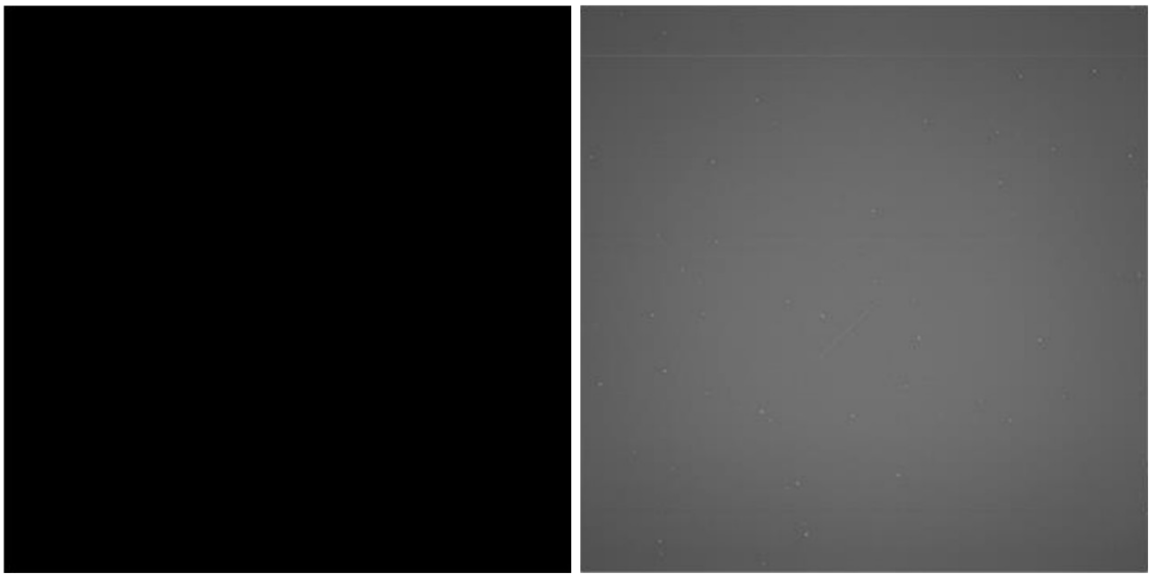
Once the FITS image is converted to a 4096x4096 matrix (it corresponds to the telescope acquisition resolution) is possible to modify it using the scikit-image exposure, util, filters, and transform modules. Finally, the image is saved in PNG. The following figures show the transformations applied to an input image array.

#### Logarithmic correction

The transformation is defined by the formula:

$$s = c \cdot \log(r + 1) \tag{3.1}$$

Where $s$ and $r$ are respectively the pixel values of the output and input image while $c$ is a constant (see Fig. 3.2) [64].
The value 1 is added to each pixel value of the image so that the logarithm is always finite. During registry transformation, dark pixels in an image are expanded relative to higher pixel values while the higher pixels are compressed. The value of $c$ in the logarithmic correction varies the intensity of the improvement.



(a) (b)

**Figure 3.2** Logarithmic correction transformation. **(a)** is the raw images, while **(b)** is the same image elaborated through logarithmic correction [64].

### Contrast Limited Adaptive Histogram Equalization

CLAHE is an algorithm for local contrast enhancement by means of histograms computed over different tile regions of the image. Details can therefore be emphasized even in regions that are darker or lighter than others. Fig. 3.3 shows the visible tracklet for the first time [64].

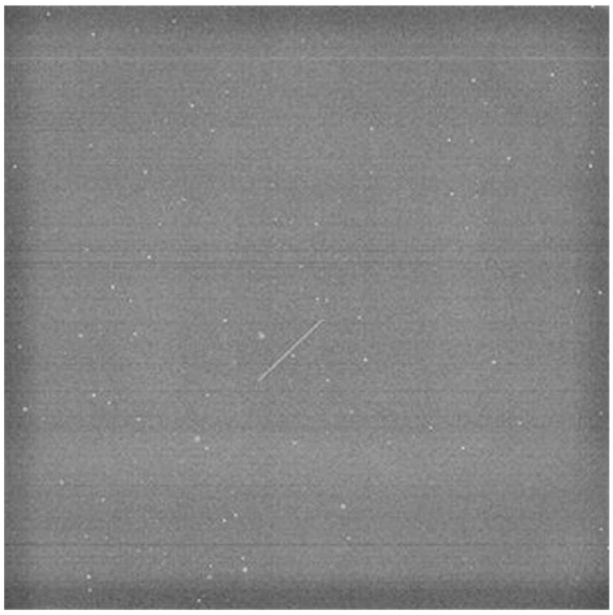


**Figure 3.3** Adaptive equalization of the Fig. 3.2 **(b)** [64].

### Performs Gamma Correction

PGC, also known as Power Law Transform, function transforms the input image pixelwise according to the equation:

$$O = I^{\gamma} \tag{3.2}$$

Each pixel is scaled into the range 0 to 1. Fig. 3.4 shows how the transformation increases contrast and improves the visibility of clear objects [64].

**Figure 3.4** Gamma correction for scaling pixels of the previous Fig. 3.3 [64].

### Noise removal

It is a common digital filtering technique to remove noise from an image or signal [65]. A well-known techinque is the median filtering because, under certain conditions, it preserves edges. The main idea of the median filter is to loop the signal input by input, replacing each pixel value with the median of the neighboring pixel values. The neighbor pattern is called a "window", which runs, entry by entry, over the entire signal. It is the last transformation undergone by the image, before being resized. Fig. 3.5 shows how noise removal gives better sharpness, and this results in better performance in the model that will be based on these images.

**Figure 3.5** Noise removal through local median of the previous Fig. 3.4 [65].

#### Image resize

This is not a transformation but consists of scaling the image by a certain scale factor. The scale factor can be a single floating-point value or multiple values, one along each axis. The images have an initial resolution of 4096 and are reduced to 1024 to improve the speed of the training.

### 3.4.2 Mathplotlib.pyplot

Compared to the previous case, the needed image tranformations from the acquisition to the *.png* conversion should not take too much, because especially with fast LEO objects (with angular velocities up to 0.5 $deg/s$) observed by narrowed FoV telescopes (roughly 3°), timing is essential to detect and track a target. The goal is to obtain a good image quality and a low computational time, i.e. at most a couple of seconds. Once the fits file is converted into a matrix of 4096x4096 elements, it is resized keeping the same color range of values and an anti-aliasing filter is applied. Fig. 3.6 shows the difference between a resolution down-scaling with preserving range and antialiasing filter options enabled on the left and not on the right. Figure 3.6 shows almost identical images, except that the one on the left is less jagged and better defined due to the antialiasing filter.

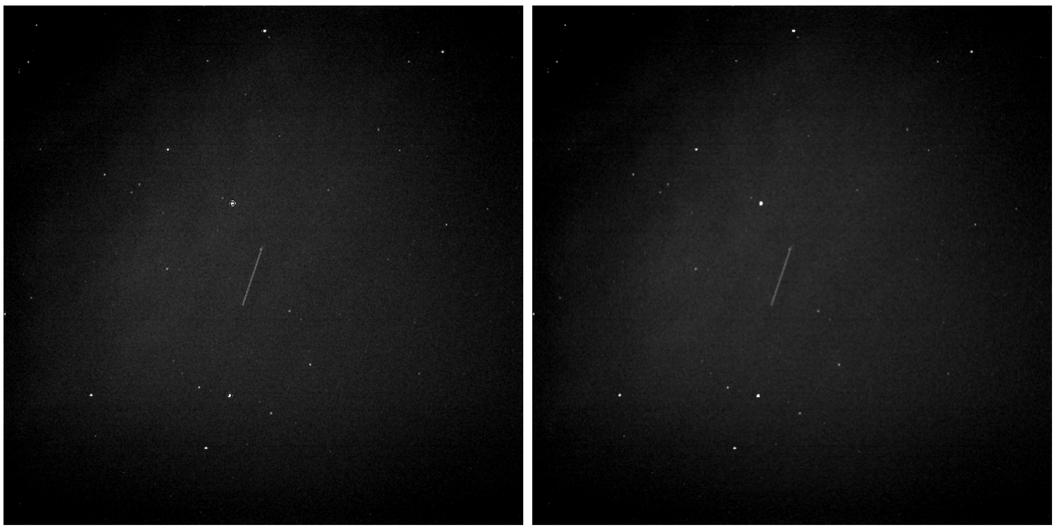


**Figure 3.6** Antialiasing filter improves track definition: image **(a)** presents the pyplot antialiasing filter, while **(b)** not.

Subsequently, thanks to the imshow visualization function of pyplot are applied some transformations to the image:

- vmin/vmax: define the data range that the colormap covers. It emphasizes low light sources but at the same time can generate noise. It allows low light or cloudy tracklets to be seen and detected;
- Lanczos interpolation: it is best suited interpolation for predominantly black images with some light sources. It consists of re-pixeling the image to make it more uniform and more grainy. Thus, the quality of the tracklets is improved and they appear smoother so easier to detect.

Fig. 3.7 shows how the transformations allow the images to be usable and subsequently analyzed by detectors. In particular, the image on the left (Fig. 3.7 **(a)**) has neither interpolation nor changes in the range of values, the central one (Fig. 3.7 **(b)**) has the only change in the range of values producing a non completely clear image with a still recognizable trail. The one on the right (Fig. 3.7 **(c)**) has undergone all the transformations listed so it appears bright and sharp.

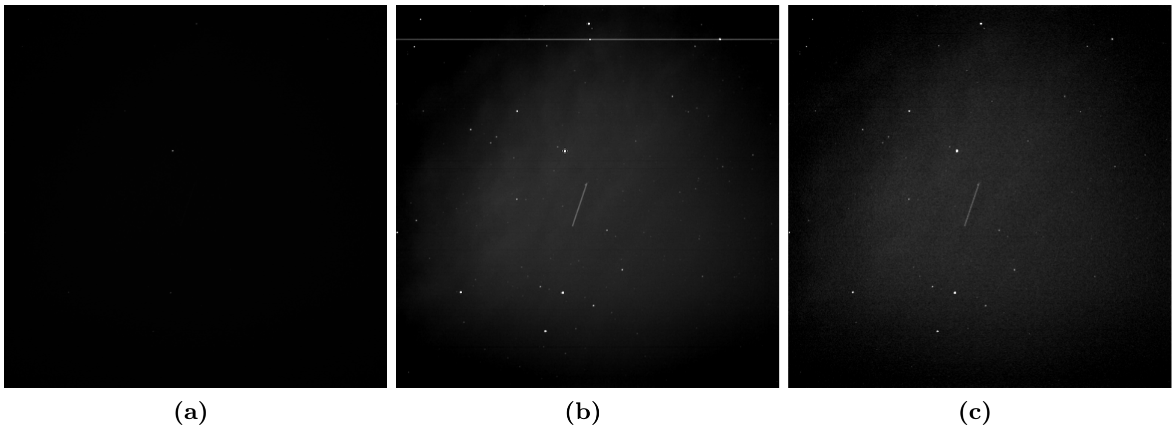

**(a)** **(b)** **(c)**

**Figure 3.7** Different Pyplot transformations. Image **(a)** is almost completely black due to low average luminosity level of the original *.fits* image (about 1100 value over a maximum value of 65536), while, image **(b)** presents a visible tracklet due to the range change transformation, and finally, image **(c)** is the interpolated version of the previous one that results more bright and sharp.

Finally, the image is saved in *.png* format.

## Conversion differences

It is important to note that the two conversion scripts that are based on the two libraries mentioned above have different purposes: the script based on skimage aims to obtain the highest possible quality for improving the output prediction in the only detection case. The other one based on PyPlot converts images in a shorter time and without penalizing quality too much in order to create an efficient and responsive tracker. The first algorithm takes on average 7 seconds to convert an image, and the second about 0.8 seconds on average. Fig. 3.8 shows on the right an image converted with Skimage, while on the left the same converted with PyPlot. The former has more details, greater sharpness, and better resolution.

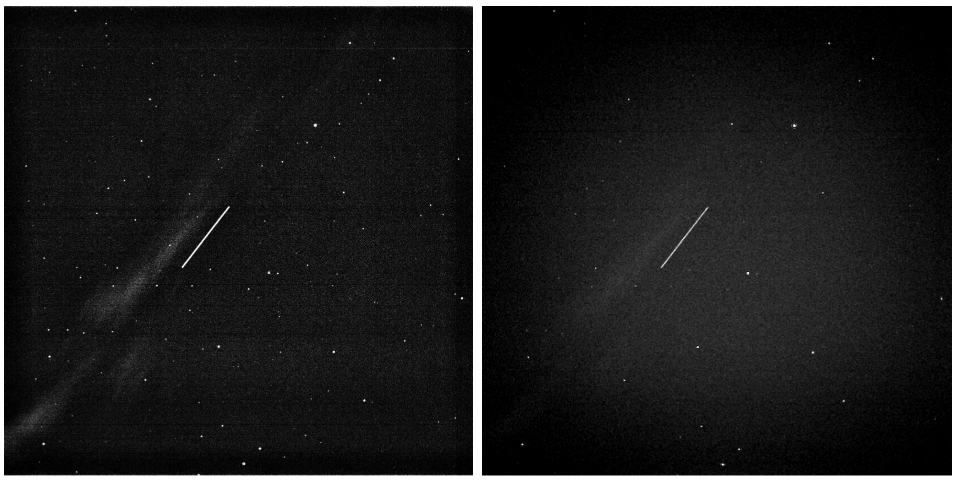

**(a)** **(b)**

**Figure 3.8** Image processing quality comparison, **(a)** is elaborated through Skimage, while **(b)** through Pyplot.

A critical case occurs when the fits image to be converted has a faintly visible tracklet or pronounced noise, such as clouds or light sources, making hard its dentification. Fig. 3.9 shows the case where the first script brings out the tracklet while the second does not.

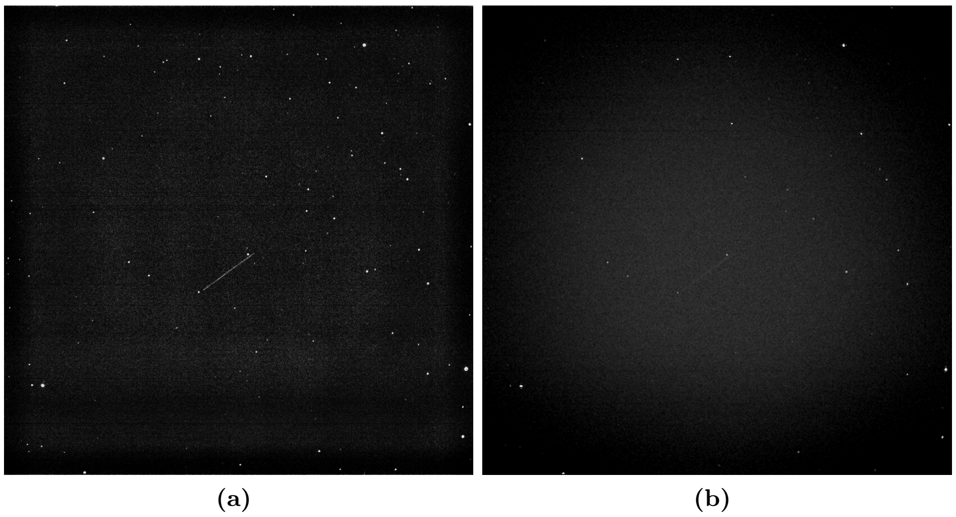


**Figure 3.9** Faint tracklet different transformation, **(a)** is elaborated through Skimage, while **(b)** through Pyplot.

# Chapter 4

# Real Images Detector

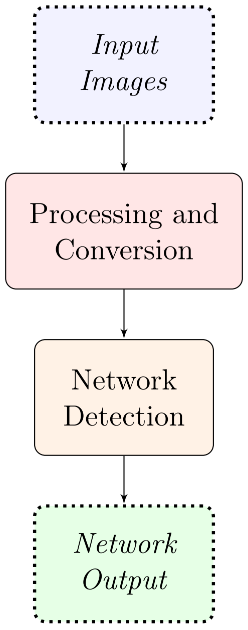


**Figure 4.1** RID working principle.

The structure of RID (Real Images Detector) can be divided into two parts (see Fig. 4.1): one dealing with the neural network design and the second focuses on real observations processing and conversion. The first section of this chapter deals with training, results, and testing of the model, while the latter outlines the structure of the algorithm, in terms of accuracy and performance. The aim of RID is to act as a filter to speed up the tracklet extraction processes. These are rather complicated and cumbersome processes that require very detailed mapping of the sky. The observations resulting from a night of acquisitions are no longer all processed through classical techniques, but are initially analysed by RID, and only the shots containing tracklets will be eventually analysed by tracklet extraction systems. This procedure should reduce the total process time of track extraction conventional techniques, which is generally around $15 - 20$ seconds on average for every observation.

## 4.1 RID architecture

As previously mentioned, the detector can be divided to read: the machine learning part, discussed in the previous sections, and the one related to the conversion of real fits observations as described in Chap. 3.
The union of these two parts forms RID. Fig. 4.2 shows the strucure:

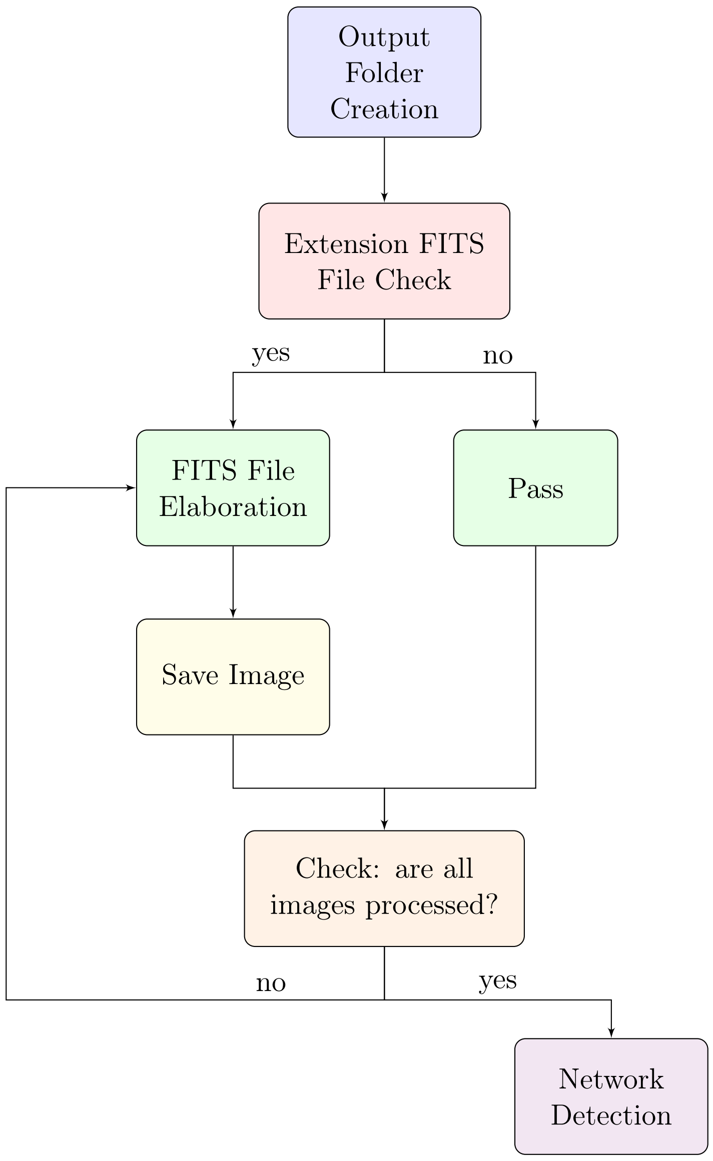
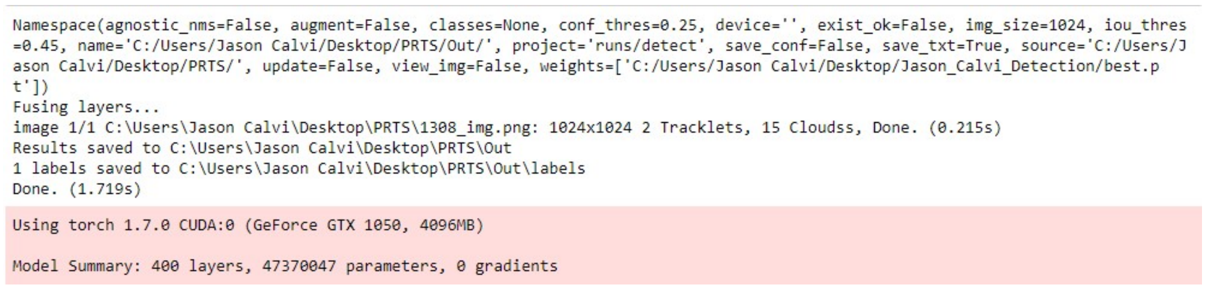


**Figure 4.2** RID flowchart architecture.

The input required by the script is a directory containing the *.fits* files to be processed. The diagram shows the whole operation. Before all, a folder is created within the input one where the processed images will be saved in *.png* format (YOLOv5 compatible).
Once all the images have been converted, the detection phase begins by means of artificial intelligence, through the following processes:

- image check: YOLO performs a check on the extension of the files to be processed, if none of them is compatible, an error message will be printed on the screen and the process will stop.
- output folder creation: if a path is indicated in the options, the script generates the desired folder, otherwise it is automatically created with a custom name. It features the *.png* images.
- loading the model: if the computer is equipped with a GPU, the model will be loaded on it via PyTorch, otherwise the CPU will be employed.
- detection: the images are analysed and then the outputs are produced.

- results: a screen shows the detected objects, the inference times for each image, and the total time of the process (see Fig. 4.3).



```
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.25, device='', exist_ok=False, img_size=1024, iou_thres
=0.45, name='C:/Users/Jason Calvi/Desktop/PRTS/Out/', project='runs/detect', save_conf=False, save_txt=True, source='C:/Users/J
ason Calvi/Desktop/PRTS/', update=False, view_img=False, weights=['C:/Users/Jason Calvi/Desktop/Jason_Calvi_Detection/best.p
t'])
Fusing layers...
image 1/1 C:\Users\Jason Calvi\Desktop\PRTS\1308_img.png: 1024x1024 2 Tracklets, 15 Cloudss, Done. (0.215s)
Results saved to C:\Users\Jason Calvi\Desktop\PRTS\Out
1 labels saved to C:\Users\Jason Calvi\Desktop\PRTS\Out\labels
Done. (1.719s)

Using torch 1.7.0 CUDA:0 (GeForce GTX 1050, 4096MB)

Model Summary: 400 layers, 47370047 parameters, 0 gradients
```

**Figure 4.3** The printed screen shows the characteristics of the image, the network, the hardware employed, the targets detected, and the inference time.

## 4.2 Folder organization

Several files are required for the detector correct working (see Fig. 5.4):

- Utils: is a folder that contains the functions useful for detection.
- Models: is a folder comprising the models of YOLOv5.
- Main.ipynb: is the main file for the functioning of the algorithm.
- FITStoPNG.py: is the python function that allows the processing of the *.fits* observations described in the previous chapter (Chap. 3).
- weight.pt: is the detector trained model.
- detect.py: is the function that allows the analysis of the images. It uses the functions in the Utils and Models folders and the weights best.pt to generate the bounding boxes and outputs.
- requirements.txt: contains the versions to be installed of all the libraries.
- README.txt: is a text file that helps a new user to set up directories and inputs.

```
/
└── RID/
    ├── utils/
    │   ├── _init_.py
    │   ├── general.py
    │   └── ...
    ├── models/
    │   ├── yolov5m.yaml
    │   └── ...
    ├── Main.ipynb
    ├── FITStoPNG.py
    ├── weight.pt
    ├── detect.py
    ├── requirements.txt
    └── README.txt
```

**Figure 4.4** RID directory tree.

The latest version of YOLO introduces an option to increase performance by about $2-3\%$ while increasing detection times by about 50%. The "augment-inference" option transforms images with resizes, distortions, and rotations to search for targets that in some cases would not be detected. It is enabled by default by RID.

## 4.3 Detectors outputs

Once the detection process is complete, the algorithm prints a list of the input images on the screen with the detected objects, the class they belong to, and the inference time.
The output generated by the YOLOv5 network is saved in the directory chosen before launching the script (if no directory is specified, the script will generate a folder in the input images folder):

- A copy of all images with bounding boxes printed around the objects, the class name, and confidence value.
- A folder named "labels" where are stored all the text files containing the characteristics of the bounding boxes.
- A folder called "Crop" where cropped images of the tracklets of the size of one degree FoV are saved (382x382 pixels).

The detect.py file created by Glen Jocher has been modified to only crop tracklets automatically. The crop procedure is designed to minimise processing time of track extraction conventional techniques. It uses the coordinates of the centroid of the bounding box. In most cases, the trace is in the centre of the cropped image, if it is on the sides of the original image, the tracklet will appear on the side of the cropped image.
The name of these files consists of three parts (considering the extension):

*[original image name] + [confidence value] + [extension .png]*

The confidence score indicates how sure the model is that the bounding box contains an object and also how accurate it thinks the box is that predicts [66]. It can be calculated using the formula:

$$C = P_{object} \cdot IoU \tag{4.1}$$

Where $C$ is the confidence value, $P_{object}$ is the precision referred to the object detected, and $IoU$ is the intersection over union value between the predicted bounding box and the ground truth.

## 4.4 Training phase

Before the training phase, two steps are required to organize data and structure:

1. Environment setup
2. Files and directories structure

### Environment setup

The training of a neural network generally requires considerable hardware resources to be performed, this demand increases as the complexity of the model, and if the size of the training set increases. In order to ensure adequate computing power to perform the necessary tasks, most of the project was developed using Google Colab. Due to Google GPU allocation restrictions, development progressed locally on Jupyter Notebook and local graphic card.

### Files and directories structure

YOLOv5 requires that the files and directories containing the dataset and the configuration files be structured in a precise way (see Fig. 4.5):

- In the dataset folder must be present two sub-folders named "images" and "labels".
- Within these there must be other two sub-folders: "train" and "valid" which contains the images if the main sub-folder is "images" and the associated text files if the main sub-folder is "labels".
- Finally, a *.yaml* file describing the directories' paths must be written, structured as follows:

*train: "train folder path file"*
*val: "val folder path file"*
*nc: "number of classes"*
*names: "names of the classes"*

```
/
└── Dataset/
    ├── images/
    │   ├── train/
    │   │   ├── Img_01.png
    │   │   ├── Img_02.png
    │   │   ├── Img_03.png
    │   │   └── ...
    │   └── valid/
    │       ├── Img_07.png
    │       ├── Img_08.png
    │       ├── Img_09.png
    │       └── ...
    └── labels/
        ├── train/
        │   ├── Img_01.txt
        │   ├── Img_02.txt
        │   ├── Img_03.txt
        │   └── ...
        └── valid/
            ├── Img_07.txt
            ├── Img_08.txt
            ├── Img_09.txt
            └── ...
```

**Figure 4.5** Directory tree organization for YOLOv5 training phase.

By structuring the data in this way, the system has all the information it needs to access the data.

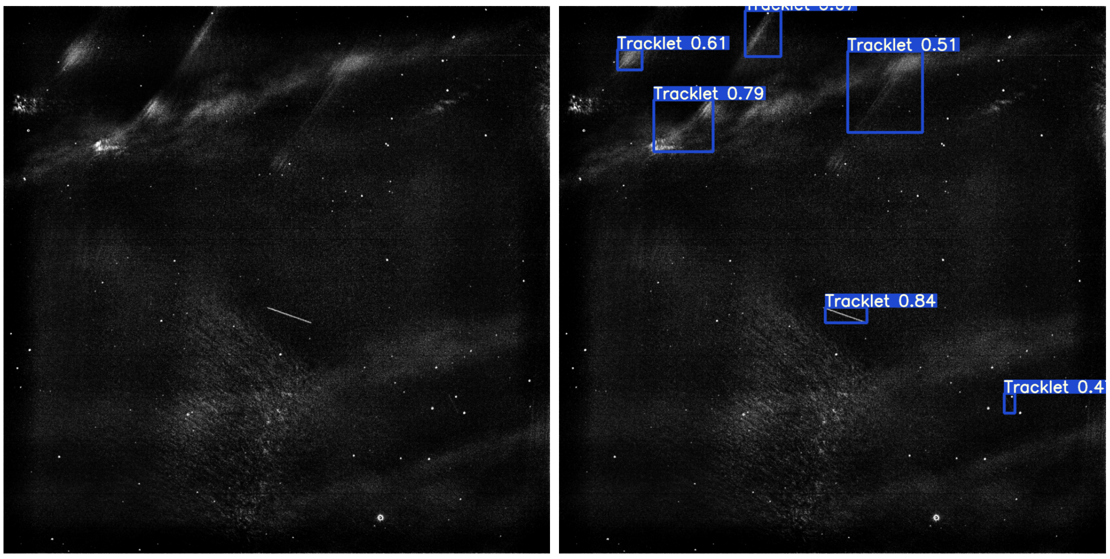
### Synthetic network training phase

Glenn proposed 4 versions of YOLOv5, the mAP values on the COCO reference dataset increases proportionally with the size of the model, but the system suffers in terms of FPS analyzed.
The search for space debris is a delicate task due to diverse tracklets length and to possible low SNR values in the captured images. Thus, the acquisition strategy has to be suitably arranged by selecting the exposure time in accordance to the orbital regime of interest, granting at the same time multiple shots and sufficient image quality for each analyzed target. Meanwhile, the algorithm aimed at trails identification has to be fast and accurate to allow follow-up observations. Hence, the YOLOv5 medium version has been chosen because it is a middle ground between performances and speed.
The following parameters have been set at the training command:

- img: the size of the image side, set at 1024 pixels to accelerate the learning process (by a factor of ten);
- batch: batch size, i.e. number of images propagated within the network in a single epoch. Several tests were made and the one that gave the best result was 1;
- epochs: number of training epochs, initially the behavior of the system was observed in a dozen epochs but the maximum validation in terms of mAP is reached in 100;
- data: the path of the streak.yaml configuration file, created in the previous step;
- cfg: the model used in this case is the "medium";
- weights: The starting weights for training the network, in this case, a first experiment was done 'from scratch' then using random starting weights, but the best performance was obtained by finetuning on pre-trained weights on the COCO dataset.

### Skimage network based training phase

Initially, as with the network built using synthetic images, the model based on real observations processed by Skimage is characterised by a single class of objects, named "Tracklets". After a training of 100 epochs, with batch equal to 1, using the medium model proposed by YOLOv5, the network reached very high performance, but one aspect that has not yet been considered is how disturbances appear in telescope shots. Clouds and some light sources can take on a variety of shapes, including tapered, rectilinear ones, sometimes resembling the tracks left by satellites or other orbiting objects. The risk in this case is that the network will not be able to distinguish the disturbances as background objects but will attribute them to the tracklet class. Fig. 4.6 shows an observation in which there are two tracklets, one very bright and the other faint, and many others (most likely clouds) quite similar to tracks. The network analysis highlighted the concern just discussed: Fig. 4.6 **(b)** shows that many disturbances and/or clouds are identified and classified as tracklets.



**(a)** **(b)**

**Figure 4.6** Output detection using one class network.

This behaviour is not an isolated case but it happens every time the disturbances have an elongated shape. Therefore, the model suffers from a significant limitation since it is not able to correctly discretize the background from the targets.

A possible solution is to eliminate from the output all those traces with a confidence value below a certain threshold according to the number of objects detected in the observations:

- 1 track: this is the most likely case. The threshold value is about 0.4. It is low in order to detect even less bright traces.
- 2 tracks: less likely than the previous case. The threshold value is about 0.55, it is increased to avoid false tracklets.
- 3 tracks: is unlikely, so the threshold value is about 0.65 to exclude disturbances.
- more than 3 tracks: the threshold value is about 0.7, it is set high, because it is highly improbable to detect four or more satellites and/or debris.

The risk using this approach is that the less bright traces are not recognized by the detector. In fact, observing again Fig. 4.6: the tracklet at the bottom right has a lower confidence value than some disturbances. Therefore, it would be excluded from the detection outputs.

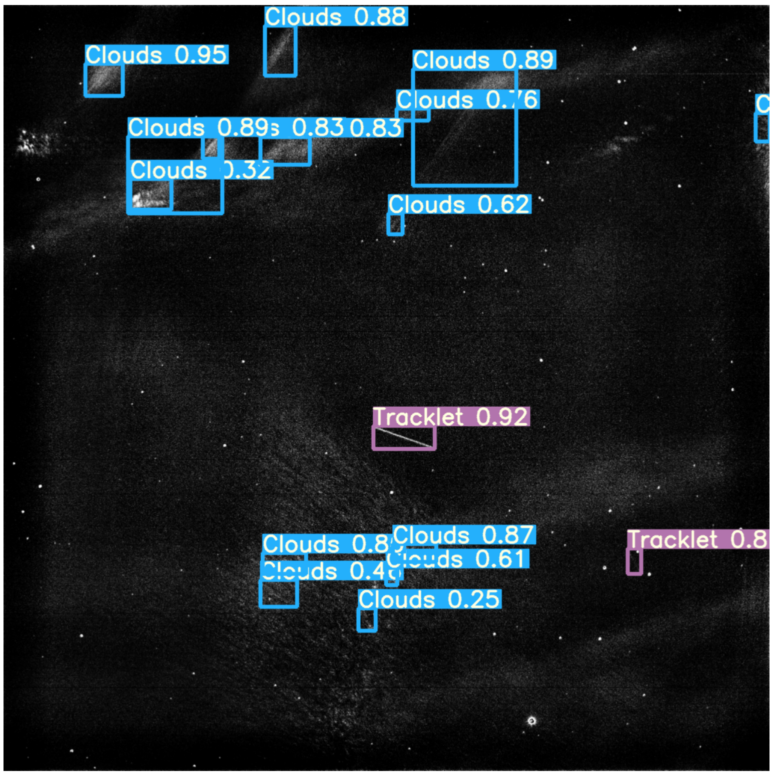


**Figure 4.7** Two classes network output works better than one class due to its ability to detect disturbances.

It is important to note that the network does not fail to identify traces but fails to distinguish disturbances with respect to the background. To overcome this problem, a new class named "Clouds" has been added in addition to the "Tracklets" one. To include this class, the dataset had to be implemented by adding new bounding boxes in the text files via CVAT [59]. The newtwork architecture and parameters are the same as the previous described implementation. Once the learning phase is complete, some images similar to the one in Fig 4.6 were analysed. Fig. 4.7 shows the correct classification of the two-class network that does not generate false positives. The model is now able to correctly and accurately detect tracklets.

## 4.5 Training results

### Synthetic network training results

As configured, the network achieves outstanding results when analysing synthetic images: maximum validation in terms of mAP is achieved in just a few epochs (see Fig. 4.8). Figure 4.8 **(a)** shows the model loss value evolution along the training epochs, while Fig. 4.8 **(d)** represents the network objectness along the epochs, that is essentially a measure of the probability that an object exists in a proposed region of interest.
If the analysis shifts to real images, performance drops considerably.

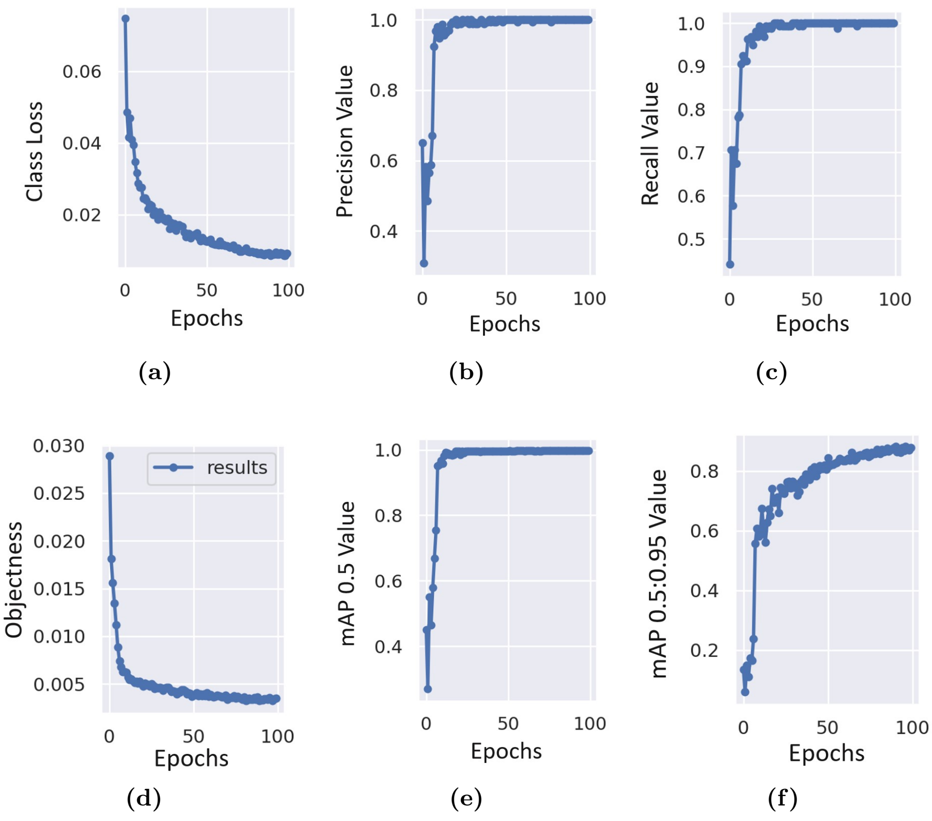


**Figure 4.8** Synthetic network results after 100 epochs.

The quality parameter of a network is the mean Average Precision (mAP), which is the average accuracy of each class. The mAP value is about 1 in case of confidence value greater than 0.5 and 0.85 when the confidence lies between 0.5 and 0.95.

Another figure of merit is the F-value ($F_1$), a function of precision and recall, defined in the Eq. 4.2 where P is the Precision and R is the Recall of the network for different confidence values. It calculates the goodness of a test, especially if there is a non-uniform distribution of classes (as in the Skimage based network case).

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{4.2}$$

Figure 4.9 shows that the synthetic model is a well trained model for almost all confidence values.

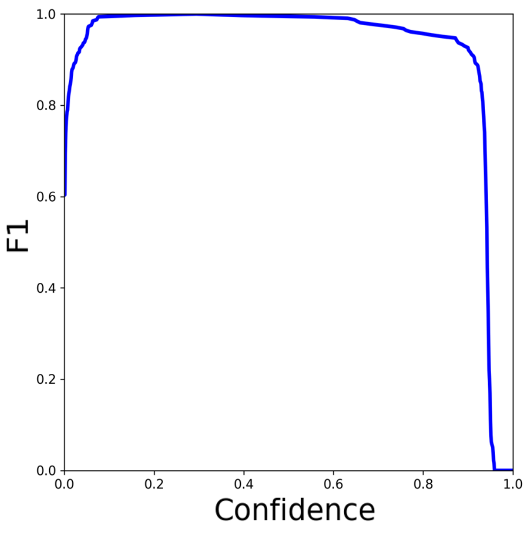
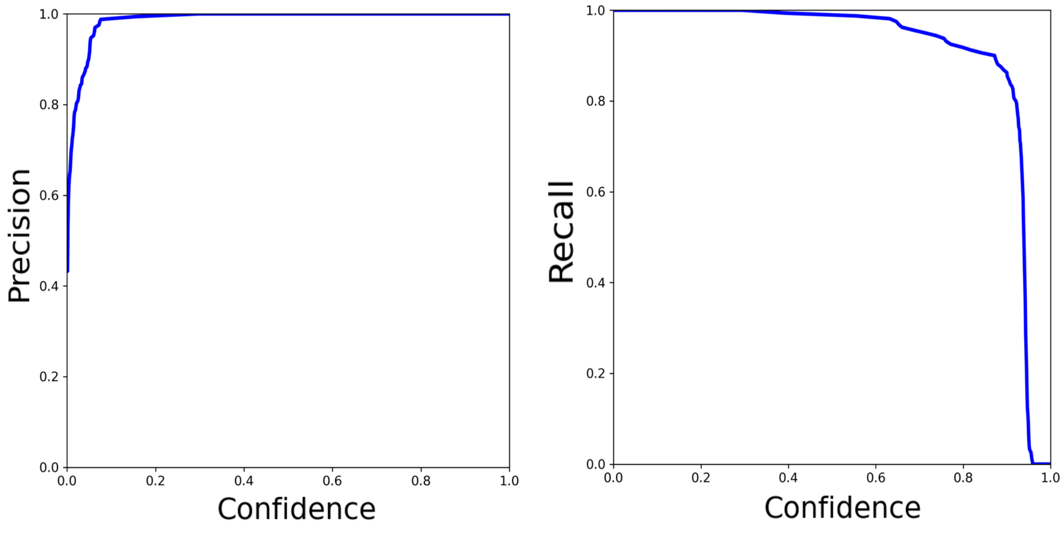


**Figure 4.9** Synthetic network $F_1$ curve.

Figure 4.10 show three performance graphs: the first one shows the precision of the network over the confidence values, i.e. how accurate is the predictions along the different confidence values. The second one is related to the recall over confidence, i.e. how many relevant target are identified. The last graph displays precision over recall, where the area under the curve is the Average Precision (AP) of the various classes (one for this case). The curve should ideally go from $P = 1$, $R = 0$ in the top left towards $P = 0$, $R = 1$ at the bottom right to capture the full AP.



(a) (b)

**Figure 4.10** in **(a)** the Precision over confidence and in **(b)** the Recall over confidence.

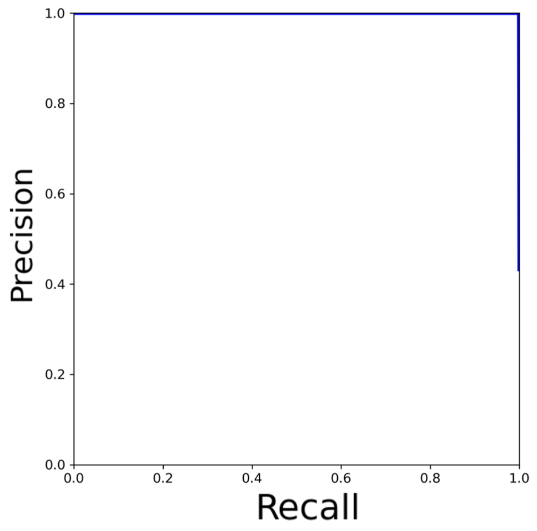


**Figure 4.11** Precision over Recall.

### Skimage based network training results

Now the network has achieved good performances, in particular Fig. 4.12 shows the quality of the network considering the performance average of the two classes.

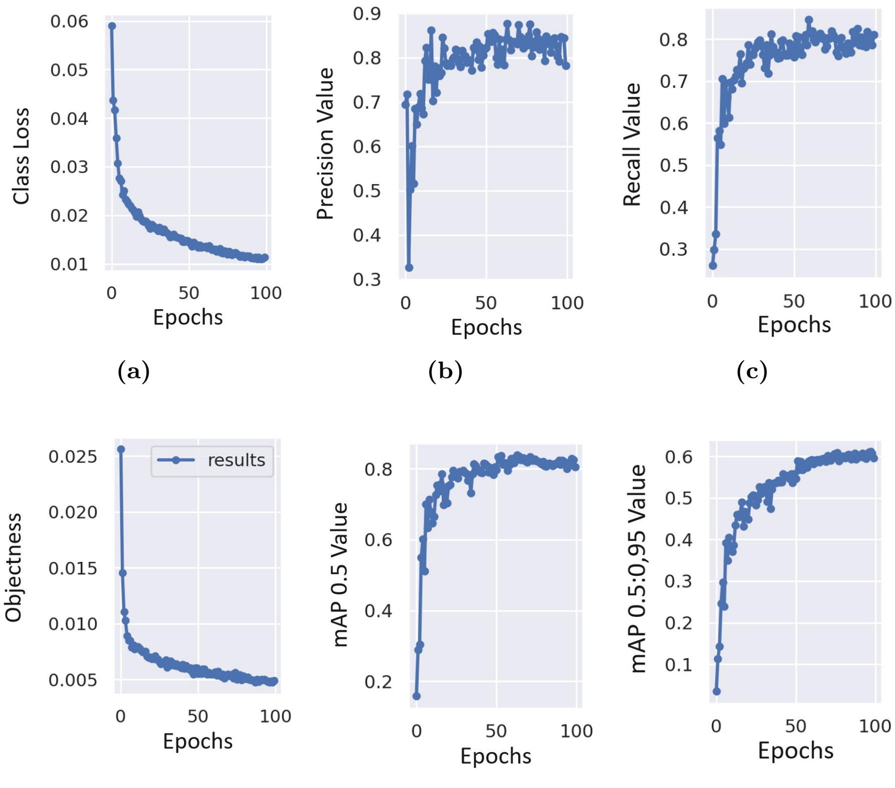


**Figure 4.12** Skimage based network results after 100 epochs.

The mAP exceeds the value of 0.8 in case of confidence values greater than 0.5 and 0.6 when the confidence lies between 0.5 and 0.95.
Fig. 4.13 shows three different curves: the light blue one refers to the class of "Traklets" and for most confidence values its $F_1$ value tends to one (this means that the class has been well

trained), on the contrary, the orange one is referred to the "Clouds" class and presents low $F_1$ values (due to few samples for this case), and finally, the bold blue one represents the average of the two curves and indicates good general behaviour of the network. The Fig. 4.13 legend emphasises that the highest $F_1$ value is achieved with a Confidence value of 0.2, meaning that the model can identify accurately poorly visible traces.

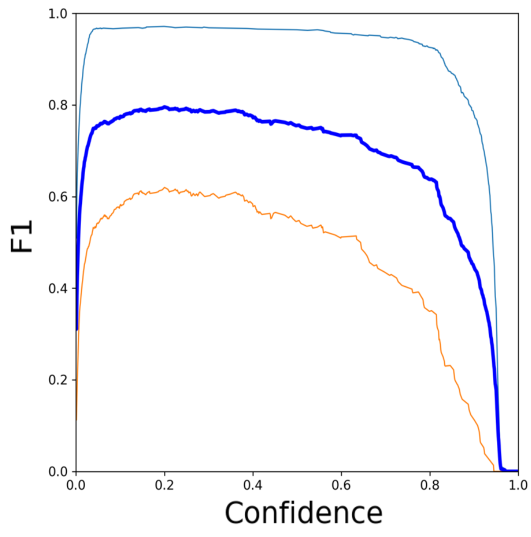
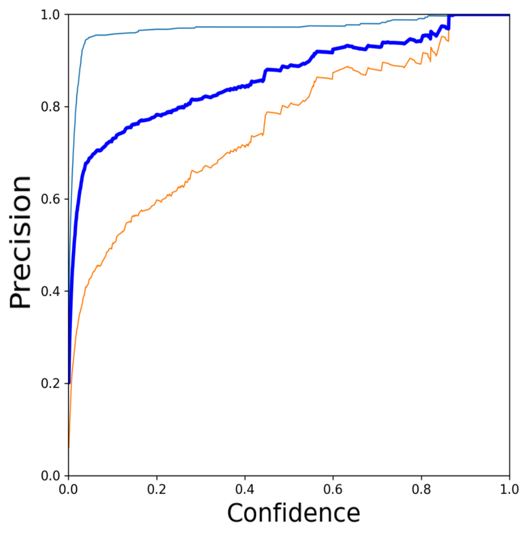
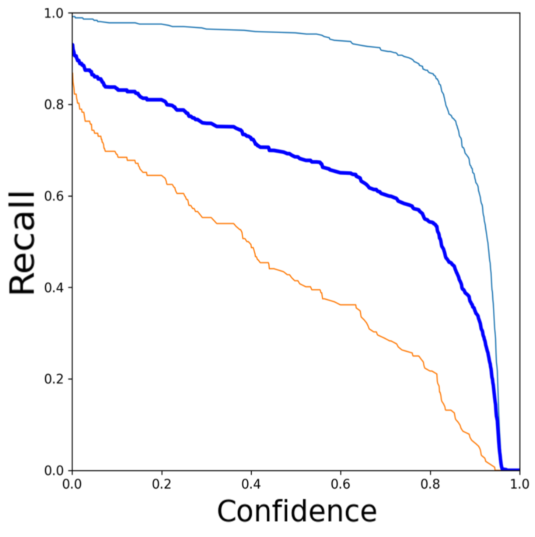


**Figure 4.13** Skimage based network $F_1$ curve.

In particular, in all three cases Fig. 5.22 shows how the tracklet class obtains excellent performance, due to a numerous dataset (about 1500 items), while the disturbance class performs worse, due to less objects (about 600 elements).



(a)



(b)

**Figure 4.14** in **(a)** the Precision over confidence and in **(b)** the Recall over confidence (the light blue curve is referred to the "Tracklets" class, the orange one to the "Clouds" class, and the bold blue to the average of the previous two).

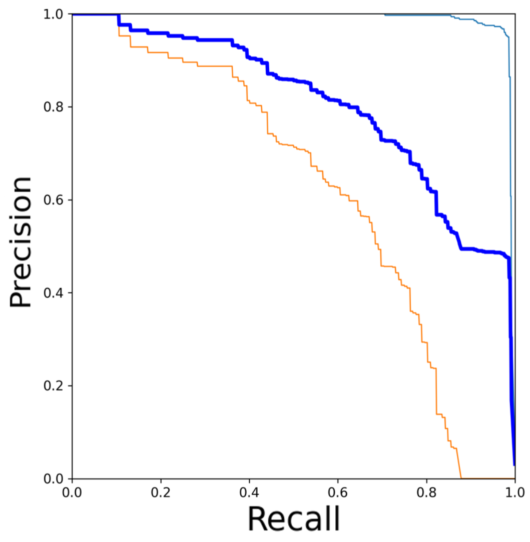


**Figure 4.15** Precision over Recall (the light blue curve is referred to the "Tracklets" class, the orange one to the "Clouds" class, and the bold blue to the average of the previous two).

## 4.6 Testing phase

The testing phase is the most challenging one because it tells how the trained model will perform with unknown inputs. It processes the remaining 10% of the dataset, which was not used for the previous phase, to calculate the mPA, Precision, and Recall values of all classes. The procedure requires input images with their labels so that the network can detect these and compare the input bounding boxes, with the generated ones.

### Synthetic network testing phase

The neural network was then tested on the remaining 10% of the dataset, which was never used during the training phase, this further demonstrates that the network has learned the information necessary to recognize the strips without overtraining, the results are illustrated in Table 4.1. Furthermore, the network was tested on an additional 250 real hand-labeled images and the results, as mentioned before, are much lower and are shown in Table 4.1.

| Class | Precision | Recall | mAP@.5 | mAP@.5 : .95 |
|---|---|---|---|---|
| **Tracklets** | 1 | 1 | 0.997 | 0.883 |

**Table 4.1** Synthetic network test results.

The trained model was aimed at verifying whether it was possible to correctly detect tracklets of real objects acquired by telescopes. The accuracy achieved is about 65%, because the model was not trained to analyse real scenarios, such as clouds, glare, light pollution, etc., which complicate the detection phase. Therefore, the model can detect real targets, but major improvements are needed to achieve a reliable and accurate detector.

### Skimage based network testing phase

Once again the tracklets class achieves excellent results, while the noises class achieves fair results. The Table 4.2 shows the output values of the test (in particular: "All" represents the average of the two classes):

| Class | Precision | Recall | mAP@.5 | mAP@.5 : .95 |
|---|---|---|---|---|
| **All** | 0.782 | 0.81 | 0.805 | 0.596 |
| **Tracklet** | 0.968 | 0.975 | 0.988 | 0.792 |
| **Cloud** | 0.597 | 0.644 | 0.622 | 0.399 |

**Table 4.2** Skimage based network testing phase results.

## 4.7 Network comparison

In order to obtain a complete overview, the model trained on real observations was compared with the synthetic model and the ASTRiDE trace detection algorithm. They were assessed by analysing the quality of their detection. Fig. 4.16 shows the number of real images correctly detected by the software out of a total of 100 randomly selected (ASTRiDE analyses the original *.fits* files, while the networks detect the corresponding *.png* elaborated images).

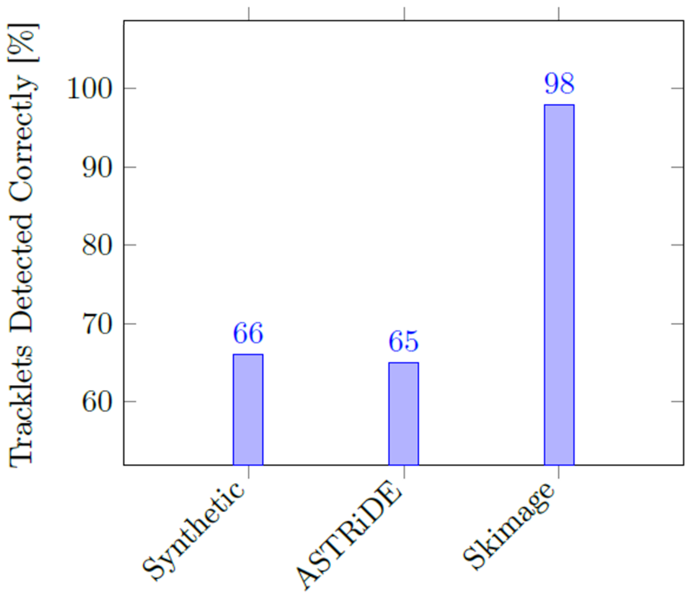


**Figure 4.16** Networks comparison bar chart. The Skimage based model achieves much better results due to efficient image processing and a well-trained model.

What has been revealed is that the synthetic network and ASTRiDE can recognise trails that are sufficiently bright if the sky is slightly covered by clouds, while Skimage based network can detect hardly visible streaks ( see Fig. 4.17, Fig. 4.18, and Fig. 4.19).

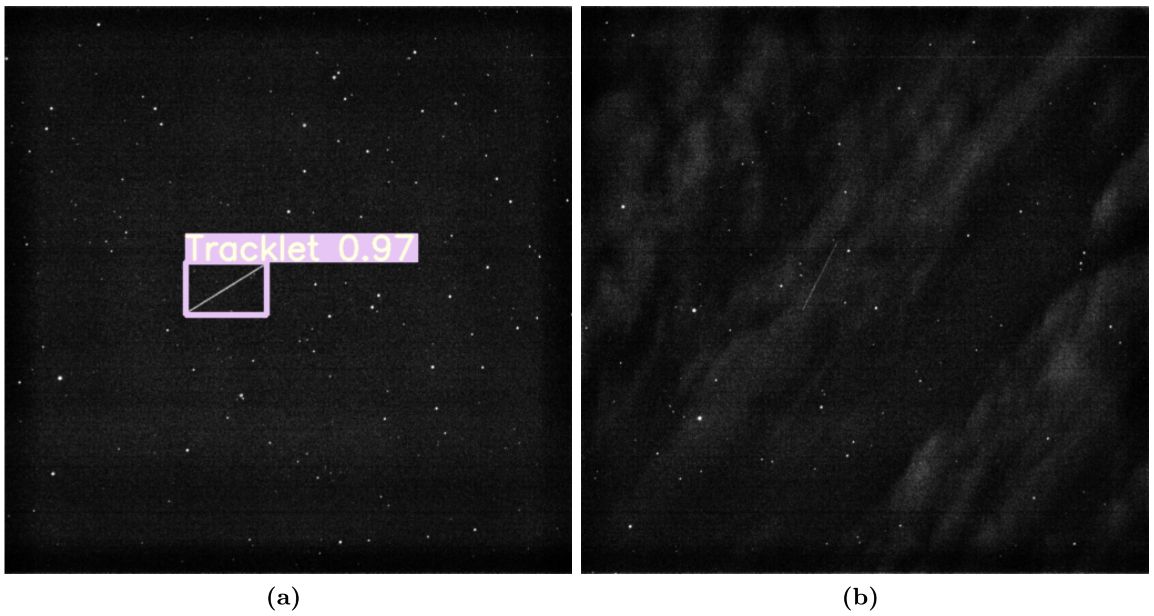


**Figure 4.17** Synthetic network can easily detect sufficiently bright tracklets, while fails if some disturbances are present or if the trace is faint.

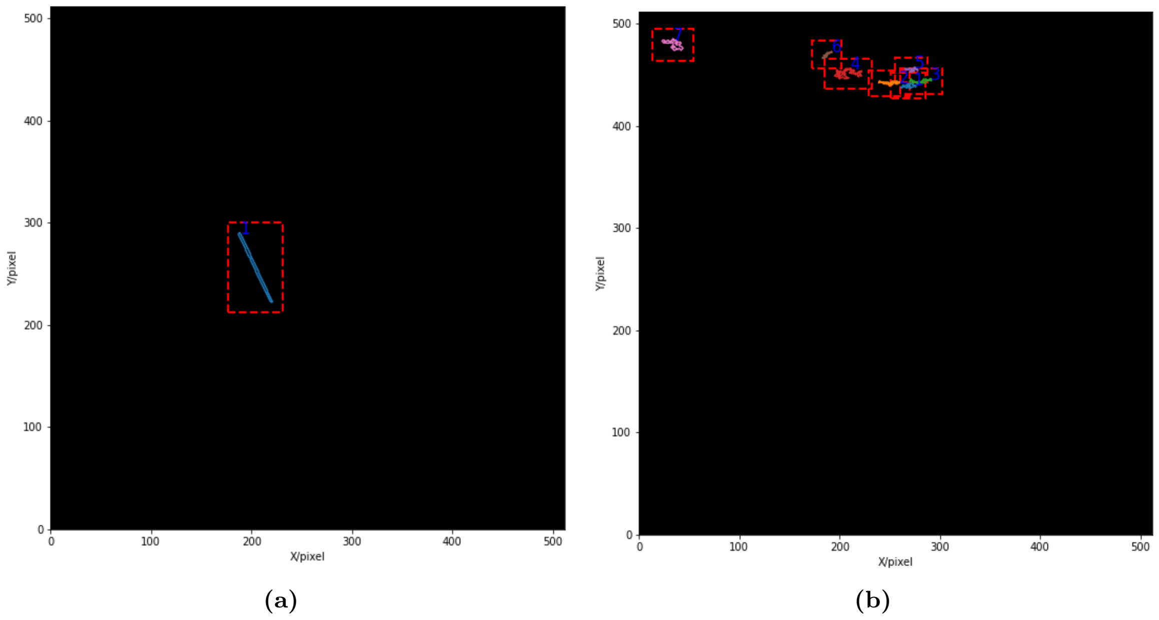


**Figure 4.18** ASTRiDE can easily detect sufficiently bright tracklets, while mistakes noises for tracks.

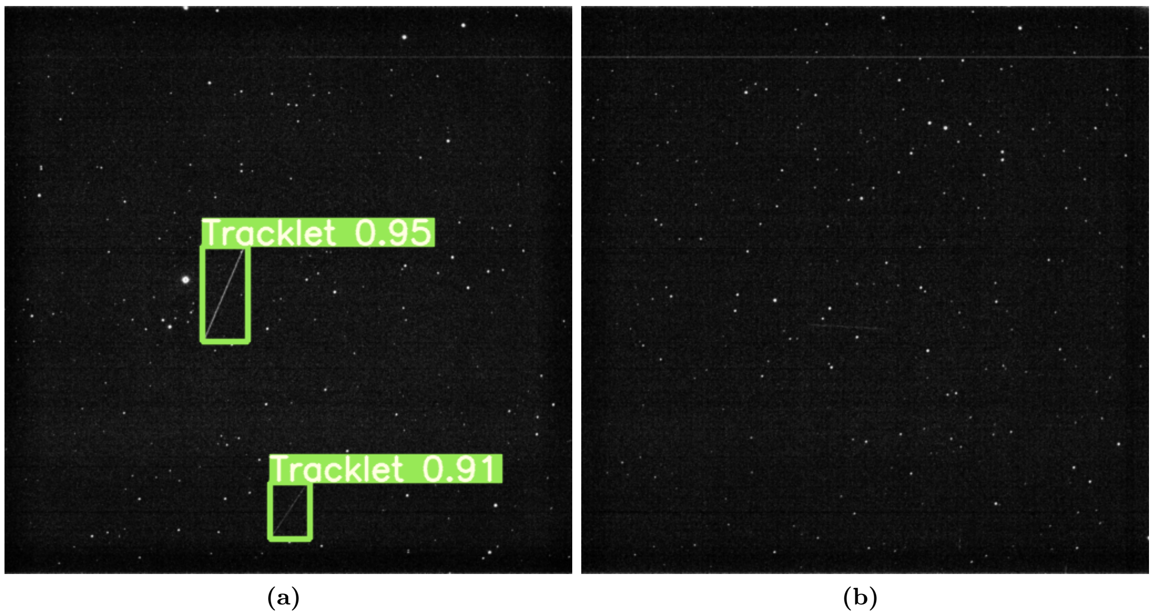


**Figure 4.19** Skimage based network detects easily most of the streaks (even in the presence of disturbances), it fails in case of very faint tracklets.

ASTRiDE and the synthetic image-based network achieve about the same performance, due to their limited ability to detect faint traces. In addition, ASTRiDE has a similar problem to the real one-class model, i.e., it mistakes noise for tracks (see Fig 4.20).

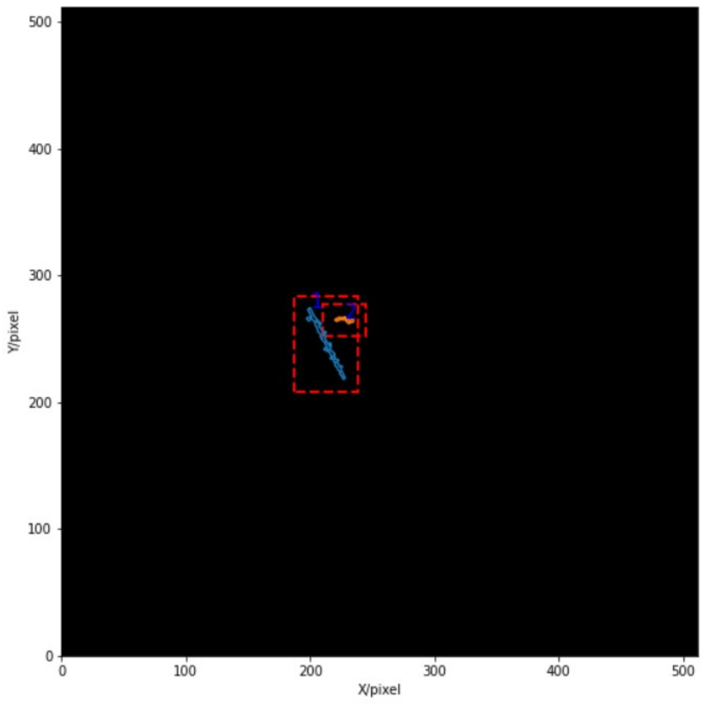


**Figure 4.20** ASTRiDE tracklet detection problem.

## 4.8 Algorithm timing performances

The times can be divided into three processes:

1. image conversion (from *.fits* file to *.png* image);
2. loading the model on the graphic card via PyTorch (if GPU available);
3. detection.

The first should be limited as much as possible by finding a fast and efficient observation processing. Unluckily, this lies in the programmer's hands who must implement an efficient algorithm and also find the most suitable image transformations to reduce possible downtime. Regarding instead the second and third one, they cannot be improved unless upgrading the computer hardware. The timing calculation was done both in the cloud on Colab and on a local computer. Table 4.3 shows the values of the two cases (in both cases, the same *.fits* file was processed):

| **Computer** | **Images Elaboration** | **Inference Time** | **Total Time** |
|---|---|---|---|
| **PC** | $6.5s$ | $0.25s$ | $6.75s$ |
| **Colab** | $7.1s$ | $0.022s$ | $7.122s$ |

**Table 4.3** Processes times required

It is quite clear from the Table 4.3 that the time required for image processing is the most time consuming, considering that the model loading time on the GPU via PyTorch is only once per process and requires about 7 seconds on average for both the local computer and Colab. Assuming 1000 images are processed (the equivalent of about one night of acquisitions): 3.7% of the time is required for detection, 0.1% for GPU model loading, and the remaining 96.2% for image conversion. Timing depends mainly on the processing, i.e. the choice of transformations to be used and partially depends also on the hardware at disposal.

The personal computer hardware is an Intel i7-7700HQ quad-core and eight-threads CPU (max frequency 2.8GHz, using Turbo Boost 3.8GHz), 16Gb of RAM, and an Nvidia GTX 1050 GPU with 4Gb of VRAM.

In terms of inference times, YOLO algorithms can use GPUs due to the support on CUDA libraries [67], while ASTRiDE uses CPUs because it does not have this option. The Table 4.4 shows the inference times required for each image using CPU:

| **Network** | **Synthetic** | **ASTRiDE** | **Two Classes** |
|---|---|---|---|
| **Time (s)** | 0.677 | 1.67 | 0.734 |

**Table 4.4** Networks inference time using CPU

While, the Table 4.5 shows the GPU inference time (in this case, ASTRiDE is not compared with other networks due to the impossibility of running on GPUs):

| **Network** | **Synthetic** | **Two Classes** |
|---|---|---|
| **Time (s)** | 0.052 | 0.061 |

**Table 4.5** Networks inference time using GPU

The processor employed is an Intel Xeon 2.20 GHz dual-core, and the GPU is a Tesla T4 with 16 Gb of VRAM.

## 4.9 Conclusion and future developments

As the results showed (see Table 4.6), the ability of the network to analyse real observations quickly and accurately is excellent. In almost all cases the detection is successful and the time of this process is about 0.25 seconds with a low-level 2017 GPU. The critical point of RID lies in the rather slow processing of the observations run on processors only. Surely the performance would improve if the algorithm ran on a newer computer with updated hardware. Therefore, RID could be a viable alternative to classical trace extraction methods used by observers, due to its low computational speed.

| **Quantity** | **Performance** |
|---|---|
| *Images Detected* | 98% |
| *Conversion Time* | $6.5s$ |
| *Inference Time* | $0.25s$ |

**Table 4.6** RID results

A fundamental advantage with respect to conventional methods, is that, once data are collected in the correct way, the algorithm can be re-trained, upgrading the training set with further information, and making the network more and more robust to new cases.
In addition, the learning capability of the network could be improved by the evolution of hyper-parameters, but this requires a lot of hardware effort and time consuming ten to one hundred times longer than normal training.
One of the most challenging tasks in object detection is to estimate the direction of the detected body from a single observation. A future development could start from this problem to find a solution involving both the telescope and the detector. This could lead to a model that not only identifies and locates an object, but also determines its direction very quickly.

Chapter 5

# Linear Orbit Tracker

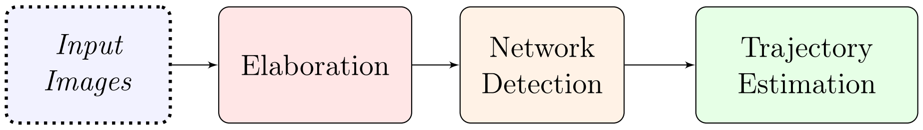


**Figure 5.1** LOT working principle.

The tracker structure can be divided into three parts: the processing of *.fits* observations into *.png* images, the trained network, and the analysis of bounding boxes (see Fig. 5.1). Initially, this chapter describes the tracker architecture with a focus on the functions used and the outputs. Folllowing, the training and testing phases are presented, and then a comparison with other networks and ASTRiDE algorithm, and finally the results of the tests carried out thanks to real satellite passages simulated with the TIG software (the part of image transformation and conversion is described in Chap. 3).

## 5.1 LOT architecture

In Fig. 5.2 a basic flowchart that underlines LOT inputs and outputs is represented. First, some directories need to be defined: the folder in which the telescope images are downloaded; the one devoted to the processed images; the output folder, which will contain subfolders divided according to the objects detected. The network is then uploaded to the GPU for a faster (almost instantaneous) response. This information combined with the actual sensor images is enough for LOT to identify and organize the detected objects into subfolders.

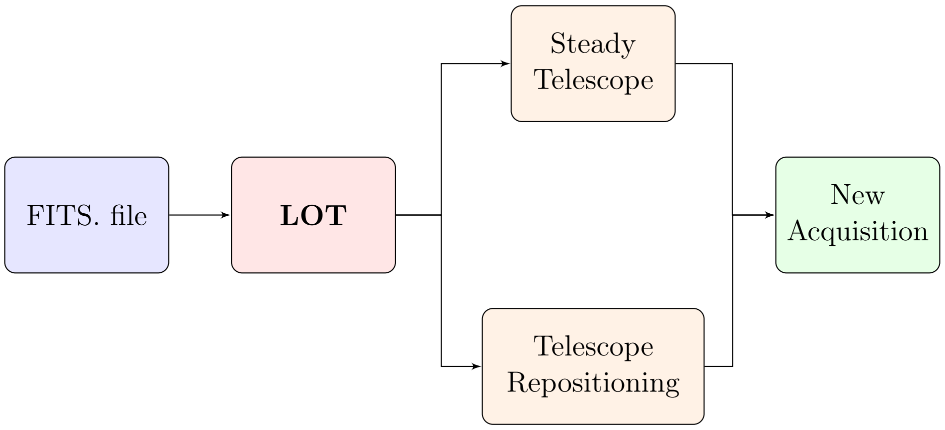


**Figure 5.2** Schematic representation of LOT I/O.

The objectives of LOT are to accurately identify LEO and MEO objects within the telescope's FoV and to track and predict their position over time through consecutive observations. The algorithm is able to identify when the observed object leaves the telescope's FoV and then the sensor position can be updated to maximise the number of observations. Once a target has been identified, the script estimates the four possible straight-line trajectories and saves the slope and intercept parameters in a text file. It is important to emphasise that the system does not 'see' tracklets as white stripes with a given direction and length but uses bounding boxes which therefore do not define a priori the inclination of a track. Fig. 5.3 shows the possible trajectories, which are calculated through statistical estimation.

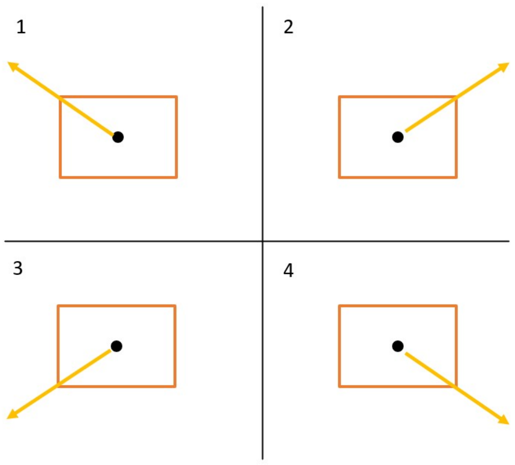


**Figure 5.3** Possible trajectories from a bounding box. The orange rectangles are the bounding boxes, the black points are the centroids, and the yellow arrows are the trajectories direction.

During this process, the telescope continues to take images and when a tracklet is identified in two consecutive images, the script compares the new bounding box with the previous one by analysing the possible trajectory and position of the centroids. This algorithm grants the possibility of acquiring more images of the same object throughout its passage in the sensor field-of-view. In the following, the detailed functioning will be described, with all the different cases, the necessary inputs, the timing, the statistical analysis, the outputs and finally the performance and the tests carried out.

## 5.2 Folder organization

In order to work correctly, LOT needs folders, functions, and the main file notebook. Hereafter, all of them are listed, together with a brief description of their function (see Fig. 5.4).

- Input_Images: is an empty folder where observations made by the telescope will be downloaded for processing.
- MODEL_FILES: is a folder containing all the functions required for the network to operate. It also contains the weights file weight.pt and the detect.py identification file.
- Imm_elab.py: enables the processing of real observations *.fits* into *.png* images through the PyPlot library (the complete process is explained in Chap. 3)
- LOT.ipynb: is the main file of the tracker. It is written in notebook, then divided into cells. It will be described in detail in a later section.
- Functions.py: is a set of functions used by LOT to organise folders, before and after observations, and to calculate the parameters of possible trajectories.

- First_det.py: is a set of functions that elaborate the network output labels when the detected object is the first in the process. This script creates a text file composed of two lines, one for each direction (if the tracklets were two, we would have two text files) and represent the inputs of the Pendenza.py script.
- Slope.py: is a Python file to calculate possible trajectories and compare the bounding boxes of successive observations. It is capable of working and tracking up to two tracklets simultaneously. It is only used after the first observation and is the core of the tracker.
- Second_det.py: is a set of functions with the purpose of comparing the current observation with the previous one. The YOLO output text files represent the inputs of this function. This script, based on the function Slope.py, decides whether the object detected in the current and the previous image is the same. When the object is the same, LOT calculates its direction and also estimates its future position. In this scenario, LOT knows the direction and heading of the object, and is able to estimate its future position. If not, the files of the previous images are moved to a folder named as the first observation. In this way, the algorithm can group multiple observations and text files of the same object, creating a new folder for each object.
- Dislocation.py: once two shots of the same body have been obtained, LOT is able to predict the position of the third shot by analysing the positions of the centroids over time. If the object leaves the telescope's FoV, the coordinates must be updated to maximise the object's detection.

```
/
└─ LOT/
    ├─ Input_Images/
    ├─ MODEL_FILES/
    │   ├─ models/
    │   ├─ utils/
    │   ├─ weight.py
    │   ├─ detect.py
    │   └─ ...
    ├─ Imm_elab.py
    ├─ LOT.ipynb
    ├─ Functions.py
    ├─ First_det.py
    ├─ Slope.py
    ├─ Second_det.py
    └─ Dislocation.py
```

**Figure 5.4** RID directory tree.

It is important to focus on the functions of the Python Functions.py file that manage the directories, to better understand how the tracker works: every time LOT is used it creates an Output folder, where all the input observations are saved, grouped in subfolders (as described before). In addition to it, the algorithm needs two auxiliary and temporary subfolders (one for the images and one for the text files) (see Fig. 5.5).

```
/
└── LOT/
    ├── Input_Images/
    │   └── Img_02.fits
    ├── Output/
    │   ├── Images_aux/
    │   │   ├── Img_01.fits
    │   │   └── Img_01.png
    │   └── Text_aux/
    │       └── Img_01.txt
    └── ...
```

**Figure 5.5** Auxiliary directory tree example.

Once the detection is done and the first trajectories calculations are performed, the image is moved from the "Input_Images" folder to the auxiliary images folder and the text files related to it are saved in the temporary text files folder waiting for a new observation. At this point, once a new capture is obtained, three scenarios can happen:

1. same object: the streaks of the new observation and the previous one represent the passage of the same object. LOT detects that the traces in the current and previous image refer to the same body, then now the auxiliary folders contain the files referring to both observations. In case there is a third observation referring again to that object the folders will continue to fill with more and more data;

2. no object: no trace appears in the new observation, thus the algorithm creates a folder named as the image and moves the photo in it. The tracklet at the second time instant may be obscured by a cloud or a disturbance, while at the third time instant it becomes visible again, so the tracker can correlate shot 1 to shot 3;

3. different object: the trace of the new observation does not represent the same object as the previous one. Therefore, if the trajectory analysis produces a negative result, all the files in the auxiliary folders are moved to a folder named as the first observation related to the previous object. For example, there are five observations related to the same body, if the sixth one is related to another body, all the first five will be moved, and only the last observation will remain in the auxiliary folders.

All the scenarios above refer to the case with one trace in each image for the sake of simplicity. In case two traces are present in the image, the system is able to track both by applying the same procedure.

## 5.3 Network loading on GPU

This tracker is designed to work for entire observation nights autonomously and automatically. Thus, the network must always be running to analyze the images just taken by the telescope. As it was described in Chap. 2, YOLOv5 analyzes the input folder looking for compatible files, then loads the model on the graphic card, and finally starts the detection. This procedure takes about 7 seconds and it represents a limitation for a real-time tracker. The adopted solution is the loading of the pretrained model on GPU via PyTorch [68]. This approach allows bypassing the control phase since it requires that the input images are first converted into matrices. In this case, it was necessary to create a script to define the directories of the weights and the device to be used for inference via CUDA library (GPU) [67].

Finally, a function was written to save the features of the bounding boxes of the detected objects in text files, as it happens with the normal inference of YOLOv5 because the output of PyTorch is a vector that describes the features of the bounding boxes.

## 5.4 Statistical analysis for trajectory prediction

In order to estimate whether two bounding boxes of two successive observations refer to the same object, it is necessary to calculate the possible trajectories and compare them through statistical analysis. Assuming that bodies move in a rectilinear motion in the images for short arcs, the trajectories are calculated from the vertices of bounding boxes through regression lines.

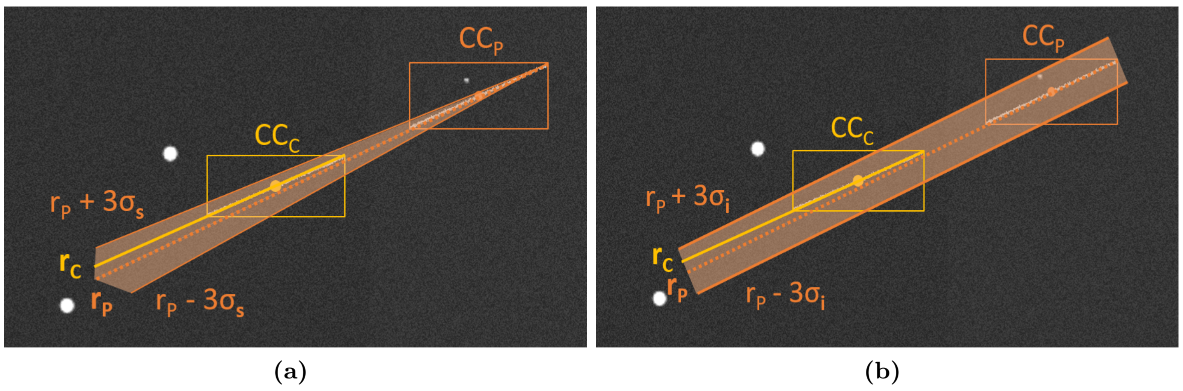


**Figure 5.6** **(a)**: shows the slope empirical rule, **(b)**: presents the intercept empirical rule. Orange elements are referred to the previous observation, while yellow ones to the current. "$CC_x$" elements are related to the centroids, "$r_x$" to the straight lines, and "$\sigma_x$" to standard deviation.

Therefore, the purpose of the analysis for each new observation is to calculate the Root Mean Square (RMS) of the slope and intercept, and subsequently, to verify that the new object simultaneously has a slope and intercept that satisfy rule Empirical rule (also known as 68-95-99.7 rule) (see Fig. 5.6). This is a statistical rule which states that for a normal distribution, almost all observed data will fall within three standard deviations (denoted by $\sigma$) of the mean or average (denoted by $\mu$). In particular, the empirical rule predicts that 68% of observations falls within the first standard deviation ($\mu \pm \sigma$), 95% within the first two standard deviations ($\mu \pm 2\sigma$), and 99.7% within the first three standard deviations ($\mu \pm 3\sigma$) (see Fig. 5.7).

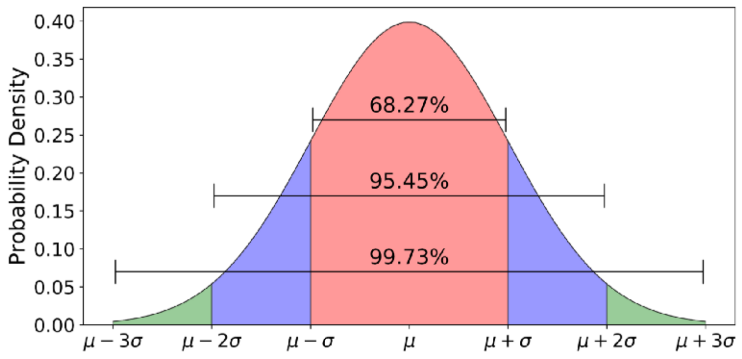


**Figure 5.7** Empirical rule: 68% of the data is within one standard deviation ($\pm\sigma$), 95% is within two standard deviation ($\pm 2\sigma$), 99.7% is within three standard deviations ($\pm 3\sigma$) [69].

Several steps were necessary to calculate the RMSs. At the beginning of the process, 250 images were generated using the TIG software, each containing a tracklet of different length, thickness and position, and their labels. As mentioned in Chap. 2, the bounding boxes are pixel accurate and therefore can be used to accurately verify the accuracy of the LOT network. Once the 250 images have been detected with the tracker model, each track has two text files: the exact one from TIG and the output from detection. The points used for the RMS calculation are those shown in Fig. 5.8, i.e. the point closest to the origin ($x1$, $y1$), and the one furthest away ($x2$, $y2$).

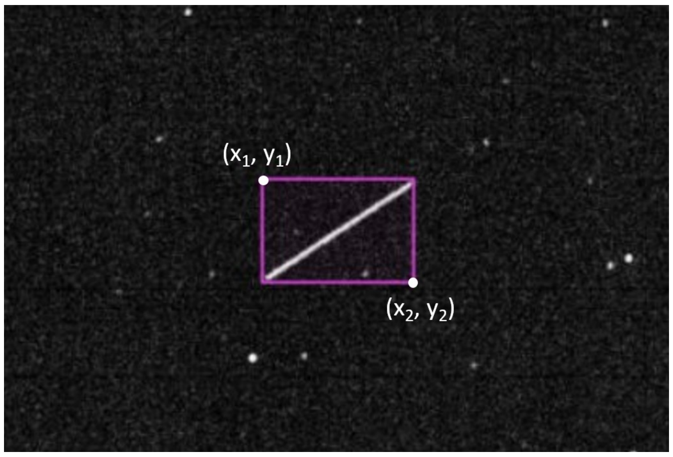


**Figure 5.8** The statistical analysis relevant points of a bounding box are the top left (i.e. the one closest to the origin) and the bottom right (i.e. the one furthest from the origin).

The procedure is now divided into two parts: the calculation of the covariance matrix and the calculation of the Jacobian of the slope and intercept vectors. For each coordinate, it is then possible to calculate the difference between the precise value and the value obtained from the network, and then to square it (the procedure should be extended to all 4 coordinates, for simplicity it is only shown on the $x_1$ coordinate), in particular, $x1_{i_\mathbf{p}}$ is referred to the correct value (obtained through TIG), and $x1_{i_\mathbf{d}}$ is the detected value (inference network output):

$$\begin{aligned} \Delta x1_i &= x1_{i_p} - x1_{i_d} \\ \sigma_{x1_i} &= (\Delta x1_i)^2 \end{aligned} \tag{5.1}$$

Therefore, all that remains is to calculate the total $\mu_{x_1}$ of $x1_i$ using the formula:

$$\mu_{x_1} = \sqrt{\frac{\sum_{i=1}^{N} \sigma_{x1_i}}{N}} \tag{5.2}$$

Where $\mu_{x_1}$ is the average, $\sigma_{x_i}$ is the standard deviation referred to the various $x_1$ analysed, and $N$ is the number of images examinated.

Once the $\mu$s of all 4 coordinates have been calculated, it is possible to compose the diagonal Covariance matrix:

$$Cov = \begin{bmatrix} \mu_{x_1}^2 & 0 & 0 & 0 \\ 0 & \mu_{x_2}^2 & 0 & 0 \\ 0 & 0 & \mu_{y_1}^2 & 0 \\ 0 & 0 & 0 & \mu_{y_2}^2 \end{bmatrix}$$

It remains constant during the calculation of the $\sigma$ of both slope and intercept.

On the contrary, the Jacobian vector varies for each pair of points, and consequently the $\sigma$ values also vary. The following formula shows the slope and intercept functions as a function of the two points considered:

$$\begin{aligned} s = f_s(x_1, y_1, x_2, y_2) = \frac{y_2 - y_1}{x_2 - x_1} \\ i = f_i(x_1, y_1, x_2, y_2) = \frac{x_2 \cdot y_1 - x_1 \cdot y_2}{x_2 - x_1} \end{aligned} \tag{5.3}$$

The Jacobian vector is formed by the derivatives of the functions $s$ and $i$:

$$J_s = \begin{bmatrix} \frac{\partial f_s}{\partial x_1} \\ \frac{\partial f_s}{\partial x_2} \\ \frac{\partial f_s}{\partial y_1} \\ \frac{\partial f_s}{\partial y_2} \end{bmatrix} = \begin{bmatrix} \frac{x_1 \cdot (y_2 - y_1)}{(x_2 - x_1)^2} \\ \frac{-x_2 \cdot (y_2 - y_1)}{(x_2 - x_1)^2} \\ \frac{-(x_2 - x_1)}{(x_2 - x_1)^2} \\ \frac{(x_2 - x_1)}{(x_2 - x_1)^2} \end{bmatrix} ; \quad J_i = \begin{bmatrix} \frac{\partial f_i}{\partial x_1} \\ \frac{\partial f_i}{\partial x_2} \\ \frac{\partial f_i}{\partial y_1} \\ \frac{\partial f_i}{\partial y_2} \end{bmatrix} = \begin{bmatrix} \frac{-x_2 \cdot (y_2 - y_1)}{(x_2 - x_1)^2} \\ \frac{x_1 \cdot (y_2 - y_1)}{(x_2 - x_1)^2} \\ \frac{x_2}{(x_2 - x_1)} \\ \frac{-x_1}{(x_2 - x_1)} \end{bmatrix} \tag{5.4}$$

Finally, it is possible to calculate the $\sigma$ of the slope and the $\sigma$ of the intercept for each pair of points:

$$\begin{aligned} \sigma_s^2 = J_s \cdot Cov \cdot J_s^T \quad &\Rightarrow \quad \sigma_s = \sqrt{\sigma_s^2} \\ \sigma_i^2 = J_i \cdot Cov \cdot J_i^T \quad &\Rightarrow \quad \sigma_i = \sqrt{\sigma_i^2} \end{aligned} \tag{5.5}$$

In conclusion, to identify if two traces refer to the same object, the algorithm calculates the direction of the trajectory by analysing the two centroids, and then checks that the slope and intercept (with the same direction as the centroids) of the most recent object satisfie the empirical rule:

$$\begin{aligned} s_{p_r} - 3 \cdot \sigma_s < s_{new} < s_{p_r} + 3 \cdot \sigma_s \\ i_{p_r} - 3 \cdot \sigma_i < i_{new} < i_{p_r} + 3 \cdot \sigma_i \end{aligned} \tag{5.6}$$

Where $s_{p_r}$ and $i_{p_r}$ are referred to the previous observation object, while, $s_{new}$ and $i_{new}$ to the current target.
If both conditions are met, the detections are assumed to refer to the same object.

## 5.5 LOT structure

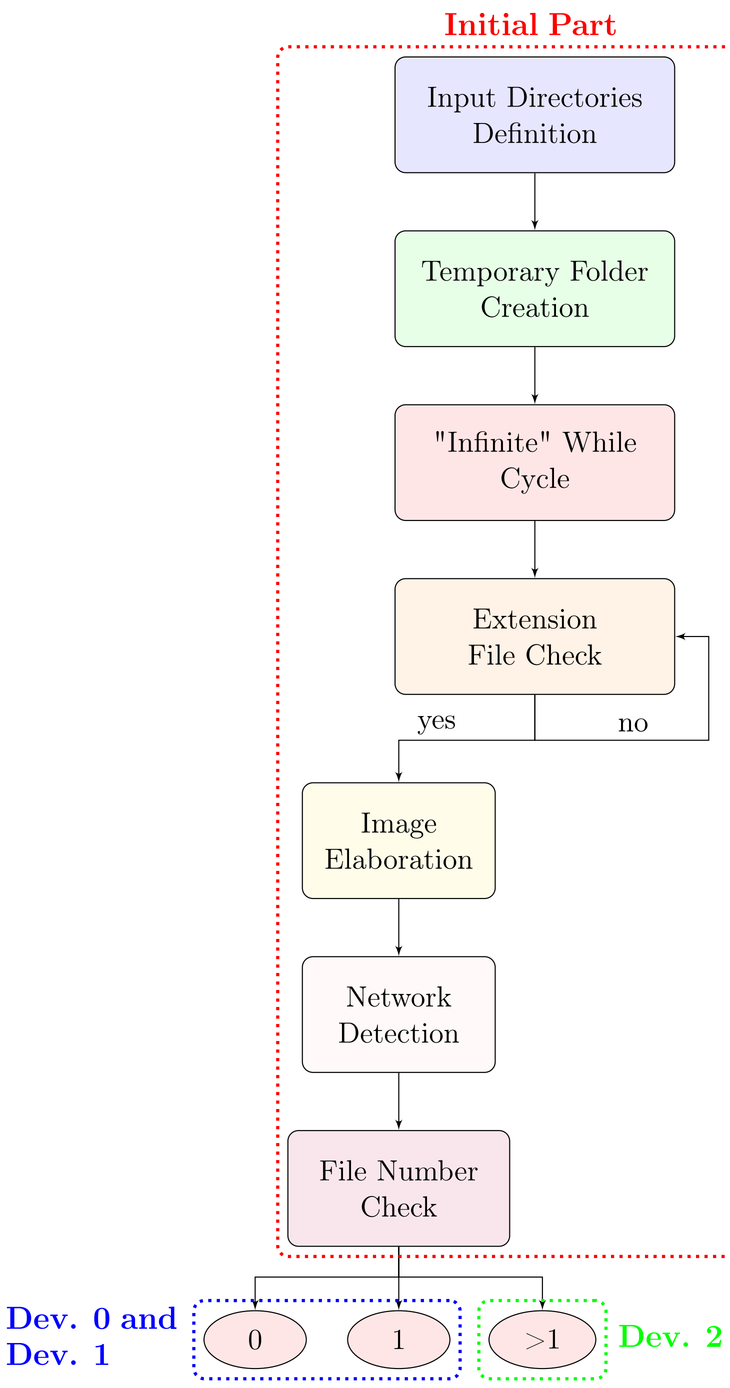


**Figure 5.9** Initial part of LOT architecture.

The implementation of this program turned out to be particularly challenging. Fig. 5.9 shows the application of LOT with a workflow, beginning with the definition of directories and ending with three cases which will be described separately to help the reader.

### Initial part

Once the inputs are defined and the network is loaded on the GPU, LOT is ready to be employed. It consists of an infinite while loop that encapsulates all the tracker processes. It requires a manual stop to allow the algorithm to run through the night without error or failure. "While"

loop will run indefinitely as long as the condition that it is given remains True. To the loop used was given True as its condition, which will never not be true. In addition, the sleep time must be specified, i.e. after how many seconds the script will restart once it has finished. The time is the machine epsilon which is the minimum possible.
Within this loop, a for loop has been added to check if there is a *.fits* file in the input folder (where the telescope observations are downloaded). If not, the algorithm continues to scan for a file with the *.fits* extension. If yes, the newly found observation is processed and converted to *.png* format and then detected by the GPU preloaded network. The observation is then moved to the auxiliary image folder to prevent the system from processing it again. Depending on how many objects are detected, the tracker now has three possible developments (the check is done in the auxiliary text files folder because the detection files and all text files needed by the algorithm are saved there).

### Development 0

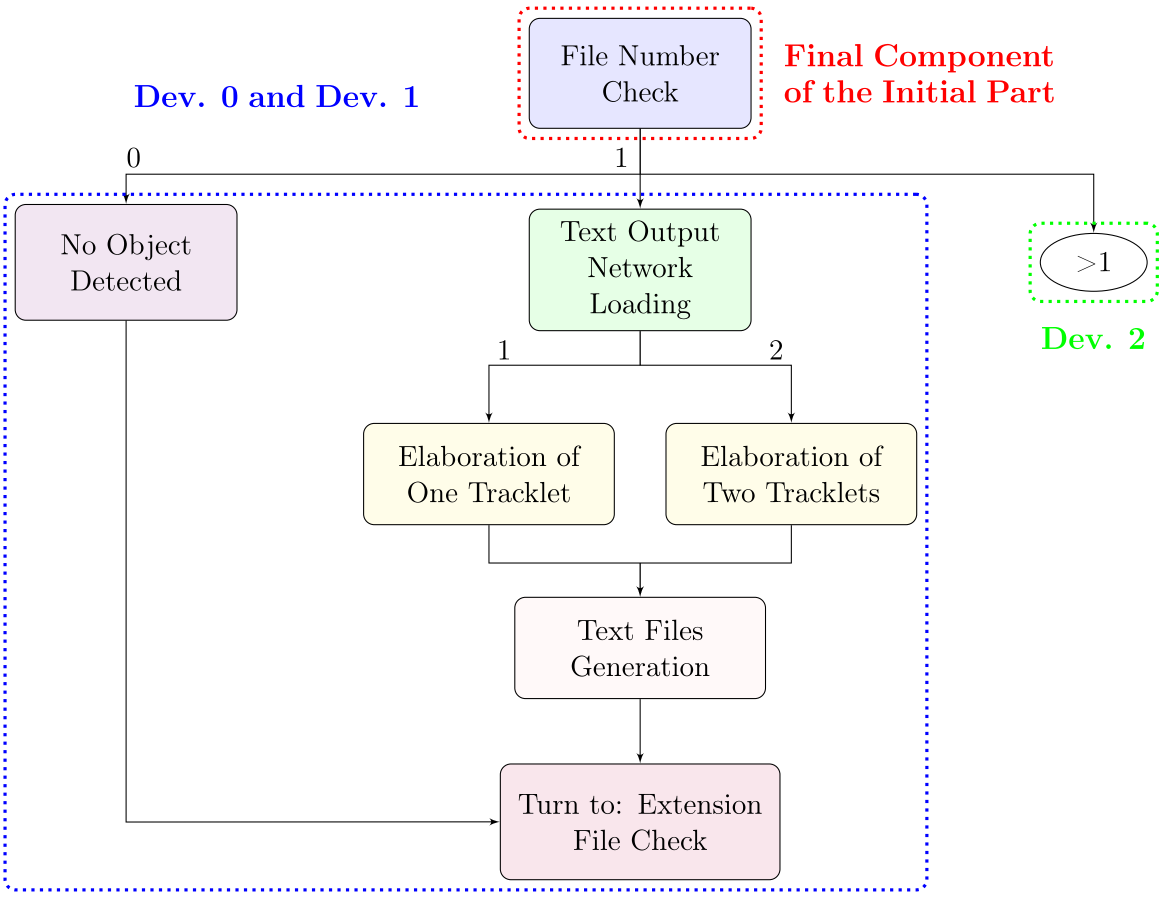

File Number Check
Final Component of the Initial Part
Dev. 0 and Dev. 1
0
1
No Object Detected
Text Output Network Loading
>1
Dev. 2
1
2
Elaboration of One Tracklet
Elaboration of Two Tracklets
Text Files Generation
Turn to: Extension File Check

**Figure 5.10** LOT flowchart for the case with no objects detected, and the case of first object detection.

This is the case when no target has been detected either in the current detection or in previous processing. The auxiliary image folder contains the *.fits* shot and the processed *.png* image, while the text file folder is empty. The script creates a folder named as the pictures, and moves the two observations into it, in order to empty the auxiliary folders for future observations. In this way, the algorithm can resume searching for *.fits* files in the input folder.

### Development 1

The auxiliary text file folder contains one file. This means that only the detection of the current image has identified one or two tracklets. Previous detections have not detected any tracklets (so they have been moved, as described above). Once the network output file is generated in the auxiliary directory, it is processed to calculate the parameters of possible body trajectories through statistical analysis, which are saved in a new text file (two in case of two tracklets) also located in the same auxiliary directory. Besides, a folder named as the observation is created and will be used to transfer the current object files when a new object is detected.

### Development 2

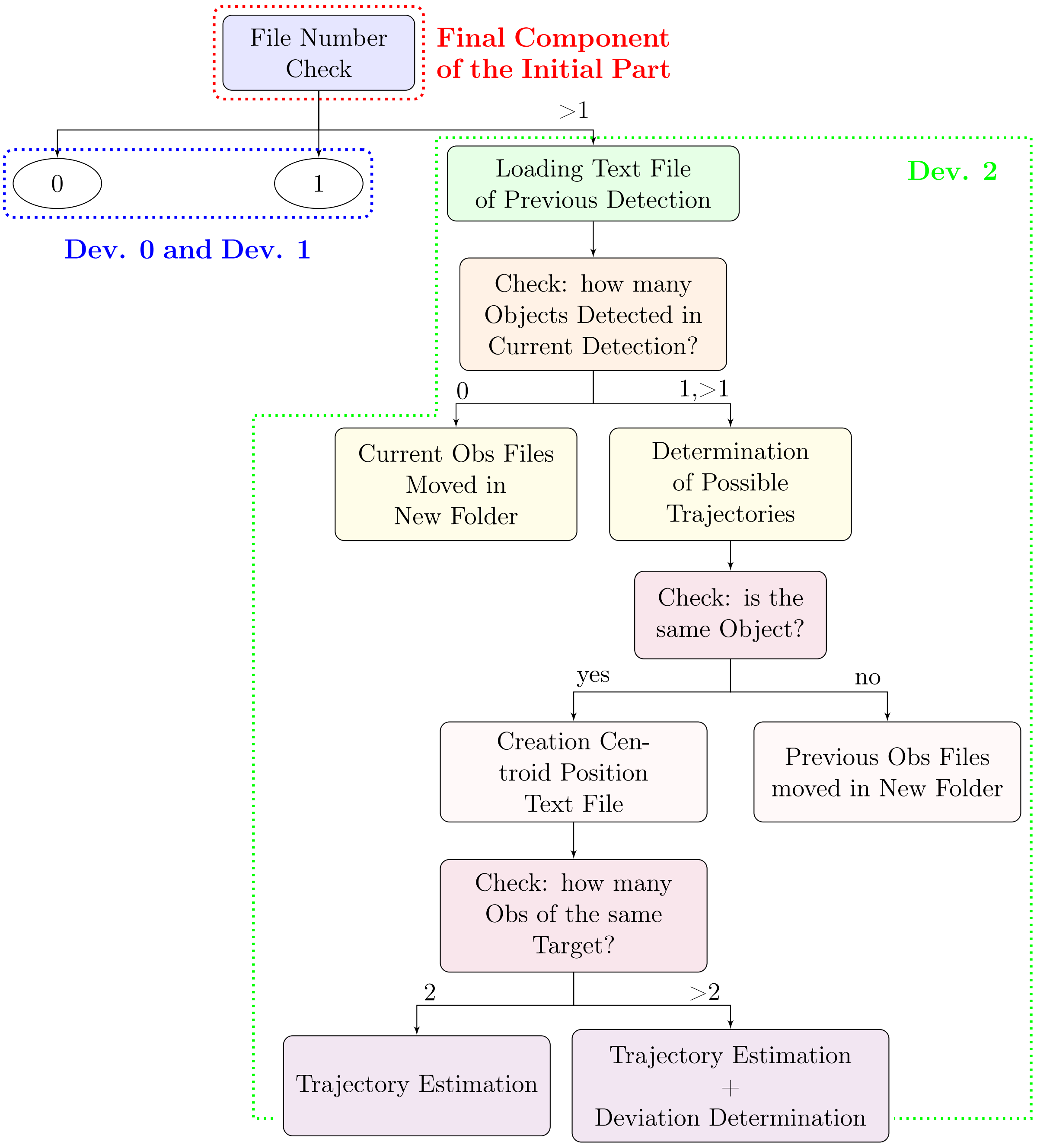

File Number Check
Final Component of the Initial Part
>1
0
1
Dev. 0 and Dev. 1
Loading Text File of Previous Detection
Dev. 2
Check: how many Objects Detected in Current Detection?
0
1,>1
Current Obs Files Moved in New Folder
Determination of Possible Trajectories
Check: is the same Object?
yes
no
Creation Centroid Position Text File
Previous Obs Files moved in New Folder
Check: how many Obs of the same Target?
2
>2
Trajectory Estimation
Trajectory Estimation + Deviation Determination

**Figure 5.11** Flowchart of the main part of the tracker. Thanks to this part, LOT correlates different observations to the same objects.

After the current image has been analysed, the script counts how many files the text support folder contains and determines that there are more than one. This means that in the previous detections (minimum one) there are traces that have been processed by LOT.
Subsequently, the algorithm checks whether the current detection has identified any targets (if the network output text file exists) and there are two possible scenarios:

- The detection has not detected any track, then a folder is created in which the *.fits* file and its processed image are moved, and then LOT continues to search for real observations. The data from the previous observation remains available for comparison in case a new track is spotted.
- The network has identified a target, so it generates a text file containing the characteristics of the bounding box (for simplicity the case with one current track is treated, later the two-track case will also be described). Using this data, LOT calculates the possible rectilinear trajectories of the object and the $\sigma$s of the lines passing through the opposite vertices of the bounding box. At this point, the tracker has all the information needed to check whether the current track and the previous one correspond to the same orbiting body (see Formula 5.6).

  If they are found to belong to the same object: the script saves the position of the centroids in a new text file and predicts the position of the next centroid. If the future position is outside the FoV, the telescope would update its position to maximise acquisitions. From the third detection of the same body on, LOT calculates the deviation of the prediction from the actual position of the centroid. These values are saved in a text file where each line corresponds to a deviation.

  This procedure is also applied in case two tracklets are present in the previous observation, i.e. the trajectory check is done on both tracks. If one of them represents the object of the current observation the procedure continues as described, and the files referring to the other trackets will be moved to a new folder.

  The algorithm works differently if there are two bodies in the current observation and one in the previous one. In this case, no file will be moved because it is possible that in the next observation there are both objects, or only one of them, without knowing in advance which one. The combination of these last two cases explains how LOT reacts if both the current and the previous image present two tracklets. It is able to track both of them in the FoV, and chase the one with a higher probability of detection, i.e. higher confidence values. Besides, at the beginning of each process it creates a folder named as the observation, which will be used to transfer the files of the current object when a new object is detected.

Once the process ends, for example after one night of acquisitions, the tracker will continue searching for *.fits* files automatically. When LOT is interrupted manually, it is necessary to launch one last function to organize the last files into auxiliary folders and then delete them. In this way, the Output folder will be composed of sub-folders divided according to the objects detected.

## 5.6 Algorithm accuray

Tracking telescopes base their observations on the predictions obtained with orbital estimates of catalogued bodies. The sensor is programmed to predict the trajectory described by the orbital parameters of the orbiting body and to acquire images at certain times and certain positions. If the orbital parameters are not accurate the observations may end up to fail. In case an unknown object enters the FoV, the instrument cannot track it to acquire measurements useful to orbit prediction and object cataloguing.

The approach proposed by LOT involves a mix of stationary and dynamic tracking. Once the tracker identifies an object through multiple observations it calculates its future linear trajectory. As long as the target remains in the telescope FoV, it remains stationary. When LOT estimates that the body leaves the FoV it calculates its future straight-line position and updates the target pointing angles of the sensor to capture again the object. The performance of the algorithm was assessed on synthetic images as no adapt real images where available. The TIG software was modified for this purpose.
Before describing the modifications of TIG and the tests performed, it is important to specify the assumptions made about the timing of the telescope at PoliMi:

- downloading a shot from the telescope to the computer on which the tracker is mounted takes about 0.5 seconds. This is a fixed time that always elapses between observations;
- repositioning the sensor after the orbiting body has left the FoV takes about 5 seconds. However, it is a variable quantity directly proportional with distance, but it was decided to consider the worst-case to perform a better analysis.This time is taken into account only when the telescope pointing angles needed to be modified during the tracking phase.

Furthermore, Table 5.1 shows all the times that LOT requires, starting from the download of the image.

| **Process** | **Time (s)** |
|---|---|
| Image Download | 0.5 |
| Image Elaboration | 0.9 |
| Detection | 0.1 |
| Tracker phase | 0.5 |
| Telescope Repositioning | 5 |
| **Total Time** | **7** |

**Table 5.1** LOT Timing.

The time elapsing between the current acquisition and the next one depends on the position of the object with respect to the sensor: it is equal to 0.5 seconds if LOT expects that the next tracklet will be in the FoV, while it is equal to 7 seconds if it expects that the telescope has to be repointed.

## 5.7 TIG modification

TIG was designed to plot tracklets with random thickness, length, and position on random noise backgrounds (see Chap. 2). It has been modified to generate images representing true passages of real satellites (while maintaining random background noise). Thanks to the SCOOP output file, which contains the angular coordinates of $Az$ and $El$ of the chosen satellites, the software calculates the position vectors of the tracklet. These are printed to images with a very high FoV (18°) and resolution (4096x4096 pixels) and then parts of the image are cropped to simulate the acquisitions of a sequence of images woth a telescope with a narrow field-of-view.
Fig. 5.12 (a) shows the full-size image, while Fig. 5.12 (b) is the cropped image that LOT will detect.

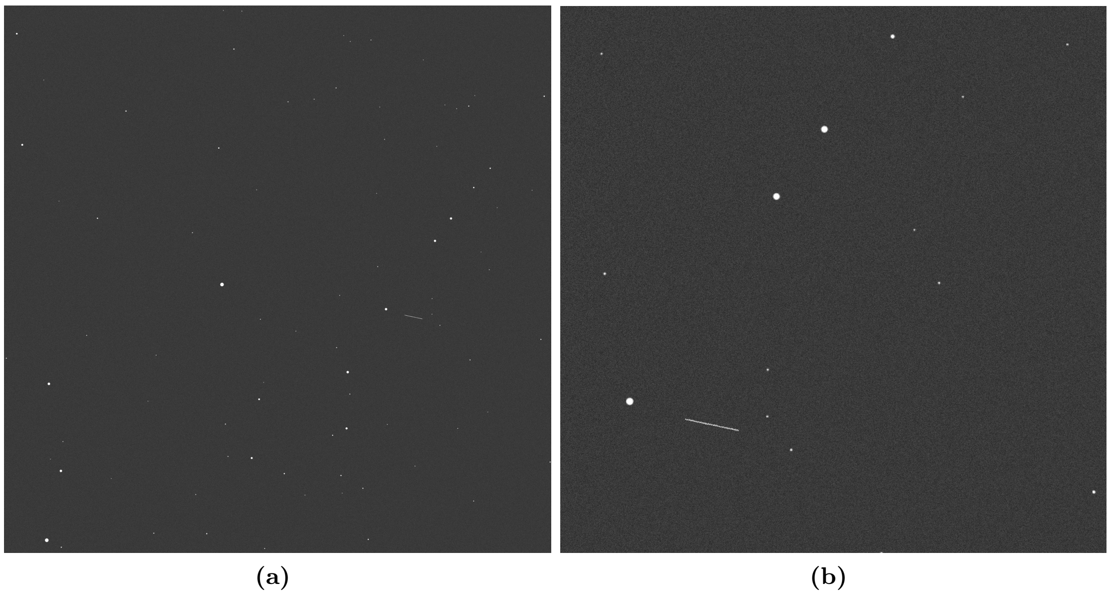


**Figure 5.12** **(a)** is the output of TIG, while **(b)** is the cropped images whit a 6° FoV.

## 5.8 Training phase

Due to less efficient and cleaner processing, the images that compose the dataset do not show disturbances that resemble tracklets. Therefore, the model is better able to distinguish targets from the background in both the one-class and two-class models. In this case, the main problem was the need of very large dataset, about 3100 images, to accurately identify the tracklets.
The directories were organised as described in Chap. 3 and in some cases, manual labelling through CVAT [59] was required.
Fig. 3.6 shows a comparison between an image processed with the method used by the detector and the other one by LOT.
As for the previously described networks, the dataset was divided into three parts: 70% for training, 20% for validation, and the remaining 10% for the testing phase. The images used have a definition of 512x512 pixels to speed up processing and detection. Thanks to the lower definition, the training phase was also faster. As in the previous case, the epochs set were 100, the batch was one and the selected model was the YOLOv5m (the medium one).

## 5.9 Training results

The network achieved excellent results, in particular after 100 epochs the mAP at 0.5 tends to one and the mAP at 0.5:0.95 reaches approximately 0.8, as shown in Fig. 5.13.

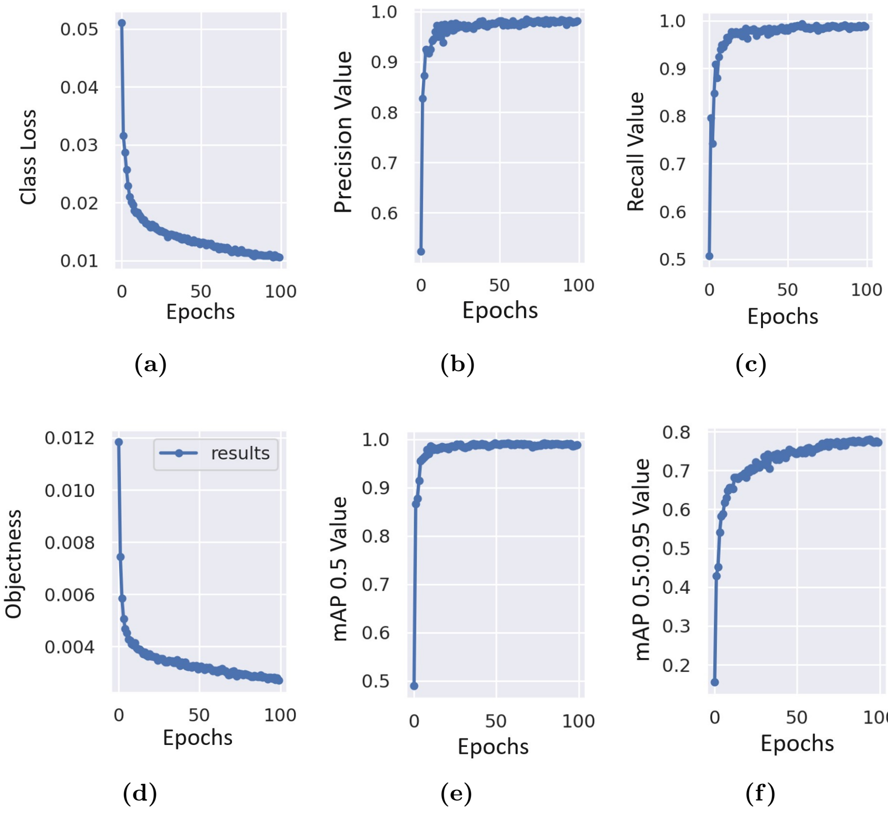


**Figure 5.13** Pyplot network results after 100 epochs training.

Furthermore, by analysing Precision and Recall from Fig. 5.15 it can be seen that the model is able to recognise traces from the background very well (in accordance with the $F_1$ parameter).

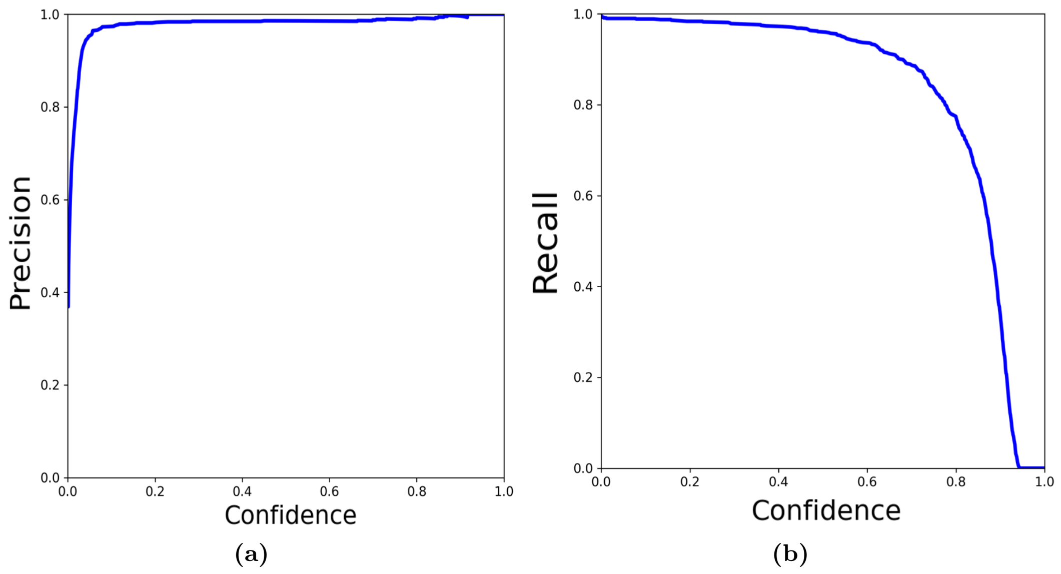
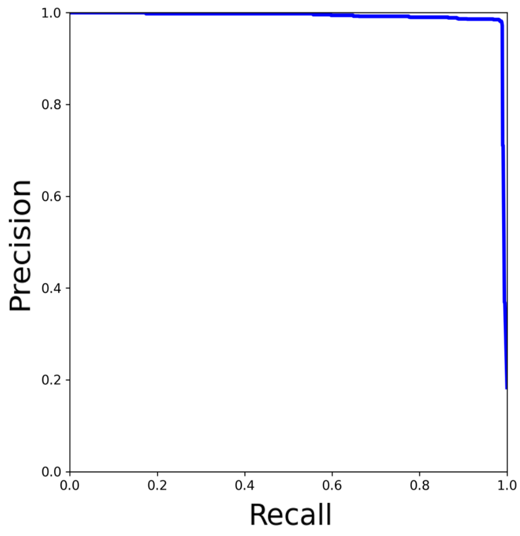
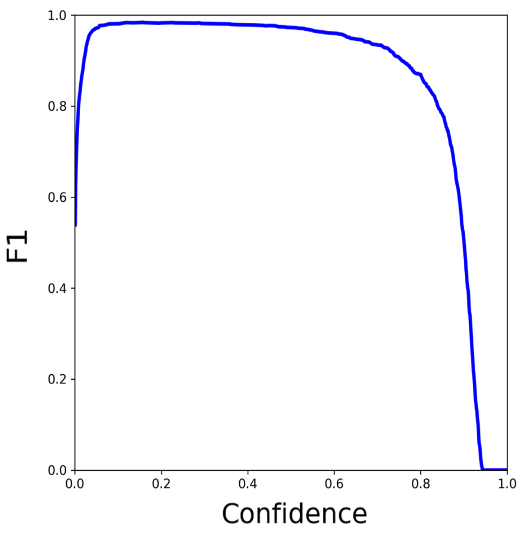


**Figure 5.14** Pyplot based network performance graphs: **(a)** is related to Precision over confidence, **(b)** to Recall over confidence.



(a)



(b)

**Figure 5.15** Pyplot based network performance graphs: **(a)** to Precision over Recall, and **(b)** is $F_1$ graph.

Analysing these graphs, leads to think that this model is better than the one used for the detector in all aspects, starting from the conversion time. On the other hand, it show its weaknesses when it comes to the data preprocessing since waekly illuminated trails do not appear on screen.

## 5.10 Testing phase

The testing procedure is described in Chap. 3. It is based on the remaining 10% of the dataset and the following Table 5.2 shows the network performance:

| Class | Precision | Recall | mAP@.5 | mAP@.5 : .95 |
|---|---|---|---|---|
| **Tracklet** | 0.981 | 0.986 | 0.991 | 0.779 |

**Table 5.2** Pyplot based network testing phase results

In order to have a complete overview of the performance of this model, it was subjected to a comparison with the other networks and the ASTRiDE algorithm through the analysis of 100 *.fits* images, in particular the same ones used in the previous test (see Chap. 4). The outputs were checked manually to be as accurate as possible. As can be seen in Fig. 5.16, the LOT network obtains good results: about 91% of the observations are detected correctly. LOT identifies more tracks than the synthetic model and the ASTRiDE algorithm, but fewer than the RID detector network. The quality of detection drops due to fast image processing (and not to the badly trained model).

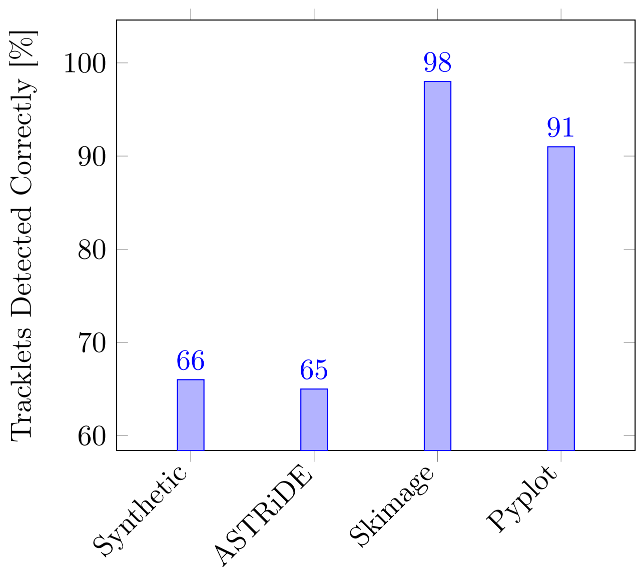


**Figure 5.16** Networks comparison bar chart. The Pyplot-based model achieves worse results than the Skimage-based model due to less effective image processing.

## 5.11 LEO satellites test

The satellites used for the algorithm testing phase are Explorer 7, Echo 1 Deb, Vanguard 1, Vanguard 2 and Solrad 3 Injun 1. Multiple passages of them were analysed to check LOT performances, and they are characterised by an elevation angle ranging from about 5° up to about 50°.
Generally, the exposure time to capture LEO bodies is about 3 seconds. Sometimes, when the elevation angle is very high, satellites travel about 0.5°/s and this means that the tracklet is about as long as half an image. The excessive length of the trace could cause issues in the detection phase if the network is not trained on such long target traces. The most challenging case is when $El$ is large, as the satellite moves very fast and its trajectory is increasingly curved. Therefore, the risk is that the estimated position of the object deviates too much from the real position, so the target leaves the FoV and the tracker fails to detect it again and loses it. The most interesting cases are reported below.

### Echo 1 Deb

Fig. 5.17 shows the passage of the Echo 1 Deb satellite at an elevation of about 25°, and Table 5.3 shows the LOT-processed values of the deviation of image (c), with respect to the prediction calculated using images (a) and (b), and the deviation after telescope repositioning of image (d), calculated from images (b) and (c).

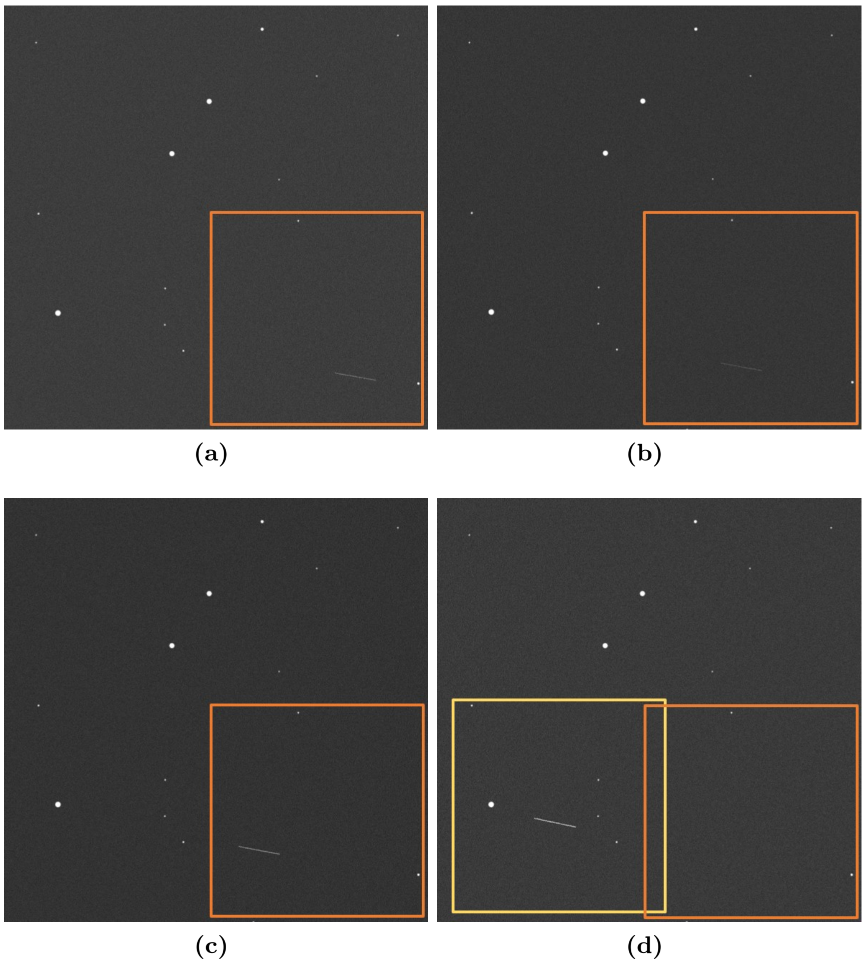


**Figure 5.17** Echo 1 Deb passage, the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

| **Deviation (pixels)** | **Images** | **Δt (s)** |
|---|---|---|
| 2.4927 | $(a)(b) \rightarrow (c)$ | 2 |
| 1.7381 | $(b)(c) \rightarrow (d)$ | 7 |

**Table 5.3** Echo 1 Deb deviation values.

The values show that the deviations are very small, so LOT can track the object by updating the telescope position.

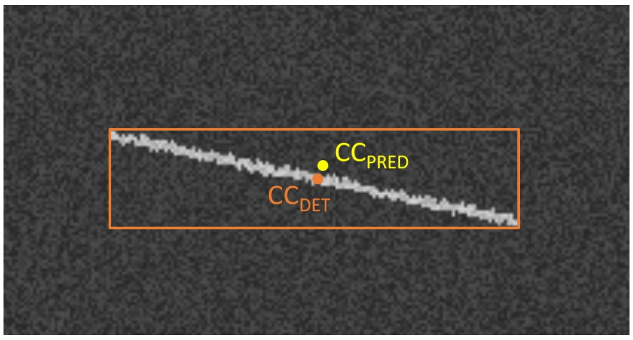


**Figure 5.18** The orange point is the centroid position of the detected bounding box, while the yellow one is the LOT predicted position centroid (this image is a zoom of the Fig. 5.17 **(d)**).

Figure 5.18 emphasises the small centroids deviation through a zoom of the Fig. 5.17 **(d)**.

## Vanguard 1

Fig. 5.20 shows the passage of the Vanguard 1 satellite at an elevation of 35°. As in the previous case, Table 5.4 shows low deviation values and therefore the tracker is able to track the object correctly.

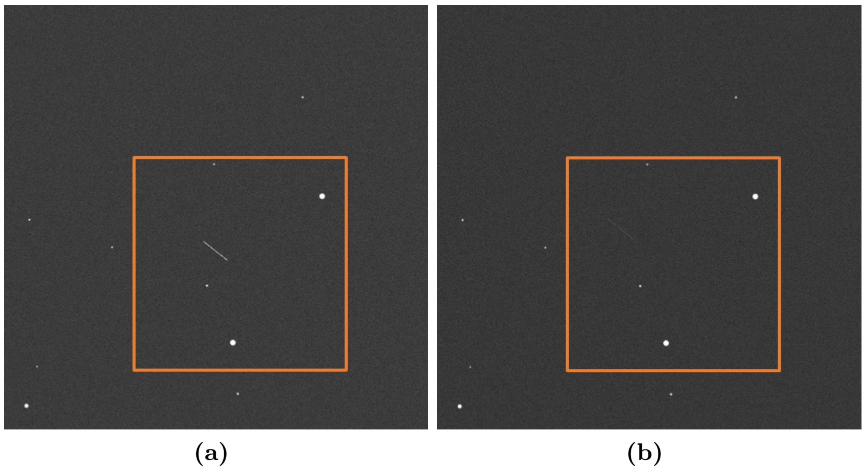


**Figure 5.19** Vanguard 1 passage part 1, the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

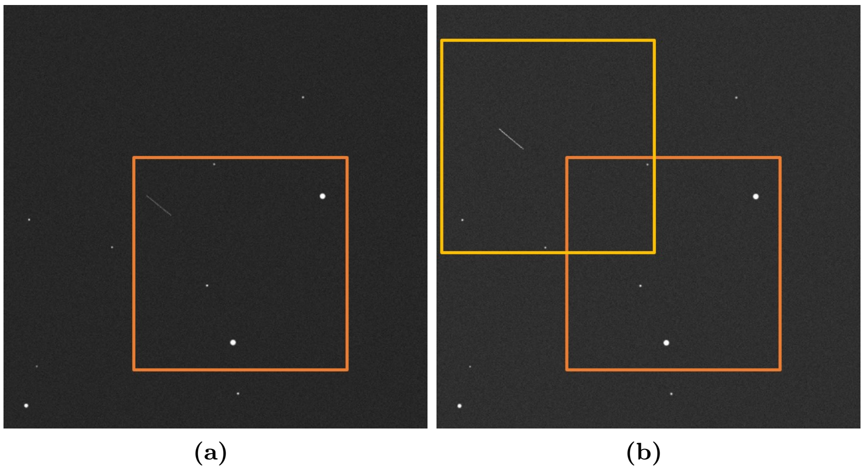


**Figure 5.20** Vanguard 1 passage part 2, the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

| **Deviation (pixels)** | **Images** | **$\Delta$t (s)** |
|---|---|---|
| 0.4196 | $(a)(b) \rightarrow (c)$ | 2 |
| 5.5215 | $(b)(c) \rightarrow (d)$ | 7 |

**Table 5.4** Vanguard 1 deviation values.

### Explorer 7

The tracklets shown in Fig. 5.22 are from the Explorer 7 satellite, with an elevation of about 50°. As described before, the object is very fast and in 3 seconds of exposure time covers about 1.5°. The network had no problems detecting the tracks and the Table 5.5 shows the deviation value. Once again, the algorithm performs well and the linear trajectory estimation reveals to be accurate.

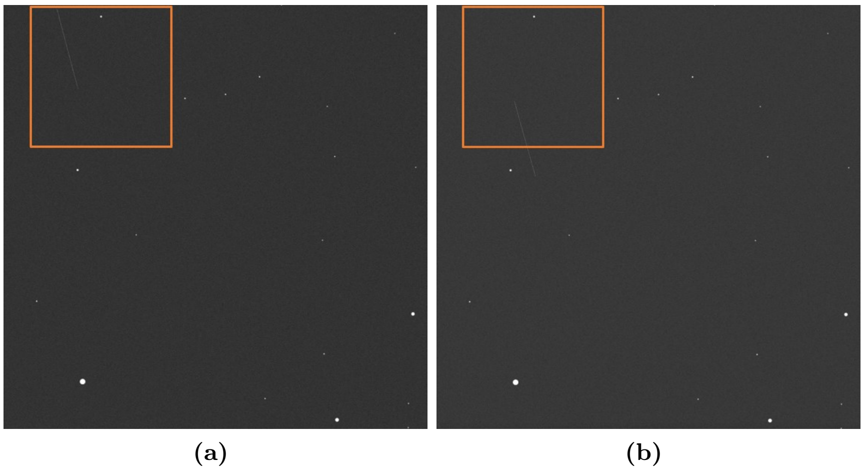
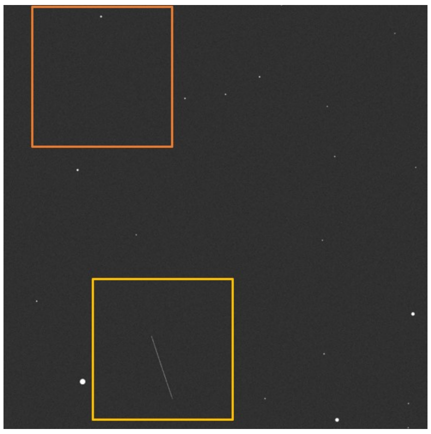


**Figure 5.21** Explorer 7 passage part 1, the rectangles represent 3 degrees of FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.



**(a)**

**Figure 5.22** Explorer 7 passage part 2, the rectangles represent 3 degrees of FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

| **Deviation (pixels)** | **Images** | **$\Delta$t (s)** |
|---|---|---|
| 4.1958 | $(a)(b) \rightarrow (c)$ | 7 |

**Table 5.5** Explorer 7 deviation values.

## 5.12 MEO satellites test

The procedure followed is the same as for the LEO case. The Arthemis P2, Ops 3662 (Vela 3), Ops 6577 (Vela 5), Ops 6909 (Vela 9), SL-12 RB(2) and Galileo FM2 satellites were analysed for the MEO case. The main difference is that these bodies, located on more distant orbits, travel at lower speeds, so the exposure time is about 30 seconds per image.
The most interesting cases are reported below.

### Ops 6577 (Vela 5)

Figure 5.23 shows the tracks of the satellite Ops 6577 (Vela 5) at an elevation of about 50°:

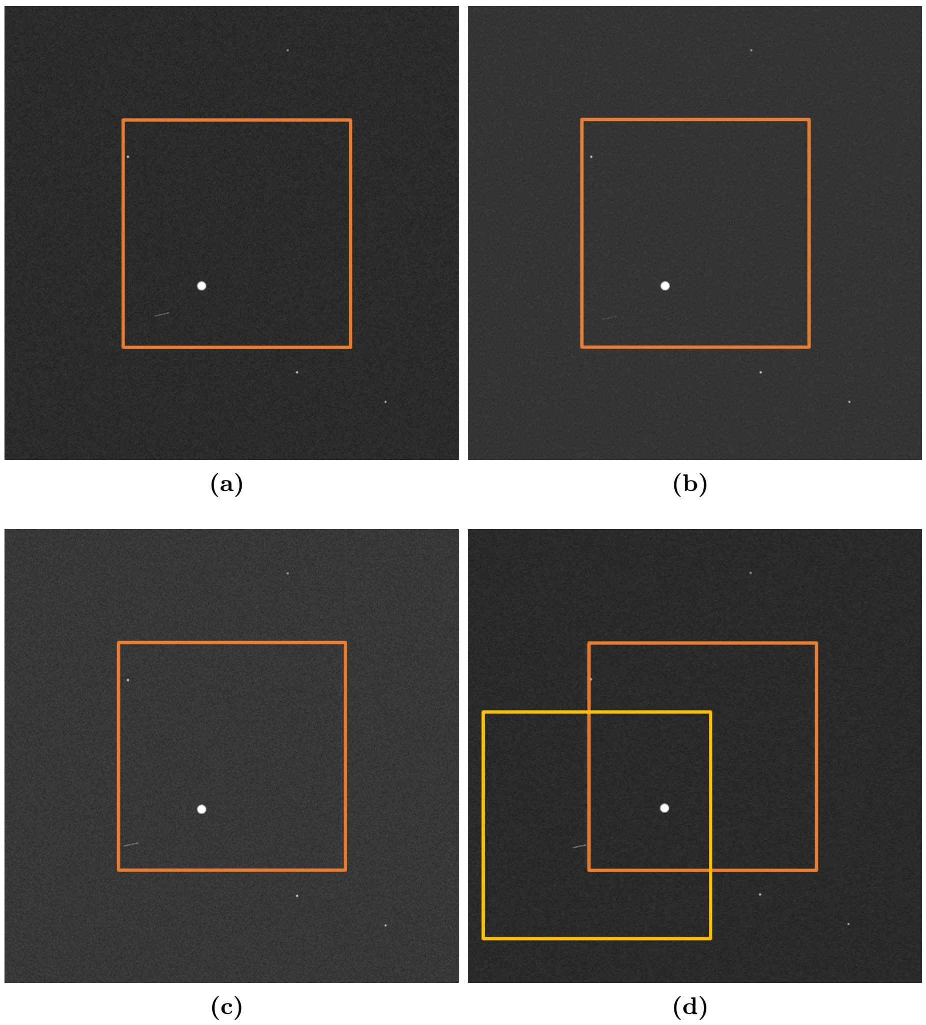


**Figure 5.23** Ops 6577 (Vela 5), the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

The deviation values shown in Table 5.6 are very low, less than one pixel, because the object's trajectory curves very slowly for short periods of time.

| **Deviation (pixels)** | **Images** | **Δt (s)** |
|---|---|---|
| 0.1664 | $(a)(b) \rightarrow (c)$ | 2 |
| 0.7453 | $(b)(c) \rightarrow (d)$ | 7 |

**Table 5.6** Ops 6577 (Vela 5) deviation values.

### SL-12 RB(2)

The case shown in Fig. 5.24 is related to the passage of the SL-12 RB(2) satellite, with an elevation of about 80°. Although the orbiting body is faster than in the previous case (higher El), the deviation values shown in Table 5.7 are similar and very small.

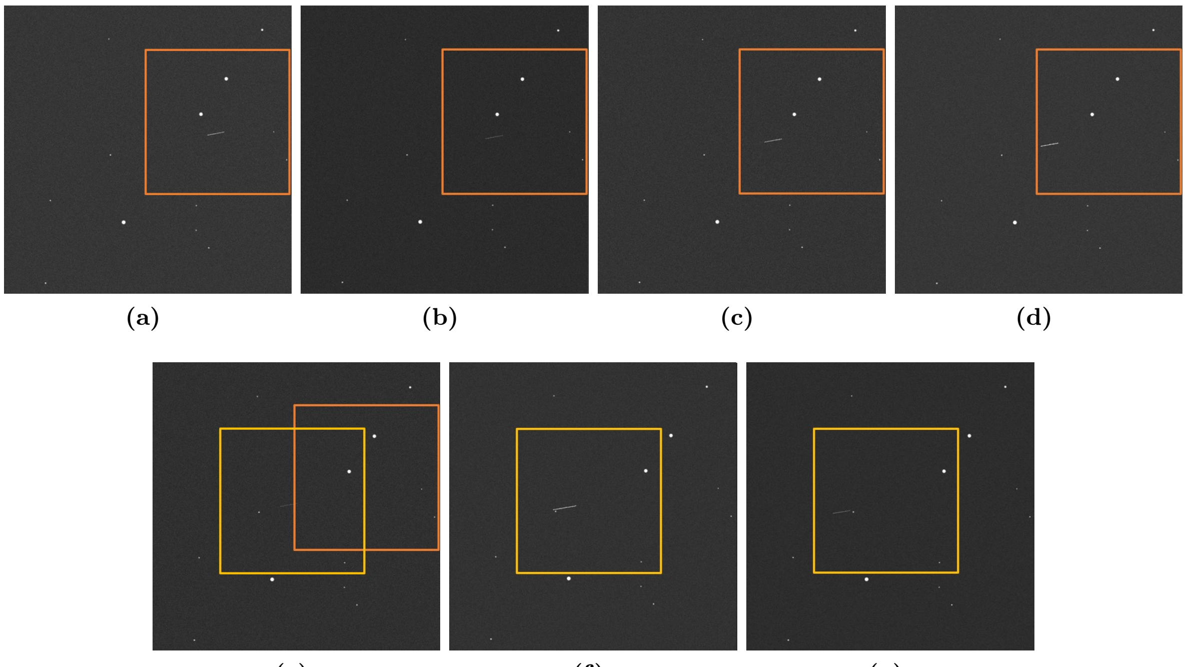


**Figure 5.24** SL-12 RB(2), the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

| **Deviation (pixels)** | **Images** | **Δt (s)** |
|---|---|---|
| 0.7113 | $(a)(b) \rightarrow (c)$ | 2 |
| 0.1435 | $(b)(c) \rightarrow (d)$ | 2 |
| 0.4809 | $(c)(d) \rightarrow (e)$ | 7 |
| 0.6003 | $(d)(e) \rightarrow (f)$ | 2 |
| 0.3801 | $(e)(f) \rightarrow (g)$ | 2 |

**Table 5.7** SL-12 RB(2) deviation values.

### Arthemis P2 (Themis C)

LOT shows more difficulties in estimating the position of satellites with very vertical tracklets. Figure 5.25 shows the Arthemis P2 (Themis C) satellite with an elevation of about 35° and Table 5.8 shows the deviation values which are much larger than in the two previous cases. These values must be compared with the image definition, which is 680x680 pixels, so the tracker will not fail to track this object, and in general, it is capable of tracking objects with very vertical passages.

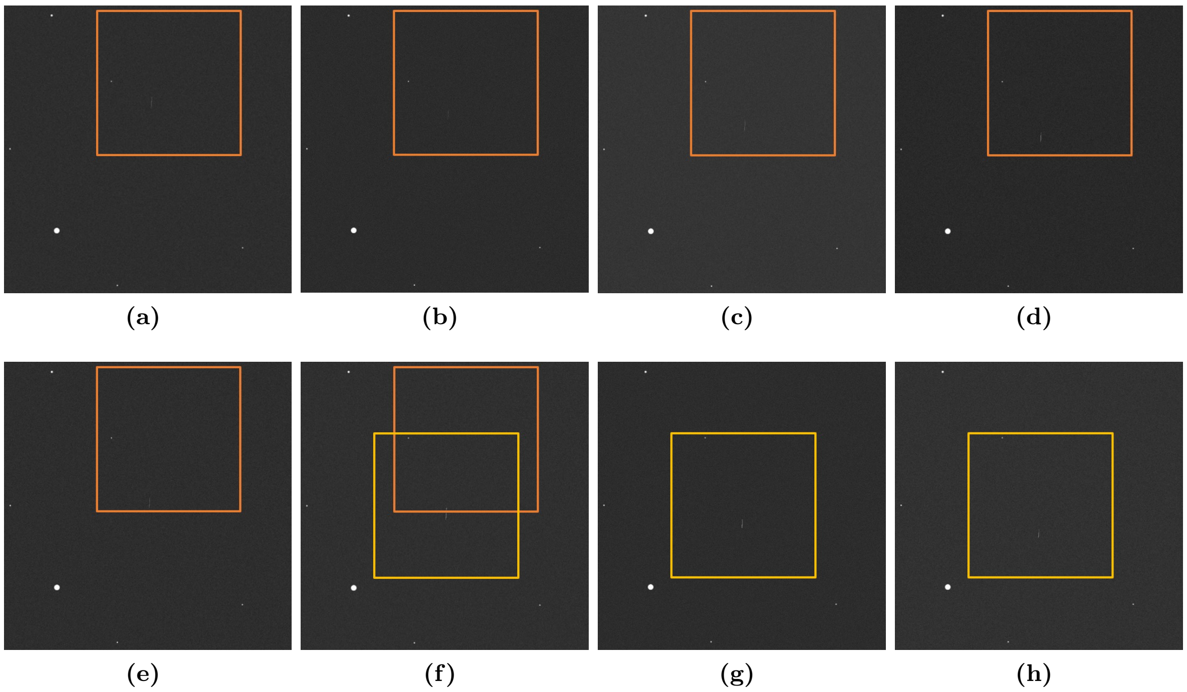


**Figure 5.25** Arthemis P2 (Themis C) passage, the rectangles represent 3° FoV, the orange ones show the first position of the telescope, while the yellow one is after the satellite has left the FoV and thus the position of the sensor has been moved.

| Deviation (pixels) | Images | $\Delta$t (s) |
|---|---|---|
| 25.02 | $(a)(b) \rightarrow (c)$ | 2 |
| 1.026 | $(b)(c) \rightarrow (d)$ | 2 |
| 8.522 | $(c)(d) \rightarrow (e)$ | 2 |
| 6.244 | $(d)(e) \rightarrow (f)$ | 7 |
| 11.33 | $(e)(f) \rightarrow (g)$ | 2 |
| 7.649 | $(f)(g) \rightarrow (h)$ | 2 |

**Table 5.8** Arthemis P2 (Themis C) deviation values.

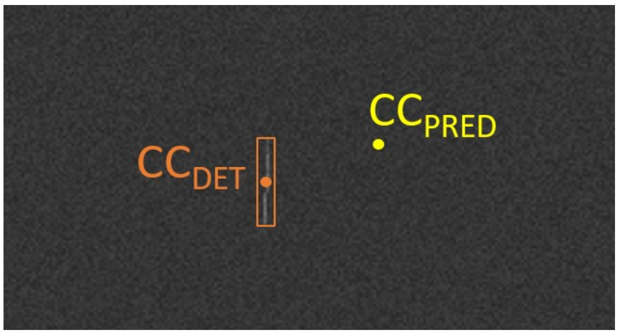


**Figure 5.26** In this case the centroids deviation is greater than the 5.18, due to the vertical tracklets. The orange point is the centroid position of the detected bounding box, while the yellow one is the LOT predicted position centroid (is a zoom of the Fig. 5.25 **(c)**).

Figure 5.26 shows a more distant centroids deviation with respect the Fig. 5.18, through a zoom of the Fig. 5.25 **(b)**.

### Galileo FM2

The latter case is based on a series of real observations made by the Pulsar2 telescope at PoliMi. The telescope was used to track the object Galileo FM2 with an exposure of about 3 seconds and an interval between one acquisition and the next of about 17 seconds. Figure 5.27 shows the passage tracklets and Table 5.9 shows the deviation values and the times between one shot and the next. The values never exceed 2 pixels and refer to images with a resolution of 512x512 pixels.

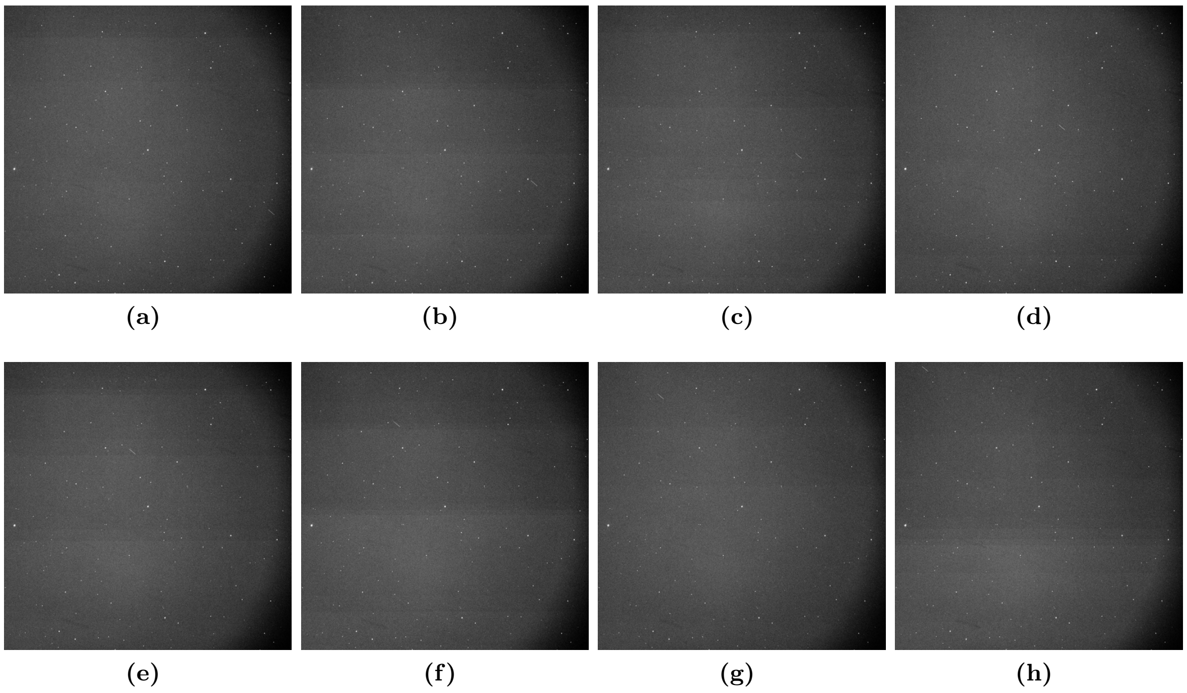


**Figure 5.27** Galileo FM2 passage, each synthetic image covers a FoV of 3°.

| **Deviation (pixels)** | **Images** | **$\Delta$t (s)** |
|---|---|---|
| 1.885 | $(a)(b) \rightarrow (c)$ | 17 |
| 1.527 | $(b)(c) \rightarrow (d)$ | 16.136 |
| 0.6897 | $(c)(d) \rightarrow (e)$ | 16.867 |
| 0.6798 | $(d)(e) \rightarrow (f)$ | 19.001 |
| 0.9443 | $(e)(f) \rightarrow (g)$ | 16.332 |
| 0.9118 | $(f)(g) \rightarrow (h)$ | 16.436 |

**Table 5.9** Galileo FM2 deviation values.

The performances of LOT are very good for the MEO case. The algorithm was stressed in the vertical case but it is still possible to track objects with such passages without any failure.

## 5.13 Algorithm timing performances

LOT consists of several processes that occur sequentially. They can be divided into two groups:

1. hardware-dependent
2. programming-dependent

The first group includes the telescope characteristics, i.e. the download times of the observations and the repointing times, and the network speed, which depend on the computer components. While, the second group includes image conversion and tracker architecture, which depend on the skills of the programmer. He must implement an efficient algorithm to try to reduce the calculation time as much as possible.
By the way, in a real observation campaign few adjustments should be done to improve this performance:

- avoid moving the real *.fits* image from the input folder to not be detected again, because normally .fist files weigh a few Mb and take a few moments to be moved.
- use the algorithm directly on the local telescope's computer. In this way, the download time of the observations would be eliminated and the process would start instantly once the photo is taken.

With these small steps, the time taken by the tracker would be reduced and it would be possible to optimise LOT to make it faster. It is quite clear from Table 5.10 that the observation processing, detection, and trajectory estimation phases take about 1.5 seconds, and represent about 75% of the time without telescope movement, and about 30% with telescope movement.

| **LOT + steady telescopes (s)** | **LOT + telescope respositioning (s)** |
|---|---|
| 2 | 7 |

**Table 5.10** LOT times.

Unfortunately, it is impossible to compare the results with a similar tracker because it is difficult to find one that works in this way and because they would have to be tested on the same computer otherwise it would lead to misleading final considerations.

## 5.14 Conclusion and future developments

As simulations have shown, LOT's ability to track satellites is excellent. For all cases analysed, the tracker never failed to track the object without a priori knowledge on its orbit. and the calculated deviations were always very small compared to the FoV. The algorithm also succeded on vertical cases, for very fast satellites, and even in the case of the actual observations of the MEO Galileo FM2 satellite. As described above, the processing phase of the observations takes a couple of seconds and this allows the software to track the objects assuming straight trajectories. The conversion process used limits the quality of the image conversion and therefore the number of tracklets detected. It could be improved by simply upgrading computer hardware or by implementing the use of graphics cards for Python's main image processing libraries.

| **Quality** | **Performance** |
|---|---|
| *Network* | 91% |
| *Elaboration image* | $0.9s$ |
| *Detection* | $0.1s$ |
| *Tracker* | $0.5s$ |

**Table 5.11** LOT performance

First of all, the behaviour of the tracker should be tested on real observation campaigns. Therefore, all the criticalities related to practical implementation may be better defined and strategies can be modified in order to cope with them and improve the algorithm's capability. A possible future development is to implement LOT with a script that calculates the orbital parameters every time it identifies and tracks a new object. In this way, the orbital estimate can be used to command new pointing angles for the telescope, instead of relying on local linearizations as in the current work. Another accessible future development is to extend the tracking procedure to GEO objects and estimate the trajectory prediction through artificial intelligence to increase the efficiency of the process since the observations are only analysed once.

# Chapter 6

# Conclusions

The observation of space debris has assumed a fundamental role in future access to space, allowing to acquire the knowledge of an increasingly number of orbiting bodies, helping in the fulfilment of SSN and SSA purposes. The space debris number is expected to increase in the next years [70] and this will require a greater effort in space mission planning and satellite maintenance if no action is taken. A network of sensors capable of surveying portions of the sky is fundamental to characterize the population distribution that turns out to be useful when designing a new space mission, and for collision risk assessment that is built upon catalogued objects. What has been proved by this thesis is that not only traditional methods can be used to perform such calculations, but also Artificial Intelligence, and neural networks, in particular, can succeed in this kind of problem too.
As regards the design of the detector, RID ensures a fairly low processing time compared to conventional techniques. The network is efficient and accurate in detecting even the least visible tracks with very limited inference times. Regarding object tracking, an innovative approach was designed for the identification and tracking of orbiting bodies (both catalogued and unknown) based on sequence of successive images. Also, this algorithm relies on artificial intelligence for target detection, obtaining good results. The biggest limitation of LOT lies in the processing of the observations due to the high computational speed required by the algorithm, which degrades the quality. The tests done produced results with a high level of accuracy and show that this new technique can be a valid alternative to conventional ones.
An interesting aspect of both RID and LOT software is that the performance of the algorithms can be improved by augmenting the datasets with new observations for deeper learning. Besides, to reduce computation time the complete algorithm could be implemented through CUDA libraries by moving the onerous computational processes on the GPU.
The space sector is increasingly approaching artificial intelligence techniques intending to automate many of its processes. Major companies and entities, such as ESA, collect "Big Data", which can then be analysed to facilitate and improve certain tasks. This could lead to the development of new techniques both for debris population characterisation collision avoidance and collision risk assessment manoeuvre planning. In the future, these automated decisions could take place onboard satellites that would communicate decisions to ground stations to ensure that there is no interference between the various manoeuvring plans of other satellites [6]. As these intelligent systems gather more data and experience, they get better at predicting how risky situations evolve, meaning errors in decision making would fall as well as the cost of operations. This new kind of approach will become fundamental in a near future, where more and more companies will access to space by means of constellation satellites. In conclusion, Artificial Intelligence, employed both for ground and space applications, can be the means to the next step of automation in the space field.

# Bibliography

[1] E. Kyle. Space launch report: Orbital launch summary by year. (accessed: 01.03.2021). https://www.spacelaunchreport.com/logyear.html.

[2] Cnes space debris website. (accessed: 01.03.2021). http://debris-spatiaux.cnes.fr.

[3] T.G. Roberts. Popular orbits 101. (accessed: 01.03.2021). https://aerospace.csis.org/aerospace101/popular-orbits-101/.

[4] A. Fostier. *Amélioration de la gestion opérationnelle des risques multiples de collision.* CST (Centre Spatial de Toulouser), 2018.

[5] M.F.Montaruli. Collision risk assessment and collision avoidance maneuver planning. Master's thesis, Politecnico di Milano, School of industrial and information engineering department of aerospace science and technology, Italy, 2019.

[6] A. De Vittori R. Cipollone. Machine Learning Techniques for Optical and Multibeam Radar Track Reconstruction of LEO Objects. Master's thesis, Politecnico di Milano, School of industrial and information engineering department of aerospace science and technology, Italy, 2020.

[7] A.Morselli. *High order methods for Space Situational Awareness.* PhD thesis, 2014.

[8] Flohrer, Juhani Peltonen, A. Kramer, T. Eronen, J. Kuusela, T. Schildknecht, E. Stöveken, E. Valtonen, Frank Wokke, and W. Flury. Space-based optical observations of space debris. *European Space Agency, (Special Publication) ESA SP*, 587:165, 07 2005. https://www.researchgate.net/publication/234373506_Space-Based_Optical_Observations_of_Space_Debris.

[9] United space in Europe. About space debris. (accessed: 01.03.2021). https://www.esa.int/Safety_Security/Space_Debris/About_space_debris.

[10] G.Lavezzi. Image processing of multiclass satellite tracklets for initial orbit determination based on optical telescopes. Master's thesis, Politecnico di Milano, School of industrial and information engineering department of aerospace science and technology, Italy, 2018.

[11] J.Africano et al. Understanding photometric phase angle corrections. In *4th European Conference on Space Debris*, volume 587 of *ESA Special Publication*, page 141. Danesy, Aug.2005.

[12] J. Brownlee. A gentle introduction to computer vision. (accessed: 01.03.2021). https://machinelearningmastery.com/what-is-computer-vision/.

[13] J.M. Juan Zornoza J. Sanz Subirana and M. Hernández-Pajares. Automated streak detection for astronomical images. (accessed: 01.03.2021). https://github.com/dwkim78/ASTRiDE.

[14] Y. Guo J. Ye Z. Zou, Z. Shi. Object detection in 20 years: A survey. *Online Article*, 2019. DOI:https://arxiv.org/pdf/1905.05055.pdf.

[15] F. Liu L. Jiao, F. Zhang. A survey of deep learning-based object detection. *Online Article*, 2019. DOI:https://arxiv.org/pdf/1907.09408.pdf.

[16] Z. Zou. Object detection milestones. (accessed: 01.03.2021). https://www.researchgate.net/figure/A-road-map-of-object-detection-Milestone-detectors-in-this-figure\protect\discretionary{\char\hyphenchar\font}{}{}VJ-Det-10-11-HOG_fig2_333077580.

[17] R. Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. (accessed: 01.03.2021). https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[18] F. Li & J.Johnson & S.Yeung. Convolutional neural networks for visual recognition. 2017. http://cs231n.stanford.edu/slides/2017/.

[19] G. Jocher. yolov5. (accessed: 01.03.2021). https://github.com/ultralytics/yolov5.

[20] D. Erhan et al. W. Liu, D. Anguelov. Ssd: Single shot multibox detector. *Online Article*, 2016. DOI:https://arxiv.org/pdf/1512.02325.pdf.

[21] A. Sachan. Zero to hero: A quick guide to object tracking: Mdnet, goturn, rolo. (accessed: 01.03.2021). https://cv-tricks.com/object-tracking/quick-guide-mdnet-goturn-rolo/.

[22] Intel. Opencv. (accessed: 01.03.2021). https://opencv.org/.

[23] S. Al-Rubaye A. Panico, L. Z. Fragonara. Adaptive detection tracking system for autonomous uav maritime patrolling. *Online Article*, 2019. DOI:https://www.researchgate.net/publication/343231247_Adaptive_Detection_Tracking_System_for_Autonomous_UAV_Maritime_Patrolling.

[24] S. Mallick. Goturn : Deep learning based object tracking. (accessed: 01.03.2021). https://learnopencv.com/goturn-deep-learning-based-object-tracking/.

[25] H. Oja M. Poutanen K. J. Donner H. Karttunen, P. Kröger. *Fundamental Astronomy*, chapter 3, pages 47–54, 69–71. Springer, 5 edition, 2007.

[26] Aperture, focal length and focal ratio. (accessed: 01.03.2021). https://blog.optics-trade.eu/aperture/.

[27] M.Massari. Optical instruments ppt presentation. Politecnico di Milano, School of industrial and information engineering department of aerospace science and technology, March 2019.

[28] TELEDYNE DALSA. Ccd vs. cmos. (accessed: 01.03.2021). https://www.teledynedalsa.com/en/learn/knowledge-center/ccd-vs-cmos/.

[29] G.M.D Genio J.Paoli E.D.Grande, F.Dolce. Italian air force radar and optical sensor experiments for the detection of space objects in leo orbit. Amostech, Technical Papers, February 2020.

[30] Officina stellare website. (accessed: 01.03.2021). https://www.officinastellare.com/.

[31] Serg. A. Mangiacapre. Forniti i primi servizi di space surveillance & tracking. (accessed: 01.03.2021). https://www.jeremyjordan.me/semantic-segmentation/.

[32] IBM Cloud Education. Machine learning. (accessed: 01.03.2021). https://www.ibm.com/cloud/learn/machine-learning.

[33] F. Li & J.Johnson & S.Yeung. Lecture 11: Detection and segmentation. 2017. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf.

[34] J.Brownlee. Difference between classification and regression in machine learning. (accessed: 01.03.2021). https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/.

[35] J. Chen. Neural network. (accessed: 01.03.2021). https://www.investopedia.com/terms/n/neuralnetwork.asp.

[36] S. Sharma. Activation functions in neural networks. (accessed: 01.03.2021). https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[37] A.Soares T.Oliveira, J.Barbar. Multilayer perceptron and stacked autoencoder for internet traffic prediction, 2014. page 6, https://hal.inria.fr/hal-01403065/document.

[38] A.Al-Masri. What are overfitting and underfitting in machine learning? (accessed: 01.03.2021). https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690.

[39] J. Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. (accessed: 01.03.2021). https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[40] A. Al-Masri. How does back-propagation in artificial neural networks work? (accessed: 01.03.2021). https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7.

[41] S.Saha. A comprehensive guide to convolutional neural networks. (accessed: 01.03.2021). https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-network\s-the-eli5-way-3bd2b1164a53.

[42] Convolutional neural network architecture: Forging pathways to the future. (accessed: 01.03.2021). https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/.

[43] J. Brownlee. How do convolutional layers work in deep learning neural networks? (accessed: 01.03.2021). https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[44] V.Tatan. Understanding cnn (convolutional neural network). (accessed: 01.03.2021). https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd62\6ee7d4.

[45] A. Al-Masri. How does back-propagation in artificial neural networks work? (accessed: 01.03.2021). https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7.

[46] R. Parmar. Common loss functions in machine learning. (accessed: 01.03.2021). https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[47] J.Jordan. An overview of semantic image segmentation. (accessed: 01.03.2021). https://www.jeremyjordan.me/semantic-segmentation/.

[48] J. Solawetz. Yolov5 new version - improvements and evaluation. (accessed: 01.03.2021). https://blog.roboflow.com/yolov5-improvements-and-evaluation/.

[49] A. Cabras. Design and implementation of an efficient orbital debris detection in astronomical images using Deep Learning. Master's thesis, Università di Pisa, Italy, 2020.

[50] A. Rosebrock. Intersection over union (iou) for object detection. (accessed: 01.03.2021). https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[51] Precision and recall for object detection and instance segmentation. (accessed: 01.03.2021). https://supervise.ly/explore/plugins/precision-and-recall-75278/overview.

[52] Python website. (accessed: 01.03.2021). https://pythonprogramming.net/.

[53] The jupyter notebook. (accessed: 01.03.2021). https://jupyter-notebook.readthedocs.io/en/stable/notebook.html.

[54] M.Goossens, F.Mittelbach, and A.Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[55] Google colaboratory. (accessed: 01.03.2021). https://research.google.com/colaboratory/faq.html.

[56] Norad two-line orbital element set file. (accessed: 01.03.2021). https://ai-solutions.com/_help_Files/two-line_element_set_file.htm.

[57] Kim Dismukes. Definition of two-line element set coordinate system. (accessed: 01.03.2021). https://spaceflight.nasa.gov/realdata/sightings/SSapplications/Post/JavaSSOP/SSOP_Help/tle_def.html.

[58] Giovanni Purpura. Scoop ppt presentation. Politecnico of Milan, Faculty of Space Engineering, February 2020.

[59] Openvinotoolkit - cvat. (accessed: 01.03.2021). https://github.com/openvinotoolkit/cvat.

[60] Flexible image transport system (fits), version 3.0. (accessed: 01.03.2021). https://www.loc.gov/preservation/digital/formats/fdd/fdd000317.shtml.

[61] Multi-extension fits file format. (accessed: 01.03.2021). https://www.stsci.edu/itt/review/dhb_2011/Intro/intro_ch23.html.

[62] L.H Nielsen L.L Christensen. The esa/eso/nasa fits liberator 3. (accessed: 01.03.2021). https://www.spacetelescope.org/projects/fits_liberator/.

[63] scikit-image image processing in python. (accessed: 01.03.2021). https://scikit-image.org/.

[64] Scikit-image module: exposure. (accessed: 01.03.2021). https://scikit-image.org/docs/stable/api/skimage.exposure.html?highlight=exposure#skimage.exposure.equalize_adapthist.

[65] Scikit-image module: filters.rank. (accessed: 01.03.2021). https://scikit-image.org/docs/stable/api/skimage.filters.rank.html?highlight=median#skimage.filters.rank.median.

[66] P. Shivaprasad. A comprehensive guide to object detection using yolo framework — part i. (accessed: 01.03.2021). https://towardsdatascience.com/object-detection-part1-4dbe5147ad0a#:~:text=The%20confidence%20score%20indicates%20how,%3D%20Pr(object)%20*%20IoU.

[67] Cuda toolkit 10.2 download. (accessed: 01.03.2021). https://developer.nvidia.com/cuda-downloads.

[68] Pytorch. (accessed: 01.03.2021). https://pytorch.org/.

[69] M. Galarnik. Explaining the 68-95-99.7 rule for a normal distribution. (accessed: 01.03.2021). https://towardsdatascience.com/understanding-the-68-95-99-7-rule-for-a-normal-distribution-b7b7cbf760c2.

[70] ESA Safety and Security. The current state of space debris. (accessed: 01.03.2021). https://www.esa.int/Safety_Security/Space_Debris/The_current_state_of_space_debris.