ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN VISIÓN ARTIFICIAL

Master thesis

Multiobject tracking using deep learning and tracking by detection

Author: Alexandre Rodríguez Rendo

Tutor: José María Cañas Plaza

Academic course 2018/2019

Acknowledgement

Galician version

En primeiro lugar quero dar as grazas ao meu titor Jose María polo apoio e axuda neste proxecto. Por outra banda, agradecer a amig@s, compañeir@s de estudo, de traballo, de vivenda... tod@s os que me acompañaron e animaron neste camiño.

Aos meus avós, os que están e os que xa non están pero seguen preto, por ser o maior exemplo de humanidade que un pode ser. Sen vós non sería nada do que son agora. Gracias de verdade.

A Cintia por ser á vez psicóloga, amiga e irmá.

Especialmente aos meus pais que me ensinaron a importancia de palabras como traballo, honestidade e valores. A partir do que vós me ensinastes fun capaz de poder aprender moitas cousas mais. Quérovos moito!

Seguramente me falte moita xente por mencionar pero creo que aquí está o máis importante.

English version

First of all I would like to thank my tutor Jose Maria for the support and help in this project. On the other hand, I would like to thank friends, colleagues, work colleagues, flat mates ... everyone who accompanied me and encouraged me on this path.

To my grandparents, those who are and those who are no longer but there are still close, for being the greatest example of humanity that one can be. Without you there I would not be what I am now. Thank you really.

To Cintia for being a psychologist, friend and sister at the same time.

Especially to my parents who taught me the importance of words such as work, honesty and values. From what you taught me I was able to learn many more things. I love you!

Surely I miss many people to mention but I think that here are the most important ones.

Abstract

The visual multiple object tracking is an open problem inside the computer vision community with multiple applications in the industry such as in the autonomous vehicles or in the security field. Many efforts has been made in the past to solve this task, especially for person tracking due to its greater interest.

In the last years, the deep learning techniques have been able to beat the state of the art in tasks such as image classification or object detection in images. Thus, this work has made use of deep learning methods to built a visual multiobject tracking application. These techniques are combined with a tracking by detection scheme to perform the tracking and achieve a good balance between speed and accuracy in the tracking. The final developed software component, named as *dl-objecttracker*, has a mechanism of tracking processing speed measurement that allows for different tracking processing speed regimes and it is also configurable.

Finally, the developed solution has been experimentally validated on the MOT17Det dataset, one of the most well-known datasets of multiple object tracking (MOT).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work is focused in the problem of visual tracking of multiple objects using deep learning and tracking-by-detection techniques. In this chapter, the context of this project is presented. Finally, the main objectives of this thesis and the methodology used to fulfil them are going to be explained.

1.1 Multiple Object Tracking in Computer Vision

The Multiple Object Tracking or *MOT* is an important computer vision problem which continues to attract attention because of its potential in both the academic and commercial spheres. The real-world applications of the multiobject tracking are numerous including human-computer interaction, autonomous vehicles, robotics, video indexing, surveillance or security, among others. The computer vision community have been making big efforts in the past few decades to solve the MOT problem but the task is still open for improvement.

Many autonomous car projects are taking place globally which require solutions to various different problems including to keep an eye to all other moving objects in the area where the car is located (Figure 1.1). The outputs from the tracking module are a basic input for other modules like maneuver planning and trajectory planning. Autonomous vehicles are key in the continuous progress made in tracking and in the computer vision community in general. Many multi-object tracking algorithms have been proposed to solve the problem of real-world traffic monitoring. In these kind of tasks, the algorithms have to deal with complex occlusion situations and difficult object matching.

In an era where human-computer interaction has become particularly important, the hand

is of big interest. Therefore, the object tracking is an important part of this area. For example, it is being used for tracking the hand movements because of its non-intrusive nature compared to other types of sensing which could imply the user to wear gloves, among others. With this tracking we can develop very interesting applications that range from predicting sign language to games like playing "hand ping pong".
One of the most studied tracking areas is the pedestrian tracking, mainly because this particular kind of object can be seen in a large number of applications with commercial potential. As some studies indicate [1], about the 70% of the current research done in MOT is dedicated to pedestrians. The difficulty of MOT lies in various challenging situations that can occur such as variation of the illumination, variation of scale, target deformation or fast motion. Most of this challenges are common to Single Object Tracking (SOT) but MOT also needs to solve two main tasks: determining the number of objects and mantaining its identities over the time.

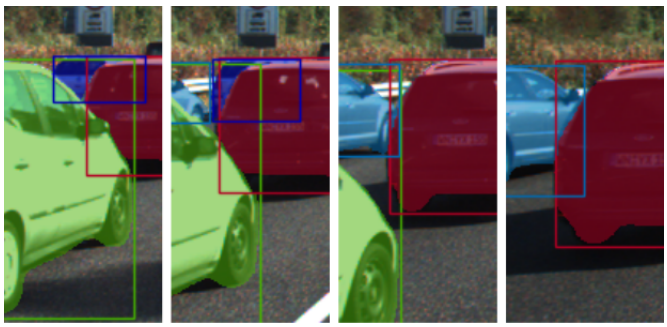

Figure 1.1: The importance of tracking in the autonomous vehicles [1]

1.2 Deep Learning in Computer Vision

Since the birth of Artificial Intelligence (AI) in 1956, the computer vision has followed a great rhythm of evolution. The AI has taking the machines to equal humans in the resolution of some tasks and, in certain cases, to overcome them. Artificial intelligence is defined in [30] as "the subfield of Computer Science dedicated to developing programs that allow computers to present behaviors that can be characterized as intelligent". Machine learning (ML) is defined in [31] as "a field of Computer Science that gives computers the ability to learn without being explicitly programmed". Therefore, given this definition, the ML can be considered a subfield of the AI.

One of the most well known and currently growing ML subfields is called *Deep Learning* [32]. This type of algorithm is intimately linked with the *Artificial Neural Networks (ANNs)* and, in practice, they are usually used in an equivalent way although they are not the same. One of the aspects to be highlighted in the Deep Learning algorithms is that it is no longer necessary to extract feature vectors for the input to the machine learning system. This is because these algorithms "learn" how to represent the data in a hierarchical way. From these networks, the *convolutional neural networks (CNNs)* have a special interest to face the problem that this project presents. This type of networks are characterized by the use of a convolution operation in at least one of the layers of the network and they are designed for the processing of two-dimensional data such as images [33].

In the *Artificial Intelligence* era the multi-object tracking makes use also from the AI to improve the tracking algorithms. These techniques are being used for a broad range of applications such as object detection, image classification, biometrics or medical imaging, among others (Figure 1.2). In most cases, the Deep Learning has beaten the previous State-of-the-Art in these areas.

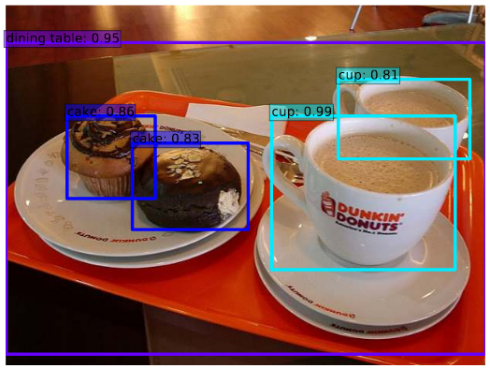

Figure 1.2: Object detection using deep learning techniques [2]

A popular example of image classification is the case of classifying a handwritten digit (multiclass classification) from the MNIST dataset. Numerous applications can surge from recognizing digits and numbers like the automated recognition of house numbers in Google Street View images (Figure 1.3).

Figure 1.3: Digit recognition on SVHN dataset [3]

Apart from the typical deep learning applications, in the last few years other applications that can be tagged as "artistic" have emerged like, for example, the style transfer. The *neural style transfer* consists on learning the style from one or more images and applying that style to a new image. This can give some very interesting results as it can be seen in Figure 1.4. There are other examples of "artistic" deep learning such as image colorization using deep learning. In this case, the neural colorization tries to convert a grayscale image to a color image which can be very helpful in areas like photography or the film industry.

Figure 1.4: Example of neural style transfer from famous artworks to a photograph [4]

Given the broad areas of application where deep learning is succeeding and the ones who are still in research it is going to be studied on this master thesis the use of deep learning techniques to tackle the multi-object tracking problem.

The deep learning for tracking has been used in previous works from other colleagues such as Marcos [34]. In his work, the task is a visual tracking on people using deep learning with a classical feature tracking based on the Lucas-Kanade algorithm [35]. The detections

obtained by the neural networks allow to build a robust hybrid tracker. However, these detections are calculated in an offline way before launching the tracking.

Other interesting works with neural networks, in this case, for object detection have been developed such as *Object Detector* [36]. This application is composed of three modules working as three asynchronous threads. These modules are: a *Camera* that provides the images, a *GUI* that provides the user interface and a *DetectionNetwork* that encapsulates an object detector neural network. As a result this node allows the user to visualize object detections, i.e. bounding boxes drawn over the image in real-time. The images can be obtained from different sources as a webcam, a video or via remote proxy. It also provides functionality to perform on-demand detection. The Figure 1.5 shows the Object Detector running.

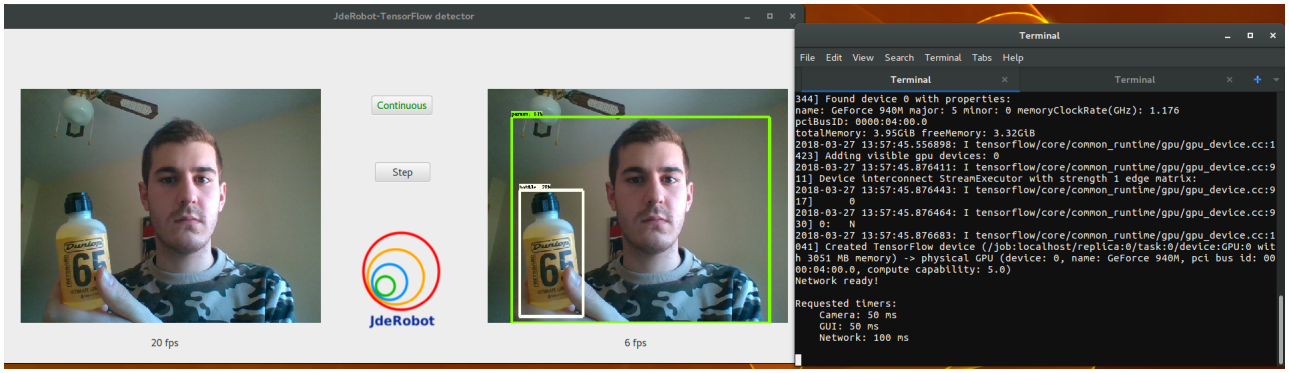

Figure 1.5: Example of the Object Detector node working in real-time (from Object Detector official repository)

The user can download pre-trained open neural network models for *Tensorflow* or *Keras* and configure the Object Detector to work with them.

As it has been seen, the deep learning and the multiobject tracking are still open areas where multiple applications can be obtained.

1.3 Objectives

The main objective of this master thesis is to build a multi-object tracking application which makes use of two techniques: deep learning and 2D-tracking. In this work we are going to study how to use the best of both techniques to build a robust and fast multiobject tracker which can be capable of run in resource constrained hardware on real-time. This work takes the form of a user application and allows multiple configurations. The developed solution will be tested with well-known datasets of multiobject tracking challenges which will provide the performances obtained by each configuration of the application and finally, it will allow the selection of the best configuration.
This task can be divided into different subobjectives:

1. **Development of an object detector using deep learning**
   Learn the fundamentals of object detection using deep learning techniques. Study the performance on both accuracy and speed of these techniques in datasets. Finally, select the default object detector.

2. **Development of a visual tracking module**
   Build the tracking module taking into account the necessity of speed in constrained resources.

3. **Combination of neural object detection and object tracking in a single software component**
   Integration of the modules needed into a thread infrastructure. This will imply a sophisticaded synchronization between them.

4. **Experimental validation**
   Finally, several experiments with state-of-the-art datasets will be performed to validate the developed solution and select the best configuration based on the extracted results.

1.4 Methodology

The following tools have been employed to follow the project progress and making it visible for the community:

- **GitHub repository:** the code of the project is publicly available on GitHub and was constantly updated. The repository can be accessed in the link[1].
- **Wiki:** it has been used as a blog of the progress of the project. In the link[2], the steps followed to achieve this target can be seen.

The development of this project has been weekly followed by the tutor. In this weekly meetings the work done in the previous week was evaluated and discussed ending in new milestones for the following week. This continuous feedback allowed a better development of the project both in terms of understanding the topic in question and also in terms of time management.

[1] `https://github.com/RoboticsURJC-students/2017-tfm-alexandre-rodriguez`

[2] `http://jderobot.org/Arodriguez-tfm`

Chapter 2

State of the Art

In this chapter, we will cover the *state-of-the-art* and the background related to the topic of the thesis. First, the object tracking will be introduced including the algorithms, data-sets and metrics used for a good development of a multiobject tracking system. Second, the object detection will be discussed following the same scheme. Other necessary tools and interesting subjects from the literature will be also briefly commented.

2.1 Object tracking algorithms

The main objective in object tracking is to estimate the state of the object (*target*) over the time in a sequence of images (*frames*). This state can be defined by different *features* such as shape, appearance, position or speed.
It is a difficult field since one or more difficulties must be solved by the algorithm. Among them the management of variations in lighting and in the point of view of the object that can lead to changes in its appearance. Likewise, the occlusions that occur when objects are mixed with other elements of the scene or the quality of the image itself may be a problem.
To confront these problems the following paradigms have been followed [37]:

- **Tracking using matching**: these algorithms make a *matching* between the representation of the model of the object created from the previous frame and the possible candidates in the next frame. These methods rely on the correct representation of the match and the similarity measurement used to perform the

matching. The most outstanding methods are *Normalized Cross-Correlation* [38], *Lucas-Kanade Tracker* [35], *Kalman Appearance Tracker* [39] and *Mean Shift Tracking* [40]. Most of them use the intensity values in the images to build the algorithm, for example, Lucas-Kanade performs spatiotemporal derivatives on these values.

- **Tracking-by-detection**: a model is built to distinguish the object from the background [41]. Once you have one detection it is associated with the previous detections. Currently, the community is turning to neural networks to compute detections.

- **Tracking, learning and detection**: it is an extension of the previous group that includes a mechanism to update the model that is *learned* during execution. For example, you can use the results of an *optical flow* tracker for this update [5]. This ensures that the algorithm is invariant to changes in the object.

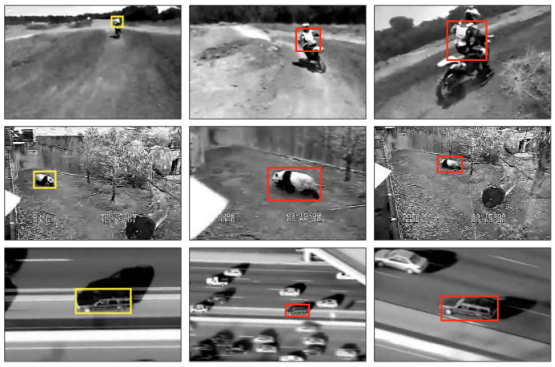

Figure 2.1: P-N learning mechanism (from [5])

Prior to the modern techniques to be discussed here there are more "classic ways" of tracking objects that can be useful in problems that require real time, for example. One of the most well-known is *feature tracking*. This technique uses characteristic points that can be found in images and that allow to estimate the movement. These points must meet some requirements to be able to be characteristic of the image such as *repeatability* (the characteristic can be found in the images even if they have undergone some transformation), *compatibility* (each characteristic must be descriptive and easy to find) or *efficiency* (the representation of the information characteristic of the image must be done with as few characteristics as possible). The characteristic points most commonly

used are *corners*. They are characterized by gradients with higher values in them in two or more directions. These techniques can be seen in Harris [42] and Shi-Tomasi corner detectors [43].

There are tracking systems that take advantage of the speed of feature tracking and the accuracy of neural networks to create a "hybrid tracking". In this type of tracking the detections are done each N frames using some type of neural network and the intermediate tracking is done through feature tracking.

With the arrival of neural networks this way of grouping the tracking methods changes to adapt to them [44]:

- **Tracking-by-detection**: they are designed to follow a certain class of object (*model-based*) and to obtain a specific classifier. In practice, the detections are obtained with neural networks and they are linked in tracking using temporal information. They are limited to a single class of objects.

- **Tracking, learning and detection**: they are characterized by being fully trained *online*. A typical tracker example of this group samples zones close to the object and considers them *foreground*, the same happens with the distant zones that would be assigned to the *background*. With this a classifier can be built that differentiates them and estimates the new location of the object in the following frame [24]. It has been tried to introduce neural networks in environments with online training but due to the slowness of the networks when training the results are slow in practice.

- **Siamese-based tracking**: multiple patch candidates from the new frame are received and the one with the highest *matching score* with respect to the previous frame is chosen as the best candidate, that is, the most similar according to the matching function.

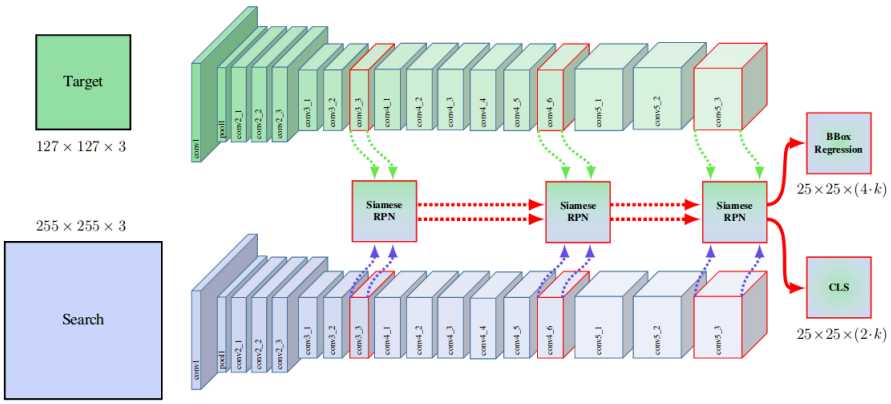

Figure 2.2: SiamRPN++ (from [6])

In the figure above, one of the last siamese network based tracker called *SiamRPN++*. This network, state-of-the-art in visual object tracking on VOT2017/18 and accepted to participate on CVPR 2019, follows the commented strategy which formulates tracking as convolutional feature cross-correlation between a target template and a search region.

- **Tracking as regression**: in this group, on the other hand, the network receives only two images (the previous frame and the current one) and directly returns the location of the object in the current frame. Since this tracker predicts a bounding box instead of just the position, it is able to model changes in scale and aspect of the tracked template. However, it only can process a single target and it needs from data augmentation techniques to learn all possible transformations of the targets [44].

- **Tracking with RNN**: this type of algorithms use *Recurrent Neural Networks* to model the sequence of movement of objects from the detection obtained. Thus improves the response to prolonged occlusions in time, for example [45]. They are the state of the art in tracking nowadays in terms of accuracy but they usually do not perform well in real-time.

2.2 Datasets for object tracking

The visual object tracking is a fundamental task in computer vision which has importance in many applications such as surveillance, autonomous vehicle or video analysis. This task, the same way as others in the field, needs datasets from which create and evaluate the algorithms. The datasets are also commonly associated with competitions that allow the benchmarking of the developed algorithms. These benchmarks often provide the most objective measure of performance and, for this reason, they are important guides for research in the area of study.
The visual tracking datasets can be divided according to their *tracking target*, that is, if they are focused on the tracking of a single object (SOT) or on the tracking of multiple objects (MOT).

2.2.1 Multiple object tracking datasets

- **MOT** [7]

  This dataset arises from the need to provide a general and standardized way to create multi-object tracking algorithms, evaluate the results and present them. In the recent past, the computer vision community has promoted several benchmarks for the evaluation of numerous tasks such as object detection, optical flow or stereo estimation that have advanced the state of the art in these areas. However, not so much effort has been made in the standardization of the evaluation of multiple target tracking.

  As many other datasets it is associated with a challenge, the *MOTChallenge*. With this challenge the organizers try to create a unified framework for the evaluation of multi-target tracking. The dataset provides a collection of datasets, some of them coming from datasets already in use and some from new challenging data. The given data are video sequences.

  The first release of the dataset named *MOT15* was focused on multiple people tracking, following the trend of other datasets. The pedestrian tracking is by far the most studied case in the tracking context. In the next releases, more significant classes generally seen in urban scenarios were added, like vehicles, bicycles or motorbikes. The challenge has had three editions: *MOT15, MOT16, MOT17*. In each of them the sequences were more challenging than the edition before. This

can include different camera viewpoints and positions, more challenging weather conditions (cloudy, night, sunny). For example, the mean crowd density in MOT16 is three times higher when compared to the first benchmark release.

Figure 2.3: An overview of the MOT16 dataset. Top: train sequences. Bottom: test sequences [7]

- **ALOV**[1]

  The *Amsterdam Library of Ordinary Videos for tracking* is another well-known visual object tracking dataset in the field. It aims to cover as diverse circumstances as possible including illuminations, transparency, zoom or low contrast, for example. The dataset consists of 315 video sequences mainly obtained from YouTube with 64 different types of targets. The sequences are normally short with an average length of 9.2 seconds and the total number of frames is 89364 (in ALOV300).

- **CAVIAR** [8]

  The CAVIAR project (*Context Aware Vision using Image-based Active Recognition*) from INRIA labs was dedicated to the development of algorithms that can describe and understand video scenes. The scenes were associated with surveillance scenarios where people performed some different activities related with the surveillance area. Those activities included *walking*, *browsing*, *resting*, *leaving bags behind* or *two people fighting*. The annotations contain, apart from the bounding boxes locations, the head and feet positions, the body direction, among others. Refering to the tracking task, the challenging problems include occlusions, appearance/disappearance, appearance changing or similar object tracking, for example. In terms of data size, the first set contains 28 video sequences and the second set contains 44 video sequences (Figure 2.4). It is a well-known dataset and is commonly used for development and testing of tracking algorithms.

[1] ALOV Dataset

Figure 2.4: CAVIAR: From left to right, two frames (ground truth superposed) from sequences of datasets 1 (entrance lobby of INRIA Labs) and 2 (hallway of a shopping center) [8]

- **BEHAVE** [8]

  Similarly to CAVIAR dataset, the BEHAVE Interactions Test Case Scenarios dataset contains various video sequences with different scenarios where people perform different interactions among which are *walk together*, *meet* or *split* (Figure 2.5). The annotations include labels in case of interactions. Proposed for behavior analysis of interacing groups, this dataset was also used for other purposes like the validation of visual tracking algorithms that consider occlusions or fast and varying motion of objects.

Figure 2.5: BEHAVE: Two snapshots of sequences, with the ground truth bounding boxes of the objects to track [8]

- **PETS** [8]

  The *International Workshop on Performance Evaluation of Tracking and Surveillance* organizes a visual tracking competition with different objectives on every edition starting from 2000. In 2013[2], two of the objectives were the tracking and counting of people in crowds to estimate the density and detecting events by crowd analysis. As BEHAVE or CAVIAR, PETS datasets are very popular among the

[2]PETS 2013

computer vision community. The latest PETS edition took place in 2017[3] and continued the evaluation theme of on-board surveillance systems for protection of mobile critical assets started in PETS 2016 [9]. On this edition, the dataset included sequences that adressed the protection of trucks (Figure 2.6) or vessels at sea, among others.

Figure 2.6: PETS 2016: A group of people detected and tracked walking by a truck [9]

- **TrackingNet** [10]

  Most of the commented datasets are limited by its small size. Even more if they are going to be used by data-hungry trackers based on deep learning. Currently, this trackers rely on object detection datasets due to the lack of dedicated large-scale tracking datasets. For this reason, the authors created TrackingNet, the first large-scale dataset and benchmark for object tracking in the wild. TrackingNet provides a total of 30643 video segments with more than 14 million dense bounding box annotations (Figure 2.7). The contributions of this work include different techniques to generate dense annotations from coarse ones and an extended baseline for state-of-the-art trackers benchmarked on TrackingNet. Referring to the latter, the authors affirm that pretraining deep models on this dataset can improve their performance on other datasets by increasing their metrics by up to 1.7%.

[3]PETS 2017

Datasets	Nb Videos	Nb Annot.	Frame per Video	Nb Classes
VIVID [38]	9	16274	1808.2	-
TC128 [37]	129	55652	431.4	-
OTB50 [4]	51	29491	578.3	-
OTB100 [5]	98	58610	598.1	-
VOT16 [10]	60	21455	357.6	-
VOT17 [11]	60	21356	355.9	-
UAV20L [40]	20	58670	2933.5	-
UAV123 [40]	91	113476	1247.0	-
NUS PRO [39]	365	135305	370.7	-
ALOV300 [17]	314	151657	483.0	-
NfS [36]	100	383000	3830.0	-
MOT16 [16]	7	182326	845.6	-
MOT17 [16]	21	564228	845.6	-
TrackingNet (Train)	**30132**	**14205677**	**471.4**	**27**
TrackingNet (Test)	**511**	**225589**	**441.5**	**27**

Figure 2.7: TrackingNet: Comparison of current datasets size for object tracking [10]

2.2.2 Single object tracking datasets

- **OTB** [11]

  There are several datasets for visual tracking in surveillance escenarios but often the target objects are humans or cars of small size with a static background. Also, some of the scenes are sometimes not annotated with bounding boxes which makes them not very useful for the comparison of tracking algorithms. To facilitate the evaluation task the authors built a tracking dataset with 50 fully annotated sequences in the first release *OTB50*. Later, the dataset was extended with another 50 sequences (*OTB100*).

  Many factors can affect the tracking performance such as illumination variation or occlussion, for this reason the authors categorized the sequences with 11 attributes according to the occurrence of any of the selected factors (Figure 2.8).

Attr	Description
IV	Illumination Variation - the illumination in the target region is significantly changed.
SV	Scale Variation - the ratio of the bounding boxes of the first frame and the current frame is out of the range $[1/t_s, t_s], t_s > 1$ (t_s=2).
OCC	Occlusion - the target is partially or fully occluded.
DEF	Deformation - non-rigid object deformation.
MB	Motion Blur - the target region is blurred due to the motion of target or camera.
FM	Fast Motion - the motion of the ground truth is larger than t_m pixels (t_m=20).
IPR	In-Plane Rotation - the target rotates in the image plane.
OPR	Out-of-Plane Rotation - the target rotates out of the image plane.
OV	Out-of-View - some portion of the target leaves the view.
BC	Background Clutters - the background near the target has the similar color or texture as the target.
LR	Low Resolution - the number of pixels inside the ground-truth bounding box is less than t_r (t_r=400).

Figure 2.8: OTB: List of the attributes annotated to test sequences [11]

Apart from the data side, the authors also integrated most of the publicly available trackers at the time to create a code library with uniform input and output formats to facilitate large scale performance evaluation. Including TLD [5], MIL [24] or CPF [46] making a total of 29 tracking algorithms.

- **VOT** [12]

  The *Visual Object Tracking* initiative started in 2013 to address performance evaluation of short-term visual object trackers. The short-term tracking means that trackers are assumed to not be capable of performing successful re-detection after the target is lost and they are therefore reset after such event. In all the previous editions the challenge considers single-camera, single-target, model-free[4], causal trackers[5], applied to short-term tracking. The main goal of VOT is establishing datasets, evaluation measures and toolkits for visual object tracking (as many other initiatives). The successive editions were made in conjunction with Computer Vision Conferences like ICCV or ECCV. In 2015, a subchallenge focussed on tracking in thermal infrared (*TIR*) was made due to the growing interest in this kind of imaging. The 7th Visual Object Tracking Challenge VOT2019 workshop will be held in conjunction with the ICCV2019. With respect to the previous edition in 2018, this challenge edition introduces the evaluation of trackers that use 4 channels (*RGB-IR* and *RGB-depth*).

[4]The only training information provided is the bounding box in the first frame

[5]The tracker does not use any future frames, or frames prior to re-initialization, to infer the object position in the current frame

Referring to the data itself, the VOT datasets try to pay more attention to the diversity of the data and the quality of the content and annotation with respect to the quantity. For example, some datasets assign a global attribute to the entire sequence when it is happenning in a fragment of it. VOT dataset tries to avoid the assumption that the quality of the data is correlated with its size. The VOT Challenge has focused on developing a methodology for automatic construction and annotation of moderately large datasets from a large pool of sequences (Figure 2.9). For example, they use sequences from datasets such as the OTB.

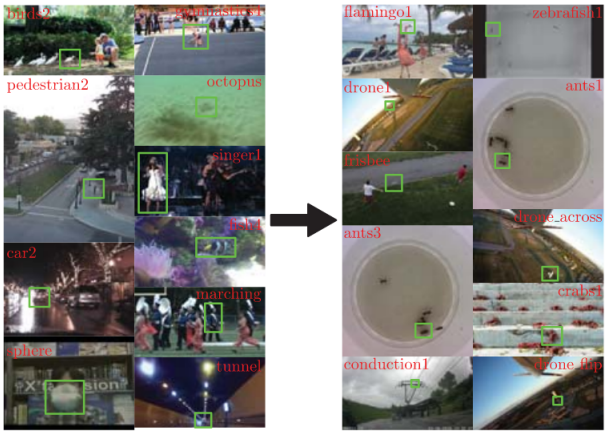

Figure 2.9: VOT: Images from the VOT2016 sequences (left column) that were replaced by new sequences in VOT2017 (right column) [12]

- **Need for Speed** [47]

  Visual object tracking algorithms have been usually evaluated at the canonical frame rate of 30 frames per second but consumer devices with cameras such as smartphones, tablets or drones are increasingly coming with higher frame rates. This can take time for the visual object track community to adapt to what *real time* means in terms of how faster frame rates affect the choice of a tracking algorithm. The authors introduce Need for Speed (*NfS*) as the first higher frame rate video dataset and benchmark for visual object tracking. The dataset consists of 100 videos captured with 240 FPS cameras from real world scenarios. The frames are annotated with bounding boxes and the sequences are manually labelled with nine visual attributes (occlusion, fast motion, etc.). The work also provides a ranking of many recent state-of-the-art trackers according to their tracking accuracy and real-time performance. One interesting conclusion the authors obtained is that at

higher frame rates, simple trackers such as correlation filters outperform complex methods based on deep learning. This must be taken into account when making the choice of a tracking algorithm in practical applications. It needs to be a tradeoff between the resources (available bandwidth, computation hardware, etc.) and the required application accuracy.

2.3 Metrics for multiobject tracking evaluation

Apart from the datasets and algorithms used to solve a given task or problem it is necessary to use a measure or set of measurements that provide an evaluation of the performance of the obtained solution. In this section, the most important metrics used for evaluating *multiple* object tracking are commented.
For the metrics used in this evaluation, the classification from MOT [7] is used as reference. As it will be seen, the performance evaluation for MOT algorithms is not so straightforward as the one presented for object detection.
The metrics for tracking can be classified into four subsets according to different attributes:

- **Accuracy:** this type of metrics tries to measure how accurately a tracking algorithm tracks targets. From this type of metric the following two are briefly commented: *IDs* [48] and *MOTA* [13].
  The IDs metric measures the ID *switches*, i.e. given an id for an object it measures how many times the MOT algorithm changes this id.
  The *Multiple Object Tracking Accuracy* or MOTA is calculated as follows:

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}$$

Figure 2.10: MOTA definition: where m_t is the number of misdetections (FN), fp_t is the number of false positives (FP), mme_t is the number of mismatches (IDs) and g_t is the sum of TP and FN (from [13])
(2.1)

This combination of FP rate, FN rate and mismatch rate into a single number is, as the authors indicate, "by far the most widely accepted evaluation measure for MOT" [7] and it gives an intuitive measure on the tracker's performance at detection and trajectory. It does not take into account the precision of that detections' location.

- **Precision:** in this metrics group the key factor is the description of the precision that the tracked objects have using criteria such as bounding box overlap or distance. The most important are *MOTP* [13], *TDE* [49] and *OSPA* [50]. MOTP, for example, uses a ratio with the distance between the ground-truth detections locations d and the associated detected locations c.

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{2.2}$$

Figure 2.11: MOTP definition (from [13])

- **Completeness:** it refers to how completely the ground truth trajectories are tracked. This set includes the results from *Mostly Tracked (MT), Partly Tracked (PT), Mostly Lost (ML)* and *Fragmentation (FM)* [51].

- **Robustness:** this last type of metrics are linked to the recovering from occlusion. Examples of this group are *Recover from Short-term occlusion (RS)* and *Recover from Long-term occlusion* [52].

2.4 Object classification and detection using neural networks

Much of the progress made in recent years on the classification field of computer vision can be directly associated with the use of neural network architectures. The first big step forward came in 2012 when AlexNet [53] beat all the proposals of the state of the art at that time in the ImageNet challenge, ILSVRC. This competition of classification in images is a reference in the computer vision community. AlexNet obtained a test error rate of 15.3% compared to the previous year's winner error which was 26.2%. This network

follows the basic design archetype of convolutional neural networks: a series of convolution layers, followed by *max-pooling* and *activation* layers before the final classification layers (*fully-connected*). The next architectures are being used as blocks that serve as the basis for numerous subsequent works (commonly known as *backbone networks*) in computer vision and are briefly commented below:

- **VGG** [54]: this architecture from the VGG Group (University of Oxford) makes the improvement over AlexNet by replacing larger kernel-sized filters of size 11 and 5 in the first layers with multiple 3x3 kernel filters one on top of each other. With multiple stacked smaller kernels the depth of the network increases allowing it to learn more complex features at a lower cost.

- **MobileNet**: is a simplified version of Xception [55] for mobile applications that is currently behind the computer vision applications used on Google mobile devices. A year after MobileNet v1, MobileNet v2 was introduced with a great improvement respect to the previous version. For example, the new models used two times fewer operations [14]. In terms of architecture, the main changes are the residual connections and the expand/projection layers in the main building block, the *bottleneck residual block* (see Figure 2.12).

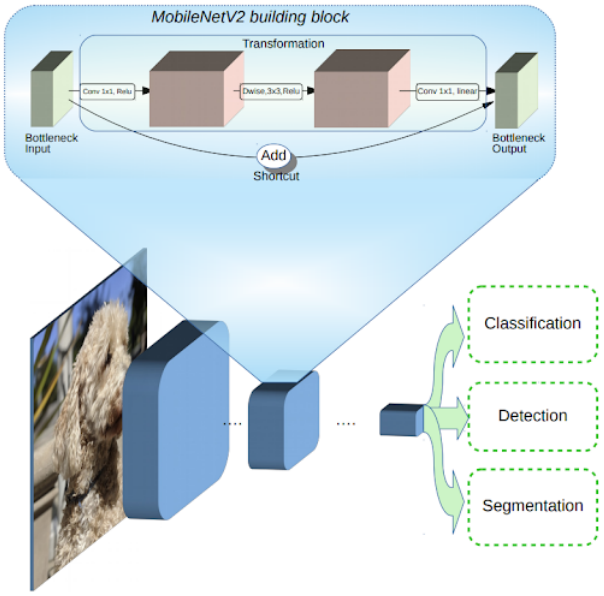

Figure 2.12: MobileNet v2 [14]

- **Inception** [56]: this family of networks looks for *wider* networks, that is, with

more intermediate operations between layers. The authors try to increase neural networks, in terms of operations, without an increase in computational cost. They try to reduce the still huge computational requirements of VGG specially in terms of reducing the number of calculations done due to large width of convolutional layers. Introducing different parallel convolution operations the density of extracted information increases but also the computational costs. To solve the problem they use 1x1 convolutions to reduce dimensionality while performing different transformations in parallel. The resulting networks are simultaneously deep and wide.

The first version of Inception, known as GoogLeNet, was the winner of the ILSVRC in 2014. It was improved later with Inception v2 and v3. The last Inception v4 creates a hybrid with ResNet, known as Inception-ResNet [57].

- **Res-Net** [15]: this network tries to solve the problem that seems to appear when adding layers to a network which is that it generally behaves worse. For this reason, the authors propose that instead of trying to learn the hidden *mapping* of the input x to the function *H(x)*, learn the difference between the two, that is, the residue (*residual net*). The original mapping is recasted into *F(x) + x*. This is a big change at the time as it solves the problem of the *vanishing gradients* that the neural networks have suffered until the date. In addition, it allows to create much *deeper* networks, that is to say, with more layers, that will allow better results.

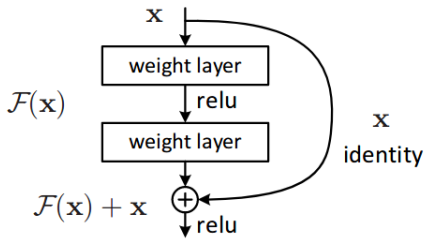

Figure 2.13: Residual learning block [15]

With the arrival of autonomous vehicles, intelligent video surveillance, face detection and numerous emerging applications, faster and more accurate object detection systems are increasingly in demand. This includes not only recognizing and classifying each object in the image but also obtaining the location with its corresponding *bounding box*. This

makes object detection significantly more complicated than traditional image classification. However, the most successful object detection algorithms today are extensions of image classification models. Usually, network architectures such as VGG or ResNet are used as backbone networks as they perform the *feature extraction*. After the backbone, the *head* of the network is stacked. The following object detection models follow the commented scheme [58].

- **Faster R-CNN** [16]: is one of the current reference models and one of the last detectors known as *region-based* from Girshick *et al.* This model basically work in the following way: it uses some mechanism to extract regions from an image that are probably an object and then classifies those proposed regions with a CNN. The father of this model is the R-CNN and it was the real driver of this type of techniques [59]. In the proposed regions obtained through an algorithm called *Selective Search* the characteristics are extracted through a CNN by region and then those regions are classified based on the characteristics. But its performance was slow.
  This performance improves with Fast R-CNN [60] for two main reasons. First, the CNN is applied over the whole image instead of over each region and then the regions are obtained from the last map of characteristics of the network. Second, the introduction of a *Softmax* activation layer simplifies classification. This mechanism was faster and easier to train than R-CNN but there was still a bottleneck in the generation of regions.
  To solve it the RPN (*Region Proposal Network*) is introduced and added to the Fast R-CNN to create Faster R-CNN. The RPN returns proposed regions based on a *score* that refers to the probability that the bounding box is an object, the *objectness* (Figure 2.14). And these regions are passed directly to the Fast R-CNN to perform the classification.

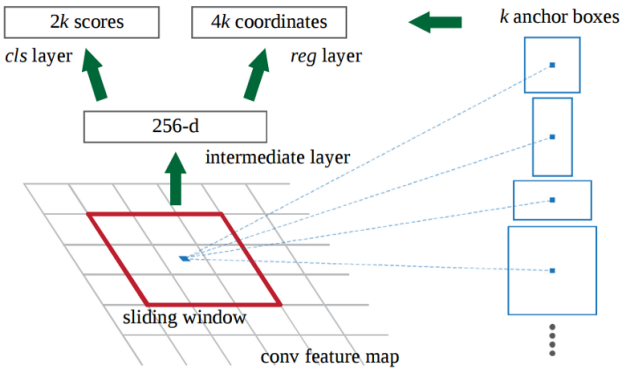

Figure 2.14: Region Proposal Network (from [16])

- **Overfeat** [61]: winner of the ILSVRC 2013 in location and detection of objects, this work showed that training a convolutional network to simultaneously classify, locate and detect objects in images can enhance the success both in classification, detection and location. Subsequently, it has been replaced by SSD and YOLO for tasks that require better performance in real time.

- **SSD** [2]: it provides great speed gains over Faster R-CNN by performing the phases of generating regions of interest and subsequent classification jointly (*Single Shot MultiBox Detector*). As a result you get a lot of bounding boxes which most of them are not useful. By applying the techniques known as *non-maximum suppression* and *hard-negative mining* the final detections are achieved.
  In the MobileNet v2 paper [14] *SSDLite* is proposed which reduces parameter count and computational cost with respect to regular SSD. To do so, the authors replace all the regular convolutions with separable convolutions.

- **R-FCN** [62]: there are faster models than Faster R-CNN such as the *Region-based fully convolutional network* or R-FCN. The authors try to solve the problems of SSD for detecting small objects because the detection in SSD was done on the feature map when features have low spatial resolution. This network tries to improve system speed by maximizing shared computing and provides a good balance between speed and accuracy.

- **YOLO** [17]: this model, also from the "single-shot networks family", uses a different approach with respect to the above. This network divides the image into regions and predicts the bounding box and probabilities of each region. These are then

weighted with the probabilities to obtain the definitive detections (Figure 2.15). This performs, as the authors indicate, a hundred times faster than Fast R-CNN maintaining a similar accuracy.

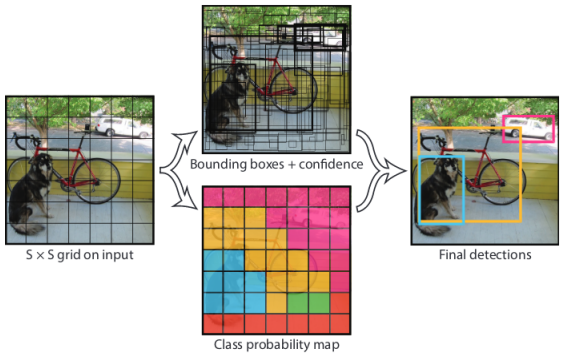

Figure 2.15: The YOLO v1 model (from [17])

The YOLO v2 model introduced big improvements: removed all fully connected layers and used anchor boxes to predict bounding boxes, used batch normalization on all convolutional layers and allowed for multi-scale training, among others. In the next table, it can be seen how YOLO v2 is almost on a par with methods like SSD or Faster R-CNN. However, it has a better balance between speed and accuracy since it manages to work in some cases at 91 FPS (*frames per second*) when Faster R-CNN barely reaches 10 FPS (see Table 3 in [28]).

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Table 2.1: Accuracy comparison in test on PASCAL VOC 2012 (from [28])

The latest version of YOLO, called YOLOv3, achieves a 57,9 mAP on COCO test-dev. The frame rate is lower than the obtained with YOLOv2 (with the same image input size) but it still performs as a state-of-the-art real-time object detection system, according to the author [63].

2.5 Instance segmentation using neural networks

The machine vision community has improved the results obtained in object detection and instance segmentation in a short period of time thanks, in large part, to powerful base systems such as Faster R-CNN. This project will use the detections coming from instance segmentation networks so this type of segmentation is going to be introduced, including some recent instance segmentation models such as *Mask R-CNN*.

The instance segmentation requires the correct detection of all objects in the image along with the precise segmentation of each instance. Thus, each pixel belongs to one of the different categories without differentiating whether or not it is in a particular object. Semantic segmentation differs from the instance segmentation in that in the first the labels are class-aware whereas in the second the labels are instance-aware.

Driven by the effectiveness of the R-CNN family many of the methods proposed for instance segmentation are based on segment proposals where segmentation precedes object type recognition [64]. This has proved to be slower and more inaccurate than if the prediction of object masks and class labels were done in parallel and separately. Li *et al.* propose a system known as FCIS (*Fully Convolutional Instance Segmentation*) [65] that tries to predict the output of a set of position-sensitive channels in a completely convolutional way. These channels perform the tasks of class, bounding box and masks calculations simultaneously which makes them faster. But it shows errors in instances that overlap creating spurious edges systematically (Figure 2.16). Recently, Mask R-CNN [18] arose to solve many of these problems and it has situated itself as a state-of-the-art technique in segmentation of instances (see Figure 2.16)

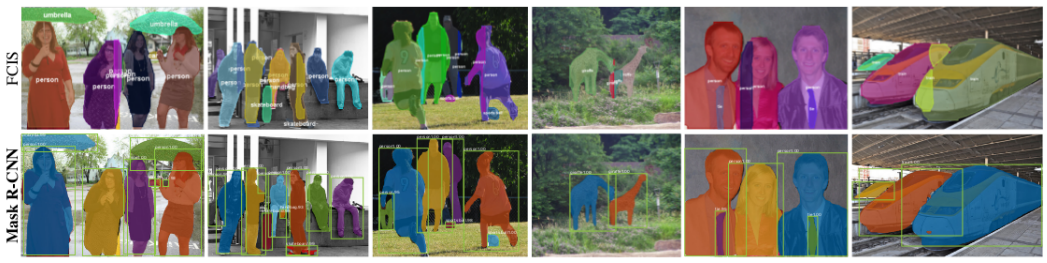

Figure 2.16: Results obtained by FCIS and Mask R-CNN in test images in COCO Dataset (from [18])

Conceptually, Mask R-CNN adds a third stage to Faster R-CNN in which it obtains the mask of the object. The first stage of RPN coincides with that of Faster R-CNN while in the second stage it calculates, in parallel with the prediction of the class and the bounding box, a binary mask for each region of interest (*RoI*). The generation of masks for each class is done without the classes competing with each other, which allows to separate the mask and class predictions from the object prediction. According to the authors, this proves to be the key to obtain good results in the final segmentation.
Another key factor in the proper functioning of this method is the correct alignment between the RoI and the extracted characteristics. This is usually done using *RoIPool* in Fast R-CNN but introduces misalignments if the purpose is to segment rather than classify. This is why Mask R-CNN authors create *RoIAlign*. To demonstrate the generality of the proposed method the authors introduce the mask prediction branch on several existing neural network architectures such as Faster R-CNN with ResNet, for example, and they manage to surpass the winners of the 2015 and 2016 COCO Challenge segmentation, MNC [66] and FCIS [65].

2.6 Datasets for object detection

The datasets used when implementing or testing a certain system are a key factor, since they influence the performance that the system can achieve. They also allow for a comparison of the solution found with respect to others that are part of the State of the Art in the task that is carried out, since they are usually associated with some type of competition. Therefore, it is necessary to correctly choose the dataset or datasets used in a computer vision problem. Here are some of the most well-known datasets used in many object detection applications:

- **COCO (Common Objects in Context)**[6]: it is a large scale dataset for detection and segmentation of objects mainly. It contains 80 categories of objects and 330000 images of which more than 200000 are labeled. It is a dataset widely used between the community and in congresses such as the ICCV[7] (International Conference on Computer Vision).

[6]COCO Dataset
[7]ICCV

- **PASCAL VOC**[8]: this dataset is linked with another challenge, the Pascal VOC Challenges. The organizers ran this competition from 2005 to 2012. This project provides standardised image datasets for object class recognition, segmentation or action classification tasks.

- **ImageNet**[9]: it consists of 14 million images approximately and an average of 500 images per category. It organizes the well-known ILSVRC[10] competition (ImageNet Large Scale Visual Recognition Challenge) of location and detection of objects in images and videos. It is one of the reference datasets in this area.

- **KITTI**[11]: centered in the autonomous driving field, this vision benchmark suite introduces itself as a novel challenging real-world computer vision benchmark. The main areas of interest include 3D/2D object detection, 3D tracking or stereo vision. The type of objects for object detection available are focused in the ADAS field such as car, van, truck, pedestrian or cyclist.

- **Cityscapes**[12]: this dataset focuses on semantic segmentation in urban scenes. It contains 30 kinds of objects, 5000 images labeled with a *fine* label (more precise) and 20000 labeled with a *coarse* label in 50 different cities.

- **OpenImages**[13]: it is a dataset of about 9 million images. This makes it the "largest existing dataset with object location annotations". It also has a bigger number of classes than other challenges as the previously cited COCO and PASCAL VOC, exactly 600 object classes. It must be mentioned that the label distributions are usually *skewed* and with OpenImages it ocurres too. This means that there are many more objects of some kinds than others.

There are many other datasets such as those from research centers like INRIA, MIT or Caltech that contribute to the continuous improvement of the available data.

[8]PascalVOC Dataset
[9]ImageNet Dataset
[10]ILSVRC
[11]KITTI Dataset
[12]Cityscapes Dataset
[13]OpenImages Dataset

2.7 Metrics for object detection evaluation

Before going deeper with the most common metrics in the evaluation of object detection, the basic concepts need to be mentioned. When talking about object detection, the following definitions usually appear:

- **Intersection over Union (IoU)**: also known as Jaccard index, this measure evaluates the overlap between two bounding boxes, a predicted bounding box and the ground truth bounding box. With this definition a prediction can be classified into valid (TP) or invalid (FP). See Figure 2.17.

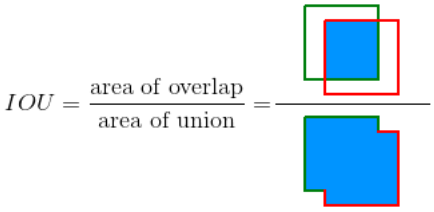

Figure 2.17: IoU definition (from link)

- **True Positive (TP)**: is a correct detection. The condition is that the IoU must be above or equal to a given threshold. This threshold is usually defined in percentage to 50%, 75% or 95%. The results obtained by a system with these three thresholds can define its behavior. For example, a given object detector can easily have good results at a 0,5 IoU but not so easily at a 0,95 IoU.
- **False Positive (FP)**: is a false detection. The IoU of the detection must be below the threshold.
- **False Negative (FN)**: is a detection not detected.
- **True Negative (TN)**: it is not important but it is defined as all the possible bounding boxes that were correctly not detected. It is not used in metrics.

It is very common to see that the metrics are established by a given challenge or associated with it. It is the case of the Pascal VOC challenge that uses the *precision/recall curve* and the *average precision*. These terms are now defined:

- **Precision**: is the proportion of correct positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

- **Recall**: is the proportion of positive predictions with respect to all positives.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

- **Precision/Recall curve**: this curve plots the performance of an object detector as the confidence is changed for each object class. A good precision/recall curve has a high precision while recall increases, i.e. if the confidence threshold varies, the precision and recall stay high.

- **Average precision (AP)**: the AP summarizes the shape of the previous curve allowing to obtain the *Area Under the Curve (AUC)*. This is done because of the nature of the precision/recall curve in form of "zigzags" that does not permit an easy comparative between different curves (detectors). This numeric metric is the precision averaged across all recall values between 0 and 1.

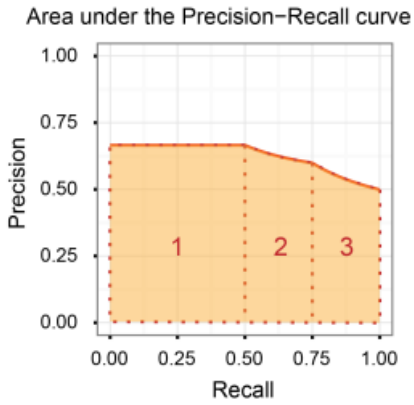

Figure 2.18: AUC example: the areas from the trapezoids are 0,335, 0,15875 and 0,1375 respectively, giving an AUC score of 0,63125 (from link)

This average can be done in two main ways: 11-point interpolation or interpolating all points.

  - **11-point interpolation**: is defined as the mean precision at a set of eleven equally-spaced recall values ranging from 0 to 1 (Equation 2.5). The precision at each recall value is obtained by taking the maximum precision measured value for a method for which the corresponding recall is above r (Equation 2.6). This was the method used in Pascal VOC 2008.

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \rho_{interp(r)} \tag{2.5}$$

$$\rho_{interp(r)} = \max_{\tilde{r}:\tilde{r} \geq r} \rho(\tilde{r}) \tag{2.6}$$

– **All points interpolation**: in this case the mean precision is done interpolating through all recall points (Equation 2.7). The precision at each level r is obtained now taking the maximum precision which has a recall value greater or equal than the recall value at the level $r + 1$ (Equation 2.8). This method of interpolation is used in Pascal VOC metrics from the year 2010 onwards.

$$\sum_{r=0}^{1} (r_{n+1} - r_n) \rho_{interp(r_{n+1})} \tag{2.7}$$

$$\rho_{interp(r_{n+1})} = \max_{\tilde{r}:\tilde{r} \geq r_{n+1}} \rho(\tilde{r}) \tag{2.8}$$

In the next figure, these calculations are presented in a graphical way. With this interpolation the AUC obtained is exact.

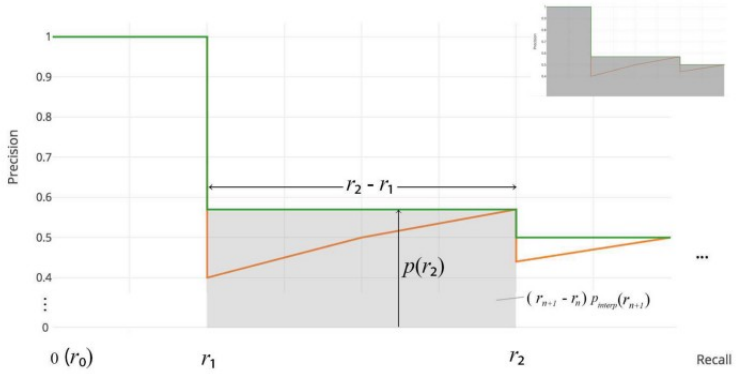

Figure 2.19: All points interpolation (from [19])

2.8 Deep learning frameworks in computer vision

For the use of Deep Learning techniques numerous *frameworks* have emerged. A deep learning framework allows us to build deep learning models more easily and quickly, without getting into all the details of the underlying algorithms. In this section the most important frameworks used for deep learning and computer vision tasks are introduced:

- **Tensorflow** [67]: open-source platform developed by Google, it offers a low-level API that allows complete control over model designs and also a more simplified high-level API with limited functionality. For debugging purposes, it provides the Tensorboard tool which allows for the visualization of the model training, among others.

- **Keras** [68]: provides a high-level API for the use of neural networks. Compared to Tensorflow, it offers a more friendly and modular environment which is very interesting when taking the first steps into the deep learning field. As the official Keras documentation indicates, it "is a model-level library" and "it does not handle low-level operations". For this reason, Keras relies on a optimized tensor manipulation library which serves as "backend engine". It can run on different *backends* such as Theano or Tensorflow.

- **Caffe** [69]: Convolutional Architecture for Fast Feature Embedding was originally developed by the University of California (Berkeley). It supports many different types of deep learning architectures orientated towards image classification and image segmentation. It is written in C++ and provides a Python interface. It is quite common to see this type of architecture in terms of programming languages. Due to the speed of C++ compared to Python, C++ is commonly used for deployment environments whereas Python is often used for quick prototyping because of its user-friendly nature.

- **PyTorch** [70]: is a machine learning library created originally by the Facebook AI research group for the Python programming language. Recently, it has been gaining importance in the "frameworks' battle". This is mainly due to its tensor computing functionality (similar to *NumPy*) that makes the programming easier.

Chapter 3

Used software infraestructure

Before going deeper inside the software implementation made to solve the task of multiobject tracking it is necessary to introduce the software infraestructure needed to accomplish it. The name selected for the component or application built is *dl-objecttracker*. This component is completely written in Python. In particular, the Python version used is the 2.7.12 mainly due to the compatibility with ROS. The component needs a set of parts for its perfect operation which, due to the nature of the task, the majority of them are related to the fields of computer vision, machine learning, deep learning and programming. The following are the most important ones.

3.1 OpenCV

OpenCV[1] (Open Source Computer Vision Library) is a computer vision and machine learning software library originally developed by Intel. It was built to provide a common infraestructure for computer vision applications (Figure 3.1). OpenCV is written in optimized C and C++ and takes advantage of the IPP instructions of the Intel processors which makes it highly efficient. OpenCV is a multiplatform library with versions for GNU/Linux, Mac OS, Windows and Android.

It offers more than 500 functions that provide solutions for areas ranging from the object recognition to the robotic vision. It includes computer vision algorithms from both classic and recent periods (including machine learning and deep learning).

In this work, the version used of OpenCV is the 4.0.1.

[1] OpenCV

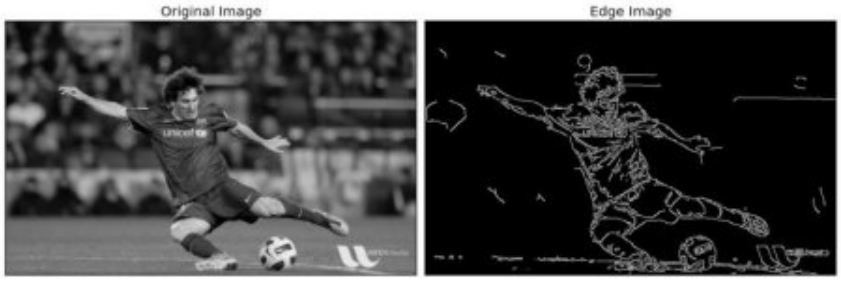

Figure 3.1: Canny edge detection using OpenCV (from [20])

3.2 ROS

The Robot Operating System (ROS[2]) is a framework to write robot software. Its aim is to simplify the task of creating complex and robust robot behavior across different robotic platforms (Figure 3.2). ROS was designed to enhance the collaboration between groups, allowing to build solutions using collaborative robotics.

In this project, ROS is used to interface with standard USB cameras as a source of images with the *rospy* client API for Python. To do so, it is necessary to install the `usb_cam` driver. For more information on the installation process visit my wiki[3]. ROS uses a request and response mechanism which interactuates between them using ROS messages. The `Publisher` publishes a ROS topic which can be seen by a `Subscriber`. This can be used to read from a V4L USB compatible camera (most of the commercial cameras are compatible).

The *rospy* version used is 1.12.12 and the ROS version used is Kinetic Kame.

[2]ROS
[3]Using ROS

Figure 3.2: ROS-based snake robot (from [21])

JdeRobot[4] is an opensource toolkit which facilitates the development in the fields of robotics and computer vision. Mainly written in the C++ programming language it provides a programming environment based on distributed components which can work together in an asynchronous and concurrent way to build applications.

3.3 Deep learning frameworks

This project makes use of the deep learning potential for the object detection task and, for this reason, deep learning frameworks are needed. In particular, the following will be used: *Tensorflow* and *Keras*.

As introduced in section 2.8, Tensorflow is an open-source platform for machine learning and that includes deep learning. It is often seen in deployment environments due to its flexibility allowing for more complex optimizations that are necessary when working in the real world. Keras can run on top of Tensorflow and other backends such as Theano or CNTK. This framework is more focussed on prototyping and fast experimentation.

Both frameworks will be used as building blocks for the neural network module. The object detection models selected in this project are going to run over the Python API of Tensorflow and Keras. The first one is used to prepare the inputs to the neural network models and to get the outputs (object detections) from the Tensorflow pre-trained object detectors (see section 4.4.1). The Tensorflow models are saved using the *pb*[5] format from Google. The Protocol Buffers (commonly known as *protobufs*) define data structures in

[4]JdeRobot

[5]Protocol Buffers

text files that can be loaded or saved using different programming languages. Keras is used to build an SSD-VGG object detection architecture (see section 4.4.2) and obtain the object detections. This framework uses the *HDF5*[6] file format for the data management. The Tensorflow version used in this project is the 1.12.0 and the Keras version running is the 2.1.1.

3.4 dlib library

Written in C++, dlib[7] is a general purpose cross-platform library that follows the idea of component-based software engineering, i.e. a set of independent software components. This toolkit contains machine learning algorithms and tools to solve real world problems in domains including robotics, embedded devices or mobile phones. For example, dlib has an interesting face landmark detector. Landmarks in the face are basically key points in the face that can help in tasks such as person recognition. In the automotive industry landmarks are widely used to perform interior monitoring, i.e. to monitor the driver status. In this project, the dlib library is used to perform object tracking. A correlation filter carries out the tracking using scale pyramid representation to handle large-scale variations in tracking [71]. The version used is the 19.17.0.

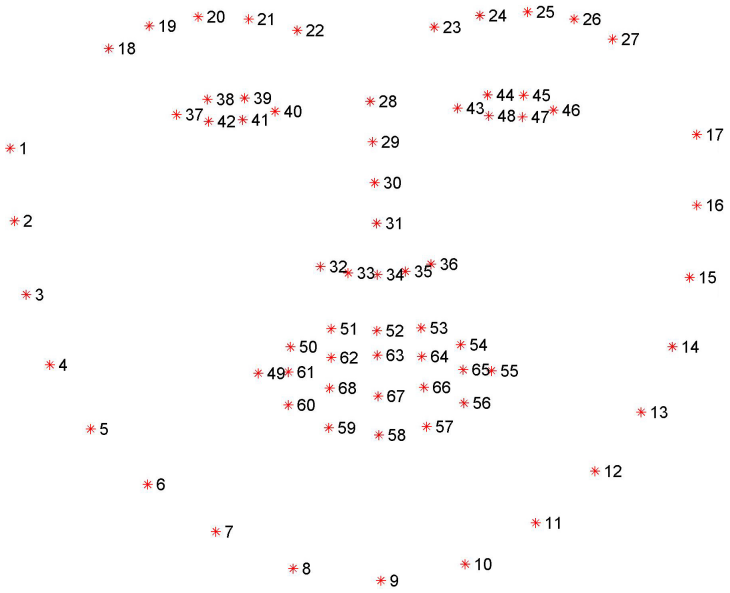

Figure 3.3: Facial landmarks with dlib, from the pre-trained model iBUG300-W [22]

[6]HDF5
[7]dlib

3.5 Object Detection Metrics

To obtain the results or statistics from the project the *Object-Detection-Metrics* tool has been used. This tool written in Python by Rafael Padilla (thanks for sharing) provides the metrics used in the Pascal VOC object detection challenge: Precision x Recall curve and Average Precision.
It offers a simple format to work with results from an object detection application. In this case, it was used to obtain results from an object tracking application. Apart from the metrics it has also options to control the IoU threshold or the bounding boxes format, among others.
For installation and how-to-use instructions please refer to his GitHub repository.

3.6 NumPy and PyQt

Apart from the commented libraries and tools used for the project the following two need to be mentioned.

- **NumPy**
  NumPy is the fundamental Python package for scientific computing specially for working with N-dimensional arrays such as matrixes. Images are basically matrixes at the end so here comes the necessity of this package. The version used of NumPy is the 1.15.4.

- **PyQt**
  PyQt provides a Python interface to the Qt library. Qt is a group of C++ libraries and development tools which include functionality to create graphical user interfaces, networks or threads, among others. In this project, PyQt is used in the *GUI* module allowing the visualization of the results and the user interaction. The PyQt version used is the PyQt5.

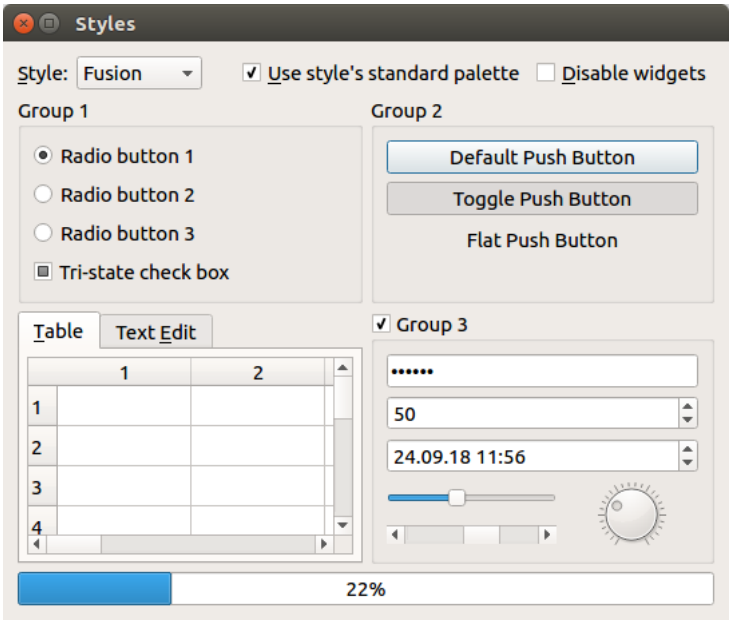

Figure 3.4: PyQt5 example: common widgets (from [23])

Chapter 4

Multiobject tracking using deep learning and tracking by detection

In this chapter, the solution developed for solving the multiobject visual tracking problem using deep learning and tracking-by-detection is explained.

4.1 Design

The main contribution of this work is the development of a visual tracking algorithm capable of tracking different types of objects using deep learning techniques. To achieve this task the selected tracking methodology is the tracking by detection. Our method combines detections coming from an object detection neural network with classic visual tracking techniques. With this combination, the final system provides a balance between speed and accuracy. The detections from the neural networks are usually slower than a pure tracking but more accurate and robust whereas the tracker results are often quick but slightly more inaccurate.

A software application, named *dl-objecttracker*, has been designed and developed which implements the proposed multiobject visual tracking algorithm. The module architecture of this application is summarized in the diagram of Figure 4.1.

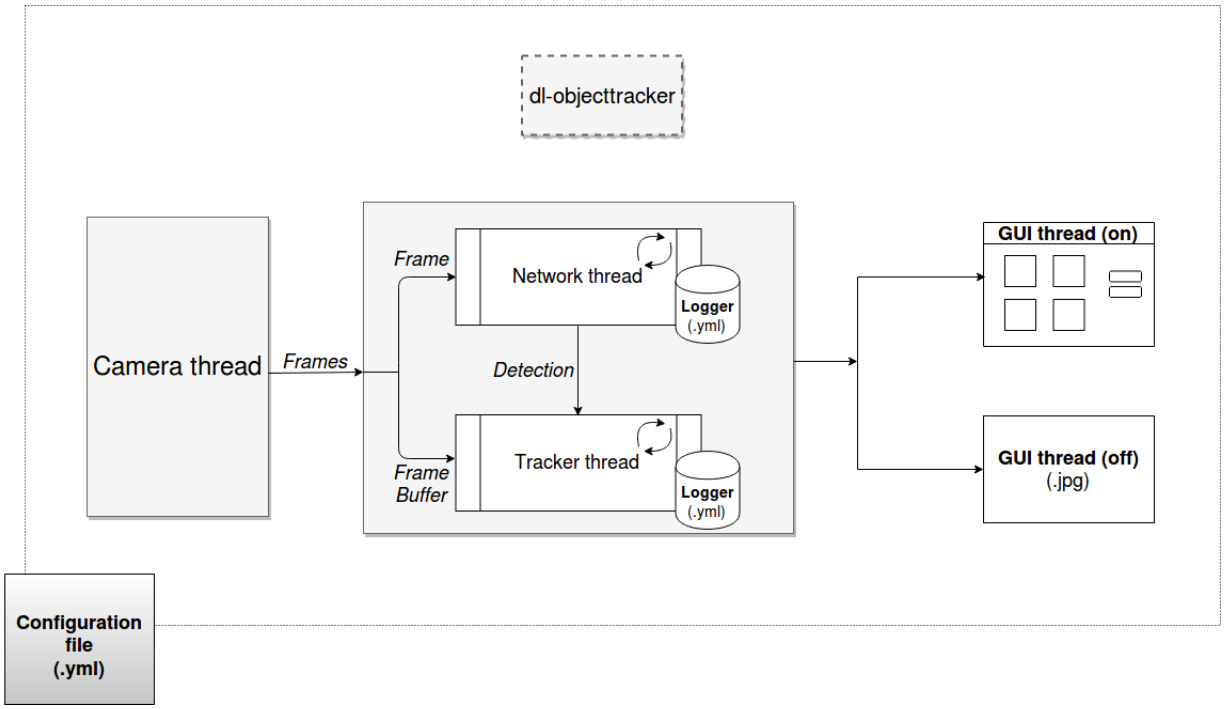

Figure 4.1: Modular architecture of the dl-objecttracker application

As it can be seen the system is built in a modularized way with different threads. There are four threads: Camera, GUI, Network and Tracker. All of them are going to be discussed more in depth on its corresponding sections of this chapter but the general workflow of the system is going to be explained here (Figure 4.3).

First, the camera thread provides the images or frames to the rest of the threads, i.e. it is in charge of the input to the system. The output of the system can be provided using a mechanism of *logging* of the results per frame or using the GUI. If the GUI is configured to show the graphical interface (*on*), the results are shown on the screen. But if the graphical interface is not configured (*off*) the results are saved in JPG files.

The core of the computing is divided into the Network thread and the Tracker thread. This Tracker has a buffer of frames of different size coming from the Camera to work on *delayed real-time*. So, when the first frame is available it is given to the Network thread which starts doing the inference, i.e. it starts detecting objects. Meanwhile, the buffer is accumulating the incoming frames from the Camera until the detection from the Network comes.

When the neural detection is available, the Tracker thread starts the tracking of the detected objects inside the buffer of frames. The last frame in the buffer is used to feed

again the Network allowing for a synchronism between the neural detections and the tracking in the frames. This timing can be observed more clearly in the Figure 4.2.

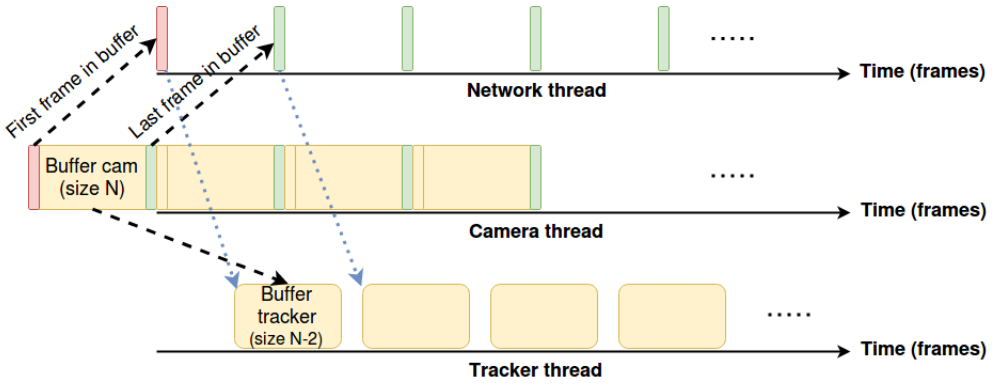

Figure 4.2: How the buffer is handled

It can be seen that given a complete buffer in Camera of size *N*, the first frame is assigned to the Network and, when this frame is processed, a buffer of size *N-2* is assigned to the Tracker to be processed. The last frame of the given buffer is passed to the Network again. This iterative mechanism continues in time ensuring that no frame is lost or ignored while the neural processing takes place at its own pace, which can be much slower than the input frame rate. After the first iteration, the length of the size passed to the Tracker is *N-1*. The application has always a delay in output frames of 1 buffer (*delayed real-time*).

As commented before, the buffer changes its size in every iteration but we can not allow the buffer to increase or decrease this size in an uncontrolled way because that will end blocking the application. If the buffer is too big, the tracking will take more time and the neural network will finish before the tracking is done. This is, the Network is underused. In the other side, if the buffer is too small the Tracker will end its work before the inference is done in the Network so the Tracker will have to wait much more time to the Network to finish (Tracker underused). For these reasons a balance is needed.

The neural network inference time is approximately always the same so this time is taken as reference. Then, the only part of this processing core where we can change is on the Tracker side. The obtained solution consists of a Tracker which constantly measures its frame rate (FPS) allowing it to slow down or speed up depending on the Tracker processing speed.

Once all the frames have been processed the Network and Tracker thread stop. After that, the results are logged into the YML files (a file for each frame) and the user can

close the application.

The general behavior of the application when running video sequences or raw frames was presented. Nevertheless, it can also handle live stream videos coming from local cameras connected. In this case, the logging of the results is not performed but the tracking by detection scheme is the same.

The "main" is done in the `update` function of the Camera which is continuously called by the Camera thread. As the system architecture is based in multiple threads, the synchronism between them is crucial. Because of this, the control of the application is done taking into account the synchronism between all the threads, their internal status and variables.

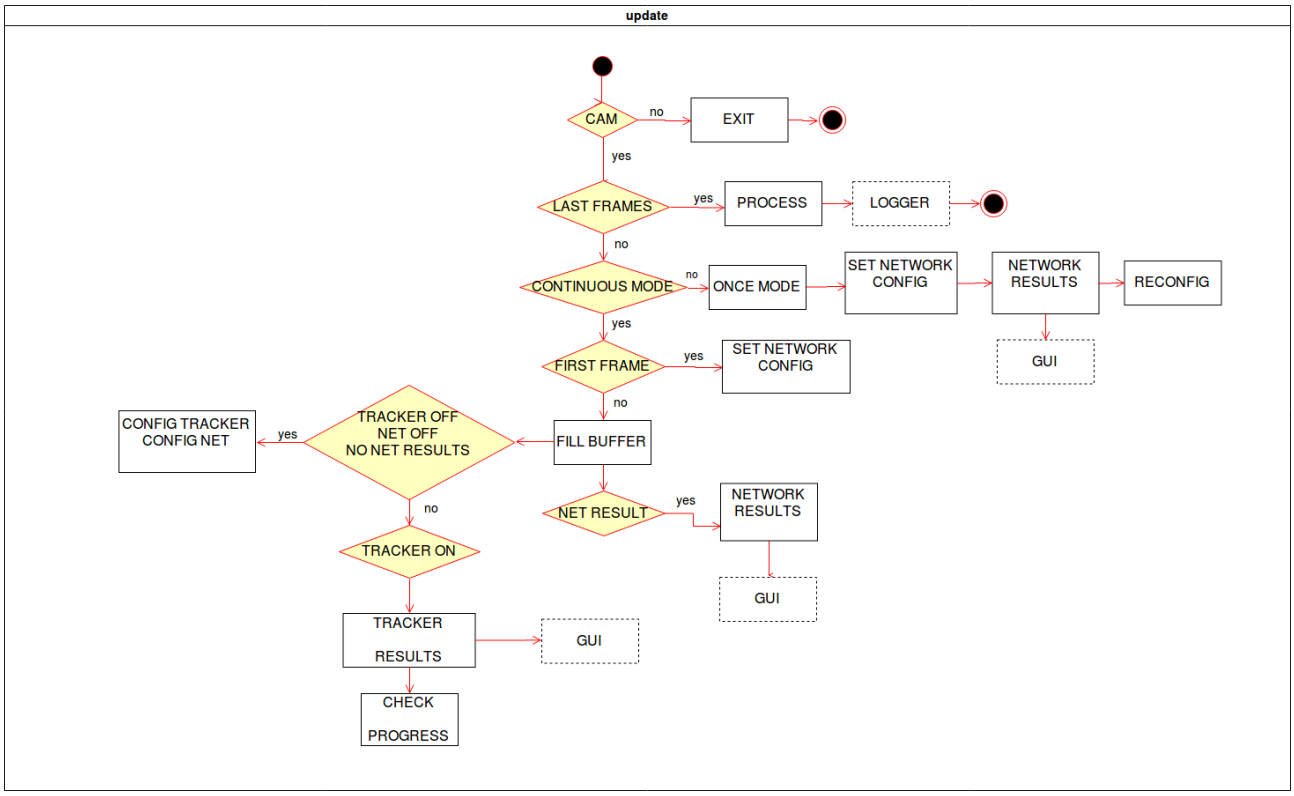

Figure 4.3: The control flow

The application is configurable using an YML file (*objecttracker.yml*). For the instructions on how to run the application please refer to the wiki of the project (see 1.4).

In the next sections the implementation of each thread of the system is explained, including the available configuration modifications.

4.2 Video source module

The Camera module provides the input images to the rest of the system. These images can be obtained using four different sources:

- **Local camera (with OpenCV)**: the Camera can read from a local camera using the OpenCV routine `VideoCapture` indicating the device number of the camera.
- **Local camera (with ROS)**: the Camera can use ROS to read from the camera device. In order to do so, the user needs to launch a terminal and type `roslaunch usb_cam.launch`. This will publish a ROS topic `/usb_cam/image_raw` that can be subscribed by the Camera module and it will start reading frames from the device. For more information about the launching process please refer to the wiki of the project[1].
- **Local video**: to read from a local video the OpenCV routine `VideoCapture` is also used but indicating the video path in the configuration file.
- **Local image files**: the path containing all the image files is required to be passed to `VideoCapture`. This video source is very useful because most of the datasets are provided as sequences of frames instead of videos. And it can avoid problems such as creating sequences of videos with the wrong duration or frame rate.

The Camera thread source and options can be modified at the configuration file. The source is selected at `ObjectTracker->Source`. After that the user is required to indicate the device number `ObjectTracker->Local->DeviceNo` if using a local camera with OpenCV, the video path `ObjectTracker->Video->Path` if using a local video or the images path `ObjectTracker->Images->Path` when using local image files.
The user needs to modify the `usb_cam.launch` to change the Camera configuration when using ROS.
Before being sent to other threads the image is rescaled according to the neural network input size. It continues all the process with this standard size. When the final results are obtained this scaling is taken into account to rescale again the coordinates of the detections or trackers with respect to the original image size.
Apart from providing images the Camera thread also controls the flow of the application.

[1] Wiki: ROS

It has been implemented in this way because the application offers the option of not having GUI. In the first versions of the project the control of the application (the "main") was implemented in the GUI thread but when the no-GUI option was added this was moved to the Camera thread.

4.3 GUI module

The GUI module provides the interface with the human user and, as commented before, is optional. It was implemented using the tools provided by PyQt5, in particular with the packages *QtGui, QtCore* and *QtWidgets*.

The graphical interface has four windows and two buttons. The top-left window shows the input frames in real-time while the top-right one shows the final results. The intermediate results obtained from the Network and the Tracker will be provided at the bottom part of the display. The application with GUI has two modes with its respective buttons: "run continuous" and "run now". In the first one, the application runs continuously until it finishes the processing (the program finishes depending on the image source). In the second one, the user can push the *Run now* button to make a single Network detection on the current frame and to continue the tracking from that frame onward.

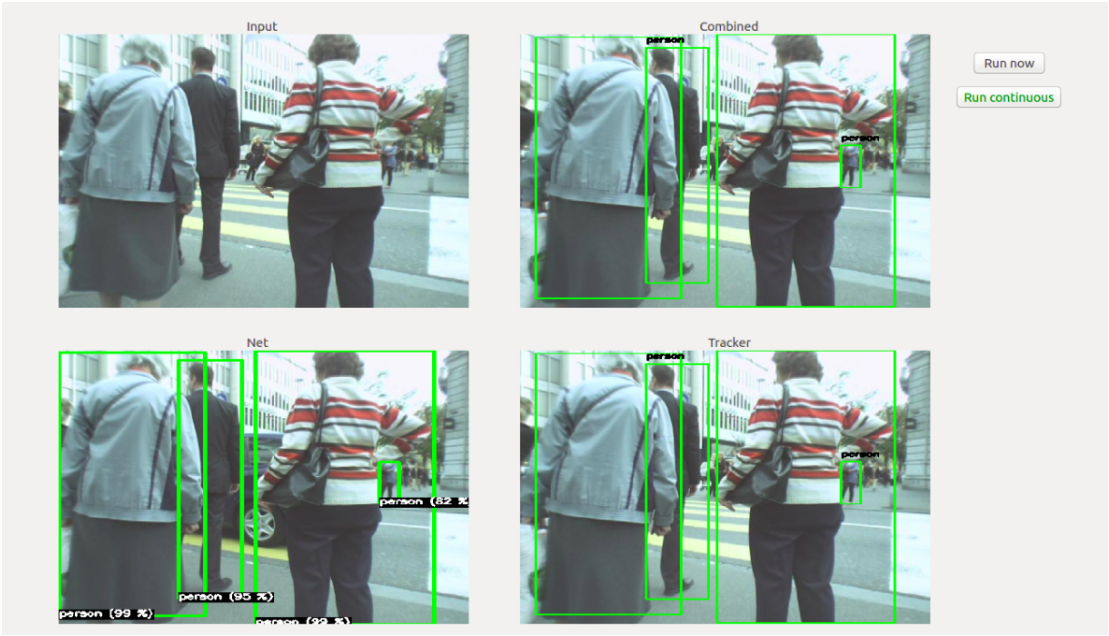

Figure 4.4: The graphical user interface of the dl-objecttracker application

4.4 Neural Network module

The Network module is tasked with the object detections in the images, which feed the Tracker on every iteration. It supports Tensorflow and Keras object detection models. The Tensorflow models can be obtained from the Tensorflow detection model zoo[2] and include SSD and R-CNN detectors. The Keras models are limited to SSD architectures and it can be obtained from this set[3].
The datasets on which the model was trained need to be specified to assign the labels to the objects. The supported labels include the VOC, COCO, KITTI, OID and PET datasets.
As it ocurred with the Camera module, the Network module has configurable options available in *objecttracker.yml*:

- `Framework`: Keras or Tensorflow
- `Model`: the model file
- `Dataset`: VOC/COCO/KITTI/OID/PET
- `Input size`: this input size can be modified depending on the selected model. Some models do not allow to change the input image size
- `Confidence`: the confidence threshold for the detections obtained. If a detection obtains a confidence value below that detection is discarded

The Network thread basically receives an image (previously resized) and performs the inference. As a result it outputs the detections obtained in the image (if any) and it draws these detections in form of bounding boxes containing also the label and the confidence value.
The logging of the Network and Tracker results are optional and it can be changed at the YML configuration file in `ObjectTracker->Logger->Status`. These results are logged in the `logNetwork` and `logTracker` functions respectively.

[2]Tensorflow detection model zoo
[3]Keras models

4.4.1 Tensorflow models

These models are obtained from the Tensorflow detection model zoo which provides models for inference out-of-the-box, i.e. to be directly used. Among the available models only some of them were tested for its use in the project. The pre-trained models used were trained on the COCO dataset because the available classes in this dataset were considered enough for the type of objects that can be seen in the tracking sequences used.
In the table 4.1, the Tensorflow models performance can be seen:

Model name	Speed (FPS)	COCO mAP
ssd_mobilenet_v2_coco	32	22
faster_rcnn_inception_v2_coco	17	28
mask_rcnn_inception_v2_coco	13	25

Table 4.1: Tensorflow models performance (from Tensorflow detection model zoo)

The COCO mAP numbers here are evaluated on COCO 14 minival set using the MSCOCO evaluation protocol. The reported running time in ms is measured for 600x600 images (including all pre and post-processing). The AP scores are averaged over multiple Intersection over Union (IoU) values as the MSCOCO evaluation protocol indicates. As the authors say, "these timings depend highly on one's specific hardware configuration (performed using an Nvidia GeForce GTX TITAN X card) and should be treated more as relative timings in many cases". However, some characteristics of each model can be extracted in terms of speed and accuracy. As expected, the R-CNN models achieve better accuracy while performing a little slower with respect to SSD.

4.4.1.1 SSD MobileNetV2

This SSD implementation uses MobileNetV2 as backbone. As commented in section 2.4, MobileNetV2 is a network proposed to work on mobile devices, this can be interesting to the project because of the hardware limits (it needs to work on CPU only).

4.4.1.2 Faster R-CNN InceptionV2

This region-based model was discussed in section 2.4. The implementation uses InceptionV2 [72] as the feature extractor. This backbone follows the idea of its predecesor

(InceptionV1) and adds two main ideas: reduce the representational bottleneck and use smart factorization methods. Refering to the first one, the intuition is that neural networks usually perform better when the convolutions do not alter the dimensions of the input in a drastical way (may cause loss of information). To solve this problem they expand the inception module making it wider (instead of deeper). Apart from that, the convolutions are made more efficient in terms of computational complexity. The authors propose factorizing the 5x5 convolution into two 3x3 convolution operations making it 2,78 times faster, among other improvements.

4.4.1.3 Mask R-CNN InceptionV2

This state-of-the-art instance segmentation network is used as object detection network because it offers the bounding boxes locations, apart from the instance masks (Figure 4.5). The selected implementation also makes use of InceptionV2 as backbone.

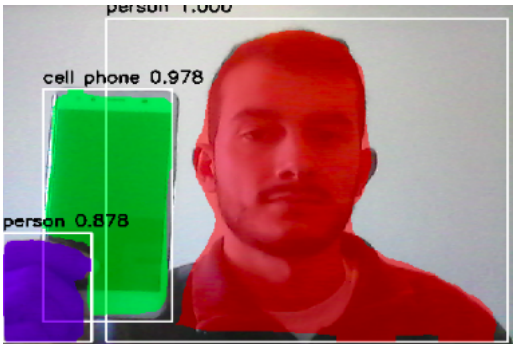

Figure 4.5: First tests with Mask R-CNN using live video

4.4.2 Keras models

The Keras network is a Keras port of the SSD model architecture introduced by Wei Liu et al. in [2] from SSD-Keras[4]. The repository offers pre-trained models and allows the model training from scratch. The base network architecture used is VGG (see section 2.4). The pre-trained models used were trained on the PASCAL VOC dataset. The pre-trained models available included COCO and ILSVRC datasets but PASCAL was selected because COCO dataset was already used in the previous SSD model in Tensorflow. In the next table, it can be seen the author reported performance [73]:

[4]SSD-Keras

Model name	Speed (FPS)	VOC2007 test mAP @ 0,5
SSD300_VOC0712	39	77,5
SSD512_VOC0712	20	79,8

Table 4.2: Keras models performance. Evaluated in the official Pascal VOC 2012 test server using an NVIDIA GeForce GTX 1070 mobile.

The authors claim that their implementation performs slightly better than the original SSD implementation in Caffe[5].

4.5 Tracker module

The Tracker module is the core of the project and therefore it is going to be explained more in depth. As previously commented, the Tracker receives an input detection coming from the Network module and performs the multiobject visual tracking over a frame buffer of variable size following a tracking by detection scheme. This hybrid tracker it is intended to show the advantages of the tracking by detection with deep learning over a pure neural network tracking or a pure classic feature tracking.

The tracker can work in three operating regimes: *slow, normal* and *fast*. The calculation of this regime is performed using an internal buffer of FPS rates of size 3. The length selected is due to the fact that the tracker is required to respond quickly to changes in its velocity avoiding slowing down or speeding up in an excessive way. The tracker speed calculation is explained in Algorithm 1. Three frame rate thresholds are established to distinguish between a tracker which is behaving in a "normal way" or a slower or a faster processing time.

If the tracker is performing slower than normal the next three frames that are to be tracked are discarded from the buffer in the `imageToTrack` function. In the other hand, if the tracker is going too fast it is slowed down in `getOutputImage` by waiting some frames to return the image to be output.

With this mechanism the buffer does not increase or decrease too much in size allowing a more stable behavior and a good synchronization between the threads. This dynamic calculation allows the tracking to behave differently depending on the operating regime in

[5]SSD-Caffe

which it is on each instant. However, it will be seen in the next chapter that this regime is very dependent on the tracker that is being used and its own performance.

Algorithm 1 Tracker speed mode

Inputs: averageFPS, lastFPSBuffer, trackerSlow, trackerFast, counterSlow, counterFast

Output: trackerSpeedMode

procedure TRACKERSPEEDMODE

if not 0 in $lastFPSBuffer$ and $averageFPS$ <10 **then**

$counterSlow$ + 1

if $counterSlow$ == 3 **then**

$counterSlow$ = 0

$trackerSlow$ = True

end if

else if not 0 in $lastFPSBuffer$ and 10 <$averageFPS$ <25 **then**

$trackerSlow$ = False

$trackerFast$ = False

else if $averageFPS$ >25 and $counterFast$ <1 **then**

$counterFast$ + 1

$trackerFast$ = True

end if

end procedure

4.5.1 Confidence in tracking

To check if the obtained tracking is acceptable or not the confidence value is used. The two libraries available to perform the tracking offer this feature so it is used with small differences. OpenCV returns a boolean value to show the confidence in the tracking, and that is directly used to threshold the tracking results. In the case of dlib, it returns a numerical value. The dlib threshold value was established to 7 based on examples of other works and some experiments.

4.5.2 OpenCV trackers

The tracking will be performed using several already built tracking implementations of two libraries: OpenCV and dlib. This will also allow for a good comparative between them that can lead us to select the preferred option for the tracking algorithm.
OpenCV is known for the great variety of algorithms for which it provides implemented solutions, one of them is tracking. Tracking libraries are included in the OpenCV extra modules (`opencv contrib`).
The tested OpenCV trackers include *BOOSTING, MIL, MEDIANFLOW, TLD, KCF, MOSSE* and *CSRT* (in release order). They are now introduced:

- BOOSTING [74]: based on an online version of AdaBoost, the tracker is trained at runtime with positive and negative examples of the object to track. An initial bounding box needs to be provided by the user or other object detection algorithm. The classifier looks over the pixel neighborhood of a previous location to find the new location. The classifier is constantly updated with these new positives.

- MIL [24]: the *Multiple Instance Learning* algorithm tries to solve the problem of learning an adaptive appearance model for object tracking. To achieve this, the authors train a discriminative classifier online to separate the object to track from the background, i.e. positive and negative examples are extracted from the frame (Figure 4.6). Similarly to the Boosting algorithm, the model searches inside of the window of the old location. It obtains a probability map with most probably new location of the object and updates the tracker model.

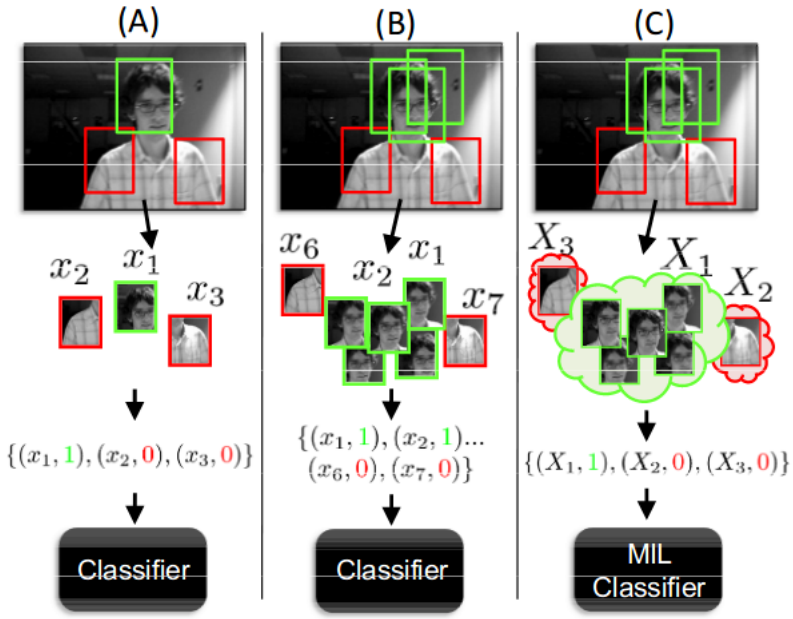

Figure 4.6: Updating a discriminative appearance model: (A) using a single positive image patch. (B) using several positive image patches. (C) using one positive bag of several image patches (from [24])

- MEDIANFLOW [25]: the Median Flow algorithm introduced a novel method for tracking failure detection based on the Forward-Backward error. This basically consists of performing the tracking forward and backward in time in a given frame and measure the discrepances between trajectories (see Figure 4.7). The authors claim that these discrepances are highly correlated with real tracking failures.

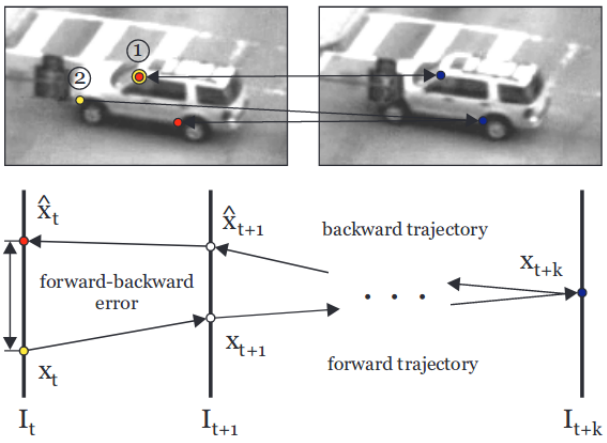

Figure 4.7: The forward-backward error in Point 2 (from [25])

- TLD [75]: in the original paper on which is based the implementation, the authors investigate the long-term object tracking and propose a novel framework that descomposes this type of tracking into *tracking, learning and detection* (TLD). The tracker based on Median Flow follows the object in every frame. The detector is composed by a patch variance module, followed by an ensemble classifier and finally a Nearest Neighbor classifier. The function of this detector is to correct the tracker if necessary. The learning step estimates the errors of the detector and updates it using a novel method called *P-N learning*.

- KCF [76]: the *Kernelized Correlation Filter* is a tracking framework that utilizes properties of circulant matrix to enhance the processing speed. The authors observed that the translated and scaled patches used to train discriminative classifiers contain redundancies and the resulting data matrix from these patches is circulant. With kernel regression as classification method they derive the KCF tracking.

- MOSSE [26]: correlation filters can track complex objects in common tracking scenarios that may include rotations, occlusions or other distractions at high frame rates. The *Minimum Output Sum of Squared Error* filter is another type of correlation filter. Filter based trackers model the appearance of objects using filters trained in example images (Figure 4.8). With a given initial target in the first frame the tracking and the filter training start to work together. The idea behind MOSSE is an optimization problem, given a set of training images f_i and training outputs g_i, MOSSE finds a filter H that minimizes the sum of squared error between the actual output of the convolution and the desired output of the convolution (see Formula 4.1).

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \tag{4.1}$$

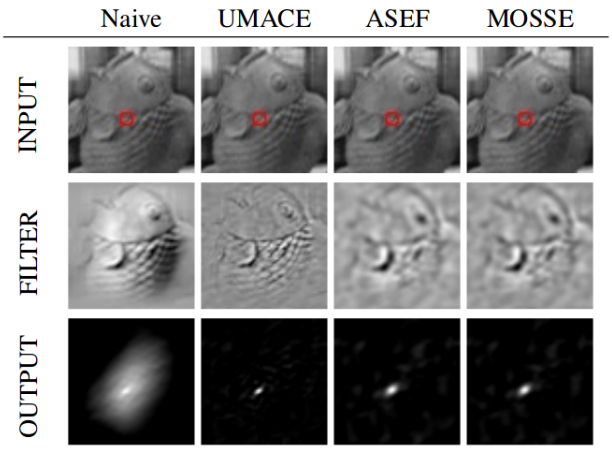

Figure 4.8: Comparison of the output peaks produced by different correlation filters (from [26])

- CSRT [27]: the CSRT tracker is based on the paper *Discriminative Correlation Filter with Channel and Spatial Reliability*. Here the authors introduce the channel and spatial reliability concepts to DCF tracking to improve the filter update and the tracking process (Figure 4.9). The spatial information is used to restrict the searching to the parts suitable for tracking. In the other hand, the channel information aims to reduce the noise of the weighted-averaged filter response.

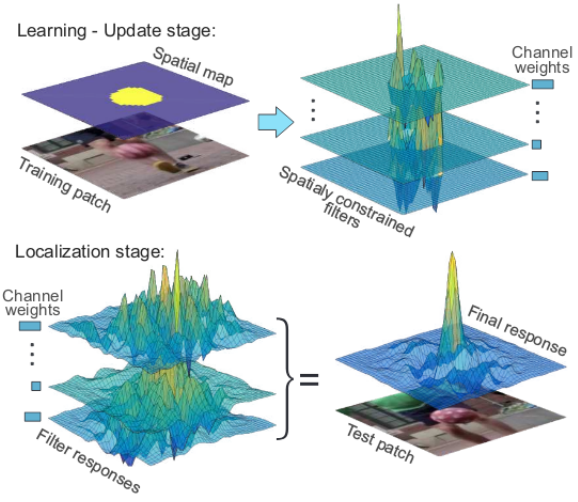

Figure 4.9: Overview of the CSR-DCF approach (from [27])

In the next chapter, some experiments are performed to put into practice the advantages and disadvantages of each OpenCV tracker. Furthermore, the dlib tracking will also be

discussed allowing for a comparison between all the proposals and allowing us to select the best tracker option.

4.5.3 dlib trackers

In chapter 2 the `dlib` library was introduced as a set of independent software components that provide different utilities, one of them is tracking. The `dlib.correlation_tracker` is going to be used for this project. As its name indicates, it is another implementation of a correlation filter for tracking which has been widely used. This tool is an implementation of the method described in [71].

In the proposed solution, the authors are centered in solving the challenging problem of handling large scale variations in visual object tracking. They propose a method for a robust scale estimation in a tracking by detection framework, as it is the case in this project. To do so the learning of discriminative correlation filters is based on a scale pyramid representation.

Chapter 5

Experiments

In this chapter, the quality of the Network and Tracker modules is characterized. First, the available neural networks will be evaluated with a common tracker to select the final neural network used in the dl-objecttracker application. Second, several trackers implementations will be also characterized using the selected neural network. The configurable parameters will be adjusted to select the best performing values. This will give us the best combination of Network and Tracker modules for the final dl-objecttracker, which will also be experimentally validated.

5.1 Experimental setup

The experiments were performed on a laptop PC with *Intel® Core™ i7-4510U CPU @ 2.00GHz x 4* and no GPU acceleration.

As commented in section 3.5, the Object Detection Metrics tool was used to compute the following metrics: precision, recall and AP. It is necessary to mention that the tool was slightly modified to provide the TP, FP and GT numbers. The speed measuremenents are obtained directly from the `dl-objecttracker` for both the Network and the Tracker modules.

The selected dataset for evaluating the tracking application and its modules is the MOT17Det [7] *train* set (see Table 3). The results were not evaluated on the *test* set due to the fact that the official web of the challenge does not include in the provided data the annotated ground truth of the test set. To obtain the ground truth from this dataset and adapt it to the metrics tool a small Python script was developed in this

project following the official reference [7]. However, some modifications were done to allow the compatibility between the metrics tool and the labels of the detections (the neural networks are trained in COCO or PASCAL) (see Table 5.1). Following the official MOT interpretation of ground truth detection files, the final ground truth obtained from the train set only includes the *person* class.

ID	Label in MOT gt	Label in our gt
1	Pedestrian	Person
2	Person on vehicle	Car
3	Car	Car
4	Bicycle	Bicycle
5	Motorbike	Motorbike
6	Non motorized vehicle	Bicycle
7	Static person	Person
8	Distractor	-
9	Occluder	-
10	Occluder on the ground	-
11	Occluder full	-
12	Reflection	-

Table 5.1: Label equivalences with MOT ground truth in our ground truth

5.2 Neural network selection

The correct selection of a neural network model for object detection is crucial in this project as it provides the previous detections the tracker module needs to track. As commented in section 4.4, the avalable neural networks models which have been integrated into dl-objecttracker are:

- SSD MobileNetV2, pretrained on COCO (Tensorflow)
- Faster R-CNN InceptionV2, pretrained on COCO (Tensorflow)
- Mask R-CNN InceptionV2, pretrained on COCO (Tensorflow)
- SSD VGG, pretrained on Pascal VOC (Keras)

Three sequences from the MOT17Det dataset were selected to evaluate the performance of the models. The reason is that these sequences represent most of the possible difficulties that may appear in multiple object tracking tasks such as occlusions, new targets, fixed

camera, big motion from frame to frame, etc. The selected sequences are MOT17-05, MOT17-09 and MOT17-11 (Figure 5.1).

Figure 5.1: MOT17Det train set samples: left image from MOT17-05, center image from MOT17-09 and right image from MOT17-11

The MOSSE tracker was selected as regular tracker and the threshold for all neural network detection was fixed to 0,6. This was done to allow a fair comparison between the models.

The MOT17-09 sequence has a fixed camera with several pedestrians walking in groups or alone. In this sequence, the tracking may find fast motion difficulties as well as continuous people coming in and out from the scene. To evaluate the performance in that sequence, two input image sizes were selected due to the Keras SSD-VGG fixed image input size.

From the table 5.2, it can be seen that the maximum AP value is obtained by the Faster R-CNN using 512x512 whereas Mask R-CNN gets the best AP score for 300x300 images. As expected, the R-CNN detectors obtain the best accuracy. However, this accuracy is not linked with the speed in the object detection. The SSD MobileNetV2 gets the best speed rate in both experiments.

	AP @ 0,5 (%)	**FPS Net**
SSD MobileNetV2	16,06	5,282
Faster R-CNN InceptionV2	31,03	1,026
Mask R-CNN InceptionV2	27,89	0,286
SSD VGG 512	23,76	0,339

Table 5.2: Experiments on MOT17-09 sequence with 512x512 images

	AP @ 0,5 (%)	FPS Net
SSD MobileNetV2	11,48	8,25
Faster R-CNN InceptionV2	24,36	1,067
Mask R-CNN InceptionV2	26,25	0,292
SSD VGG 300	19,08	0,988

Table 5.3: Experiments on MOT17-09 sequence with 300x300 images

The influence of the image input size in the processing speed and the accuracy of the models is clear. The smaller the input size of the image the faster the detections are obtained. In the opposite way, with a bigger input image the final AP result is better. This trend will be confirmed in following experiments.

The SSD-VGG models were discarded from other experiments due to its lack of flexibility about the input image size. In the table 5.4 it can be observed the different performances for 800x800 images of the region-based object detectors and the single-shot object detectors. As it ocurred with smaller image input sizes, the region-based models have a better AP performance almost doubling the AP obtained by the SSD (in the case of Mask R-CNN). For this reason, the SSD MobileNet V2 was eliminated from the neural net selection procedure despite being the fastest.

	AP @ 0,5 (%)	FPS Net
SSD MobileNetV2	17,13	7,372
Faster R-CNN InceptionV2	32,00	0,981
Mask R-CNN InceptionV2	34,23	0,272

Table 5.4: Experiments with 800x800 images

The final experiments were done with Faster R-CNN and Mask R-CNN in MOT17-09, MOT17-11 and MOT17-05. The last two sequences have a common characteristic which is that the camera is in motion. Thus, the sequences are enumerated in order of increasing degree of motion, starting from MOT-09 to MOT-05.

AP @ 0,5 (%)	MOT17-09	MOT17-11	MOT17-05
Faster R-CNN InceptionV2	35,25	26,21	19,51
Mask R-CNN InceptionV2	31,74	26,44	12,98

Table 5.5: Neural network experiments with an image input size of 1000x1000

From these experiments on table 5.5, it can be seen that the final average precision is similar in the sequence MOT17-11, however, the use of Faster R-CNN model outperforms Mask R-CNN in the other two sequences. These AP scores can be related with the frame rate obtained from each neural network which is about 0,9 FPS for Faster R-CNN and 0,2 FPS for Mask R-CNN. A higher speed can help the tracking procedure to be "refreshed" more frequently which may lead to better performance on sequences with varying motion between frames as it occurs on the evaluated sequences.
In the figure 5.2 it can be observed an example of the detections obtained with Faster R-CNN. Generally, the results seem to be pretty accurate but they include some false positives such as the person assigned to a confidence of the 69%.
Given these cuantitative results, the final neural network chosen to perform the object detection in dl-objecttracker is Faster R-CNN InceptionV2.

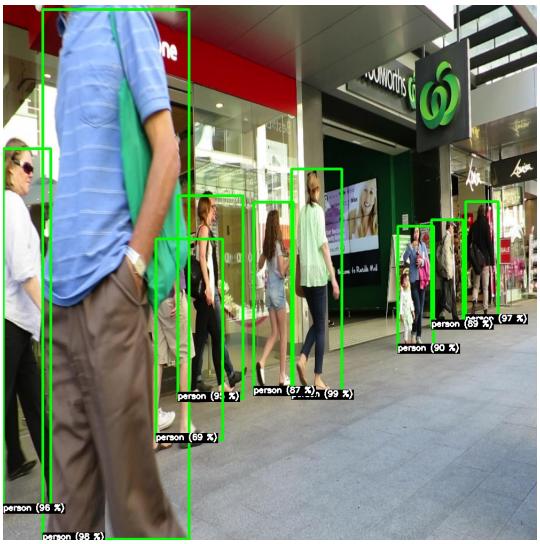

Figure 5.2: Faster R-CNN Inception V2 object detections on MOT17-09

5.3 Tracker's performance

Once the neural network was selected, it is time to evaluate the performance of the second module involved in the core of the multiobject tracking application, the tracking module. The following experiments were done on the same sequences as the Network experiments. In this case, the default configuration includes Faster R-CNN as neural network with a confidence threshold for detection set to 0,6. The confidence of the tracker is being used if no comment is made on it. This confidence is going to be modified later in this section to

see its influence on the results. The following tracking algorithms were evaluated: KCF, BOOSTING, MIL, TLD, MEDIANFLOW, CSRT, MOSSE and CF-dlib (section 4.5).

	AP @ 0,5 (%)	**FPS Tracker**
KCF	*23,07*	*6,39*
BOOSTING	13,06	4,78
MIL	15,29	2,21
TLD	8,38	2,22
MEDIANFLOW	*32,13*	*12,01*
CSRT	11,78	2,78
MOSSE	*34,60*	*47,07*
CF-dlib	*27,99*	*9,51*

Table 5.6: Tracker experiments on MOT17-09 sequence with 1000x1000 images

This experiment provides clear results on the performance of the trackers in the MOT17-09 sequence. The KCF, MEDIANFLOW, MOSSE and CF-dlib outperform in a significant way the accuracy of the rest of the trackers (in terms of AP). A good AP seems to be related with a good frame rate in the tracking.
The scheme of evaluating the performance with sequences of increasing difficulty was followed, the second experiment was performed with the MOT17-11 sequence and its results are shown in table 5.7.

	AP @ 0,5 (%)	**FPS Tracker**
KCF	19,17	4,7
MEDIANFLOW	*27,76*	*12,88*
MOSSE	*26,23*	*33,56*
CF-dlib	*25,49*	*9,95*

Table 5.7: Tracker experiments on MOT17-11 sequence with 1000x1000 images

The results in this sequence seem to indicate that the KCF tracker is not adecuated for the task. Its results in both AP and the speed measurements are below the overall average.

	AP @ 0,5 (%)	**FPS Tracker**
MEDIANFLOW	*24,01*	*13,05*
MOSSE	16,15	18,14
CF-dlib	23,97	9,51

Table 5.8: Tracker experiments on MOT17-05 sequence with 1000x1000 images

The results from table 5.7 led us to three final tracker options: MEDIANFLOW, MOSSE and CF-dlib. In the experiment on MOT17-05 shown on table 5.8, MEDIANFLOW gets the overall best performance. It achieves the highest AP score of the three tracker options and the second faster tracking. The faster tracker is MOSSE, following the trend of previous experiments.

5.3.1 Confidence in tracking

In section 4.5.1, the mechanism of confidence of the tracker was introduced. The tracker itself continuously checks if the tracking obtained for each object is reliable enough. In this section, the importance of this parameter is going to be evaluated. To do so, the performance of the three best tracking algorithms is measured when the confidence is taken into account and in the opposite case. The selected sequences are MOT17-05 and MOT17-09 with a frame size of 500x500 and the neural network used in both cases is Faster R-CNN InceptionV2.

	AP tracker on @ 0,5 (%)	**AP tracker off @ 0,5 (%)**
MEDIANFLOW	36,09	32,35
MOSSE	18,60	10,33
CF-dlib	23,74	30,06

Table 5.9: Confidence influence on tracking performance on MOT17-05

	AP tracker on @ 0,5 (%)	**AP tracker off @ 0,5 (%)**
MEDIANFLOW	37,80	36,17
MOSSE	30,16	21,83
CF-dlib	28,06	31,60

Table 5.10: Confidence influence on tracking performance on MOT17-09

In the tables 5.9 and 5.10 the results seem to indicate that the influence of taking into account the confidence parameter with OpenCV trackers is positive. However, it occurs the opposite for the `dlib` tracking with CF.

The MEDIANFLOW tracker was finally selected to perform the tracking in the dl-objecttracker application given the performace shown in the previous experiments. Figure 5.3 shows an example of the tracking using that tracking algorithm.

Finally, the image input size was modified to check which size was best appropriate to our problem. In table 1 of the *Annex*, the image size of 400x400 gets the best balance between speed and accuracy for the tracking task. Given the tracker and the image size, the threshold for the confidence of the detections from the object detection neural networks was evaluated. Using values ranging from 0,3 to 0,7 the experimental threshold selected was 0,5.

Figure 5.3: Medianflow multiobject tracking on MOT17-05 (selected frames are not sequential)

5.3.2 GOTURN tracking

The GOTURN (*Generic Object Tracking Using Regression Networks*) is a deep learning based tracking algorithm available in OpenCV which learns the motion of the object in an *offline* manner. Many real-time trackers rely on *online* learning that is usually much faster than a deep learning based tracking solution. The authors claim in the original paper [44] that their system is "the first neural-network tracker that learns to track generic objects at 100 FPS" (using GPU acceleration with an Nvidia GTX 680). However, when using only a CPU the tracker runs at 2,7 FPS according to the authors. This was the main reason to discard this tracker for the project. Apart from the speed side, in some tests performed using live video the tracking results were not specially good as it had problems with occlusion and motion during the tracking.

5.4 Experimental validation of the final solution

After the unit test experiments, the dl-objecttracker application follows this configuration:

1. Neural network: Faster R-CNN InceptionV2, image input size 400x400, confidence threshold 0,5

2. Tracker: MedianFlow using tracking confidence

Given this configuration, the whole application was evaluated on the complete train set of MOT17Det to obtain the results of our final solution.

dl_objecttracker	**AP @ 0,5 (%)**	**FPS Net**	**FPS Tracker**
MOT17-02	11,59	0,93	31,4
MOT17-04	17,25	0,869	23,96
MOT17-05	36,53	0,98	37,28
MOT17-09	43,53	0,95	35,83
MOT17-10	23,26	0,943	36,18
MOT17-11	35,74	0,96	41,56
MOT17-13	14,04	0,941	42,01

Table 5.11: Final results on MOT17Det train set

The best results in terms of average precision occur in the MOT17-09 sequence, followed by MOT17-11 and MOT17-05. This may indicate that the procedure used influences the results giving the better scores in the sequences used to evaluate the performance of the Tracker and Network module. However, from table 2 and table 3 it can be observed that the three mentioned sequences have in common that they have a small number of total ground truth ocurrences. It can be easier for the developed system to get good results in this type of sequences. The results indicate that the developed application performs best on sequences with lowly crowded densities.
Refering to the speed of the developed application, the object detection in the neural network is returned with a stable frame rate of about 1 FPS and the tracker runs above 20 FPS in every sequence.

Chapter 6

Conclusions

This chapter summarizes the main contributions of this work. Possible lines of future work are also outlined.

This master thesis studied the use of deep learning techniques to build a multiobject visual tracking system using the tracking-by-detection scheme. To solve this task we have designed a modular application composed of a Neural Network module, a Classic Tracker module, a Camera module and a GUI module. The first module provides object detections using neural network models. These detections are handled by the Tracker module to track objects inside a buffer of frames before the new detections come. The application works in delayed real-time. The Camera module controls the general flow of the application which includes the logging mechanism and the user interface (GUI).

The Tracker module can work on three operating regimes depending on the frame rate of the tracking at each time: slow, normal and fast. This allows the tracking to adapt its speed to the processing difficulties (different image sizes, occlusions, appearance changes, ...). It may also help to adapt the tracking processing speed to the hardware on which it is being run. Refering to the user side, the project application allows the user to change many configuration options. This feature helps, for example, the quick tests of neural networks or trackers for multiobject tracking.

Once the multiobject tracking system was developed, the first three objectives of this project were fullfilled:

1. Development of an object detector using deep learning
2. Development of a visual tracking module
3. Combination of neural object detection and object tracking in a single software

component

After this, the last objective was accomplished as the application was evaluated on a well-known international multiobject tracking dataset (MOT17Det), allowing us to choose the best configuration based on some experiments. This included the selection of the best neural network model, the best tracking algorithm and other parameters such as the confidence thresholds or the image input size (section 5.4). The results obtained allowed us to extract the following conclusions:

- Region-based object detection neural networks obtain better accuracy than single-shot based ones. They can be used to perform inference on CPU at low frame rates.
- MedianFlow seems to be the best tracker available in the OpenCV library because of its balance between speed and accuracy. MOSSE is the fastest one.
- The confidence is useful to discard bad tracking performance when working with OpenCV trackers. However, dlib tracking seems to be less influenced by the confidence thresholding.
- The image input size is a key factor when working with resource limited hardware to achieve higher throughput.
- The final solution seems to perform better on sequences with lowly crowded density.

We have built a visual multiobject tracking system that performs reasonably well on a MOT dataset despite is not into the State-of-the-Art. It combines the robustness of deep learning approaches with the speed of classic tracking methods.

6.1 Future works

This master thesis is a first step into the multi-object tracking with deep learning. Once it has been developed and the results seen, the following lines of future work are proposed:

1. Train neural network models used (or new ones) in multiobject tracking datasets such as MOT. This could lead to better results.

2. Refering to the tracking, the multiprocessing with dlib for tracking is available but it was not introduced in the final application. Its integration may speed up the tracking.

3. Obtain the best configuration in a different way. For example, trying more possible combinations of parameters in other dataset sequences.

4. Improve the metrics calculation by assigning IDs to the tracked objects allowing for the calculation of tracking metrics (MOTA, for example). There are official Python implementations of metrics for benchmarking MOT such as *py-motmetrics*.

5. Test the application in other non-GPU devices as Raspberry Pi or Intel Computer Stick and in devices with graphic acceleration.

6. Try weights quantization techniques in neural networks with high number of parameters (region-based, for example). It can help to speed up the inference time and it will be more efficient in terms of memory consumption.

Bibliography

[1] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. Mots: Multi-object tracking and segmentation. *arXiv preprint arXiv:1902.03604*, 2019.

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[3] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[4] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[5] Zdenek Kalal, Jiri Matas, and Krystian Mikolajczyk. Pn learning: Bootstrapping binary classifiers by structural constraints. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 49–56. IEEE, 2010.

[6] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. Siamrpn++: Evolution of siamese visual tracking with very deep networks. *arXiv preprint arXiv:1812.11703*, 2018.

[7] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[8] Séverine Dubuisson and Christophe Gonzales. A survey of datasets for visual tracking. *Machine Vision and Applications*, 27(1):23–52, 2016.

[9] Luis Patino, Tom Cane, Alain Vallee, and James Ferryman. Pets 2016: Dataset and challenge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2016.

[10] Matthias Muller, Adel Bibi, Silvio Giancola, Salman Alsubaihi, and Bernard Ghanem. Trackingnet: A large-scale dataset and benchmark for object tracking in

the wild. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 300–317, 2018.

[11] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2411–2418, 2013.

[12] Matej Kristan, Ales Leonardis, Jiri Matas, Michael Felsberg, Roman Pflugfelder, Luka Cehovin Zajc, Tomas Vojir, Gustav Hager, Alan Lukezic, Abdelrahman Eldesokey, et al. The visual object tracking vot2017 challenge results. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1949–1972, 2017.

[13] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *Journal on Image and Video Processing*, 2008:1, 2008.

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[17] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

[19] Jonathan Hui. mAP (mean Average Precision) for Object Detection. `https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173`.

[20] OpenCV documentation. Canny Edge Detection. https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html.

[21] Filippo Sanfilippo, Erlend Helgerud, Per Anders Stadheim, and Sondre Lieblein Aronsen. Serpens: A highly compliant low-cost ros-based snake robot with series elastic actuators, stereoscopic vision and a screw-less assembly mechanism. *Applied Sciences*, 9(3):396, 2019.

[22] Christos Sagonas, Epameinondas Antonakos, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: Database and results. *Image and vision computing*, 47:3–18, 2016.

[23] Michael Herrmann. PyQt5 tutorial. https://build-system.fman.io/pyqt5-tutorial.

[24] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE, 2009.

[25] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *2010 20th International Conference on Pattern Recognition*, pages 2756–2759. IEEE, 2010.

[26] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550. IEEE, 2010.

[27] Alan Lukezic, Tomas Vojir, Luka Cehovin Zajc, Jiri Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6309–6318, 2017.

[28] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[29] MOT17Det. https://motchallenge.net/data/MOT17Det/.

[30] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12, 2006.

[31] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.

[32] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[33] Tianyi Liu, Shuangsang Fang, Yuehui Zhao, Peng Wang, and Jun Zhang. Implementation of training convolutional neural networks. *arXiv preprint arXiv:1506.01195*, 2015.

[34] Marcos Pieras Sagardoy. Visual people tracking with deep learning detection and feature tracking. `https://gsyc.urjc.es/jmplaza/students/tfm-visualtracking-marcos_pieras-2017.pdf`, TFM, Máster Oficial en Visión Artificial, curso académico 2016-2017.

[35] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[36] Ignacio Condés and José María Cañas. Person following robot behavior using deep learning. In *Workshop of Physical Agents*, pages 147–161. Springer, 2018.

[37] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, 2014.

[38] Kai Briechle and Uwe D Hanebeck. Template matching using fast normalized cross correlation. In *Proc. SPIE*, volume 4387, pages 95–102, 2001.

[39] Hieu Tat Nguyen and Arnold WM Smeulders. Fast occluded object tracking by a robust appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1099–1104, 2004.

[40] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149. IEEE, 2000.

[41] Hieu T Nguyen and Arnold WM Smeulders. Robust tracking using foreground-background texture discrimination. *International Journal of Computer Vision*, 69(3):277–293, 2006.

[42] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK, 1988.

[43] Jianbo Shi et al. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[44] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*, pages 749–765. Springer, 2016.

[45] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. *arXiv preprint arXiv:1701.01909*, 2017.

[46] Patrick Pérez, Carine Hue, Jaco Vermaak, and Michel Gangnet. Color-based probabilistic tracking. In *European Conference on Computer Vision*, pages 661–675. Springer, 2002.

[47] Hamed Kiani Galoogahi, Ashton Fagg, Chen Huang, Deva Ramanan, and Simon Lucey. Need for speed: A benchmark for higher frame rate object tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1125–1134, 2017.

[48] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In *CVPR 2011*, pages 1345–1352. IEEE, 2011.

[49] Louis Kratz and Ko Nishino. Tracking with local spatio-temporal motion patterns in extremely crowded scenes. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 693–700. IEEE, 2010.

[50] Branko Ristic, Ba-Ngu Vo, Daniel Clark, and Ba-Tuong Vo. A metric for performance evaluation of multi-target tracking algorithms. *IEEE Transactions on Signal Processing*, 59(7):3452–3457, 2011.

[51] Yuan Li, Chang Huang, and Ram Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2953–2960. IEEE, 2009.

[52] Bi Song, Ting-Yueh Jeng, Elliot Staudt, and Amit K Roy-Chowdhury. A stochastic graph evolution framework for robust multi-target tracking. In *European Conference on Computer Vision*, pages 605–619. Springer, 2010.

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[55] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[56] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[57] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.

[58] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[59] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[60] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[61] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[62] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[63] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[64] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *Advances in Neural Information Processing Systems*, pages 1990–1998, 2015.

[65] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. *arXiv preprint arXiv:1611.07709*, 2016.

[66] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158, 2016.

[67] Tensorflow website. `https://www.tensorflow.org/`.

[68] Keras website. `https://keras.io/`.

[69] Caffe website. `https://caffe.berkeleyvision.org/`.

[70] PyTorch website. `https://pytorch.org/`.

[71] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014.

[72] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[73] SSD-Keras. `https://github.com/pierluigiferrari/ssd_keras`.

[74] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[75] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2011.

[76] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European conference on computer vision*, pages 702–715. Springer, 2012.

Annex

MEDIANFLOW	AP @ 0,5 (%)	FPS Tracker
200x200	33,31	114,74
300x300	39,26	64,57
400x400	*43,31*	*41,21*
500x500	40,25	31,49
600x600	34,92	27,36
700x700	40,30	20,58
800x800	37,80	16,58

Table 1: Image input size experiments on MOT17-09

dl_objecttracker	AP @ 0,5 (%)	Precision	Recall	TP	FP	GT	FPS Net	FPS Tracker
MOT17-02	11,59	0,722	0,158	2953	1135	18581	0,93	31,4
MOT17-04	17,25	0,822	0,209	9943	2152	47557	0,869	23,96
MOT17-05	36,53	0,744	0,486	3367	1154	6917	0,98	37,28
MOT17-09	43,53	0,853	0,510	2717	468	5325	0,95	35,83
MOT17-10	23,26	0,797	0,274	3528	896	12839	0,943	36,18
MOT17-11	35,74	0,831	0,429	4052	819	9436	0,96	41,56
MOT17-13	14,04	0,833	0,160	1874	375	11642	0,941	42,01

Table 2: Final results on MOT17Det train set (detailed). TP: true positives, FP: false positives, GT: ground truth

Sequence	FPS	Resolution	Length	Boxes	Density	Description
MOT17-02	30	1920x1080	600 (00:20)	18581	31.0	People walking around a large square
MOT17-04	30	1920x1080	1050 (00:35)	47557	45.3	Pedestrian street at night, elevated viewpoint
MOT17-05	14	640x480	837 (01:00)	6917	8.3	Street scene from a moving platform
MOT17-09	30	1920x1080	525 (00:18)	5325	10.1	A pedestrian street scene filmed from a low angle
MOT17-10	30	1920x1080	654 (00:22)	12839	19.6	A pedestrian scene filmed at night by a moving camera
MOT17-11	30	1920x1080	900 (00:30)	9436	10.5	Forward moving camera in a busy shopping mall
MOT17-13	25	1920x1080	750 (00:30)	11642	15.5	Filmed from a bus on a busy intersection
Total			5316 (215 s)	112297	21.1	

Table 3: Description of MOT17Det train set data (from [29])