ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

MASTER OFICIAL EN VISIÓN ARTIFICIAL

Master thesis

Visual people tracking with deep learning detection and feature tracking

Author: Marcos Pieras Sagardoy

Tutor: José María Cañas Plaza

Academic course 2016/2017

Acknowledgement

En primer lugar quiero dar las gracias a mi tutor JoseMaria por su apoyo, confianza y sus *ánimos* durante estos meses de trabajo. ¡ He aprendido mucho !
Por otro lado, también querría agradecer el apoyo de mis amigos, colegas y familía durante estos meses de trabajo.

Abstract

Deep learning has rised by drastic improvements over reigning approaches towards the hardest problems in Artificial intelligence (AI), massive investments from industry giants, and exponential growth in research publications. Deep learning is a tool inside the machine learning toolbox, the goal is to make machines learn.

In some areas of artifical vision, deep learning techniques have been very succesful, however, in the field of visual tracking are not yet mature, therefore we have developed the multiple people tracking algorithm with deep learning techniques. Thus, in this work we have designed and build a software component that uses the paradigm tracking-by-detection. We mixed deep learning techniques, with feature tracking, using the Lucas-Kanade method. Combining these techniques, we make use of their advantages and reducing the effect of their drawbacks. In addition, the software component, utilize a mechanism of person reidentification.

Finally, the software component, has been validated experimentally and tested on a well-known database, Multiple object tracking dataset.

Resumen

Deep learning ha surgido por sus grandes mejoras respecto a las técnicas reinantes en los problemas más complicados en Inteligencia Artificial, inversiones masivas de gigantes industriales y por un crecimiento exponencial en el número de publicaciones científicas. Deep learning es una herramienta más dentro del conjunto de herramientas de Machine Learning, cuyo propósito es hacer aprender a las máquinas.

En ciertas áreas de la visión artificial han sido muy exitosas, sin embargo, en el campo del seguimiento visual aún están por desarrollar, por eso hemos abordado el problema del seguimiento visual de múltiples peatones con técnicas de deep learning. Así, en este trabajo se ha diseñado y construido un componente software que usa el paradigma de *tracking by detection*. Empleando técnicas de deep learning, con *tracking by matching*, usando el algoritmo de Lucas-Kanade. Combinando estas dos técnicas, recogemos sus ventajas, minimizando el efecto de sus inconvenientes. Además, el componente, también incorpora un mecanismo de reidentificación de peatones que mejora el seguimiento.

Finalmente, el componente desarrolado se ha validado experimentalmente y se ha probado en la conocida base de datos de seguimiento visual *Multiple object tracking*.

Contents

List of Figures

List of tables

Chapter 1

Introduction

As engineers we want to build systems that are better than our brain, to point out the difficulty of this endeavor, we can summarize the brain characteristics as follows: it has 100 billion computing elements, processing and memory are performed by the same components, works as parallel recurrent paradigm, it solves problems not soluble by previous machines, and it only requires 20 watts of power.

There are several computation challenges very interesting but, despite recent success in most of them, we struggle to reach brain performance and efficiency. Machines have beaten us in extracting information for large collection of data, they can process larger amount of data than the humans. Also, in memory, they beat us, they can store more information and access faster to it than humans. It is not a matter of speed computation, they also exceed us in reasoning tasks, like playing chess or Go. However, in low-level sensorimotor skills, like seeing or walking, our brains perform better than machines. These kinds of tasks, that humans perform unconsciously, for a computer are really complex to achieve.

This fact is called the Moravec's paradox, this paradox came out during the dawn of Artificial Intelligence back in the 80s when M.Minsky, R.Brooks, and H. Moravec tried to mimic human skills by reverse engineering on the brain. This paradox says that, contrary to traditional assumptions, high-level reasoning requires very little computation, but tasks involving perception, attention, visualization, motor, and social skills require enormous computational resources and are difficult to transfer to machines.

One possible explanation of this paradox, is based on evolution. Human skills are implemented biologically, improved over years of natural selection. The older a skill is, the more time natural selection has had to improve its design. In contrast, abstract

thought was developed only very recently and it is easy to implement due this shorter development.
These categories of intelligence will take much more time to implement in machines, but research keeps going.
This master thesis lies in the context of AI and computer vision, and more precisely in the problem of visual object tracking.

1.1 Computer vision

In the late 60s, computer vision began at universities that were pioneering artificial intelligence. It was meant to mimic the human visual system, as a stepping stone to endowing robots with intelligent behaviour. In 1966, it was believed that this could be achieved through a summer project, by attaching a camera to a computer and having it *describe what it saw*.
This describes the excitement of that time and their underestimation of the field. Although it is a complex area of study,there have been a lot of developments along fifty years, real world applications have been developed and are part of daily use.
One recent application of computer vision is the usage of these techniques on robotics, in particular on autonomous cars. These cars developed by technological giants are being used in some States of US. In these systems, the surrounding information is extracted by cameras. As we can observe in figure 1.9 computer vision is used to detect other types of vehicles on the road.

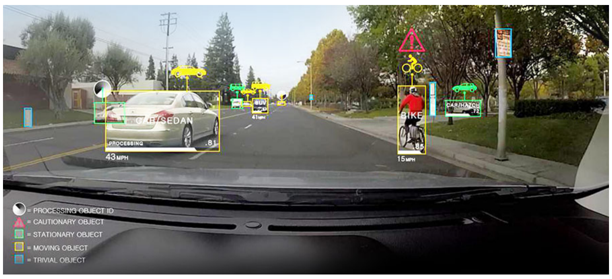


Figure 1.1: Frontal view of autonomous car.

Another computer vision's area is medical imaging, this area studies the techniques and process of creating visual representations of the internals of a body for clinical analysis. One example is the tractography map, like the one in figure 1.2 created from a diffusion weighted images, it allows us to establish connections between different areas of the brain.

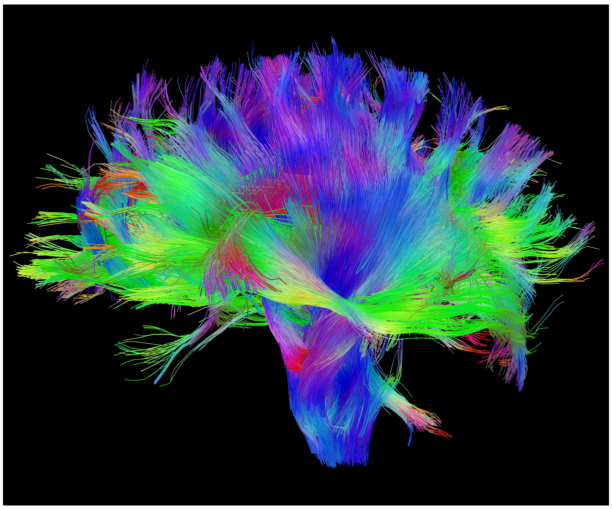


Figure 1.2: Tractography map.

One pioneer in the computer vision technology was the automation industry, where this technology is used to manufacture quicker and better. Computer vision is deeply used in factories, for instance in quality inspection of manufactured products, it can check whether a product fulfills quality characteristics, as we can observe in figure 1.3.

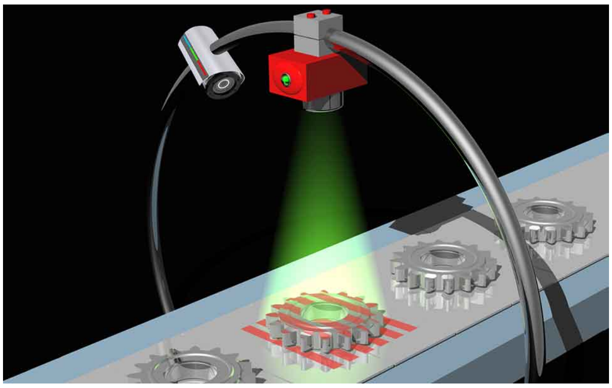


Figure 1.3: Camera for inspection.

Finally, another application of computer vision is to mix the information provided by the camera with graphics, this is called, augmented reality. One example of it is the work of the Snapchat company, it allows you to render different artifacts on an image and share it with your friends. We can observe one example on figure 1.4.

Figure 1.4: Augmented reality image.

1.2 Object tracking

In the widely computer vision field there are several study areas, one of them is Object Tracking. It estimates target state over time from image sequences. As state we can embed the position, velocity, shape, appearance or any other interesting characteristic. It is very challenging field due to:

- Variations because of geometric changes, some targets might be deformed as they move in the scene which would change their structure.
- Variations due to photometric factors, the appearance of the targets might change due to changes in illumination.
- Occlusions, targets might mix with other elements of the scene from the camera perspective.
- Image quality, the image sequences could incorporate noise or low resolution.
- Similar objects in the scene, this could cause problems to maintain the identity of targets.

To solve these problems the community has used the several paradigms:

- **Tracking using matching**, this kind of methods performs a matching of the representation between the current and the possible candidates in the next frame. Key points of these methods are the representation and the similarity measurement helps to perform the matching. Maybe the most famous methods are Normalized Cross-Correlation [1], Lucas-Kanade tracker [2], Kalman appearance tracker [3] and Mean shift tracking [4].

- **Tracking-by-detection**, this kind of methods builds a classifier to distinguish target pixels from the background. Once you have the detection, you need a data association method to link those detections. Traditionally the community has used kernel methods with support vector machines [5] to perform the detections, but in the recent years people are shifting to neural networks. In the data association algorithms graph theory techniques are dominant [6] [7].

- **Tracking learning and detection**, this is an extension of the previous category. It includes a mechanism to update the classifier during the execution of the system. This learning procedure allows the algorithm to be invariant to changes in the target. Maybe the most famous algorithms are the Predator [8] and the Alien [9].

These kinds of algorithms are quite mature and are deployed in real life applications. Like several the computing applications they allow us to process a huge quantity of information really quickly.
In video surveillance, Object Tracking allow us to track all the targets without human intervention and notify when there are dangerous situations.

Figure 1.5: Control room.

For science, they allow us to study the environment, in the case of the figure 1.6, for humans it will be difficult avoid missing the correct identity of any ant. With information supplied by the tracking, scientist can study how animals move and interact with others.

Figure 1.6: Visual tracking for science.

These algorithms are deeply used in all kind of sports like the NBA and NFL. In these situations the algorithms track the players during the game and allow to analysis his/her performance and the strategy of the team, like in figure 4.3 .

(a) Input image (b) Layout

Figure 1.7: Visual tracking for sports analysis.

In artistic performance, these algorithms are used to track the subject and render some graphics in the scene, like an extension to video of augmented reality. In the example of the figure 1.8, the systems track the singer's head and projects a visualization of his voice.

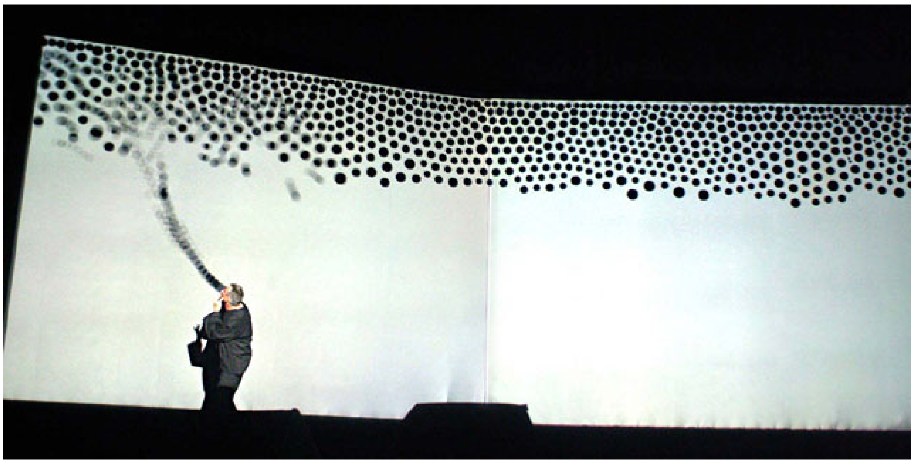


Figure 1.8: Visual tracking for art.

1.3 Deep learning in computer vision

Deep learning has raised by drastic improvements over reigning approaches towards the hardest problems in Artificial intelligence (AI). It has gathered massive investments from industry giants, and exponential growth in research publications. Deep learning is a tool inside the machine learning toolbox, whose goal is to make machines learn.
The first incursion was made by Frank Rosenblatt, the Percetron [10]. Rosenblatt conceived the Percetron as a simplified mathematical model of how the neurons in our brains operate. This model of the neuron built on the work of McCuloch-Pitts [11], who showed that a neuron model could replicate the basics OR/AND/NOT functions. That was great in the early days of Artificial intelligence, because the predominant thought at that time was that making computers able to perform formal logical reasoning would essentially solve AI. However, the McCuloch-Pitts model lacked a mechanism for learning, which was crucial for it to be usable for AI. This is were the Perceptron suceeded, Rosenblatt came up with a way to make such artificial neurons learn, inspired by the Hebb's Rule. This learning method was as follows: if the output of the perceptron was low, increase the weights, otherwise decrease the weights if the output is too high. Also, another researchers came with ADALINE [12] learning procedure. They used the signal before the activation function to compute the derivative, how much the error changes when each weight is changed can be used to drive the error down and find the optimal weight values. This is similar to the way we train the networks nowadays.
Researchers were really excited about this idea of Connectionism: those networks of such

simple computational units could be vastly powerful and solve the hard problems of AI. But in 1969, Minsky and Papert published an analysis on the limitations of perceptrons [13]. The biggest criticism was that a perceptron could not learn the simple boolean function XOR because it is not linearly separable. However, they stated that it could be learnt with multiple layers perceptron but the learning procedure did not work for multiple layers. After this book, the interest on Neural networks decreased, and it initializes a period called *AI winter*, AI shifted to logic programming and common sense reasoning. This period lasted till 1986 when Rummelhart, Hinton, and Williams published the algorithm of backpropagation [14], which specifically addressed the problems discussed by Minsky in Perceptrons, a method to train multiple layer neural nets. With this discovery in 1989, LeCun showed a real world application, recognized handwritten digits [15]. The architecture of this model was a convolutional neural network. It was inspired by the Neurocognitron [16] of Fukushima, which took ideas from studies of the brain. In particular, the studies of Hubel and Wiesel, they propose that the visual cortex is formed by a hierarchical model, primarily for simple cells that respond for simple structures and then complex cells that respond to a more complicated feature. As we can observe in figure 1.9, in the lower layer, the network learns Gabor like features, and while going upwards, the networks learn more abstract concepts.

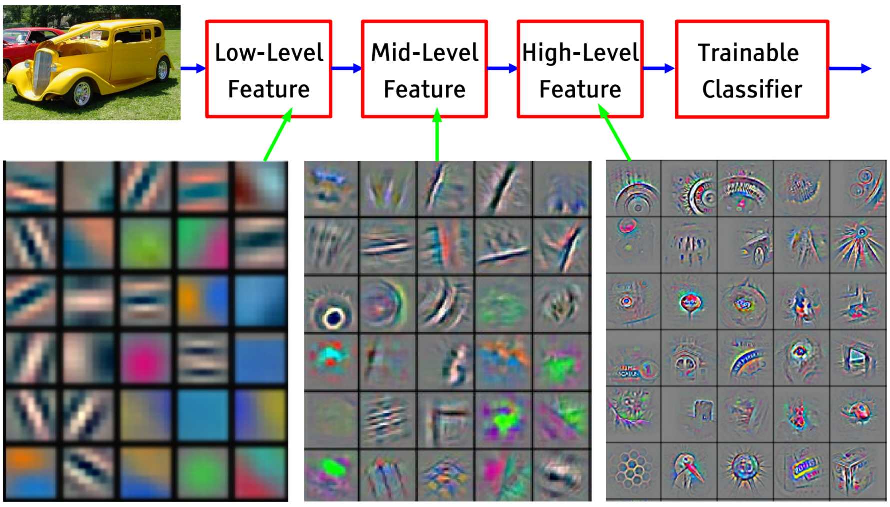


Figure 1.9: Representation of a Convolutional neural network.

But this approach did not scale to larger problems, the biggest source of problems was the vanishing gradients problem. When the backpropagation gradients backpropagates trough the network, in some nodes the local gradient is very low (in the extreme of the sigmoid functions) and the signal vanishes or saturates. By the 90s other techniques became the method of choice, like support vector machine (SVM), although some progress was made for other kinds of problems.

- Unsupervised learning. This type of architectures is used to find a smaller representation of some data from which the original data can be reconstructed, it is useful for compression, visualization, and classification. One example of this architecture with neural networks is the Restricted Boltzmann machine [17], developed by Hinton.
- Reinforcement learning. The goal of this type of learning is to learn how to make good decisions, it requires rewards, not labels. One example of this sort of systems, is the TD-Gammon [18], a neural network that learned to be a backgammon player.
- Recurrent neural networks. Plain neural networks could not process sequences due to they do not have memory, they need mechanism to remember the pasts outputs. With memory, it can process sequences like audio or text. One approach to this is Waibel [19] in 1989.

In 2006, there was a breakthrough [20], Hinton realized that a neural network with many layers really could be trained well if the weights are initialized in a clever way. The basic idea was to train each layer one by one with unsupervised training (like an autoencoder architecture) and finally stack all together and train it in a supervised way.
Although these improvements, the big step forward came in 2012, when AlexNet [21] beat the state of the art in the ImageNet challenge, an image classification challenge, where the error rate was 15.3% whereas the winner of the previous year was 26.3%. In the figure 1.10 we can observe the advance in the state of the art of the ImageNet challenge with the inclusion of deep learning techniques.

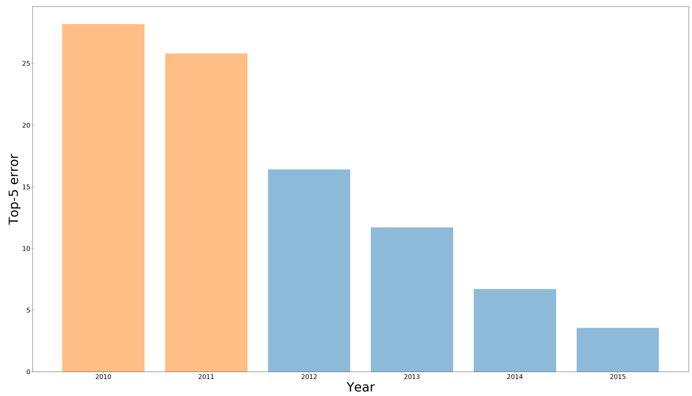


Figure 1.10: Classification error in ImageNet challenge.

The emergence of these techniques were the culmination of decades of research but the step forward was due by three aspects:

- **Appearance of large and high quality datasets**, The increasing size and quality of the datasets help the networks to converge easily.
- **Parallel computation**, The increasing of computing capabilities helped train larger models in less time.
- **Optimization details**. With the discovery of the proper initialization and activation functions larger networks can be trained.

Chapter 2

Objectives

Once we have put in context our work, in this chapter we explain the objectives of this thesis, its requirements and the methodology to accomplish them.

2.1 Description of the problem

The main objective of this thesis is to develop and characterize an algorithm of multiple people tracking merging two diferents methods: deep learning techniques and feature-based tracking. The essence of this work is to study how combine them, and reach real time operation and high performance while keeping robustness at the same time. Finally, validate our solution with a international dataset, the Multiple Object Tracking dataset. We divided this target into several sub-objectives:

- **Object detector using deep learning**. Study the fundamentals of object detectors with deep learning techniques. Analyze the performance on the main datasets and finally, we choose one to our task.

- **Development of a tracking module**. Study of which feature-based tracking technique would fit best our problem, and implementation of it in software code.

  We studied which tracking technique would fit our problem, when it was selected, we implemented on code.

- **Integration of these two techniques**. We would integrate these two techniques to perform a complete robust and fast tracking algorithm.

- **Test the component on an international databases**. We should validate our solution on well-known international databases already used in the scientific community.

2.1.1 Requirements

In addition to the previous objectives, our solution must also hold the following requirements:

- The solution will make use of the JdeRobot framework, release 5.5, which is the developing environment of the Robotics Laboratory of *Universidad Rey Juan Carlos*.
- The software will run on the GNU/Linux Ubuntu 16.04 environment.
- The algorithm will only make use of video sequences, not other information.
- The algorithm must achieve an execution on real time and guarantee a precision.

2.2 Methodology

To achieve our objectives we have used several tools that helped to monitoring the project for all members of the team. They allowed to comment or correct the task. The main tool has been the videoconference. We established a weekly meeting with all the members of the team. In these meetings we showed the results so far and shared our feedback with the other members of the team.

As complementary tools, we used a website and Github repository, they helped to control the development of our work. The website was developed using the wiki of JdeRobot [22], and it shows the weekly tasks and results. The Git Hub repository [23] allows to access to the code by all the members of the team.

Our development plan was based on the spiral model. It consists of four steps per iteration. In the first step, we determine the objectives of that development iteration, in the second one, we analyze the risks and evaluate which problems we will face, then we develope and test our prototype and the last step evaluates the results. We apply several iterations of this process till we get a satisfactory final project and the global targets are achieved.

Chapter 3

Theoretical background

In this chapter we explain the theoretical concepts of our work, these include the theory of tracking and person re-identification.

3.1 Tracking

As we explained in previous chapters, there are a traditional family of methods to solve the tracking problem. But with the incursion of deep learning techniques, they have been adapted to it or create new paradigms. The main ways to apply deep learning techniques to tracking are the following [24]:

- **Tracking-by-detection**. These methods use a specific class classifier and there is not need to train it online. So, these methods use a neural network to extract instances of the frames and then linked with temporal restrictions.
- **Tracking learning and detection**. Starting from the first frame of a video, a tracker will sample patches near the target object, and they are used to train a foreground-background classifier, and this classifier is used to score patches from the next frame to estimate the new location of the target object. These methods showed a state-of-the-art performance results. Unfortunately, neural networks are slow to train, therefore the speed of the method is reduced [25] [26].
- **Siamese based tracking**. In this approach, many candidate patches are passed through the network, and the patch with the highest matching score is selected as the tracking output [27].

- **Tracking as regression**, these methods are an extension of object localization using neural networks, these methods given an image predict the bounding box which contain the object in every frame [24]. They are restricted to one object.
- **Tracking with RNN**. From the output of an object detector, these tracking algorithms model the sequence of movement of objects using an recurrent neural network [28]. These methods represent the current state of art in tracking.

For this thesis we chose the tracking-by-detection paradigm. We get the detections with an object detector based on deep learning networks, and we link those detections with a tracking by matching, particularly, feature-based tracking.

3.1.1 Detection in tracking

In object detection too, the emergence of the neural networks has supposed a turning point. As we can observe in 3.1, the mean average precision, has almost doubled since the appearance of deep neural networks.

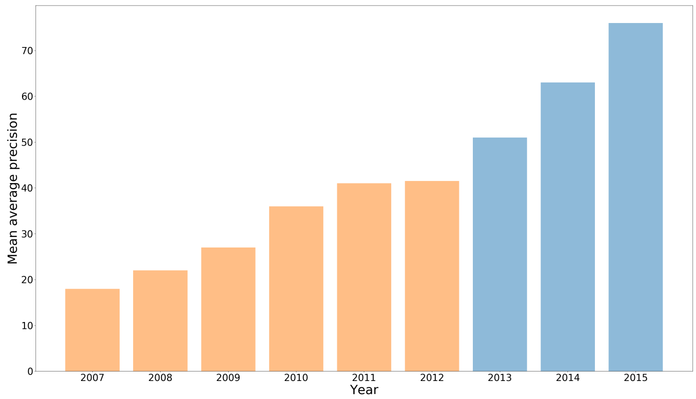
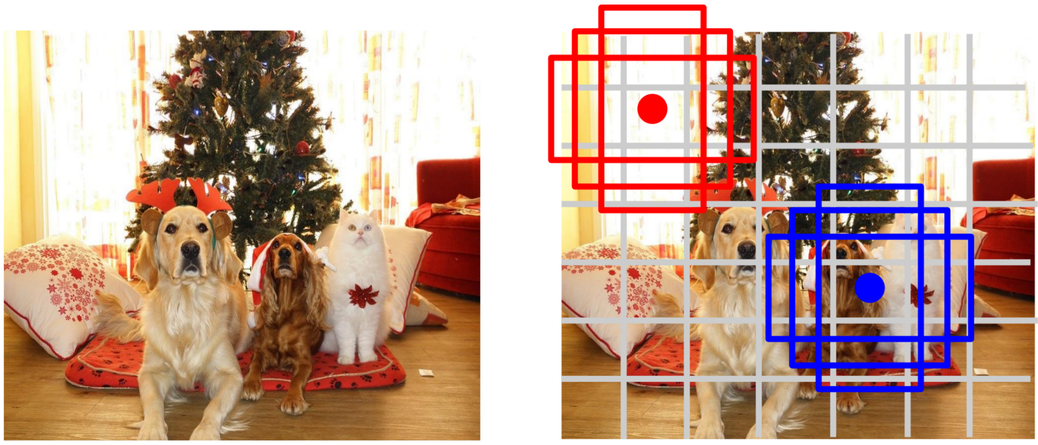


Figure 3.1: Mean average precision over the years in PASCAL dataset.

Present deep learning object detectors are based on three main family of architectures [29], named by the reference algorithm of the category: FasterRCNN, SSD, and RFCN, the characteristics of these systems are:

- Faster RCNN [30], it is the last output of a trilogy of detectors developed by R. Girshick and his team. Which are called Region-Based object detectors. They work as follows: Use some mechanism to extract region of an image that are probable to be an object and then classify those proposals with a CNN. The first paper to do so, was [31], and suppose a breakthrough in the field, increasing the precision of the state of the art of those days. But, it had a messy pipeline, slow and difficult to train. Later on, they developed [32], in this paper they applied the region proposal algorithm in the cnn feature map, so, they avoid to compute the features for each proposal. They increase the speed and it could be trained much easily. Finally, they showed FasterRCNN [30], in this algorithm, they eluded the external region proposal algorithm and they implemented a CNN to compute those proposals. This CNN share parameters with the main net and they saved a lot of time. This network, has become the standard object detector with CNN. With the association of novel net architecture like ResNet [33], Inception [34], and [35] they have won all the contests.

- SSD, it stands for Single shot multibox detector. These family of method differs from previous ones considering that these treats the problem of object detection as a regression problem. So, they are called Regression-based object detector or single shot object detector due it does not have a region proposal algorithm, they classify the image with one mechanism. The maximum exponent of these algorithms are [36] and [37]. These work as follows, they discretize the image at the features level in a fixed grid and for each grid it predicts a class and some number of bounding boxes with different shapes and sizes. It merges all, and apply a Non-Maximum suppression algorithm to obtain a set of detections. We can observe this process in 3.2. In addition, they apply this process in a multiresolution scheme as we can observe in 3.3 to deal with objects of different sizes.



(a) Input Image. (b) Divide image into grid.

Figure 3.2: SSD detector scheme.

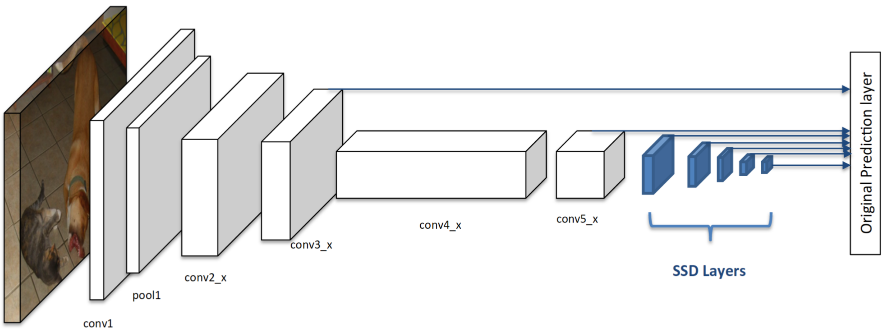


Figure 3.3: SSD architecture.

- RFCN [38], it stands for Region-based fully convolutional network and it was developed by the same authors of SSD. They noticed the lacks of the SSD, the SSD algorithm computes the object detector on the feature map, and at this level the features have a low spatial resolution, this involves do not detect small objects. So the authors inspired by the fully convolutional architectures, upsample those feature maps and compute the object detector like the SSD algorithm.

In the survey [29], they compared the different methods including changing the features extractors (ResNet, Inception, VGG) and they measured the precision (mean average precision) and computing time. This results are showed in 3.4. The conclusion are as follows, SSD is the fastest detector, RFCN it has the best balance between speed-accuracy, and FasterRCNN, is the most accurate detector although is slower than the other ones.

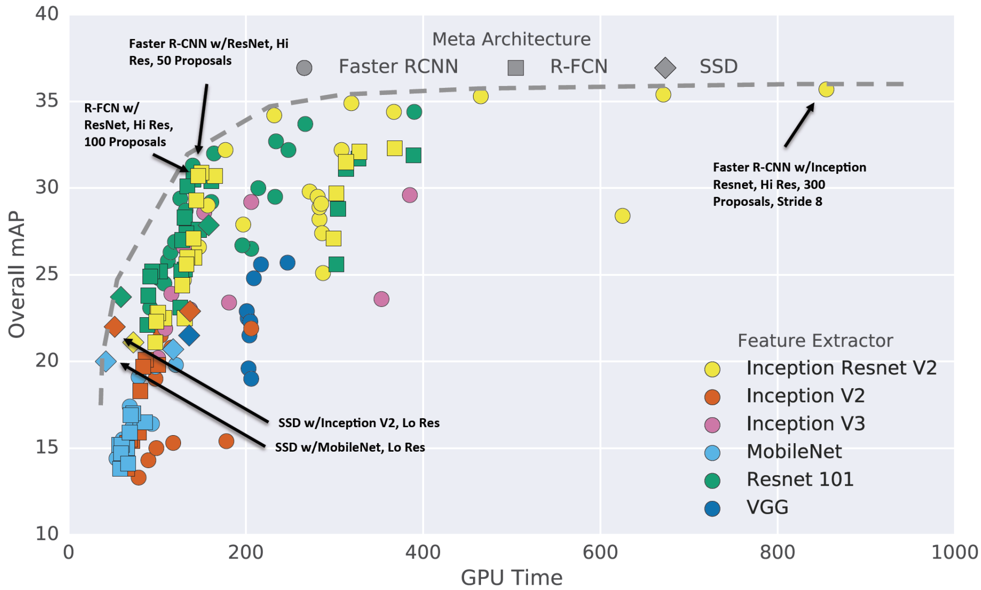


Figure 3.4: Comparison architectures.

We will finish off our review with a numeric comparison of the methods, as we can observe in the table 3.1. This information is extracted from the original papers with their implementation, all of them are trained with the union of the training set of VOC07, VOC12, and COCO, and subsequently evaluate on VOC07 test set on a Nvidia Titan X GPU. These results give us an intuition on which detector will be suitable for our task.

	mAP	**mAP_person**	**FPS**	**Proposals**
RCNN	66	64.2	0.077	2000
FastRCNN	70	69.9	6.7	2000
FasterRCNN	85.6	82.3	7	6000
SSD300	81.2	81.4	46	8732
SSD512	83.2	84.6	19	24564
YOLO	66.4	63.5	45	98
YOLOv2	78.6	81.3	40	-
RFCN	83.6	-	10	-
PVANET	84.9	-	31.3	300

Tabla 3.1: Summarize of the object detectors.

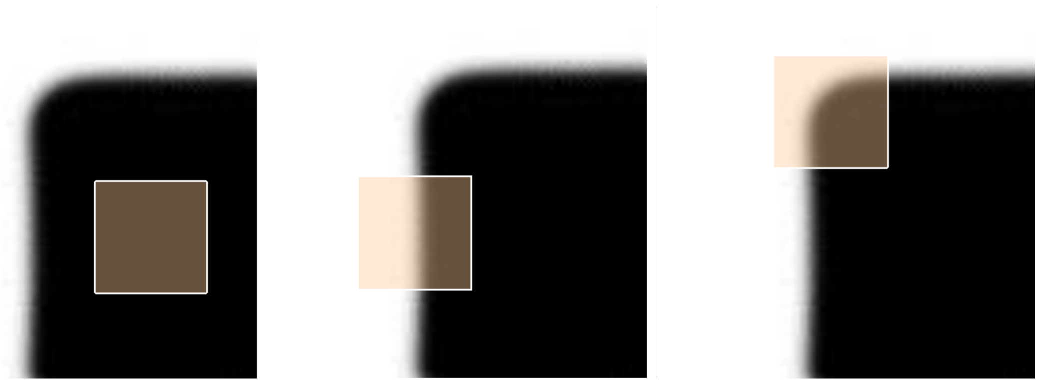
3.1.2 Feature tracking

3.1.2.1 Features

Our goal is to find points in an image, which can be found in other images and then compute some information, in this case, the movement. The characteristics of good features are:

- Repeatability, the same feature can be found in several images despite geometric and photometric transformations.
- Matchability, each feature has a distinctive description, thus easy to find.
- Efficiency, few features have to compact much more possible information.
- Locality, a feature occupies a relatively small area of the image, so therefore it is robust to clutter and occlusion.
- Performance, computation speed of features is a critical parameter.

Features points are used in all sort of operations in computer vision: Image alignment, 3D reconstruction, Motion Tracking, Object recognition, Index database retrieval, robot navigation and so on.
Looking at the figure 3.5, the flat patch, is a patch without texture and impossible to localize. Patches with large contrast edges are easier to localize, although straight lines segments at a single orientation suffer from the *aperture problem*, are also impossible to localize. Finally, patches with large gradients in at least two different orientations are the easiest to localize.



(a) Flat region. (b) Edge region. (c) Corner region.

Figure 3.5: Types of patches.

These intuitions can be formalized by looking at the simples possible matching criterion for comparing two images patches, their weighted summed square difference:

$$E(u) = \sum_i w(x_i)[I(x_i + u) - I(x_i)]^2$$

where $I(x)$ is the image, $I(x+u)$ is the shifted image, and $w(x,y)$ is a window function like a box or gaussian kernel around the pixel, and the summation i is over all the pixels in the patch. Then we are looking for points, which if we move according to u we have a change.

When performing feature detection, we do not know which other image locations the feature will end up being matched against. Therefore, we can only compute how stable this metric is with respect to small variations in positions Δu by comparing an image patch against itself:

$$E(\Delta u) = \sum_i w(x_i)[I(x_i + \Delta u) - I(x_i)]^2$$

Using a Taylor series expansion of the image function $I(x_i + \Delta u) \approx I(x_i) + \nabla I(x_i) * \Delta u$ we can approximate the expression as follows:

$$E(\Delta u) \approx \sum_i w(x_i)[I(x_i) + \nabla I(x_i)\Delta u - I(x_i)]^2$$

$$E(\Delta u) = \sum_i w(x_i)[\nabla I(x_i)\Delta u]^2$$

With algebraic notation it transforms to:

$$E(\Delta u) = \Delta u^T M \Delta u$$

where $\nabla I(x_i) = [I_x, I_y](x_i)$ is the image gradient and M is the second moment matrix:

$$M = \begin{pmatrix} I_x^2 & I_{xy}^2 \\ I_{xy}^2 & I_y^2 \end{pmatrix}.$$

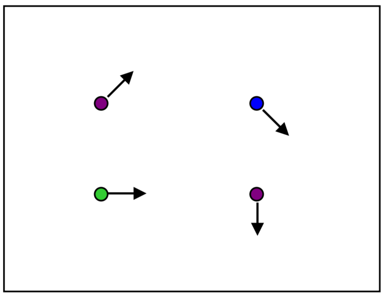
Computing the eigenvalue decomposition of this matrix, shows the directions of the fastest change, thus a measure of the *cornernes*. There are several algorithms that use in different ways this eigenvalues:

- Harris [39], they propose a corner detection response function. So for each pixel, they compute a matrix M and with it, they compute the function R, $R = det(M) - a\, trace^2(M)$. if R is large, that pixel is a corner, if R is negative with larger magnitude, it is a an edge, and if R is small it is a flat region. So the they a threshold to classify those pixels as a corner.

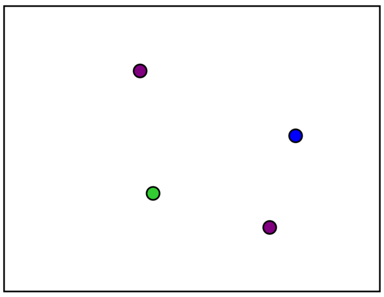
- Shi-Tomasi [40], they define the *cornerness* in another way. The image has a maximum value (e.g. 255), so λ_1, λ_2 also have an upper bound, then it is only necessary to check that $min(\lambda_1, \lambda_2)$ is large enough, this is how they define *cornerness*. This feature is called good features to track, because the authors defined a *good* features those whose motion can be estimated reliably, and they reached the same conclusions as Harris. This method is implement in the OpenCV's routine `goodFeaturesToTrack()`.

3.1.2.2 Motion estimation

Now, we have invariant points, we want to estimate the motion of those points. In order to do so, we compute the optical flow. This is the apparent two-dimensional motion of brightness pattern in the image. In the next figure 3.6 we visualized this idea.



(a) $I(x, y, t)$.



(b) $I(x, y, t + 1)$.

Figure 3.6: Optical flow example.

So, the question is: How do we estimate pixel motion from image $I(x, y, t)$ to image $I(x, y, t + 1)$. We need to solve the pixel correspondence problem. Given a pixel in $I(x, y, t)$, look for nearby pixels of the same color in $I(x, y, t + 1)$. Solving this problem is what is referred as the optical flow problem. By nearby pixels and same colour we have two assumptions:

- Colour constancy: a point in $I(x, y, t)$ looks the same in $I(x', y', t+1)$. For grayscale images, this is called *Brightness constancy constraint.* Stated in mathematical formulation:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

$$0 = I(x + u, y + v, t + 1) - I(x, y, t)$$

- Small motion: Subsequent points do not move very far, so we can estimate the motion by Taylor expansion:

$$I(x + u, y + v) \approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + higher\ order\ terms$$

Then, combining these two equations, we get:

$$0 \approx I(x, y, t + 1) + I_x u + I_y v - I(x, y, t)$$

where $I_x = \dfrac{\partial I}{\partial x}$, isolating the terms we obtain:

$$0 \approx [I(x, y, t + 1) - I(x, y, t)] + I_x u + I_y v$$

$$0 \approx I_t + I_x u + I_y v$$

In the limit of t, u and v approaches zero (assumption of small motion), so it becomes, what it is called the the *brightness constancy constraint equation*:

$$0 = I_t + I_x u + I_y v$$

If we look closely, we realized that we have two unknowns u, v and one equation. This is an underdetermined system. Intuitively, this means, that locally we can only determine the component of the flow in the gradient direction, the component of the flow parallel to an edge is unknown, this is the called the aperture problem. To recover the motion we need to add some extra constraints. There are several types of constraints to solve this problem:

- **Global constraint**, adding a smooth constraint to the brightness constraint, this new constraints penalizes for changes in u and v over the images, it assumes that the motion fields vary smoothly over the image. This approach was developed by Horn and Schunk [41].
- **Local constraint**, locally the motion field is almost the same, so we add the neighbours pixels to the equation. This approach was developed by Lucas and Kanade [42].

Local constraint

In this thesis we use the Local constraint to solve the optical flow problem. As we stated above, we add a local constraint to get more equations, this assumes that the motion field is the same in the locality. From the brightness constraint equation:

$$0 = I_t(p_i) + \nabla I(p_i)\,[u\ \ v]$$

Adding the neighborhood equations:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_t(p_1) \\ \vdots \\ -I_t(p_n) \end{bmatrix}$$

Now, there are more equation than unknows, it is an overdetermined system, we have to solve it with the least squares technique. It is based on the optimization of the function:

$$(A^T A)\, d = A^T\, b$$

Using the image notation:

$$\begin{bmatrix} \sum I_x I_x & \sum I_y I_x \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

The system has a solution when A^tA is invertible, it will be invertible when is well conditioned, this is when the ratio of the great and the small eigenvalues of the matrix is large but no too much. The matrix A^tA in terms of image formulation is the second order matrix that we stated in the section 3.1.2.1 developing the *cornerness*, then in order to be solvable it should have a strong gradient in both directions. After checking the invertability, we can solve the problem and extract the motion field:

$$d = (A^T A)^{-1} \, A^T \, b$$

Thus, using the image notation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_y I_x \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix}$$

In practice motion is large, the assumption that it is small fails, consequently the approach using Taylor expansions. For two reasons, the linearity does not hold, in order to solve it, we apply an iterative refinement, which consists in compute the displacement, apply it to the pixels, and compute it again till it converges. The other one is there are local minimum and it will fail into it. To solve it, we need to utilize a coarse to fine approach, the idea is to use multiresolution to compute optical flow, the basic is that in a low resolution image the motion between pixels is very small and we can compute optical flow.
So, in order to do so, we use image pyramids, this consists in downsample these images to specific resolution, then in top level, we compute the motion field using the previous stated method, then we upsample the motion field and the images, We apply a transformation to one image according to the motion field computed in the previous level and then compute the optical flow between that transformed image and the other image, we apply this algorithm in all the resolutions

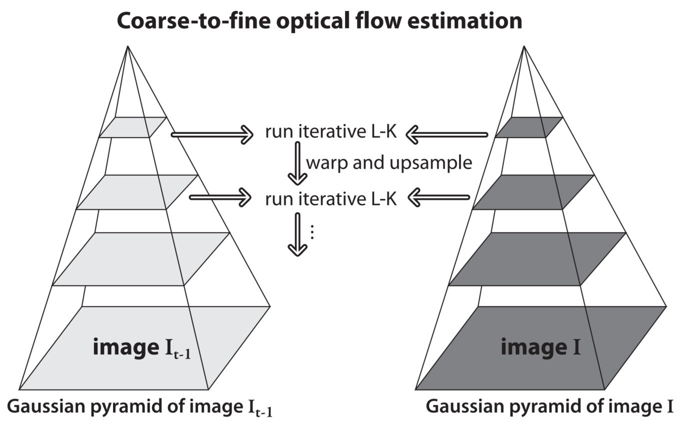


Figure 3.7: Optical flow with pyramids.

3.2 Person reidentification

One of the problems of tracking is the reidentification of pedestrians. Add a module of reidentification helps to maintain the identity of the pedestrians.
Person identification is thoroughly studied in the field of Biometrics, it consists of knowing one biometric characteristic, and comparing with a claiming identity query. Specifically, the topic of pedestrian identification based on images has raised in the last years, this is due to the growth of surveillance applications. Also inside the topic of tracking there is a subfield called data association which studies this problem, it consists in matching trackets with further pedestrian detections.
Let's define mathematical the problem, consider G is a gallery composed of N images, denoted as $(g_i)_{i=1}^{N}$. They belong to N different identities $1, 2, ..., N$. Given a query image q, its identity is determined by:

$$i^* = \underset{i \in 1,2,\ldots,N}{\arg\max} \quad sim(q, g_i)$$

where i^* is the identity of probe q, and $sim(.,.)$ is some kind of similarity function. There are several categories of this similarity function [43]:

- **Hand-crafted systems**. This involves two components, an image descriptor and a distance metric algorithm. The most common image descriptors are those used in computer vision too, like colour [44], texture [45], SIFT [46], bag of word [47]. The general idea of distance metric learning is to keep all the vectors of the same class closer while pushing vectors of different classes further apart. The most commonly used formulation is based on the class of Mahalanobis distance function [48], [49]. Other works focus on learning discriminative subspaces [50].

- **Deep learning techniques**. Two types of CNN models are commonly employed in the community, the first is the classification model as used in image classification, the output is an identity label, and the second one is the siamese model using image pairs as input. The major drawback of the classification models is that they need a great quantity of training data by category, and most of the identifications datasets only provide a few examples for identity. So currently methods focus on siamese models.

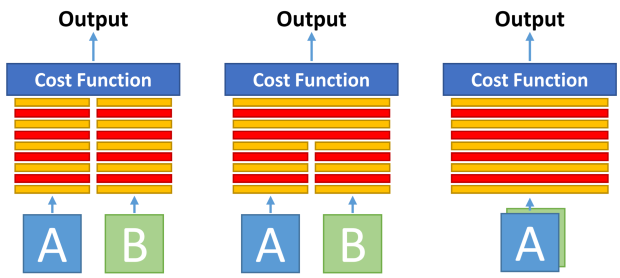
The main differences between them, is that in hand-crafted methods, feature representation of the data and the metric are not learned jointly, instead, deep learning techniques jointly optimize the representation of the input data conditioned on the *similarity* measure being used [51].

3.2.1 Siamese networks

The first work with siamese architectures were developed by LeCun [52], [53] and they addressed the identification of signatures, besides the siamese networks are used in a variety of problems like: image recovery [54], feature descriptor [55], comparing patches [56], one shot learning [57], and learning visual similarity [58].
Siamese CNN topologies can be grouped under three main categories, depending on the point where the information from each input is combined:

- **Cost function**. Input patches are processed by two parallel branches featuring the same network structure and weights. Finally, the top layers of each branch are fed to a cost function.
- **In-network**. The top layers of the parallel branches processing the two different inputs are concatenated and some more layers added on top of that.
- **Joint data input**. The two input patches are stacked together forming a unified input to the CNN.

Graphical we can observe those differences in 3.8.



(a) Cost function. (b) In-network. (c) Joint data input.

Figure 3.8: Siamese CNN topologies.

While the two first approaches have yield good results and historically were dominant, the best performance is obtained with the joint data input strategy. As pointed out by [56] and further corroborated by [59], and [60], jointly using information form both images from the first layer tends to deliver a better performance.
In the field of person re-identification, the community has used these architectures, and they also, have developed their own loss function, what is called *contrastive loss*, this loss is an extension of the Hinge loss of the SVM. This loss longs for getting close similar pairs and moving away according to one defined margin, dissimilar pairs. Although, the binary cross entropy is used by the community. Also, the community has focused in the developing of the datasets, increase the size and quality but there are not any landmark dataset.
There are several papers in the literature, one of the most famous is developed by Ahmed [61], they used In-network architecture although in order to join to the convolutional layers, they used *cross-input neighborhood differences* layer, this layer tried to increase the differences between the features of the inputs and obtain richer representation to the classification layer.
Another paper was published by Leal-Taixé [60], they are also the authors of the MOT challenge, their network used a cost function architecture besides they used as inputs the two images and their optical flow. They used the network as part of a data association algorithm.

Chapter 4

Software implementation

In this chapter, we explain the algorithm that we have designed for solving the visual people tracking problem and its software implementation.

4.1 System overview

The main contribution of this work is to develop a robust pedestrian tracking algorithm that utilizes both a neural network and does not miss the real time operation. To do so we use the tracking-by-detection framework, and combine people detection using a neural network, somehow slow but very accurate, and a regular feature tracking, very quick but prone to drift.
The architecture of the system is summarized in the diagram 4.1. Its input is a sequence of frames coming from a directory and its output is a CSV file. This file has the structure that the MOT's evaluation software requires. We divided the computing in two threads, the *object detector thread* and the *feature-based tracking thread*. The first is responsible of computing pedestrian detections using a neural network and sending them to the *feature-based tracking thread*. The second one computes the tracking procedure frame to frame. In addition, when a new detection appears it is combined with the blobs of the tracker, this is called *data association*. There are two different data associations in our algorithm. First, one in the feature-based tracking thread between the blobs in the previous frame and the new features in the current frame. Second, one that links the blobs of the feature-based tracking with the new pedestrians detected by the neural network in the object detector thread. This is different to the data association concept in the tracking-by-detection nomenclature. In our algorithm the data association, is the feature-based

tracking module which links the detections with the blobs.

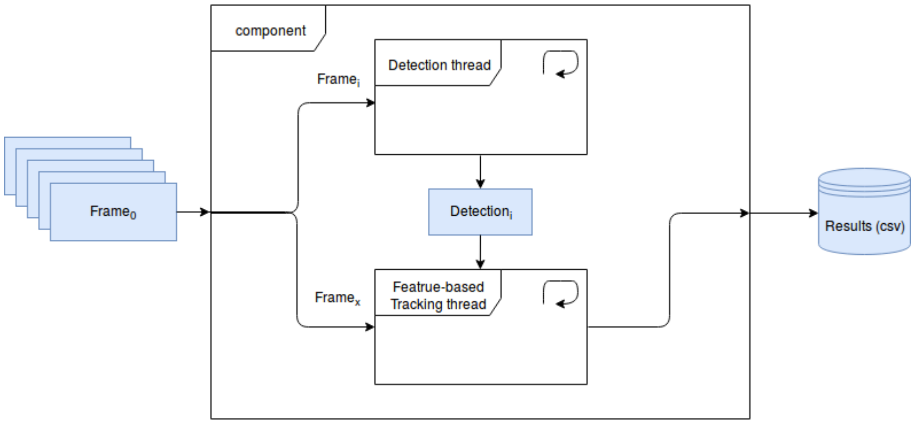


Figure 4.1: Block diagram of the component.

The system works as follows. When the algorithm starts, first of all it launches the *object detector thread*, which begins to compute the detections on the first frame. Meanwhile, the *feature-based tracking thread* remains idle, waiting for the initial detections. When the object detector finishes for the first time, it sends the initial detection to the *feature-based tracking thread* through a buffer and at the same time starts to compute the detections on the next frame. When the *feature-based tracking thread* receives from the buffer, it begins to compute the tracking between frames. Thus, the object detector thread introduces a suspension on the tracking system and in order to synchronized them we need to introduce a controlled delay on the *feature-based tracking thread*. With this controlled delay, we are able to mix the *feature-based tracking* estimation with its correspondent temporal detection. We can not process all the frames with the neural network, it would delay to much the system, the neural network is able to throw a detection every 30 frames, so we sample the sequences of frames every 30 frames. The process of mix the detections with the tracking estimation it is called data association module, in addition it has a person re-identification module, to solve some possible identity incongruities.
We can observe this temporal process in the next figure 4.2, when T represents a temporal step. With this controlled delay we are able to mix the detections of the neural network and do not miss the real time operation. When the object detector thread finishes computing all the detections it will *die* and when the tracking thread processed all the frames it dies and the component too, it has finished the work.

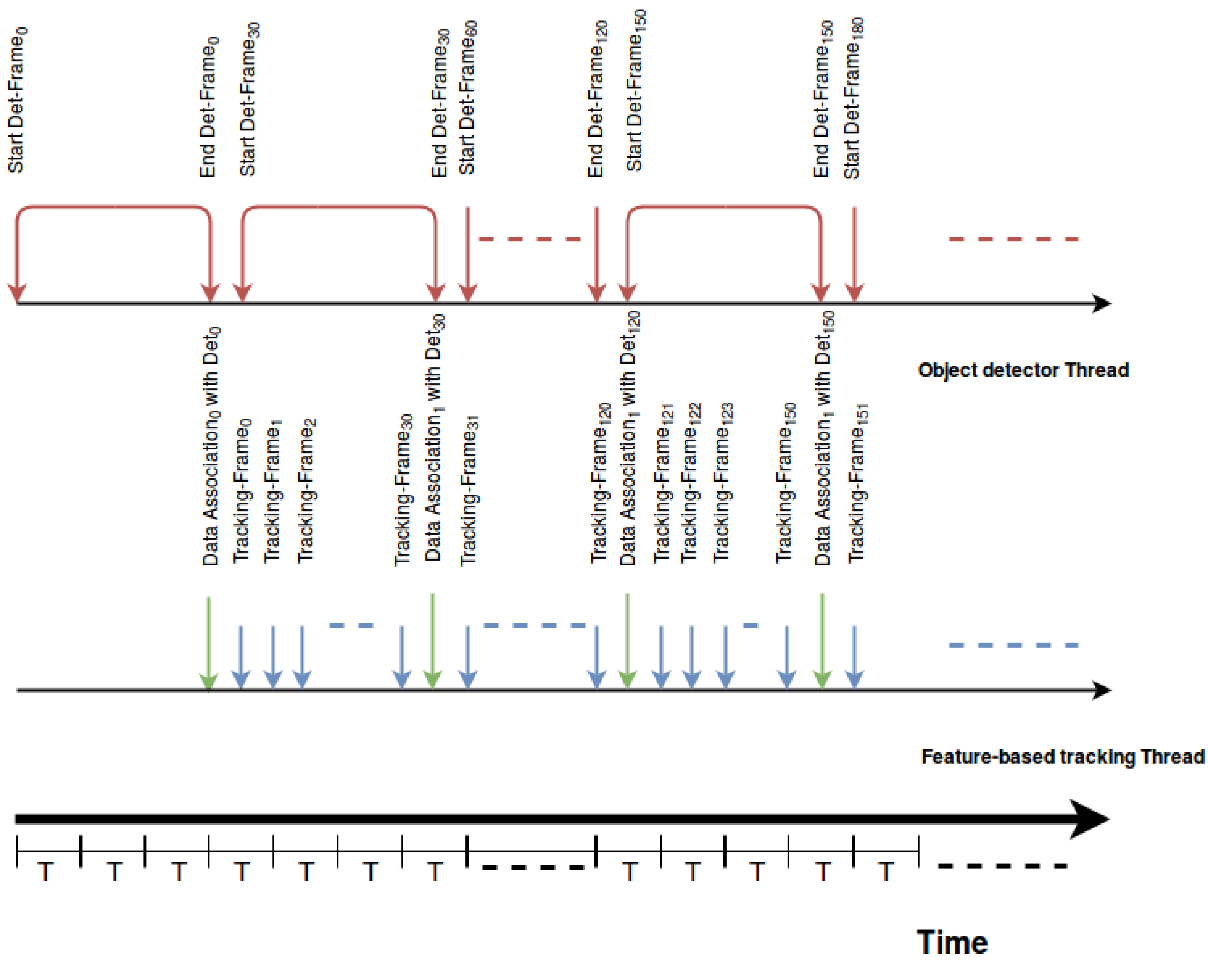
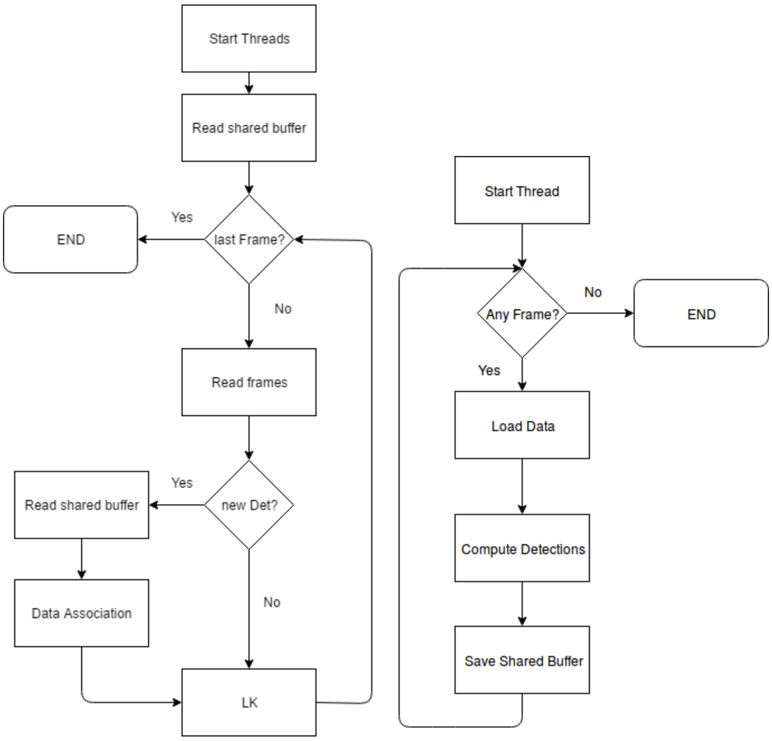


Figure 4.2: Timing of the component.

In the figure 4.3 there is a flow chart of the algorithm. The object detector thread reads images, processes the forward pass of the neural network and saves the detections in the shared buffer, it repeats this sequence until it has processed all the periodic detections. In another hand, the main tread activates the object detector thread and waits till it gets the first detection, after this, it starts the tracking algorithm. It reads the images and computes the motion of all the regions of interest. However, at beginning of each cycle it checks whether it has got newer detection coming from the *object detector thread* to mix it in.



(a) Tracking thread. (b) Object detector thread.

Figure 4.3: Flow chart of the system.

We represent each person with a bounding box, in this bounding box we extract some features and compute how they move through the frames. Based on the movement of those features we will infer the movement of the bounding box, therefore, the movement of the person. This process is represented in the figure 4.4.

(a) Detection. (b) Points. (c) Displacement.

Figure 4.4: Image and motion vectors of a moving camera sequence.

With this approach we accomplish to join two different technologies, we can exploit their benefits and reduce their drawbacks. We can get an accurate detection every 30 frames

and in between, we link those detections with the feature-based tracking module. In this way, we reduce the fragility of the tracking, which without the detections it tends to miss feature points and cause a drift of the estimation. With the periodic neural-net based detection, that corrects that drift. In addition, we compensate the slowness of the neural-net based detection with the speed of the feature tracking.
Next we explain each part in detail.

4.2 Object detector thread

We compute the pedestrian detector based on a CNN. This type of systems is very accurate but slow. We are constrained by execution time of the chosen detector, it takes 0.92 seconds for compute each detection, this allows us to get a new detection after 30 frames. The *object detector thread* reads images from the directory, processes the forward pass of the neural network and saves the detections in the shared buffer, it repeats this sequence until it has processed all the predefined list of frames. We ensured that we avoid the race condition between the threads by testing the worst scenario, when the tracking computation load is very low, this is when the tracking module has got only two blobs to process. In this case the controlled delay is enough to synchronized the threads. This process is summarized in Algorithm 1.

Algorithm 1 Object detection thread

Input: sequencesOfImages
Output: sharedVariable
$fpsRate = 30$
$numberFramesSequences = size(sequencesOfImages)$
$network = network.init()$
$listIndex = createList(FPS, numFramesSequences)$
procedure Run
    for $indexImage$ in $listIndex$ **do**
        $image = read(indexImage)$
        $detection = network.forward(image)$
        $sharedVariable = detection$
    end for
end procedure

We selected the Single Shot multibox Detector (SSD) as object detector, because it has got the best balance between performance and speed, in section 6.1 we make a comparison of the available detectors.
This detector uses the VGG network trained on ImageNet dataset for image classification purpose as feature extractor. The authors add the SSD layers to this feature extractor to build an object detector implemented on TensorFlow. To train the whole system, they only modify the weights of the SSD layers, the weights of the feature extractor are frozen, this process is called fine-tune. In this way we benefit the capabilities of the trained feature extractor and we need less example to train the whole network. The dataset for training the network is formed by the junction of the VOC07, VOC12 and COCO datasets. Although they are a generic datasets, the biggest category is the person instance. The network is trained for 12000 iterations and the weights are saved in a TensorFlow checkpoint. To use this detector we only need to load the weights in the initialization of the system. Finally, in the figure 4.5 we can observe the result of this step.

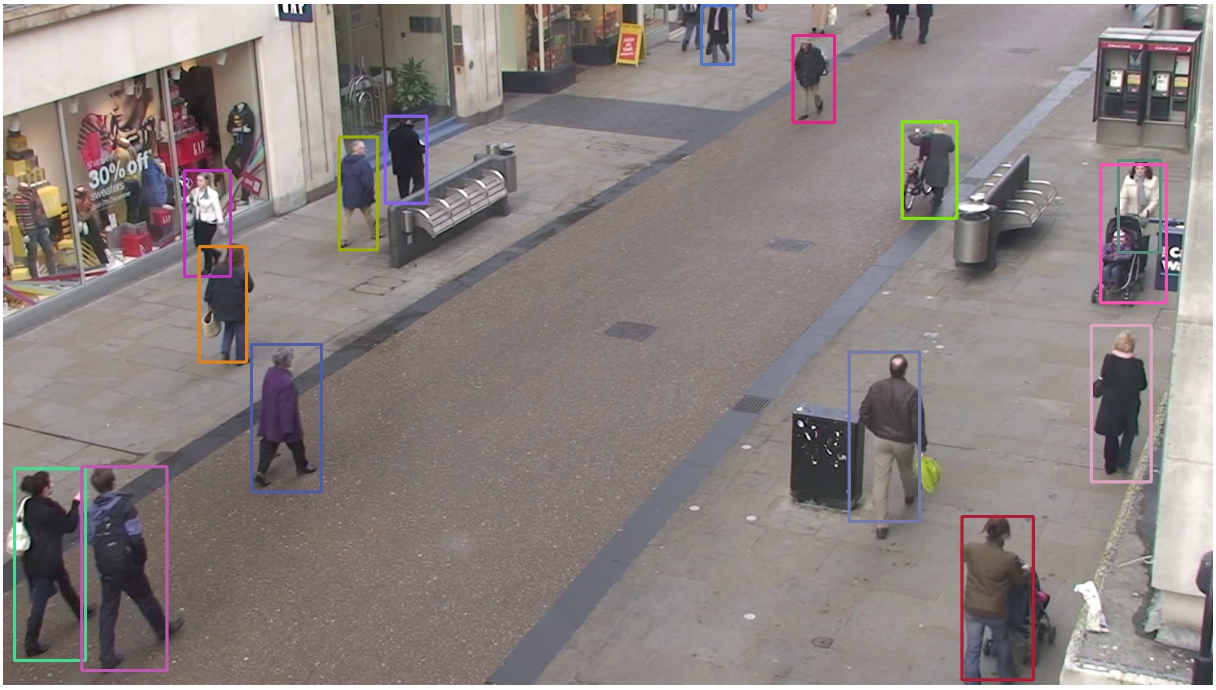


Figure 4.5: Detections of the algorithm.

4.3 Feature-based tracking thread

The essence of this thread is the tracking module, called LK in the figure 4.3. It stands for Lucas-Kanade algorithm. This module computes for each blob its displacement by computing the displacement of the features points inside its bounding box. It will have one blob for each person detected and it will update the position of each blob as it evolves and moves in the image flow. The blobs are also called trackets.

4.3.1 Feature extraction

For extracting the features, we use the OpenCV routine `goodFeaturesToTrack()`, this function determines strong corners on an image, according the Shi-Tomasi method. Its parameters and values are the following:

- `image`, input image
- `maxCorners`, maximum number of corners to return. If more corners than this maximum are found, the strongest of them are returned. We set this value experimentally to 60.
- `qualityLevel`, the minimal accepted quality of image corners. We set this value experimentally to 0.1.
- `minDistance`, minimum possible Euclidean distance between the returned corners. We set this value experimentally to 2.
- `mask`, optional region of interest. Not used.
- `blockSize`, Size of an average block for computing a derivative covariation matrix over each pixel neighborhood. We set this value experimentally to 7.
- `useHarrisDetector`, Parameter indicating whether to use a Harris detector. Not used.
- `k`, Free parameter of the Harris detector. Not used.

We applied an equalization transformation to the image before the feature extraction, to obtain more high contrast points. In the experiment described 6.2.1 we performed a comparison of several preprocessing techniques. We can observe the results of this feature extraction in the figure 4.6 and for all the blobs it looks like figure 4.7.

Figure 4.6: Shi-Tomasi points on a person.

Figure 4.7: Blobs with their feature points.

4.3.2 Feature matching

Once we have all the blobs with their feature points, we superimpose each current bounding box on the next frame. We extract the features of these ahead bounding boxes and match them with the previous ones. To computing the matching, we used the OpenCV's routine `calcOpticalFlowPyrLK()`. This function implements a sparse iterative version of the Lucas-Kanade optical flow with pyramids. And his parameters and values are the following:

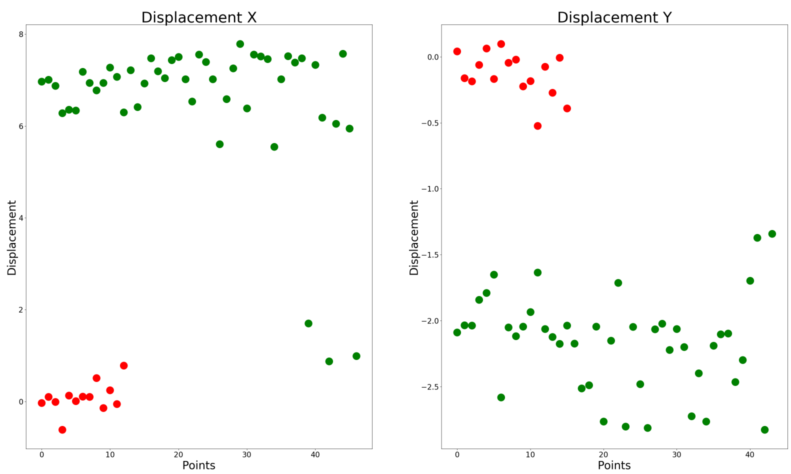
- `prevImg`, first image.
- `nextImg`, second image.
- `prevPts`, vector of 2D points for which the flow needs to be found.
- `nextPts`, output vector of 2D points containing the calculated new positions of input features in the second image.
- `status`, output status vector, it tells you whether the optical flow has been found.
- `err`, each element of the vector is set to an error for the corresponding feature.
- `winSize`, size of the search window at each pyramid level. We set this value experimentally to 15.
- `maxLevel`, number of pyramid levels. We set this value experimentally to 4.
- `criteria`, parameter specifying the termination criteria of the iterative search algorithm. We set this value experimentally to 10 iterations.

For the example we can observe the matching between the points of consecutive frames in figure 4.12.
Once we have the correspondances between feature points of consecutive frames, we can compute the motion of the blob as the displacement of those features. After this step we have a bunch of motion vectors, but some vectors in the bounding box do not belong to the pedestrian, and if we do not erase them, they will contribute to the motion computation. Usually these points belong to the static elements of the scene, like the floor or urban furniture, these points in terms of motion between subsequent frames will be very low or almost static. We can observe this fact plotting the displacement of these points and

Figure 4.8: Matched feature points.

drawing them in the image, we can observe it at figure 4.9, the red points in the plot and in the image are considered static and the green ones are not. So, we erase the points with a displacement in both dimensions smaller than a threshold. We set this value experimentally to 1.0. If all the points are static, the algorithm considers the blob as static and does not apply a displacement to his estimation. But it does not considered it lost blob.

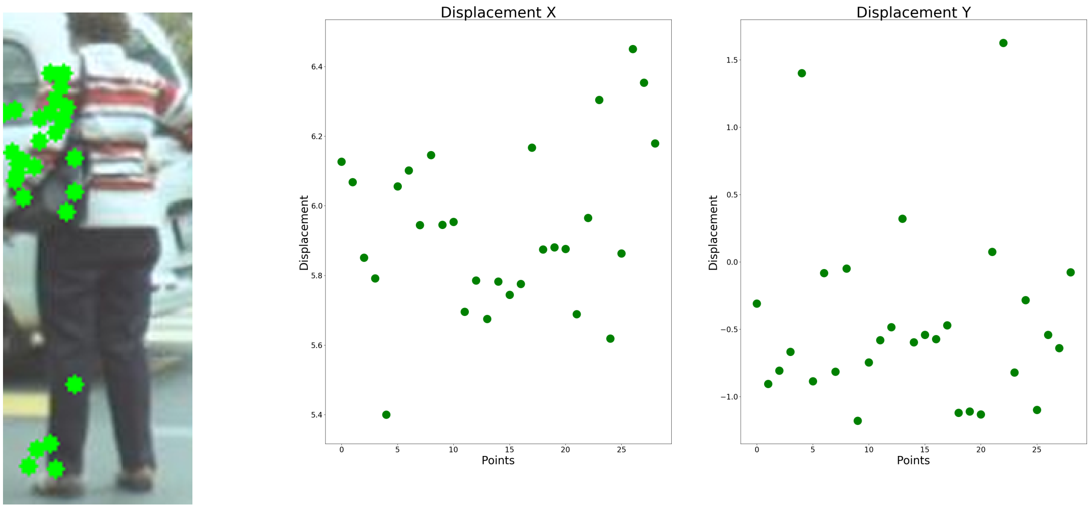
(a) Image feature points.



(b) Displacement graphics.

Figure 4.9: Image and motion vectors.

But, this behavior will only work on sequences where the camera is fixed, in sequences produced by a moving camera it will not work, we can observe the plot of displacement vectors on a sequence acquired by a moving camera in 4.10.



(a) Image feature points.

(b) Displacement graphics.

Figure 4.10: Image and motion vectors of a moving camera sequence.

4.3.3 Blob matching

Once we have the correct correspondances and erased those static feature points, we compute the displacement of the blob as the median of all of displacements in each dimension. We also compute the change of the scale of the blob, it is computed as follows: for each matched feature point, a ratio between the current feature point position and the next feature point position, is computed. Thus, the bounding box scale change is defined as the median over these ratios. In the figure 4.11 we can observe a representation of this displacement for each blob.

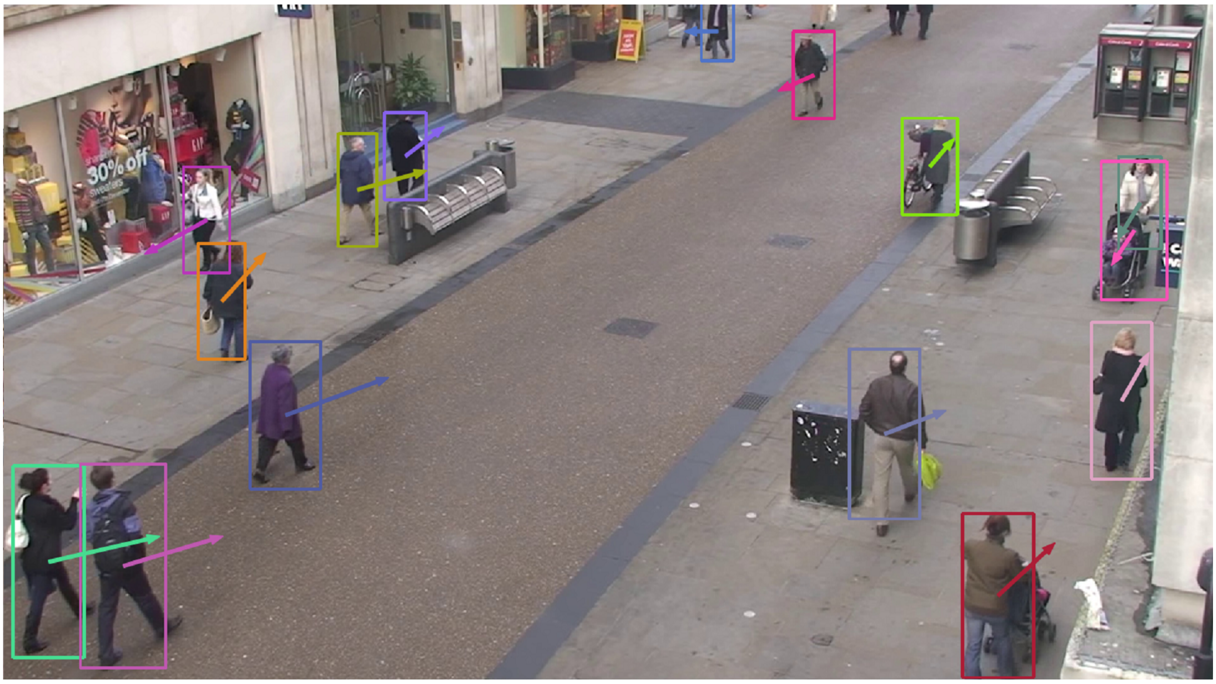


Figure 4.11: Displacement of each blob.

With this displacement and the change in scale we can update the predicted position of the blob in the next frame. In the figure 4.12 we can observe the previous estimation and the new one. At this point we have solved the data association problem in the tracking-by-detection nomenclature.

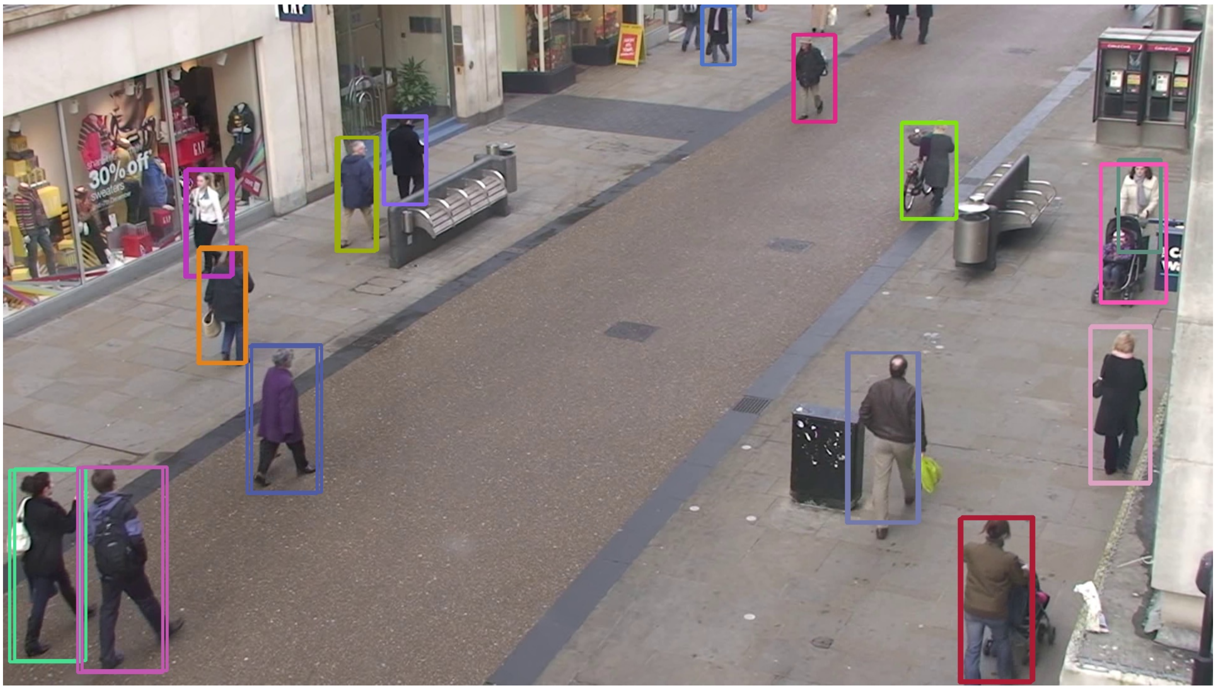


Figure 4.12: Uploaded estimation.

In 2 there is a pseudocode that summarizes these steps, with the current blob, *blob1* and the superimposition of the bounding box on the next frame, what is called *Blob2*, it computes the new estimation of the blob. At this point, we have an implementation of the tracking algorithm given a set of bounding boxes.

Nevertheless, the feature tracking is very sensitive to crossings between pedestrians, although the bounding boxes are fitted to body of the pedestrian it could include points belonging to another pedestrian. We can observe this event in figure 4.13. In this case the movement estimation is wrong and eventually the pedestrian will not be embedded by the bounding box.

Algorithm 2 LK module

Input: Blob1,Blob2

Output: displacementX,displacementY,diffScale

$blob1Equ = equalize(Blob1)$

$features1 = cv2.goodFeaturesToTrack(blob1Equ)$

$blob2Equ = equalize(Blob2)$

$features2 = cv2.calcOpticalFlowPyrLK(blob1Equ, blob2Equ, features1)$

$displacement = features2 - features1$

if displacement > threshold **then**

  delete(displacement)

end if

$displacementX = median(displacement[:, 0])$

$displacementY = median(displacement[:, 1])$

$diffScale = median(features2/features1)$

Figure 4.13: Tracking failure.

So, we need a mechanism to detect these failures. Therefore we studied how the motion algorithm behaves in these situations. When it has got a trajectory without crossing with other pedestrian, the vertical and horizontal displacements roughly behave like a damping sine wave (if it goes away of the camera) or amplified sine wave (if it goes closer to the camera). But when it has got an interference with another pedestrian, it has an steep change in that wave. We can measure that change as the differences between the current displacement and the previous one normalized by current displacement. If this value overtakes a threshold, we consider then that the tracker has lost a track. We can observe this process in the next figure 4.14, it belongs to previous trajectory shown at Figure 4.13

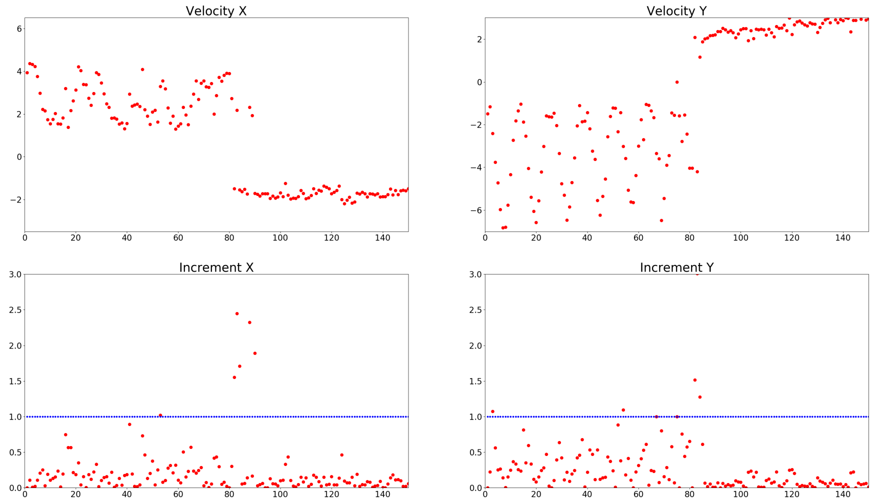
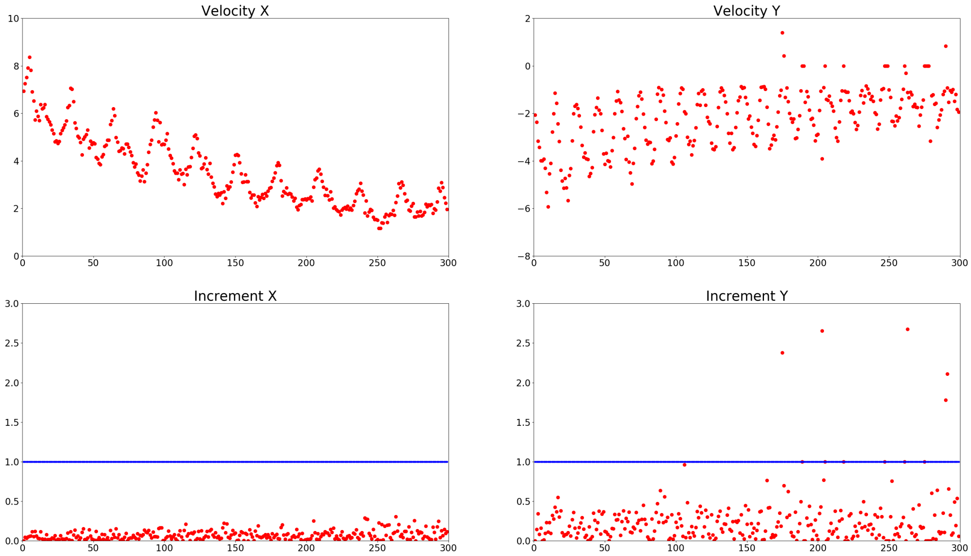


Figure 4.14: Tracking failure displacements.

In contrast, when it does not cross with another pedestrian, the displacement does not get disrupted, then the normalized differences with the previous displacement gets a low value. We can observe this process in figure 4.15. We set a threshold to notice this interference and delete this bounding box. We delete them from the current tracking execution, but we save the bounding box for following processings.

(a) Trajectory.



(b) Plots movement.

Figure 4.15: Wrong trajectory.

4.4 Data association with detected pedestrians

Once we computed the trajectories, in the next iteration we might have to add a detection, so we need a module to combine these trajectories with detections coming from the neural net. Thus, for each pedestrian we distinguish three situations:

- **Situation 1**, the tracket has got a nearby detection, then the detection replaces the tracket bounding box. This is what is called spatio-temporal constraint.
- **Situation 2**, the tracket has not got a nearby detection, then the tracking of the

bounding box, that is the blob continues.

- **Situation 3**, the detected pedestrian does not have any close blob. In this case we need to decided whether this pedestrian is new in the scene or it has been seen before (it is a lost tracket).

We can observe the procedure for first and second situations in the figure 4.16. In green colour we can observe the detections and in blue colour the trackets. We defined *nearby* as the distance between the centres of the bounding boxes, this distance has to be lower than a threshold to be considered nearby.

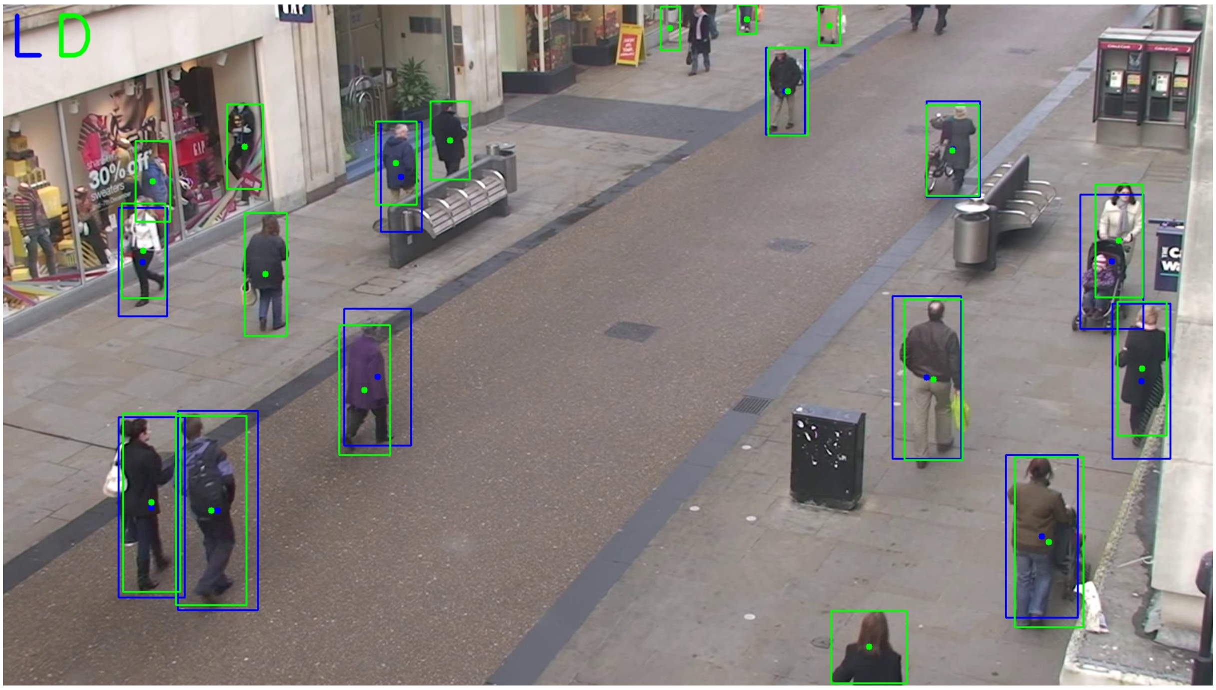


Figure 4.16: Spatio-temporal data association.

To maintain the identity of the pedestrian we need a method to compare missed trackets those with no associated detections. We decided to solve it with deep learning techniques. In particular, a Siamese convolutional neural network, with In-network architecture in figure 4.17 we can observe a diagram and the feature dimension of each layer. This network concatenates two blobs, substracts the channel means and normalized the images to the range 0 and 1, and computes a probability to belong to the same identity. It has got 6 convolutional layer and 1 fully connected layer, it was implemented on Keras with a Theano backend. In section 6.3 we explain why we selected this architecture and how we trained it.

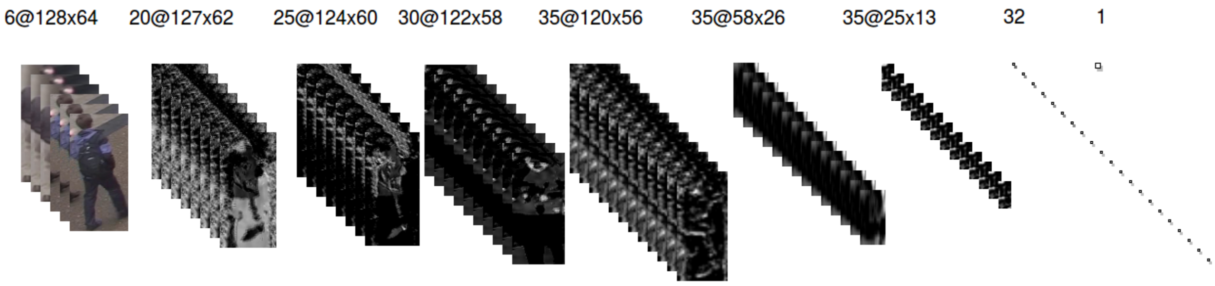


Figure 4.17: Siamese network: In-network.

For each detection we compare with all the missed trackets, if the maximum value of this comparison is greater than a threshold we assign it to that identity. If it is not we consider that detection as a new identity. We can observe this data association process in Algorithm 3.

Algorithm 3 Data Association

```
Input: listBlobs,listDetections,listLostBlobs
Output: listBlobs
procedure Run
    siamese = siamese.init()
    for i in listBlobs do
        distance = euclideanDistance(blob[i], listDetections)
        distanceOrdered = argmin(distance)
        % Situation1
        if distanceOrdered[0] < threshold then
            newBlobs.append(listDetections[idx])
            delete(listDetections[idx])
            % Situation2
        else
            newBlobs.append(listBlobs[i])
        end if
    end for
    % Situation3
    for i in listDetectionsNotAssigned do
        similarity = siamese.forward(listLostBlobs, DetectionsNotAssigned[i])
        similarityOrdered = argmin(similarity)
        if similarityOrdered[0] < threshold then
            newBlobs.append(listDetectionsNotAssigned[i])
            delete(listDetectionsNotAssigned[i])
        else
            newBlobs.append(listDetectionsNotAssigned[i])
        end if
    end for
end procedure
listBlobs=newBlobs
```

Chapter 5

Datasets and evaluation procedures

In this chapter we explain the datasets and evaluation procedures that we have used to adjust our algorithm, or parts of it and for experimentally validating the developed solution, allowing an objective comparison between solutions. We explain the three main dataset used in this thesis: for object detection, for tracking, and for person reidentification. In addition, in order to evaluate and compare the candidate solutions, the quality measurements on each of them are also described.

5.1 Datasets for object detection

This section describes the most common datasets used in object detection tasks. Throughout the history of computer vision research datasets have played a critical role. They not only provide means to train and compare fairly the algorithms, they also drive research in new and more challenging directions. In order to accomplish this, they provide:

- a collection of challenging images and high quality annotations.
- an standard evaluation methodology, so the performance of the algorithms can be objectively compared.

In the next subsections, we will explain several well known international datasets for object detection. These datasets are provided in the context of international challenges, these challenges look for an improvement on the state of the art on the object detection algorithms. In table 5.1 we show the comparison of the datasets according to two key parameters: number of categories and instances per category. These parameters are critical in the selection of one of them.

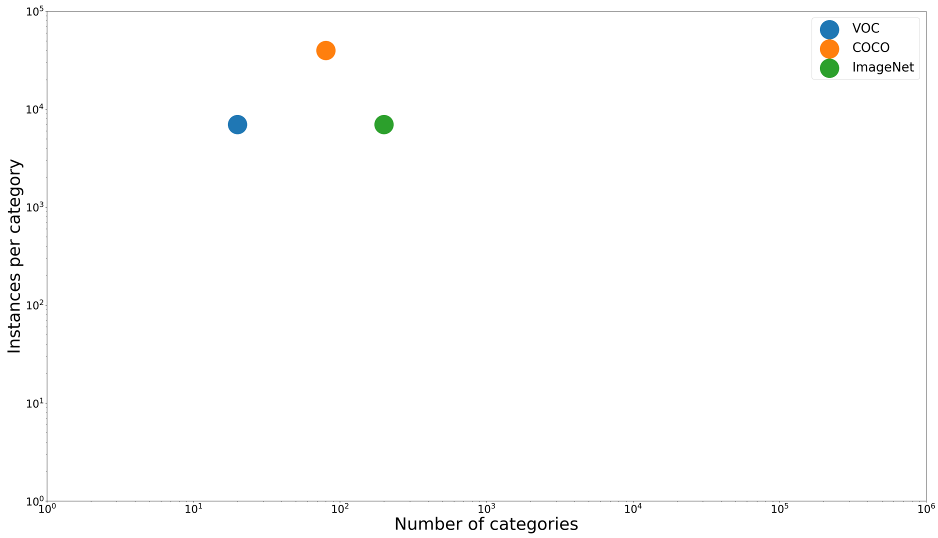


Figure 5.1: Comparison of available datasets.

5.1.1 Pascal Visual Objects Classes

The Pascal Visual Object Classes (VOC) challenge [62] is a benchmark in visual object category recognition and detection. It has been organised annually from 2005 to 2012. The challenge and its associated dataset has become accepted as one of the landmark benchmarks for object detection. All the images are taken from the `flickr` consumer photographs website and annotated with the Amazon Mechanical Turk tool [63]. The most popular editions of the challenge for object detection are those from years 2007 and 2012.

The challenge of the year 2007 [64] contains 5000 images in the trainval (training + validation) and test sets, with almost 12000 objects. This was one the first datasets for object detection before the deep learning era. Also, it is very useful for researchers, due it has 2.5 mean object per image and it is very challenging. In figure 5.2 we can observe the distribution of images and objects instances.

In figure 5.3 we can observe an example of several images with their ground truth annotation.

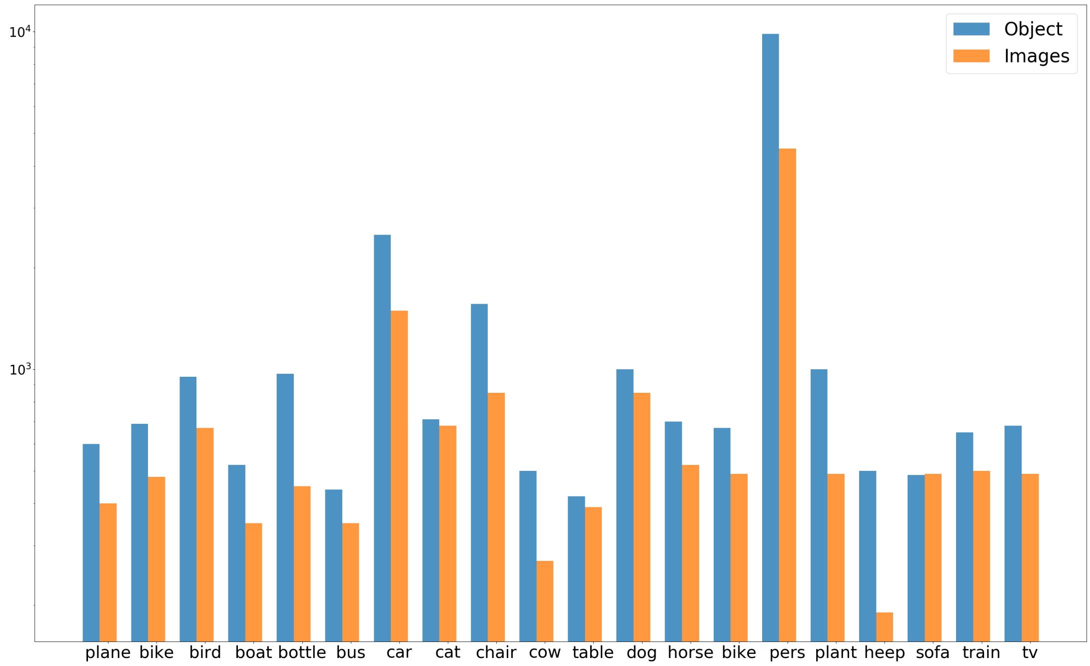
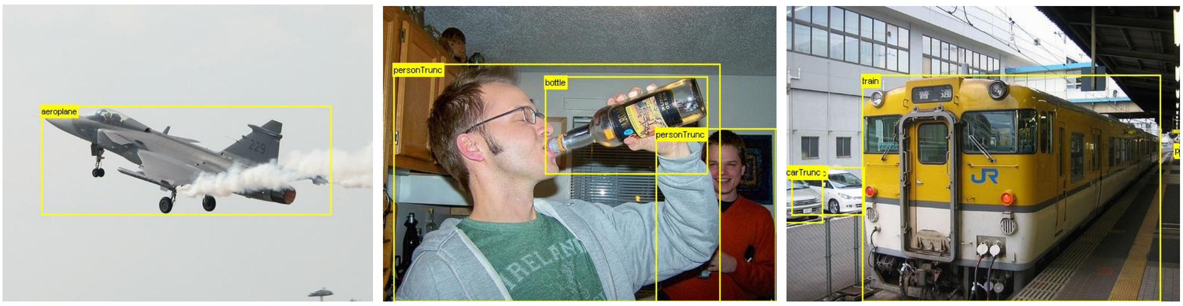


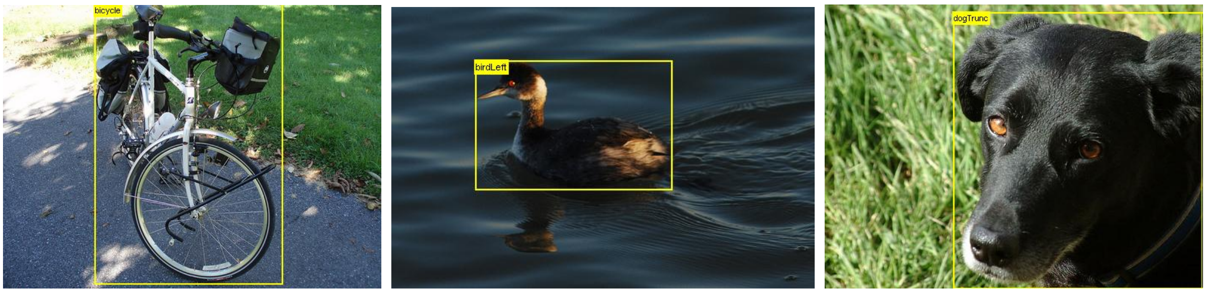
Figure 5.2: Distribution of VOC07 dataset.



(a) (b) (c)

Figure 5.3: Few samples of the VOC07 dataset .

The 2012's edition [65] is also one of the most used datasets in object detection tasks. It increases the volume of images of the 2007 edition up to 10000 images on trainval and test sets and similar quantity of instances per image. In the figure 5.4 we can observe an example of several images with their ground truth annotation.



(a) (b) (c)

Figure 5.4: Few samples of the VOC12 dataset .

The datasets from Pascal challenge are very useful to test object detection algorithms, their size is very handy (a few thousands of images) and contains a challenging quantity of objects per image, very interesting for the algorithms. But its little amount of images does not permit to train a network on this dataset, although it can be used to finetune the network.

5.1.2 ImageNet

ImageNet project [66] with the challenge ImageNet Large Scale Visual Recognition Challenge [ILSVRC] was the first large-scale database, temporally developed to supply the deep learning techniques, eager of feed with tons of images. ImageNet aims to populate the majority of the 80000 synsets of WordNet with an average of 500-1000 clean and full resolution images. The collection was based on the query of that words on several image search engines and human refined on the Amazon Mechanical Turk platform. It can be downloaded from here [67].

In 2016, the project collects more than 10 million of annotated images with 1000 classes. Although its main purpose is image classification, it has an object detection challenge with 200 categories with over a 1 million images with annotated objects. In the figure 5.5 we can observe an example of several images.

(a) (b) (c)

Figure 5.5: Few samples of the ImageNet dataset .

The dataset for the ImageNet challenge is not used too much in object detection tasks, it contains several instances per image. This did not encourage researchers to use it. Although it is used to train neural networks in image classification tasks. Although those trained architectures can be incorporated in the object detection algorithms.

5.1.3 COCO

The Microsoft Common Objects in Context also known as COCO dataset [68], is a dataset that addresses the three core research problems in scene understanding:

- detecting non-iconic views of objects. For many datasets most of the objects have an iconic representation, they appear unobstructed, near the center of the photo and with their canonical shape. So in this dataset, they included images to struggle the object recognition task, like objects in the background, partially occluded, amid clutter. Therefore, it reflects the composition of actual everyday scenes.

- contextual reasoning between objects. Nowadays natural images contain multiple objects, and their identity can only be solved using context, due to small size or ambiguous appearance in the image. So in this dataset, images contain scenes rather isolated objects.

- the precise 2D localization of objects, also the detailed spatial understanding of object layout will be a core component of an image understanding system, so this dataset struggle to do so.

So, the three main tasks of this challenge are object classification, object detection and semantic scene labelling. This dataset contains 91 object categories, with 2.5 million labelled object instances in 328 thousand images, labeled with the Amazon Mechanical Turk tool. It can be downloaded from here [69]. In the figure 5.6 there is an example of it.

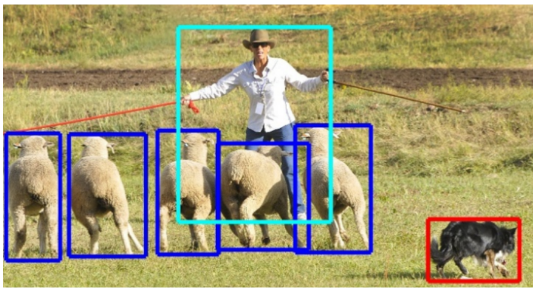


Figure 5.6: Sample of the COCO dataset .

The COCO dataset is the most recent one. Is the one focus on object recognition and the detection supposes a challenge due the objects are in common places and are very challenging to detect. And it is very interesting to due of the quantity of instances per image. The COCO challenge contains 91 object categories with 82 of them having more than 5 thousand labeled instances. In total the dataset has 2.5 million labeled instances in 328 thousand images.

In contrast to ImageNet dataset, COCO has fewer categories but more instances per category. Also, it has more instances per category than the VOC dataset. This fact aids in learning detailed object models capable to cope with the variability and also with their unknown 2D location in the images. In addition, another prominent feature of the COCO over the other two, is the number of labelled instances per image which may aid in learning contextual information.

Moreover, the COCO dataset uses images from non-canonical point of views, allowing to the algorithm to be robust to everyday views. This feature can be observed in the plot 5.7, in which we can observe different views of the same category. And clearly the COCO's images are the most not iconic representation.

(a) Pascal VOC. (b) ImageNet. (c) COCO.

Figure 5.7: Distribution of pascal.

Finally, the table 5.1 summarizes the main statistics of the dataset stated previously.

	VOC07	**VOC12**	**ImageNet** [**2014**]	**Coco** [**2015**]
trainval set	5011	11540	476688	165482
test set	4952	10991	40152	81434
Number of classes	20	20	200	80
Mean obj per image	2.5	2.4	1.1	7.2
Number person instances	4690	8566	-	300000

Tabla 5.1: Datasets tables

5.2 Evaluation of object detection algorithms

In order to compare the performance of the different algorithms, each challenge establishes a clear measure. In this thesis, we used the interpolated *average precision* (AP), used in the Pascal VOC challenge (based on [70]).

For each class, the precision-recall curve is computed from a method's ranked output.

- Recall is defined as the proportion of all positives examples ranked above a given threshold.
- Precision is the proportion of all examples above the threshold which are from the positive class.

The AP summarises the shape of the precision/recall curve, and is defined as the mean precision at a set of eleven equally spaced recall levels [0,0.1,...,1]:

$$AP = \frac{1}{11} \sum_{r\epsilon(0,...,1)} p_{interp}(r)$$

The precision at each recall level r is *interpolated* by taking the maximum precision measured for a method for which the corresponding recall exceeds r:

$$p_{interp}(r) = max_{\hat{r}:\hat{r}>r} p(\hat{r})$$

The authors justified this measurement as a way to reduce the impact of the 'wiggles' in the precision/recall curve, caused by small variations in the ranking of examples. In the figure 5.8, we can observe this effect on the curve.

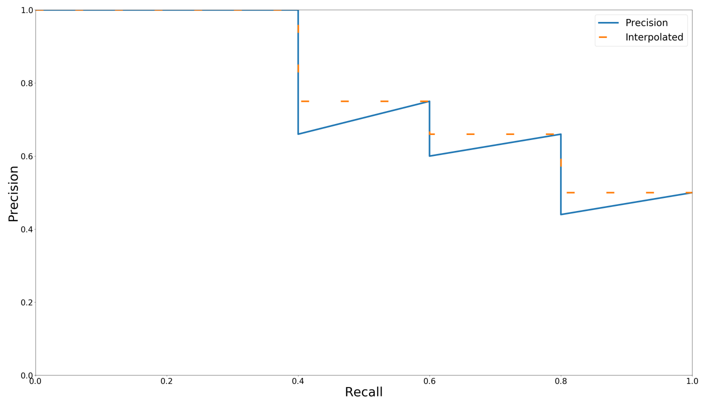


Figure 5.8: Comparison interpolated and normal curve.

In addition, detections were assigned to ground truth objects and judged to be true/false positives by measuring bounding box overlap. To be considered a correct detection, the area of overlap a_0 between the predicted bounding box B_p and ground truth bounding box B_{gt} must exceed 0.5 by the formula:

$$a_0 = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

where $B_p \cap B_{gt}$ denotes the intersection of the predicted and ground truth bounding boxes and $B_p \cup B_{gt}$ their union. The treshold of 50 % was set deliberately low to account for inaccuracies in bounding boxes in the ground truth data. Multiple detections of the same object in an image were considered false detections.

Finally, setting the threshold IoU to a value of 0.5 could cause misdetections of small objects, in [66] they propose an adaptive setting of that threshold based on the size of the ground truth and so detect correctly small objects. In practice, this change only affects 5.5% of objects in the detection validation set.

5.3 Datasets for multiple object tracking

Evaluating and comparing multi-target tracking methods is not trivial for numerous reasons.

- First, the perfect solution is difficult to define clearly. Partially visible, occluded, or cropped targets, reflections, and objects that are very close resemble targets; all of them impose intrinsic ambiguities, such that even humans may not agree on one particular ideal solution.
- Second, a number of different evaluation metrics with free parameters and ambiguous definitions often lead to inconsistent quantitative results across the literature.
- Finally, the lack of pre-defined test and training data makes difficult to compare different methods fairly.

In contrast to other research areas in computer vision, multiple object tracking still lacks large-scale benchmarks.

5.3.1 PETS

Targeted primarily at surveillance applications [71], the 2009 version consisted of 3 subsets: S1 targeted at person count and density estimation, S2 targeted at people tracking, and S3 targeted at flow analysis and event recognition. In the figure 5.11 we can observe one image from this dataset.

Figure 5.9: Example of Pets.

Even for this widely used benchmark, we observe that tracking results are commonly obtained in an inconsistent fashion: involving using different subsets of available data, different detection inputs, inconsistent model training that is often prone to over-fitting, and varying evaluation scripts. Results are thus not easily comparable [72].

5.3.2 Town Centre Dataset

Developed by the Active Vision group of the University of Oxford [73], is one of the standard datasets of the Tracking community. The video sequence is high definition, 1920x1080 with 25 FPS and has got ground truth consisting of sixteen people visible at any time. It is higly used because its excellent ground truth and a good rate of density pedestrians. They also provide a ground truth of the pedestrian's heads, thus it could be used to gaze estimation. In the figure 5.10 there is a snapshot of the sequence with its ground truth.

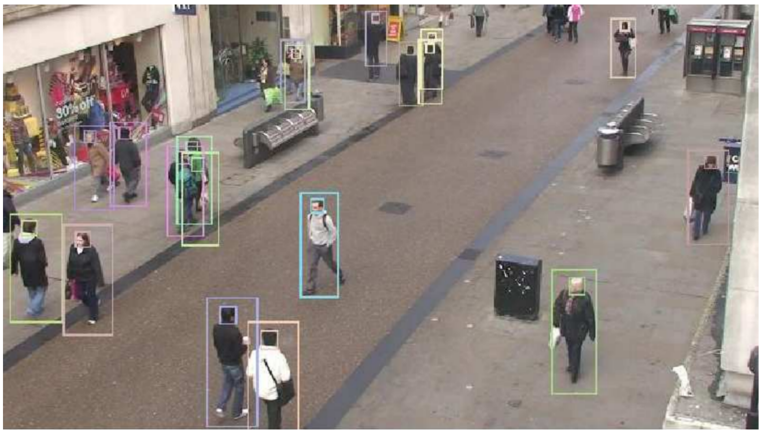


Figure 5.10: Snapshot of the Town Centre dataset.

5.3.3 MOT challenge

In the tracking community there is no a standard dataset like other fields of computer vision. Even for the widely used benchmark Pets, the tracking results are commonly obtained in an inconsistent fashion: involving using different subsets of the available data, inconsistent model training that is often prone to overfitting, varying evaluation scripts, and different detections inputs. Results are thus not easily comparable.
The MOT's authors realized these problems while analysing existing dataset. In order to make advande the field, they decided to create the Multiple object dataset. This dataset had got three main components: a collection of publicy available and new datasets, a centralizaed evaluation method, and an infrastructure that allows for crowdsourcing of new data, new evaluation methods and even new annotations.
They also organized a yearly workshop *MOTChallenge*, where they share the winner of the challenge and show the best tracking algorithms.

5.4 Evaluation of multiple people tracking algorithms

A critical point with any dataset is how to measure the performance of the algorithms. A large number of metrics for quantitative evaluation of multiple target tracking have been proposed. Choosing unique general evaluation is still ongoing.
On one hand, it is desirable to summarize the performance into one single number to enable a direct comparison. On the other hand, one might not want to lose information about the individual errors made by the algorithms and provide several performance

estimates, which precludes a clear ranking.
We will explain two sets of measures that have been established themselves in the literature: the CLEAR metrics [74], and a set of track quality measures [75].
As in the object detection metrics, we can classify each tracket, whether it is a true positive, that describes an actual (annotated) target, whether the output is a false alarm (or false positive, FP). This decision is typically made by the well-known thresholding measure of Intersection over Union [IoU]. Also a target that is missed by a tracker is a false negative.
Due to we are working with multiple object, we assume that each ground truth trajectory has one unique start and one unique end point, that is not fragmented. So we need to penalty re-identification. This is called, identity switch [IDSW], and it is counted as if a ground truth target i is matched to track j and the last known assignment was $k = j$. The next figure summarizes the stated measures (the grey area indicate the matching threshold).

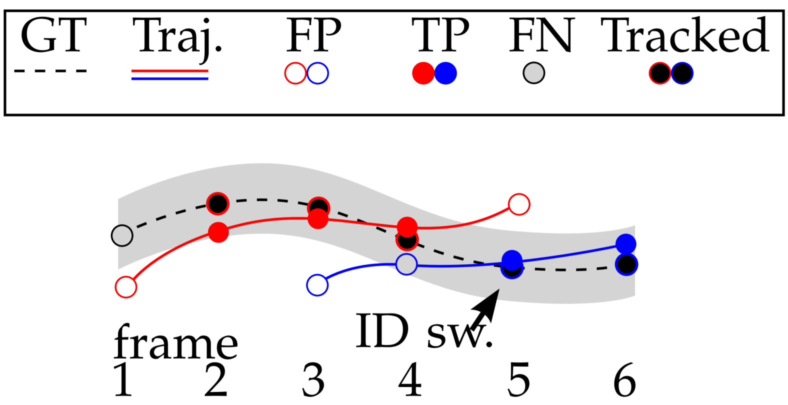


Figure 5.11: Example of measures.

Then, after determining true matches and establishing the correspondences it is possible to compute the metrics over all the sequences. The multiple object tracking accuracy [MOTA] [74] is perhaps the most widely used figure to evaluate a tracker's perfomance. The main reason for this is its expressiveness as it combines three sources of errors defined above:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t}$$

where t is the frame index and GT is the number of ground truth objects. This measure gives an indication of the overall performance.
The multiple object tracking precision [MOTP] is the average dissimilarity between all true postives and their corresponding ground truth targets. For bounding box overlap, that is computed as

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}$$

where c_t denotes the number of matches in frame t and $d_{t,i}$ is the bounding box overlap of target i with its assigned ground truth object. Thereby, it gives the average overlap between all correctly matched hypotheses. So, the MOTP is a measure of localization precision.
As we have stated above, another metric is the tracking quality. Each ground truth trajectory can be classified as mostly tracked (MT), partially tracked (PT), and mostly lost (ML). This is done based on how much of the trajectory is recovered by the tracking algorithm. A target is mostly tracked if it is successfully tracked for at least 80% of its life span, without considering if there was an identity switch. If a track is only recovered for less than 20% of its total length, it is said to be mostly lost (ML). All other tracks are partially tracked. Finally antoher quality measure is track fragmentations (FM), it counts how many times a ground truth trajectory is resumed at a later point.

5.5 Datasets for pedestrian identification

A number of datasets for image-based re-identification have been released, and some commonly used datasets are summarized in table 5.2.

Name	**Date**	**Images**	**IDs**	**Cameras**	**Label**	**Evaluation**
VIPeR [76]	2007	1264	632	2	hand	CMC
iLIDS [77]	2009	476	119	2	hand	CMC
GRID [78]	2009	1275	250	8	hand	CMC
CAVIAR [79]	2011	610	72	2	hand	CMC
PRID2011 [80]	2011	1134	200	2	hand	CMC
WARD [81]	2012	4786	70	3	hand	CMC
CUHK01 [82]	2012	3884	971	2	hand	CMC
CUHK02 [83]	2013	7264	1816	10	hand	CMC
CUHK03 [84]	2014	13164	1467	2	hand/DPM	CMC
RAiD [85]	2014	1264	43	4	hand	CMC
PRiD 450S [86]	2014	900	450	2	hand	CMC
Market-1501 [87]	2015	32668	1501	6	hand/DPM	CMC/mAP

Tabla 5.2: Statistical comparision datasets.

Over recent year, dataset's size is increasing. Many of these datasets are relatively small in size, especially those of early days, but recent datasets, such as CUHK03 and Market-1501, are larger. Both have over 1000 ID's and over 10000 bounding boxes, and both datasets provide good amount of data for training deep learning models. In adition, the bounding boxes tend to be produced by pedestrian detectors, instead of being hand-drawn. Also, more cameras are used during collection, this helps to increase generalization. Although there are several datasets, there is not a prominent one in the literature.

5.6 Evaluation for pedestrian identification

When evaluating identification algorithms, the Cumulative Matching Characteristics (CMC) curve is usually used. CMC represents the probability that a query identity appears in differentiated candidate lists.

Formally [88], for each probe p from P_G we sort the similarity scores against gallery G, and obtain the rank of the match. Identification performance is then stated as the fraction of probes whose gallery match is at rank r or lower. The set of probes with a close match is:

$$C(r) = \Big\{p_j : rank(p_j) \leq r\Big\} \quad \forall p_j \in P_G$$

where the rank is defined as before. We now define the Cumulative Match Characteristic (CMC) to be the identification rate as a function of r:

$$P_I(r) = \frac{|C(r)|}{|P_G|}$$

which we plot as the primary measure of identification performance. It gives an estimate of the rate at which probe images will be classified at rank r or better. One drawback of the characteristics is its dependence on gallery size, $|G|$.

Chapter 6

Experiments

In this chapter we characterize the quality of each module and explain the validation experiments of our solution. Also, we explain several alternatives that we considered for each module.

6.1 Detection experiments

For the final choice of the detector to be included in our application we compared sevveral detectors studied in the theoretical review 3.1.1 and we tested them on the *MOT16* dataset. In the figure 6.1 we can observe the ROC curves of different detectors.

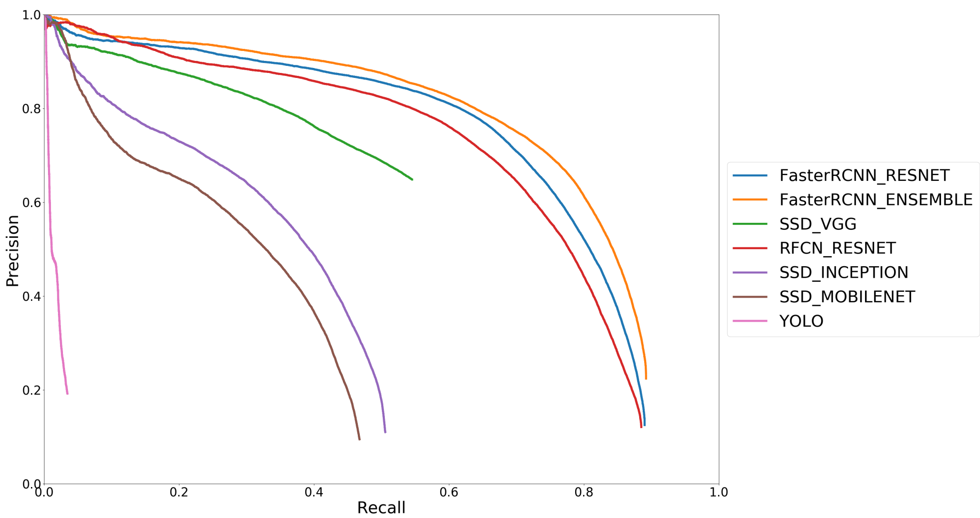


Figure 6.1: ROCs curves on the MOT16 dataset.

In the figure 6.2 we can observe the mean average precision against the time consumption.

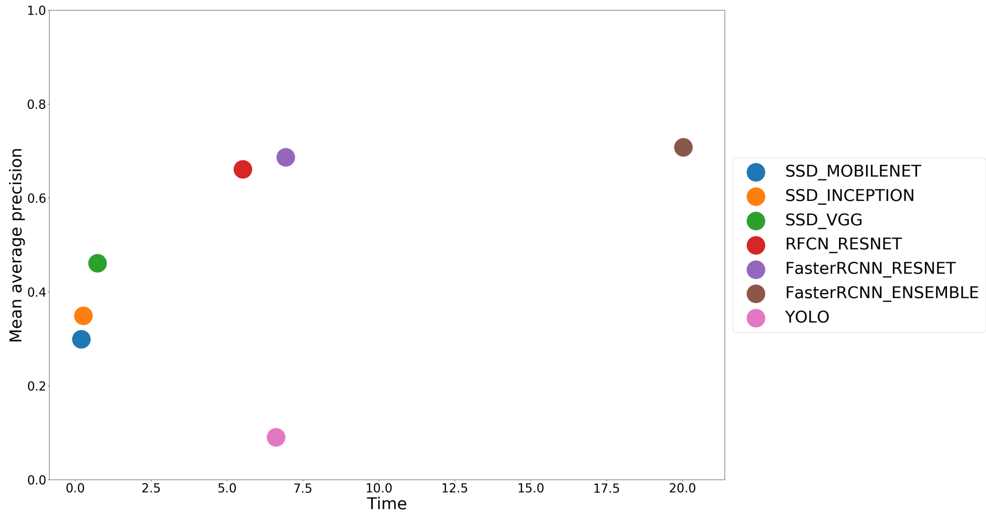


Figure 6.2: Mean average precision against time.

With this information we can summarize the conclusions of the study of the detectors:

- **Faster-RCNN**, we used the TensorFlow implementation [89], the original code required a Nvidia GPU. This repository includes the Faster-RCNN model with ResNet as feature extraction and the ensemble model compound by Inception-ResNet. It scores 0.6872 and 0.7081 average precision with a time consumption of 6.93 and 20.029 seconds respectively. These are the highest accuracy value obtained in this comparison, but also the slowest.

- **R-FCN**, the original code is not publicly available. We used the TensorFlow implementation [89]. It scores 0.6614 average precision and 5.514 seconds. This accuracy value is similar to the previous one but is slow.

- **YOLO**, we used the original and it scores 0.09 average precision on the dataset, it takes 6 seconds per image [90]. It takes too time for this awful score. According to the theory 3.1.1 this is the fastest detector, but it does not optimized to run on cpu.

- **PVANET**, the code is not publicly available.

- **SSD**, we tested several feature extractors with this model. Their scores are the following: the SSD model with VGG as feature extractor, it scores 0.4612 average precision and takes 0.73 seconds; SSD with Inception as feature extractor it scores 0.3499 average precision and takes 0.73 seconds; and SSD with MobileNet as feature extractor it scores 0.2995 average precision and takes 0.198 seconds. The original code [91] is not optimized for CPU execution, it takes about 3.5 seconds and the Caffe framework does not allow to run it in a multithreading way, so we discarded it. The VGG version comes from a particular developer [92] and the Inception and MobileNet from TensorFlow organization [89].

According to these results the object detector with the best balance between precision and time consumption is the SSD detector with VGG feature extractor. Detectors like SSD-Inception and SSD-MobileNet are really fast but their performance is 23% lower than the SSD with VGG. In contrast, RFCN is more accurate but it takes 700% more time than SSD with VGG.
The MOT organization provides a set of detections, they include FasterRCNN, DPM v5, and SPD [93]. We were not able to reproduce their results, because we can not access to the original code. In the figure 6.3 we can observe those detections.

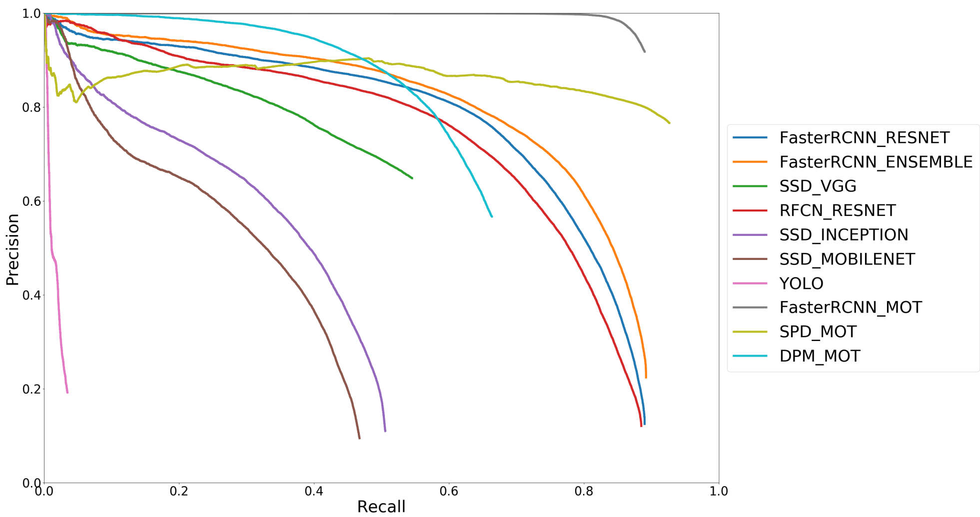


Figure 6.3: ROCs curves on the MOT16 dataset.

6.2 Feature-based tracking experiments

We started developing our feature-based tracking module with simple artificial objects like the one in the figure 6.4. As soon we had got expertise we shift to much complex models like people. Finally, the last version of the tracking module was inspired by the well-known tracking algorithm *MedianFlow* by Zalal et al[94] with its correspondent implementation in Python[95].

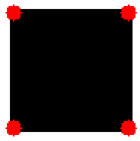


Figure 6.4: Artificial object to start tracking.

This tracking uses matching based on the optical flow, explained in section 3.1.2. It computes the new position through gradient descent in several frames. We assume that the motion is pure translational. As we can observe in the sequences of frames 6.5 of the dataset, the typical pedestrian moves in translation in the image plane, so this assumption may hold for most of the people appearing in the scene.

Figure 6.5: Sequence of translational movement.

In contrast to the previous figure, we can observe the figure 6.6 where the assumption of translation motion is not fulfilled (this sequences does not belong to the used dataset, only showed to contrast the previous idea) and a translational assumption will fail. The car in that image significantly rotates in the same scenario, in addition with a traslation.

Figure 6.6: Sequence of no translational movement.

We tested other tracking algorithms, like MeanShift, but we discarded it due to its problems to track pedestrians in a messy background.

6.2.1 Feature extraction improvement

The strength of the further processing greatly depends on the quality and quantity of features detected in the images. In order to enhance the general performance, we apply some prepreprocessing to the images. We tried several preprocessings techniques like sharpening, image contrast, median filter, and equalization. In 6.7, we can observe the relation between number of points extracted and time consumption of those techniques.

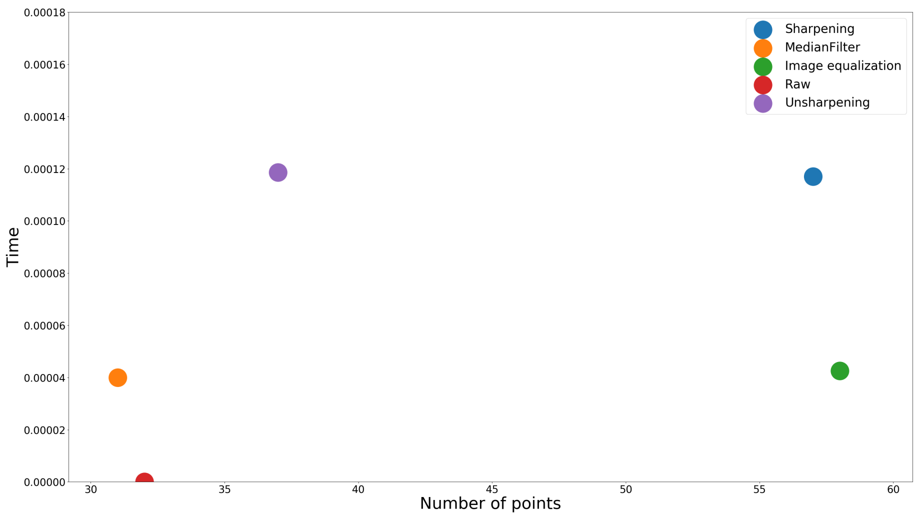


Figure 6.7: Plot of different image preprocessing techniques.

With this experiment, we realized that the best preprocessing in terms of speed and number of points, is to equalize the image. The computation is really simple, it only consists of equalizing an histogram and applying that transformation to the image. It increases over 55% the number points in comparison to not applying it to the raw image. In the figure 6.8 we can observe the different number of features in the raw and in the equalized image.

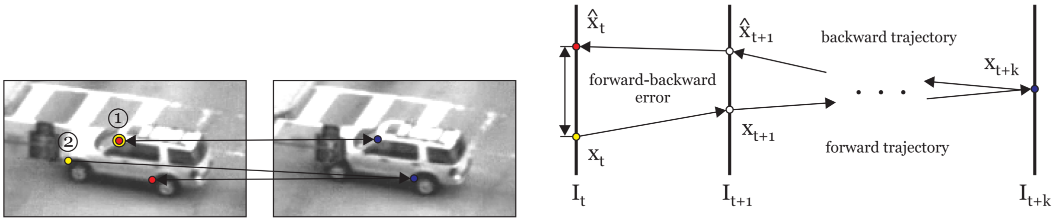
(a) (b)

Figure 6.8: Comparison between feature extraction on raw (a) and equalized image (b).

6.2.2 Matching module

We used the Lukas-Kanade algorithm to get the displacement of the features but we implemented the same method used in [94] too. The proposed method is based on so called forward-backward consistency assumption. This assumes in that correct tracking should be independent of the direction of time-flow. Algorithmically, the assumption is exploited as follows. First, a tracker produces a trajectory by tracking the point *forward* in time. Second, the point location in the last frame initializes a validation trajectory. The validation trajectory is obtained by *backward* tracking from the last frame to the first one. Third, the two trajectories are compared and if they differ significantly, the forward trajectory is considered as incorrect.
Figure 6.9 illustrates the method when tracking a point between two images. Point number 1 is visible in both images and the tracker is able to localize it correctly. Tracking

this point forward or backward results in identical trajectories. On the other hand, point number 2 is not visible in the right image and the tracker localizes a different point. Tracking this point backward ends in a different location than the original one.



(a) Image forward-backward. (b) Scheme forward-backward.

Figure 6.9: Illustration forward backward error.

We also implemented the forward method. We replaced for the tracking module in the algorithm. In the table 6.1 we observe the result of the forward and the forward-backward methods. Both have the same MOTA but the forward-backward methods takes around 10 % more time. Both have got the same accurcy but the forward-backward method is slower. For these reasons we decided to not use the forward-backward method and use the forward method in our solution.

	GT	**MT**	**PT**	**ML**	**FP**	**FN**	**IDs**	**MOTA**	**MOTP**	**FPS**
Forward	517	3	127	387	18896	78999	618	10.8	68.1	15.85
Forward-Backward	517	11	181	325	19951	78940	827	9.7	67.3	9.0

Tabla 6.1: Comparison tracking modules.

6.2.3 Tracking analysis

In this part we realize a qualitative analysis of the tracking module. The main disadvantage of the feature-based tracking is the dependence on the quality of the features, this method needs blobs with high texture to accomplish a good tracking. Thus, a sequence of low resolution there are less points with these characteristics. Also, we have problems with people who wear low texture clothes or are away from the camera, as we can observe in figure 6.10.

(a) High texture person. (b) Low texture person. (c) Far away person.

Figure 6.10: Differences texture examples.

Although a low frame rate could penalize the matching capabilities between frames, the pyramidal implementation of the Lucas-Kanade method solve it. In the figure 6.11 we show the matching procedure of a blob belonging to a low frame rate sequence, and its result is correct.

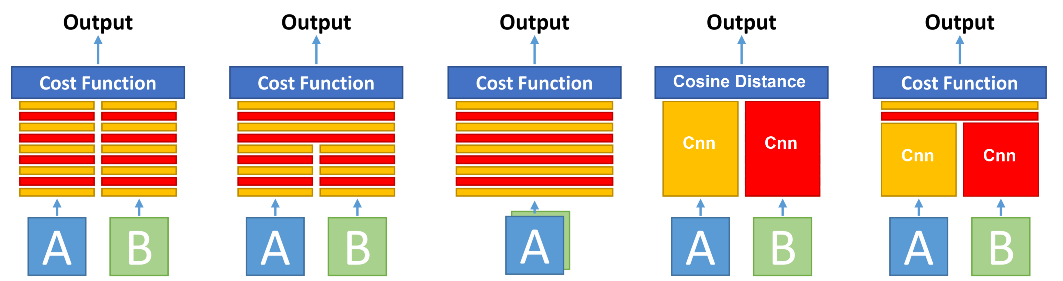
Figure 6.11: Blob matching low frame rate sequence.

6.3 Data Association experiments

With this module we want to solve the person reidentification problem generate in the feature-based tracking module. This module wants to check whether some blobs detections are new in the scene or have been seen previously. Thus, these architectures takes two blobs images as input and compute the probability to belong to the same identity. We tested how perform this task five architectures based on neural networks and we chose the best to incorporate on our tracking algorithm:

- **Siamese network: Cost function**, this is based on the idea of deep learning as feature extractor and top layers as classifier. Two branches that share parameters process the images and classify it.
- **Siamese network: In-network**, this is a mix of the previous models, where the information of the convolutional layers merges at some point before the classifier.
- **Siamese network: Joint data input**, according to the literature this architecture gives the best results compared with the other topologies. The input of the network is a concatenation of the two images and the network processes them together.
- **Feature extractor with cosine distance**, we used well-known architectures for image classification to extract features from the images and then compare those features with the cosine distance.
- **Famous network fine-tuned**, we extract features for each image with a well-known architecture and merge them with a fully connected layer.

We can observe these architectures in the figure 6.12.

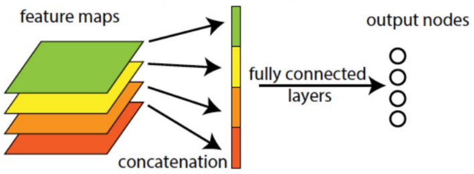


(a) Cost function. (b) In-network. (c) Joint data input. (d) Features with cosine distance. (e) Cnn fine-tuned.

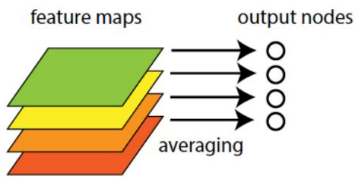
Figure 6.12: Siamese CNN topologies.

The main characteristics of the trained networks are the following:

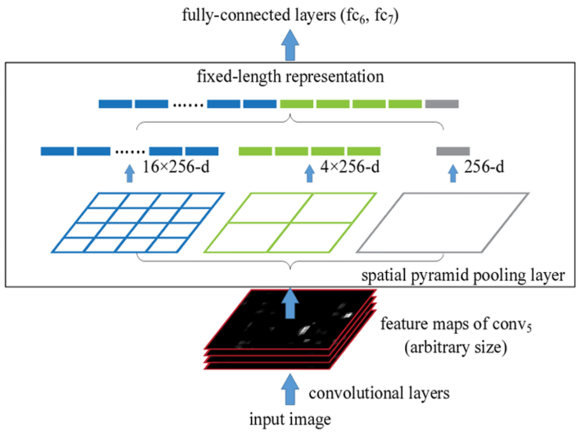
- **Loss**, we used the binary cross entropy as a loss. We tried with the contrastive divergence but it did not converge.
- **Optimizer**, as optimizer we used Adam, even though it has a mechanism to decrease the learning rate. We added an exponential decay to speed up the convergence
- **Activation**, we used ReLu. Currently, there are other activations functions, but ReLu has been established as the reference.
- **Initialization**, to initialize the weights we used the He. initialization method [96]. In addition we initialized the biases with the value of 0.1, in this way we avoid the dead neurons in the firsts iterations.
- **Batch normalization**, we tested batch normalization, but it adds to much computation time and so we discarded it.
- **Regularization**, we used Dropout in the fully connected layer to avoid overfitting.
- **Preprocessing**, as preprocessing techniques we centre the data. We subtract the mean per channel calculated over all images. Also, we normalized the data between the range 0 and 1.
- **Final layers**, historically, in the junction between the convolutional layers and the fully connected layer, a flatten mechanism has been used, but it increases dramatically the number of neurons in the fully connected layer, and it shows problems to converge. From the publication of InceptionV3 [97], the authors used a global average pooling layer, it generates the spatial average of each channel of the tensor. With this layer we achieve a reduction on the number of neurons in the fully connected layers and the nerwork converges quickly. Also we used the spatial pyramid pooling layers, they consist of a multiresolution max pooling. We can observe those differences in 6.13.



(a) Flatten.



(b) Global average pooling.



(c) Spatial pyramid pooling.

Figure 6.13: Final layers.

- **Output**, We did not use softmax as output, we only used one neuron with sigmoid activation, in this way the output is constrained between 0 and 1.

We developed our models in a VGG way, stacking several convolutional layers and finishing the network with a fully connected layer. We started with a few convolutional layers and added more till the score on the test set does not improve. We started with 3 and we end up with 6 convolutional layers as best performance.
For the dataset, there is not a prominent dataset in the field, so at the beginning we decided to use the MOT16 as dataset and adapt it to our needs. In order to do so, we extracted the detections with their identities, and then for each identity we selected all possible random pairs as positive examples. For the negative set we selected several random identities. The negative dataset is much bigger than the positive dataset, so we

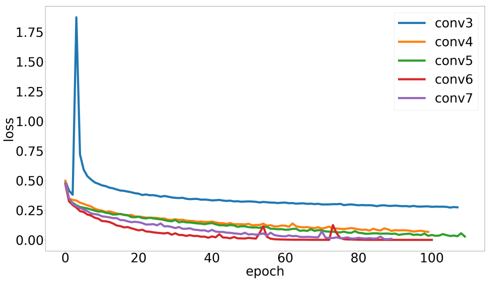
limited it to have a balanced dataset. The problem with the MOT16 dataset, is that the ground truth was built with the detections of a classifier and there is not a human intervention, resulting in a messy ground truth. We inspected the dataset and around the 70% of the dataset was wrong, there are a lot of occlusion in detections resulting in wrong pairs, pairs that are not matching with the same identity.
Then, we discarded the MOT16 dataset an used the TownCenter dataset [73] from the University of Oxford instead, which has a manual ground truth. We have got 29824 positive and negative pairs, then a dataset of 59648 image pairs. We split the dataset between training and validation set, 80% and 20% respectively. For testing we selected a set of identities of the MOT16 dataset. To regularize and enlarge our dataset we applied some data augmentation techniques to our dataset like we observe in figure 6.14. These transformations consist in: apply a random change of brightness, apply a random crop, apply a vertical flip, apply gaussian blur, appply a random shadow, apply a zoom in, apply a random rotation and translation, apply a zoom out, add gaussian noise, and apply the opposite vignetting. We tried apply all the transformations for each images but then the dataset was too noisy and the network did not converge. So we finally added only one random transformation for each pair, so we double our dataset. We stop their training when its loss do not change in 5 epochs.

(a) Original image. (b) Random brightness. (c) Random image crop. (d) Vertical flip. (e) Gaussian blur. (f) Random shadow.

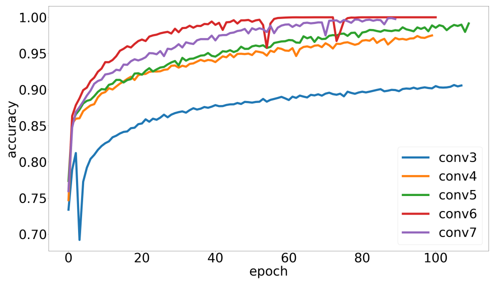
(g) Zoom in. (h) Rotation and translation. (i) Zoom out. (j) Gaussian noise. (k) Opposite vignetting.

Figure 6.14: Data augmentation.

We notice in the first test that the joint data input outperforms the other siamese configurations for the same number of convolutional layers, so we centre in this model. We increased the number of layers of that architecture, *conv I*, refers to this type of model, with *I* the number of layers. So, we explored conv3, conv4, conv5, conv6, conv7. In the figure 6.15 and 6.16 we observe the loss and the accuracy on the training and validation sets respectively. We notice that the best models are the conv6 and conv7, they have got a similar results. Despite of the regularization techniques applied to the models are a little bit noisy but it is tolerable (except conv3, it is too noisy).

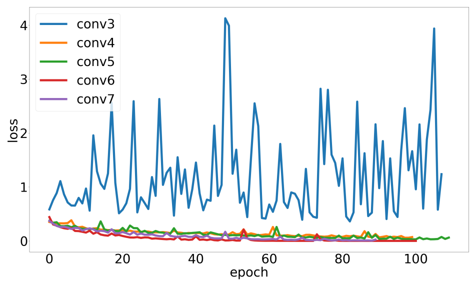


(a) Loss.

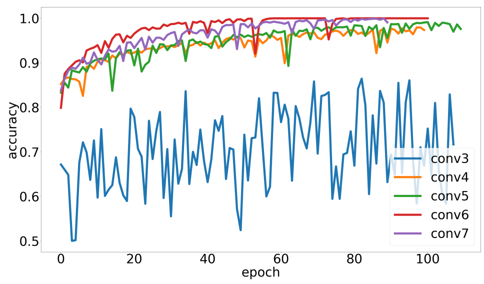


(b) Accuracy.

Figure 6.15: Results on training set.



(a) Loss.



(b) Accuracy.

Figure 6.16: Results on validation set.

We can observe the comparison using the CMC measure in the figure 6.17, we notice that the siamese network with joint data input with less layers performs better than bigger models like Inception. This remarks the idea of training jointly the feature extractor and the classifier and the need of task specific networks. Also, the siamese network with the configuration joint data input outperforms the other siamese networks. Among the siamese joint data input, the performance increases till the 6 convolutional layer architecture, then performance of the model with 7 convolutional layers drops.
Also in the figure 6.18, we observe the performance against the time consumption. The siamese network with the joint data input with 6 convolutional layers gets the best balance between performance and execution time. So this was the network included in our robust people tracker, in the reidentification module.
In the table 6.2, we studied the impact of the reidentification module, checking the algorithm with and without it. With the reidentification the identity switching (ID's)

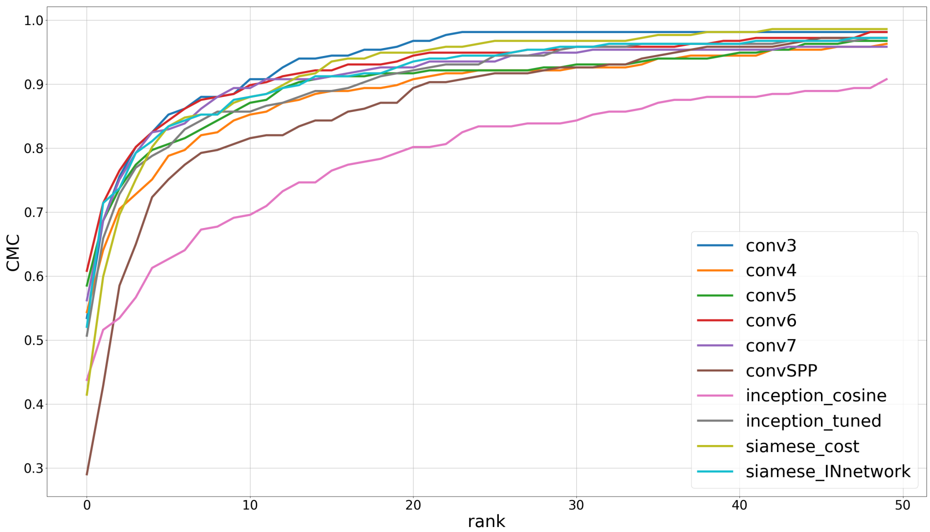


Figure 6.17: CMC plot.

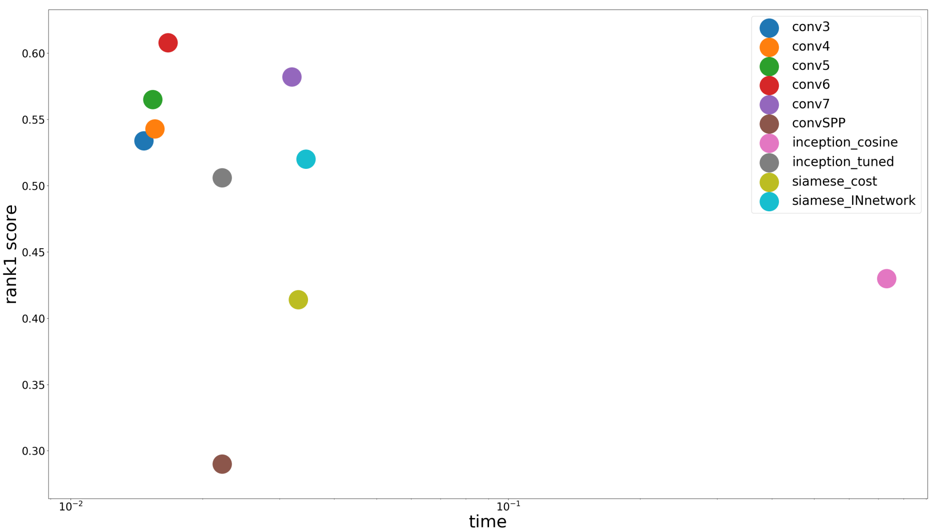


Figure 6.18: Performance-timing comparision.

is reduced around 24%, as can be seen in the numer ob ID's detected in the same dataset.

	GT	**MT**	**PT**	**ML**	**FP**	**FN**	**IDs**	**MOTA**	**MOTP**	**FPS**
Without reidentification	517	12	180	325	19098	78329	827	10.3	69.1	18.2
With reidentification	517	3	127	387	18896	78999	618	10.8	70.3	15.85

Tabla 6.2: Impact of the reidentification module.

6.4 Global validation experiments

Once we have explained the experiments for each module, in this sections we show some results of the global system.

6.4.1 Typical execution

Using the code provided by the MOT16 challenge organization, we evaluate our solution on the training set. The evaluation procedure and dataset are explained in previous sections, 5.4 and 5.3 respectively. The principal measure to compare the algoirthms is the MOTA score, this measure combines three error sources: false positives [FP], missed targets [FN] and identity switches [IDs]. Another measure is the track quality, this measure classifies each trajectory as mostly tracked [MT], partially tracket [PT], and mostly lost [ML].
We show the results of our algorithm in the table 6.3. We reach 10.8 of MOTA at 15.85 FPS, also around of 24% of the blobs are partially tracket.

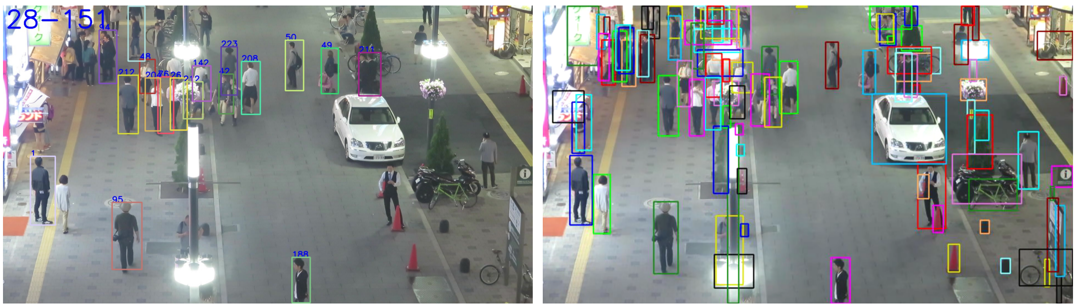
	GT	**MT**	**PT**	**ML**	**FP**	**FN**	**IDs**	**MOTA**	**MOTP**	**FPS**
Our algorithm	517	3	127	387	18896	78999	618	10.8	70.3	15.85

Tabla 6.3: Results of our algorithm.

In addition, we show the results for each sequences, we can observe the results in the table 6.4.
The algorithm gets the best performance on sequences with a fixed camera from an elevated view point and a low angle recording and close targets like sequences 4, 9, and 11. In the figures 6.19 and 6.20 we can observe a snapshot of these sequences.

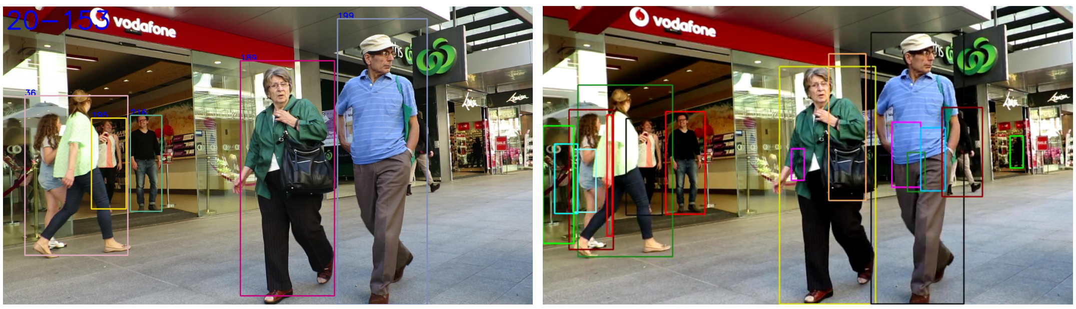
	GT	MT	PT	ML	FP	FN	IDs	MOTA	MOTP	FPS
02	54	0	13	41	2181	15526	113	0.1	67.1	9.02
04	83	0	41	42	5495	33980	290	16.6	71.1	12.3
05	125	3	43	79	28571	4713	109	-12.2	67.8	17.94
09	25	1	19	5	932	3225	71	19.7	62	10.52
10	54	0	4	50	404	11647	81	1.5	68.4	14.23
11	69	0	16	53	948	7366	72	8.6	71.4	17.49
13	107	0	9	98	1315	10743	32	-5.6	67.1	20.5
Global	517	3	127	387	18896	78999	618	10.8	70.3	15.85

Tabla 6.4: Results algorithm by sequences.



(a) Our algorithm (b) Ground truth

Figure 6.19: Comparision between our algorithm with MOT-04 ground truth.



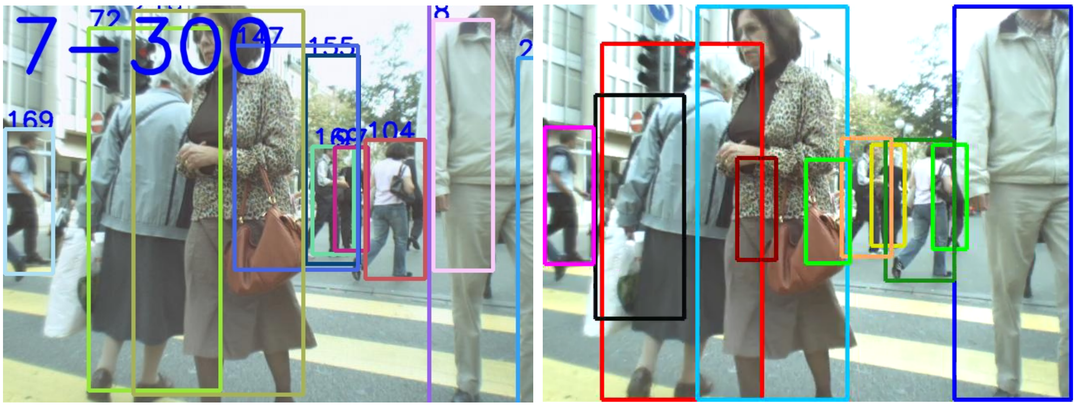
(a) Our algorithm (b) Ground truth

Figure 6.20: Comparision between our algorithm with MOT-09 ground truth.

In contrast, our algorithm struggles in sequences with low resolution like sequences 5, and when the targets are away from the camera like 13. In the figure 6.21 and 6.22 we can observe a snapshot of these sequences.

(a) Our algorithm (b) Ground truth

Figure 6.21: Comparison between our algorithm with MOT-13 ground truth



(a) Our algorithm (b) Ground truth

Figure 6.22: Comparison between our algorithm with MOT-05 ground truth

We did not evaluate on the test set because the ground truth it is not provided by the organization. You have to submit yours results on their website to get an evaluation on this set, but the website it is blocked because an upcoming conference. Evaluating our results on the test set will not reveal any furhter analyse because they are different cuts of the same video sequences

6.4.2 Comparison with other algorithms

We have compared our algorithm with the MOT16 leaderboard [98], we only include the algorithms which belong to a research paper, in the table 6.5 and the figure 6.23 we can observe those results. We observe that these algorithms overtake our solution on

	GT	MT	PT	ML	FP	FN	IDs	MOTA	MOTP	FPS
DP_NMS [99]	517	28	169	320	1123	121578	972	32.2	76.4	212.6
CEM [100]	517	40	198	279	6837	114322	642	33.2	75.8	0.3
SMOT [101]	517	22	253	242	17426	107552	3108	29.7	76.3	0.2
LP2D [72]	517	44	211	262	5084	111163	915	35.7	75.8	49.3
MDPNN [28]	517	72	287	215	2681	92856	774	47.2	75.8	1.0
LMP [102]	517	98	222	197	8886	85487	852	48.8	79	0.5
Our method	517	3	127	387	18896	78999	618	10.8	70.3	15.85

Tabla 6.5: Comprarison with the MOT's results

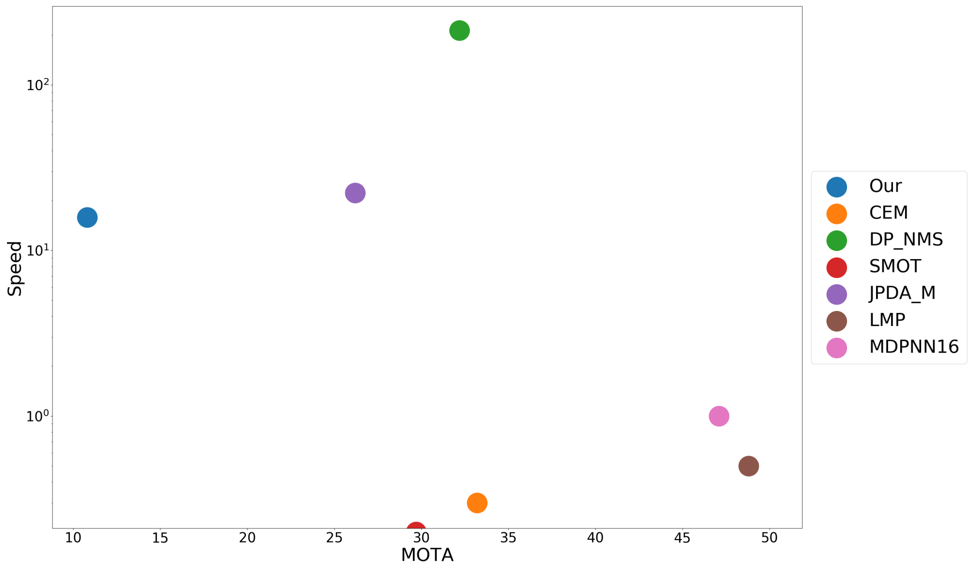


Figure 6.23: Comparision with other algorithms.

the MOTA measure but we pass them in processing speed, even their processing speed parameter does not include the execution of their detector.

These algorithms are focused on solving the data association module from the tracking-by-detection paradigm, to do so they have access to the detection at each frame. They focus on how to link those detections and do not consider the processing speed. Instead, we were focused on how to develop a tracking-by-detection with neural networks on real time.

6.5 Timing performance

To analyse the timing performance of our algorithm we used the Town Centre sequence. We chose it because it has got a representative density of pedestrian. The mean frame rate of the algorithm with the person reidentification mechanism is 15.86. In figure 6.24 we can observe a barplot of time consumption and distribution of our algorithm per frame. We notice the peaks each 30 frames, these belong to the execution of the siamese network and it depends on how many blobs detections without assignment there are.

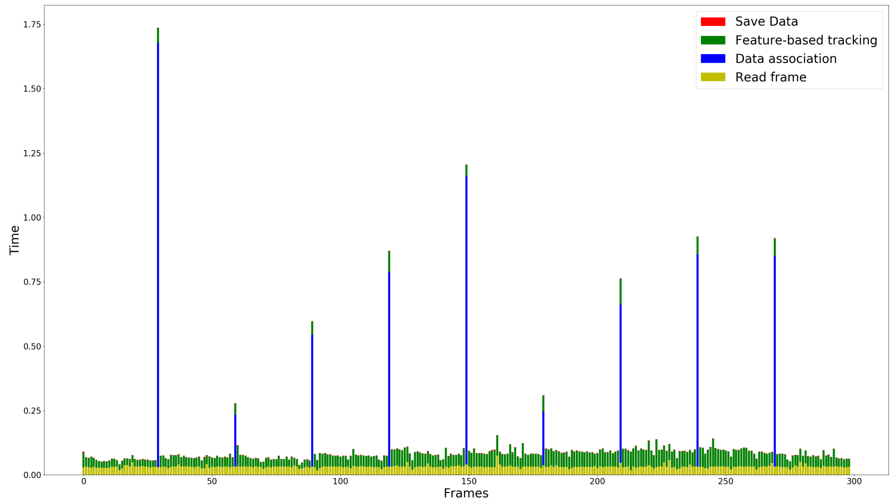


Figure 6.24: Barplot of the timming.

Getting a zoom in the figure 6.24, we can sobserve the time of reading the frame remains constant. The feature-based tracking time distribution gets a peak every the detection and after that decreases due to erasing blobs with the lost mechanism and losing some feature points. The time for save the detections mantains costant and very small.

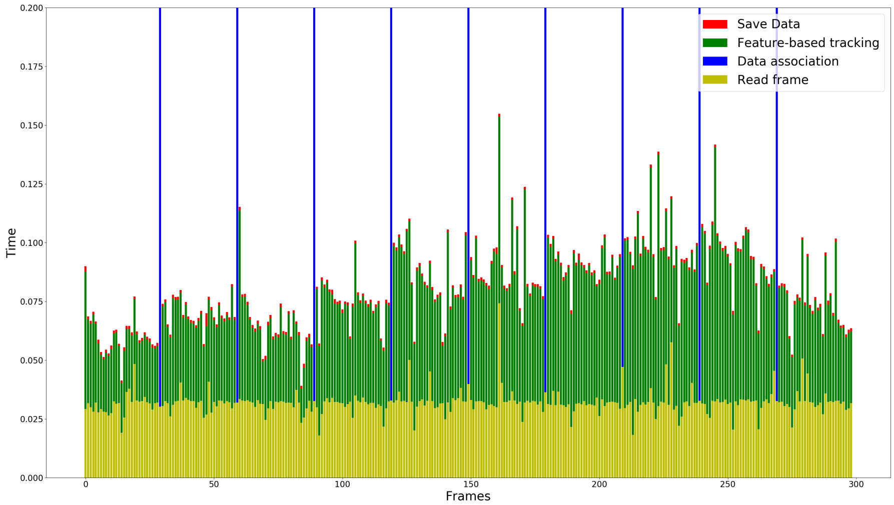


Figure 6.25: Zoom in of the barplot.

One we have analysed the global timing performance of our algorithm, we are going focus on the temporal evolution for different parameters of the feature-based tracking. The first of them it is the number of processing blobs. In the figure 6.26 we observe an histogram of the number of blobs with their correspondant execution time. When more blobs process the tracking module it increases the time of their execution. The feature-based runs in a sequential way, with more blobs it increases the number of operations of the algorithm.

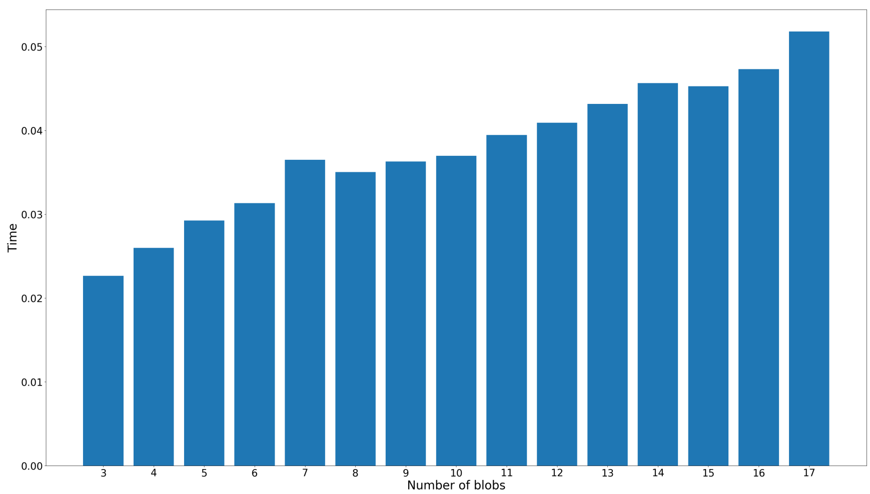


Figure 6.26: Time histogram of number of blobs.

Another parameter that we analysed it is the number of feature points that the feature-based tracking process. Increasing the number of feature points it also increases the execution time. We compute the displacement of each blob by the displacement of the features points, increasing the number of feature points, augments the execution time.

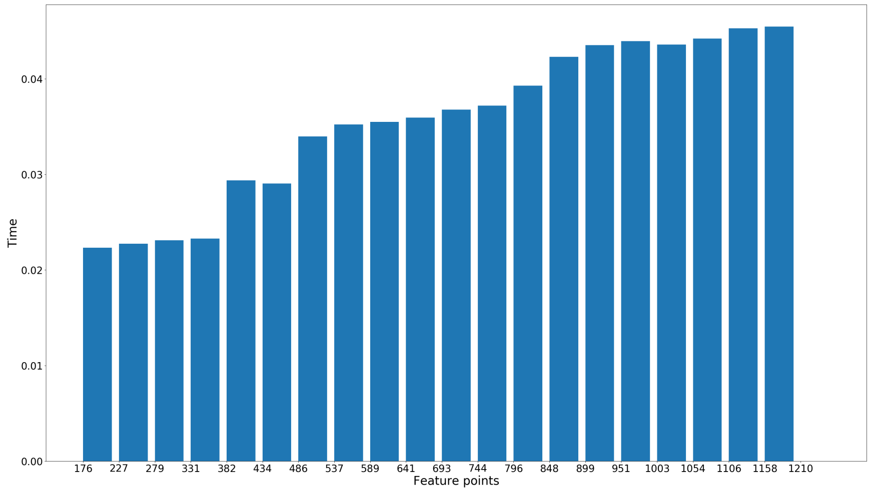


Figure 6.27: Time histogram of feature points.

Finally, the last parameter that we analysed it is the total blobs' area processed by the feature-based tracking. The execution time increase as increases the area of the blobs but at some point it reduces its rate of change.

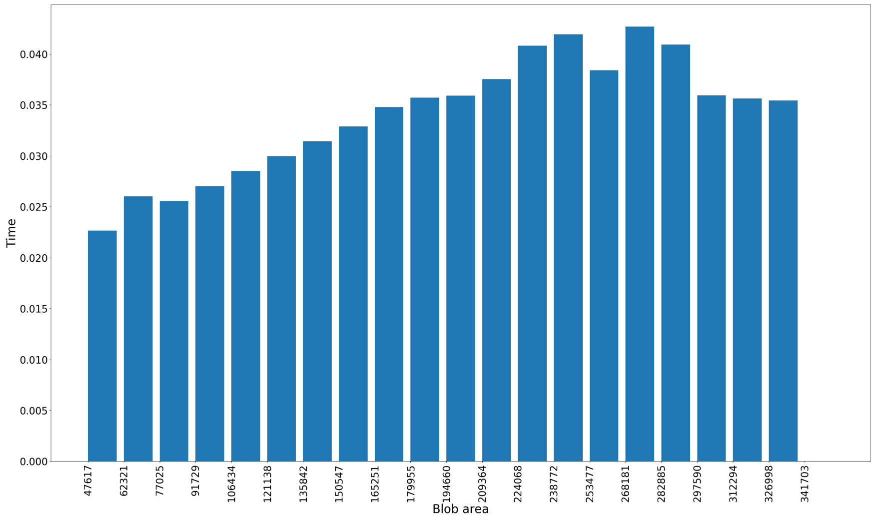


Figure 6.28: Time histogram number of area blob.

Chapter 7

Conclusions

In this chapter the main contributions of this work are summarized, and a few lines of future development are sketched.

7.1 Contributions

In this master thesis we have studied deep learning techniques and their application in a hybrid robust tracking algorithm. We were able to build a people tracking algorithm in videos that utilizes a neural network and does not miss the real time operation. To do it we used the tracking-by-detection framework, and combined people detection using a neural network with feature-based object tracking. The tracker has three functional blocks: a neural net detector, a feature-based tracker and a data association which merges tracked blobs with detected ones. The person blobs obtained with a trained neural network are associated to the tracked blobs from the feature-based tracking. All the frames are analyzed for feature-based tracking, but only a subset of them (time sampling) is used for person detection with the convolutional neural network, because this processing is slow. All the frames in the feature-based tracking thread are delayed in a buffer, waiting for the result from the CNN. This way both threads are synchronized. Thus, the system works on real time but with a time offset compared to the input video, but at the same frame rate. The data association is based on spatial proximity and distance between the tracked blobs and the new detected blobs. This have been improved integrating a person reidentification network which enhances the data association, particulary in people overlaps and crossing inside the image flow. In addition, we have studied the person reidentification problem to improve data association. We trained several siamese CNN architectures and tested with

our own dataset.
Finally we have experimentally evaluated our algorithm in a well-known challenge, MOT16, and analysed its performance and timing capabilities on it. The algorithm performs reasonably well in sequences of high frame rate and resolution, but in low frame rate and resolution sequences its performance drops dramatically.
To develop this task, in section 2 we divided the main objectives in several subtasks. Next discuss their fulfilment:

- **Object detector using deep learning**. We studied the main family of deep learning architectures for object detection like Faster Region Proposal Networks (Faster-RCNN), Region-based Fully Convolutional networks (RFN) and Single Shot Multibox Detector (SSD). We carried out a statistical comparison of them, explained in section 6.1, and with these experimental results we chose the Single Shot Multibox Detector (SSD).

- **Development of a Featured-based tracking module**. We studied several tracking methods like MeanShift and Lucas-kanade, which could fit our problem. We realized that the feature-based tracking Lucas-Kanade fits our requirements of speed and accuracy. It consist on track feature points with Optical flow and developing a simple blob matching based on the movement of those tracked points.

- **Merging detections and feature-based tracking**. Once we have developed the two previous blocks, we joined them. This combination allows to put together too different techniques, which work at very different speeds. First, a fast but brittle technique (the feature-based tracking). Second, a slow but robust one (neural network detection). This combination in the programmed prototype is one of the main contributions of this work. In addition, we added a person reidentification module based on CNN to solve the identity incongruities.

- **Testing of the component on an international databases**. We tested our solution on an international database Multiple Object Tracking 2016, and analysed the experimental results 6.4.

We can say that we have fulfilled the objectives of this work. We built a robust and hybrid people tracking algorithm with neural networks that gets a satisfactory accuracy on a dataset, and reaches a real time operation.

7.2 Future works

This work is a first entrance on robust tracking algorithm using deep learning techniques, we have reasonable results. After developing this first hybrid prototype there is room for improvement. We propose several lines to improve it.

- Migration to C++. We used a scripting programming language Python, if we switched to a compiled programing language we would increase the time performance.
- GPU implementation. Computing displacement for each blob could be computed in a parallel way and GPU hardware are the best for this kind of tasks.
- Probabilistic framework. Consider the feature-based tracking and the detections of the neural netowrk as observations of the state of the system and mix them with bayesian filter techniques.
- Improve siamese architectures. Study new siamese architectures to increase the accuracy of the reidentification module, like inception stem of InceptionV3 or include the optical flow information into the neural network.
- Enhance data association. Use more confident techniques to associate the detections, current state of the art methods relay on probabilistic graphical models to solve the data association problem.

Bibliography

[1] Kai Briechle and Uwe D. Hanebeck. Template matching using fast normalized cross correlation, 2001.

[2] Simon Baker, Ralph Gross, and Iain Matthews. Lucas-kanade 20 years on: A unifying framework: Part 3. *International Journal of Computer Vision*, 56:221–255, 2002.

[3] H. T. Nguyen and A. W. M. Smeulders. Fast occluded object tracking by a robust appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1099–1104, Aug 2004.

[4] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift, 2000.

[5] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M. M. Cheng, S. L. Hicks, and P. H. S. Torr. Struck: Structured output tracking with kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2096–2109, Oct 2016.

[6] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[7] Anton Milan, Stefan Roth, and Konrad Schindler. Continuous energy minimization for multitarget tracking. *IEEE transactions on pattern analysis and machine intelligence*, 36(1):58–72, 2014.

[8] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1409–1422, 2012.

[9] Federico Pernici and Alberto Del Bimbo. Object tracking by oversampling local features. *IEEE transactions on pattern analysis and machine intelligence*, 36(12):2538–2551, 2014.

[10] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[11] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[12] B Widrow. An adaptive 'adaline'neuron using chemical 'memistors', 1553–1552, 1960.

[13] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989.

[16] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119 – 130, 1988.

[17] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147 – 169, 1985.

[18] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.

[19] Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Readings in speech recognition. chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[20] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[22] M. Pieras. Wiki jderobot: Marcos pieras. `http://jderobot.org/Marcospieras-tfm` [2017-06-11], 2017.

[23] M. Pieras. Github tfm: Marcos pieras. `https://github.com/RoboticsURJC-students/2015-TFM-Marcos-Pieras` [2017-06-11], 2017.

[24] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 FPS with deep regression networks. *CoRR*, abs/1604.01802, 2016.

[25] Hyeonseob Nam and Bohyung Han. Learning multi-domain convolutional neural networks for visual tracking. *CoRR*, abs/1510.07945, 2015.

[26] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Learning spatially regularized correlation filters for visual tracking. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[27] Ran Tao, Efstratios Gavves, and Arnold W. M. Smeulders. Siamese instance search for tracking. *CoRR*, abs/1605.05863, 2016.

[28] 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.

[29] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[31] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[32] Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[35] Kye-Hyeon Kim, Yeongjae Cheon, Sanghoon Hong, Byung-Seok Roh, and Minje Park. PVANET: deep but lightweight neural networks for real-time object detection. *CoRR*, abs/1608.08021, 2016.

[36] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[38] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.

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

[40] Jianbo Shi and Carlo Tomasi. 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.

[41] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185 – 203, 1981.

[42] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[43] Liang Zheng, Yi Yang, and Alexander G. Hauptmann. Person re-identification: Past, present and future. *CoRR*, abs/1610.02984, 2016.

[44] Douglas Gray and Hai Tao. *Viewpoint Invariant Pedestrian Recognition with an Ensemble of Localized Features*, pages 262–275. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[45] Alexis Mignon. Pcca: A new approach for distance learning from sparse pairwise constraints. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2666–2672, Washington, DC, USA, 2012. IEEE Computer Society.

[46] R. Zhao, W. Ouyang, and X. Wang. Learning mid-level filters for person re-identification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 144–151, June 2014.

[47] Joost van de Weijer, Cordelia Schmid, Jakob Verbeek, and Diane Larlus. Learning color names for real-world applications. *Trans. Img. Proc.*, 18(7):1512–1523, July 2009.

[48] M. Köstinger, M. Hirzer, P. Wohlhart, P. M. Roth, and H. Bischof. Large scale metric learning from equivalence constraints. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2288–2295, June 2012.

[49] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, June 2009.

[50] Shengcai Liao, Yang Hu, and Stan Z. Li. Joint dimension reduction and metric learning for person re-identification. *CoRR*, abs/1406.4216, 2014.

[51] M. Chatterjee, Y. Luo. Similarity learning with cnn. `http://slazebni.cs.illinois.edu/spring17/lec09_similarity.pdf` [2016-12-15], 2016.

[52] Jane Bromley, Isabelle Guyon, Yann Lecun, Eduard Säckinger, and Roopak Shah. Signature verification using a ”siamese” time delay neural network. In *In NIPS Proc*, 1994.

[53] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, June 2005.

[54] Nam N. Vo and James Hays. Localizing and orienting street views using overhead imagery. *CoRR*, abs/1608.00161, 2016.

[55] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[56] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. *CoRR*, abs/1504.03641, 2015.

[57] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015.

[58] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. on Graphics (SIGGRAPH)*, 34(4), 2015.

[59] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[60] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. Learning by tracking: Siamese CNN for robust target association. *CoRR*, abs/1604.07866, 2016.

[61] E. Ahmed, M. Jones, and T. K. Marks. An improved deep learning architecture for person re-identification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3908–3916, June 2015.

[62] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[63] Amazon Inc. Amazon mechanical turk. `https://www.mturk.com/mturk/welcome` [2017-02-21], 2017.

[64] University of Oxford. Pascal voc07. `http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html` [2017-02-21], 2017.

[65] University of Oxford. Pascal voc12. `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html` [2017-02-21], 2017.

[66] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[67] Stanford University. Imagenet. `http://www.image-net.org/` [2017-02-21], 2017.

[68] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[69] Microsoft Inc. Microsoft coco challenge. `http://mscoco.org/` [2017-02-21], 2017.

[70] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[71] J. Ferryman and A. Ellis. Pets2010: Dataset and challenge. In *Proceedings of the 2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, AVSS '10, pages 143–150, Washington, DC, USA, 2010. IEEE Computer Society.

[72] Laura Leal-Taixé, Anton Milan, Ian D. Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. *CoRR*, abs/1504.01942, 2015.

[73] University of Oxford. Town centre dataset. `http://www.robots.ox.ac.uk/ActiveVision/Research/Projects/2009bbenfold_headpose/project.html#datasets` [2017-02-21], 2011.

[74] Rainer Stiefelhagen, Keni Bernardin, Rachel Bowers, John Garofolo, Djamel Mostefa, and Padmanabhan Soundararajan. *The CLEAR 2006 Evaluation*, pages 1–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[75] Bo Wu and R. Nevatia. Tracking of multiple, partially occluded humans based on static body part detection. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 951–958, June 2006.

[76] UC Santa Cruz. Viper: Viewpoint invariant pedestrian recognition. `https://vision.soe.ucsc.edu/node/178` [2016-12-15], 2007.

[77] T. Wang, S. Gong, X. Zhu, and S. Wang. Person re-identification by discriminative selection in video ranking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(12):2501–2514, Dec 2016.

[78] University of London. Qmul underground re-identification. `http://personal.ie.cuhk.edu.hk/~ccloy/downloads_qmul_underground_reid.html` [2016-12-15], 2009.

[79] Doug Gray, Shane Brennan, and Hai Tao. Evaluating appearance models for recognition, reacquisition, and tracking. In *In IEEE International Workshop on Performance Evaluation for Tracking and Surveillance, Rio de Janeiro*, 2007.

[80] Martin Hirzer, Csaba Beleznai, Peter M. Roth, and Horst Bischof. Person re-identification by descriptive and discriminative classification. In *Proc. Scandinavian Conference on Image Analysis (SCIA)*, 2011.

[81] Abir Das, Anirban Chakraborty, and Amit K. Roy-Chowdhury. *Consistent Re-identification in a Camera Network*, pages 330–345. Springer International Publishing, Cham, 2014.

[82] Wei Li, Rui Zhao, and Xiaogang Wang. *Human Reidentification with Transferred Metric Learning*, pages 31–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[83] W. Li and X. Wang. Locally aligned feature transforms across views. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3594–3601, June 2013.

[84] W. Li, R. Zhao, T. Xiao, and X. Wang. Deepreid: Deep filter pairing neural network for person re-identification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 152–159, June 2014.

[85] N. Martinel and C. Micheloni. Re-identify people in wide area camera network. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 31–36, June 2012.

[86] Peter M. Roth, Martin Hirzer, Martin Köstinger, Csaba Beleznai, and Horst Bischof. *Mahalanobis Distance Learning for Person Re-identification*, pages 247–267. Springer London, London, 2014.

[87] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. In *Computer Vision, IEEE International Conference on*, 2015.

[88] Patrick Grother, Ross J. Micheals, and P. Jonathon Phillips. *Face Recognition Vendor Test 2002 Performance Metrics*, pages 937–945. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[89] Alphabet Inc. Object detetection in tensorflow. `https://github.com/tensorflow/models/tree/master/object_detection` [2017-02-21], 2017.

[90] J. Redmon. You only look once. `https://github.com/pjreddie/darknet` [2017-06-11], 2017.

[91] Wei Lee. Single shot detector caffe. `https://github.com/weiliu89` [2017-02-21], 2017.

[92] Paul B. Single shot detector tensorflow. `https://github.com/balancap/SSD-Tensorflow` [2017-02-21], 2017.

[93] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2129–2137, 2016.

[94] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 2756–2759. IEEE, 2010.

[95] Jay Rambhia. Median flow tracker in python. `https://github.com/jayrambhia/MFTracker` [2015-08-15], 2015.

[96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society.

[97] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[98] MOTChallenge. Mot 16 results. `https://motchallenge.net/results/MOT16/` [2017-02-21], 2017.

[99] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR 2011*, pages 1201–1208, June 2011.

[100] A. Milan, S. Roth, and K. Schindler. Continuous energy minimization for multitarget tracking. *IEEE TPAMI*, 36(1):58–72, 2014.

[101] C. Dicle, O. I. Camps, and M. Sznaier. The way they move: Tracking multiple targets with similar appearance. In *2013 IEEE International Conference on Computer Vision*, pages 2304–2311, Dec 2013.

[102] Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Multi-person tracking by multicut and deep matching. *CoRR*, abs/1608.05404, 2016.