ISCTE IUL

University Institute of Lisbon

Department of Information Science and Technology

Real Time Multiple Camera Person Detection and Tracking

Dária Baikova

Dissertation submitted in partial fulfillment of the Requirements for the Degree of

Master in Computer Engineering

Supervisor

PhD João Carlos Amaro Ferreira, Assistant Professor

ISCTE-IUL

Co-Supervisor

PhD João Pedro Afonso Oliveira da Silva, Assistant Professor

ISCTE-IUL

October 2018

Resumo

À medida que a quantidade de dados de vídeo cresce, os esforços para criar sistemas inteligentes capazes de observar, entender e extrapolar informação útil destes dados intensificam-se. Nomeadamente, sistemas de detecção e *tracking* de objectos têm sido uma àrea amplamente investigada nos últimos anos. No presente trabalho, desenvolvemos um protótipo de *tracking* multi-câmara, multi-objecto que corre em tempo real, e que usa várias câmaras *fish-eye* estáticas de topo, com sobreposição entre elas. O objetivo é criar um sistema capaz de extrapolar de modo inteligente e automático as trajetórias de pessoas a partir de imagens de vigilância. Para resolver estes problemas, utilizamos diferentes tipos de técnicas, nomeadamente, uma combinação do poder representacional das redes neurais, que têm produzido excelentes resultados em problemas de visão computacional nos últimos anos, e algoritmos de *tracking* mais clássicos e já estabelecidos, para representar e seguir o percurso dos objectos de interesse. Em particular, dividimos o problema maior em dois sub-problemas: *tracking* de objetos de uma única câmera e *tracking* de objetos de múltiplas câmeras, que abordamos de modo modular. A nossa motivação a longo prazo é implmentar este tipo de sistema em aplicações comerciais, como áreas comerciais ou aeroportos, para que possamos dar mais um passo em direcção a sistemas de vigilância visual inteligentes.

Palavras-chave: Detecção de Objectos, Tracking de Objectos, Deep Learning, Visão Computacional.

Abstract

As the amount of video data grows larger every day, the efforts to create intelligent systems able to perceive, understand and extrapolate useful information from this data grow larger, namely object detection and tracking systems have been a widely researched area in the past few years. In the present work we develop a real time, multiple camera, multiple person detection and tracking system prototype, using static, overlapped, fish-eye top view cameras. The goal is to create a system able to intelligently and automatically extrapolate object trajectories from surveillance footage. To solve these problems we employ different types of techniques, namely a combination of the representational power of deep neural networks, which have been yielding outstanding results in computer vision problems over the last few years, and more classical, already established object tracking algorithms in order to represent and track the target objects. In particular, we split the problem in two sub-problems: single camera multiple object tracking and multiple camera multiple object tracking, which we tackle in a modular manner. Our long-term motivation is to deploy this system in a commercial application, such as commercial areas or airports, so that we can build upon intelligent visual surveillance systems.

Keywords: Object Detection, Object Tracking, Deep Learning, Computer Vision.

Acknowledgements

Firstly, I must thank my supervisors, Professor João Ferreira and Professor João Oliveira, who, through countless meetings, supplied the knowledge, scientific input and suggestions that made this work possible and who have shaped my approach to research. I am very grateful for their support and patience.

I would like to thank everyone in INOV-INESC Inovação, who made work an enjoyable experience. Specifically, I want to thank Pedro Santos, for his constant advice and feedback, and for always pushing me to take incrementally difficult challenges and inspiring me to think bigger. I also want to thank to my colleges Gonçalo and Tomás, who shared this journey with me from the beginning, and without whom this experience would not be the same.

A big thanks to my friends for the endless support and encouragement to keep going. And finally, I want to thank all my family. Special thanks to my dad, who inspired me to go into computer science in the first place, and to my mom, who provided unconditional support and who always believes in me 100 times more than I do.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The ability to see, understand and interact with our surroundings is a defining human characteristic. We do this through our vision, which is the primary sensing mechanism for humans and which enables us to easily perform complex visual recognition tasks. By processing large amounts of data obtained through our visual system we are able to perform tasks such as glancing at an image and immediately understand it. Up until recently, this ability was restricted to humans and other living beings, however with recent technological advancements, this ability is now being passed on to machines. Computer vision is the field of computer science which strives at giving machines the same ability as the human visual system: to perceive the environment similarly to what we do with our eyes and brain. Namely, due to rapid technological progress, computers are now able to perform at human level in complex visual recognition tasks, and even surpassing human level in tasks such as distinguishing between thousands of fine-grained image categories (He, Zhang, Ren, & Sun, 2015). Due to these advancements, powered by the increasingly growing availability of image and video data, along with continuous improvements in hardware and software, computer vision is used today in various types of commercial applications, such as medical imaging, face detection, self-driving vehicles, image search, visual surveillance etc. However, despite the encouraging prospects, there are problems which remain unsolved. One of these problems is object detection and tracking, which has seen many advancements in recent years, however, is still not optimal and efficient enough to be deployed in commercial applications solely by means of computer vision, without the use of other technologies (e.g., traditional sensors such as LIDAR). In many cases, these applications, need to process data and output results online and in real time which is still a challenge for these systems which can not guarantee that the information is processed fast enough in a reliable way. The present work strives to relieve these problems by focusing on the research, implementation, and evaluation of state-of-the art object detection and tracking systems with the long-term motivation of building towards a solution that is reliable enough to be deployed and used by real-world applications.

1.1 Overview

This section presents an overview of the object tracking problem as well as an outline of the progress of recent years made in computer vision.

Tracking as a computer vision problem: Object tracking is the process of detecting multiple objects of interest in a video frame[1] and maintain the identities of these objects in the subsequent frames (over time) in order to identify the trajectory of each object in the video. This is a simple task for humans: due to our visual system, we are able to glance at a video and easily follow and describe the trajectory of an object of interest. However, it is still a challenging problem for computers. A video frame from the perspective of a computer is a large array of numbers that encode pixel intensities in each location of the image. In order to solve this problem, firstly, a computer must translate these pixel values into concepts, such as "person" to identify the objects to be tracked, and this must be done solely by pixel intensities and the relationships between pixels. The system has to process large amounts of pixel information multiple times in order to be able to extract useful features. Secondly, in order to create the track of one person through the video, a computer must recognize that the person in one frame

[1] Generally in the object tracking context, the detection outputs are the bounding boxes which enclose each of the objects of interest in a frame.

is the same person as in the next frame. For this to be possible the machine must correctly match vast amounts of feature information between frames or perform complex probabilistic calculus. Moreover, this process should be repeated throughout all frames in a video, despite variations, such as differences in the appearance and pose of a person, occlusions and other tracking problems which will be addressed in the next sections. In summary, the task of object tracking in video, which comes naturally to humans, involves a complex pattern recognition and probability estimation problem for machines.

Deep learning in visual recognition tasks: Regardless of the difficulty of these tasks, in the past few years, a great amount of progress has been made in computer vision. Namely, data-driven techniques became more prominent in order to create efficient algorithms to analyze the content of images and videos. Specifically, supervised deep learning methods achieved promising results in image and video analysis tasks, such as image classification (Krizhevsky, Sutskever, & Hinton, 2012; LeCun et al., 1989; Simonyan & Zisserman, 2014; Szegedy, Liu, et al., 2015; He, Zhang, Ren, & Sun, 2016) and object detection (Girshick, Donahue, Darrell, & Malik, 2014; Girshick, 2015; Redmon, Divvala, Girshick, & Farhadi, 2016; Liu et al., 2016; Ren, He, Sun, & Girshick, 2017). More recently, although less common, deep learning methods also started being employed in object tracking problems (Gan, Guo, Zhang, & Cho, 2015; Ning et al., 2017; Milan, Rezatofighi, Dick, Reid, & Schindler, 2016; Held, Thrun, & Savarese, 2016; Bertinetto, Valmadre, Henriques, Vedaldi, & Torr, 2016), which usually follow a *tracking-by-detection* framework (Andriyenko, Schindler, & Group, 2011; Choi, 2015). This framework consists of two steps: applying a detector, and subsequently, a post-processing step witch involves a tracker to propagate detection scores over time.

1.2 Motivation

Despite the encouraging progress in computer vision of recent years, the problem of multiple object tracking is still very far behind other visual recognition tasks, as some limitations remain unsolved. For context, due to the advent of deep learning techniques, the error rate of the image classification task in the ImageNet Large Scale Visual Recognition Challenge challenge (ILSVRC) (Russakovsky et al., 2015) is today below 1%[2], while in the MOTChallenge (Leal-Taixé, Milan, Reid, Roth, & Schindler, 2015), for instance on the MOT17 dataset, the current highest MOTA score is about 54.5%[3]. This difference is due to the high number of complexities, limitations, and difficulties presented by the problem of multiple object tracking when compared to other visual recognition tasks. We explore a few of these challenges below:

- **Datsets.** The first limitation is that supervised methods rely on large amounts of annotated training data in order to be able to generalize to previously unseen data. The recent deep learning methods proposed for visual tasks (Krizhevsky et al., 2012) partially own their success to the existence huge datasets such as the Pascal dataset (Everingham et al., 2014), the ImageNet dataset (Russakovsky et al., 2015) or the COCO –Common Objects in Context– dataset (Lin et al., 2014). Similary data-driven approaches to object tracking also rely on popular tracking datasets such as the Visual Object Tracking dataset (Kristan et al., 2017) or the Multiple Object Tracking dataset (Milan, Leal-Taixe, Reid, Roth, & Schindler, 2016).

  As so, in order to take advantage of the potential of deep learning and use it in order to implement a tracking system, there has to be a great amount and variety of labeled data. However, the available datasets are not always suitable for all problems and scenarios. If the problem specification has different configurations from the available datasets –for instance, different perspectives, different camera configurations, the need to have image data captured from multiple cameras simultaneously– then the need to create data is apparent, and creating data is a very slow and labor-intensive process.

[2]http://image-net.org/challenges/LSVRC/2016/results

[3]https://motchallenge.net/results/MOT17/?chl=10&orderBy=MOTA&orderStyle=ASC&det=Public

- **Differences in appearance and pose.** Some trackers heavily rely on appearance models in order to associate detections between frames (Wojke, Bewley, & Paulus, 2018). These types of trackers create an internal feature vector for each target, and if the appearance changes drastically between frames, so will the feature vectors, which can lead to wrongly associated detections and consequentially, identity switches. Many factors can contribute to the change in the appearance of a target from one frame to another, which include variations in lighting, motion blur and scale variations, such as rotation, deformation, and transformation of the targets.

- **Fast target movement.** Many trackers assume a prior motion model of some sort. For instance, the visual tracker codenamed GOTURN (Held et al., 2016) assumes that the objects do not move too fast from frame to frame. This tracker employs a frame cropping system in order to find the object from the frame at time t in the frame at time $t + 1$. Specifically the bounding box of the object at time t is centered on the same point in the frame at time $t + 1$ and re-scaled by a k factor. The resulting bounding box constitutes the search area where the tracker will be looking for the previous object. If the object moves too quickly, it may not be present in the search area, and thus, the tracker will not be able to locate it. Therefore, the FPS rate of the final system should be a compromise between the speed at which objects move, which should be low enough to guarantee that the objects do not move too fast from frame to frame, and the time taken to process frames, since with a higher frame rate more frames have to be processed, which can be costly for a multiple object tracking real-time system.

- **Partial/total occlusions.** In a video sequence, one tracked object may fall behind another object, or leave the field of view completely. Intuitively, if the object is not present at the frame the tracker will not be able to localize it. Humans are able to track a target through a video, have that target disappear for a few frames, and recognize the same target when it reapers. We do this by recognizing that the appearance and location of the new object are similar to the previous object, and thus are able to associate both as being the same object. The same applies to computers: this can be solved with, for instance, an appearance model (Wojke & Bewley, 2018), which can recognize that both objects are similar through feature vectors, or simple heuristics, e.g., if one object disappears from the field of view, and after a few frames a very similar object reapers in a position spatially close to the one where the previous object disappeared, then both objects are probably the same object. Additionally, the occlusion rate increases in scenarios in which the environment is very cluttered. A situation in which two, or more targets are very close together may create a problem for the tracker since one may occlude the other or the bounding boxes enclosing the objects have a high overlap, and thus, features are very similar, which may easily lead to identity switches.

- **Tracker drifts.** Track recovery in case of drift from the object of interest is essential for applications in which the object identifiers matter and identity switches are undesirable. A good algorithm should strive to minimize this drift in order to maintain accuracy over time.

- **Maintaining and ending a track.** One question that arises when performing object tracking is how to know when the track of a target is finished. If the tracker fails to detect a target in the current frame, does it mean that the target really left, or is it simply due to occlusion or detection failures? As objects cannot teleport themselves, the tracker should be able to determine that if a target disappears while at the center of the field of view it should be an occlusion, and be prepared for the target to reappear in the next frames. Similarly, if a target fails to be detected next to an entrance or exit, the tracker should assume that the target left the field of view. However, this creates other challenging questions, for instance, for how many frames should the tracker wait for an object to reappear? If this number is too high, the risk of identity switches is greater, if is too low, multiple identities can be assigned to the same object.

- **Multi Camera Tracking.** In a scenario with multiple overlapping cameras, the problem is how to know that the object in one frame from taken by one camera is the same object as in another frame taken by another camera. Note that frames should also be synchronized by timestamp: the multiple cameras feed should be aligned through time for the processing to be done correctly. The synchronization presents another challenge for a real-time application, as feeds are susceptible to suffering from network delays, and it is remarkably non-trivial to get a timestamp from an RTSP stream. Another consideration that arises is how should geo-referencing be done between frames of different cameras, and should it be done at all. One approach to avoid geo-referencing of frames completely is to use a high performing appearance model, in order to directly compare objects between different cameras. However, this approach is also costly since it comes with other challenges, such as building an appearance dataset and finding a model that is able to learn from top-view fish-eye like image data.

If these limitations are surpassed, the practical applications of a system able to track multiple objects through multiple cameras would be countless. For instance, smart-surveillance systems, that allow locating every person in any area at each instant in time could be used and deployed in numerous fields –from smart visual security surveillance systems to retail applications, as already is being done by Amazon Go [4]. As so, the short-term motivation for the present work is to explore the current state-of-the-art of multiple object tracking and attempt to build a multiple-camera, top-view tracker prototype that is able to respond to these limitations. Specifically, we focus on surveillance systems, and how objects can reliably be tracked as they move through a space from top-view security footage. As the present work is ongoing and will continue on beyond this dissertation, for the long-term motivation, we would like to perfect a system that is deployable in a real-world scenario, and employs solely computer vision techniques, building up towards intelligent visual systems.

[4]`https://www.amazon.com/b?ie=UTF8&node=16008589011`

1.3 Objectives

The present work tackles surveillance video analysis, specifically the problem of multiple person tracking from top view footage generated by multiple cameras. The main objective is to develop, implement and evaluate a state-of-the-art multiple person tracking prototype able to track people from top-view cameras in real time, using solely computer vision techniques. Additionally, since to the best of our knowledge, there is currently no existing multiple camera, top-view multiple object tracking dataset available for research, the following goal is the construction of these datasets that support the tracking algorithms. Therefore, one of the additional challenges of the present work is to build a fully labeled top-view multi-camera dataset. This is a challenge as it is an extremely labor intensive and difficult process, not only because the final datasets should be large in size, but also because they should be varied and clean. Additionally, challenging and ambiguous cases should be treated according to a standard in order to avoid noisy data (e.g., if there is a leg in the frame should it be considered a person? or should there be a class "leg"?) and we need to ensure that the data is balanced, i.e, too many examples of one class, and too few of another.

Finally, since we will be working with our own custom data, the next goal is to find a way in which the used algorithms can successfully be evaluated –as we have no benchmark. For this objective, we will need to generate evaluation sequences and evaluate different types of trackers on these sequences, in order to understand how different algorithms behave, and what methods work best on this kind of data. In summary, the main objectives are the following:

1. Study the current multiple object tracking state-of-the-art and understand what are the currently existing solutions: specifically, we want to explore different object tracking algorithms and understand which types of algorithms work better for our specific scenario.

2. Additionally, we want to explore how deep learning techniques, which have had success in other visual recognition tasks, are being applied to multiple

object tracking, and establish whether it is feasible to employ deep learning in our scenario;

3. Based on the research of the previous objective, we want to implement a multiple camera, multiple object detection and tracking prototype. For this objective, we would like to test multiple tracking algorithms, and combine classical solutions with more recent methods;

4. As most algorithms are data-driven, an additional necessity, which therefore becomes an objective, is to build top-view datasets, suitable for the different types of models that are employed;

5. Since we are using custom data, and no benchmark is available, we need to evaluate the algorithms and find ways to compare them in order to decide which work best;

1.4 Outline

In **Chapter 2** we provide the basic supporting concepts related with deep learning needed to better understand how these algorithms may be applied to object tracking problems, and how we can utilize them in our scenario. We provide some mathematical background, including optimization and backpropagation. Additionally, we discuss the most prominent visual recognition tasks, the advances that took place in the past few years in this field, and the most frequently used deep learning architectures for visual recognition problems.

In **Chapter 3** we focus on the state-of-the-art which supports the more specific object tracking problem. We review observation modeling algorithms, including motion models and appearance models, as well as different trackers. We explore more classical tracking variants, such as probabilistic methods, and data association methods, as well as more recent deep learning architectures aimed at object tracking. We explore both Visual Object Tracking (VOT) and Multiple Object Tracking (MOT) techniques.

As our final goal of real-time multiple camera, multiple person tracking is a complex task, we split this task into two main stages: single camera, multiple person tracking, and multiple camera, multiple person tracking. In **Chapter 4** we develop and present the proposed architecture for the single camera, multiple person tracking system. In **Chapter 5** we discuss how we scaled the single camera tracking system to multiple cameras and present the proposed architecture for the final system.

In **Chapter 6** we present the system evaluation, in which we evaluate the used detectors, trackers and appearance models. We present and analyze the results. Additionally, as we have not found a dataset suitable for our specific problem, we describe how we build our own datasets, which we used in the present work to train and evaluate the algorithms. Additionally, we discuss the process of building these datasets from scratch, and what consideration went into it.

Finally in **Chapter 7**, we reason about what was achieved by the present work so far, and identify which are the remaining future challenges.

Chapter 2

Deep Learning Supporting Concepts

In recent years, deep neural networks (DNNs) (Krizhevsky et al., 2012; LeCun, Bottou, Bengio, & Haffner, 1998) have had great success in the field of computer vision and are today the state-of-the-art for tasks such as image classification, object localization, and object detection. This chapter gives a basic overview of deep learning techniques, addressing the basic technical background necessary in order to better understand how we can build upon these concepts and create more complex tracking algorithms. We start with a brief introduction to Deep Neural Networks (DNNs), address what types of DNNs currently exist and how they are used in practice, and end with an overview of what DNNs are able to achieve in the context of visual recognition tasks, by presenting some of the most relevant results today. For more in-depth and thorough introductions we recommend the work in (Nielsen, 2015; Goodfellow, Bengio, & Courville, 2016).

2.1 Deep Neural Networks (DNNs)

A DNN is a machine learning algorithm and a type of Artificial Neural Network (ANN). The most basic type of DNN is a fully-connected vanilla feed-forward neural network. Internally, each of these networks is composed by layers of neurons, stacked on top of each other, where the output of each of neuron in each layer is the input to the following layer (Figure 2.1). The term *deep learning* arises from having multiple of these layers in one network.

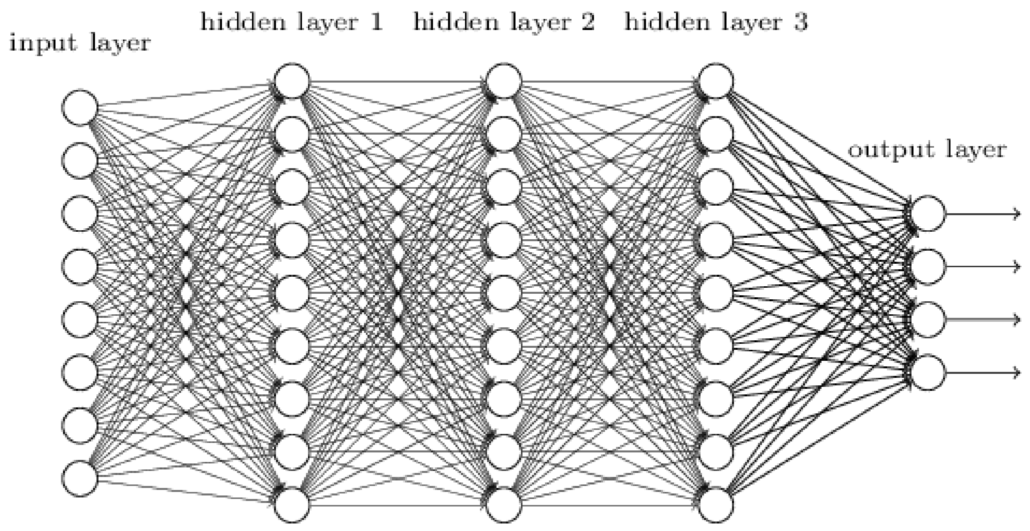


Figure 2.1: Fully connected vannila feed-forward neural network (Retrived from (Nielsen, 2015).

These algorithms allow representing complex concepts and the relationships between them out of simple features. Each hidden layer produces a higher level feature representation, more specific than in the previous layer (its input). The main task of a DNN is to find the function f^* that best describes all training examples. At a high level, the training process for a DNN consists of making it learn a representation –features– of the training data so that it is able to generalize to new data. DNNs are used in different domains: in computer vision features are constituted by elements such as edges, corners, and contours, and the task is to map a set of input pixels to a class label (Krizhevsky et al., 2012); in text processing features may be concepts attached to words and phrases, and the purpose may be to map phrases to meanings and emotions (sentiment analysis) (Socher, Perelygin, & Wu, 2013).

2.1.1 Overview

The goal of a neural network is to *learn* the optimal mapping function f^* between a set of inputs $x \in X$ and a set of outputs $y \in Y$, from a dataset of training examples $\{(x_1, y_1), ..., (x_n, y_n)\}$ which follows some probability distribution P. The goal is to find the function f^* that best describes the training examples and can produce the mapping $y = f(x, \theta)$ from an input x to an output y with learned parameters θ . The optimization problem is to learn the function f^* that minimizes the expected loss function, $L(f(x), y)$ over the data of distribution P. The loss function allows the model to quantify the discrepancy between the predicted value $f(x_i) = \hat{y}_i$ and the true label y_i, and at each iteration, the neural network will try to correct that error by slightly tweaking its parameters θ. Since not all data points of P are known, the loss approximation is averaged across all available training examples. Once we learn f^*, the original training data is discarded and f^* is employed to map previously unseen inputs $(x, y) \in P$ and predict its labels/values. In other words, the network will try to find the function $f(\theta)^*$ that approximates the following equation:

$$\theta^* \approx \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i; \theta), y_i) \tag{2.1}$$

2.1.2 Forward pass

In a vanilla feed-forward neural network, each neuron in one layer connects to each neuron in the following layer, i.e, the output of every neuron is input to the neurons of the next layer. Every connection has a weight, w, associated with it, which states how much the input is important to the output, and every layer has a bias b. The weights and biases define the parameters θ to function f that the network will learn during its training. At each iteration, each neuron will compute its output based on the weights of the connections, the output of the previous layer, and the bias as depicted in Figure 2.2. The information flows forward from the input nodes, through the hidden layers, and to the output nodes. This process of propagating information forward through the network is typically called the *forward-pass.*

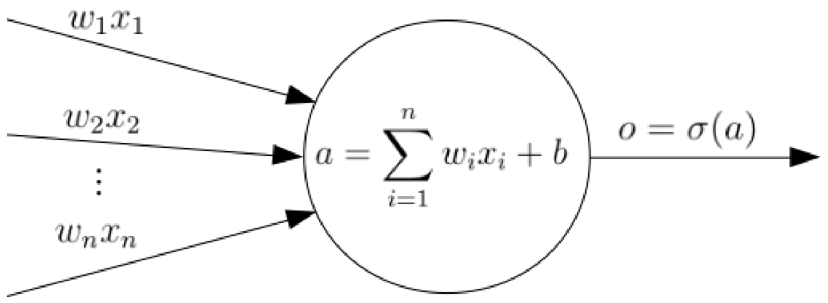


Figure 2.2: Computations performed by one single neuron. The output of each neuron is the activation function which takes the weighted sum of the product of all the weights connecting the neuron to the previous layer by the outputs of the previous layer as an input. The activation value can be seen as the degree to which the neuron responds to its input.

The activation function serves as a non-linearity so that, firstly, the output of a network does not remain a linear function of its input, and secondly, so that the outputs are squashed between a range of pre-defined values, so these are not exclusively large or small. Historically, the sigmoid non-linearity was widely used, which squashes its input to a value between 0 and 1, and therefore provides a good interpretation as the rate at which the neuron responds to its input. Today, there is a wide range of non-linearities to choose from. The ReLU activation function (Jarrett, Kavukcuoglu, Ranzato, & LeCun, 2009; Nair & Hinton, 2010) is one of the most popular and the default for most feed-forward neural networks.

At the end of each feed forward-pass, a loss –or cost– function is used in order to quantify the consistency of the expected output with the output of the network. These functions measure the discrepancy between the outputs of the model and the training data labels. Common choices include the quadratic loss function –or MSE (mean squared error)–, or the negative log-likelihood –or cross-entropy loss.

2.1.3 Regularization

The problem of overfitting happens when a model is able to perform well for the training data but fail to produce good results for new situations, therefore fail to generalize to new data. The whole purpose of supervised machine learning is to create models that are able to achieve good results in situations they have not seen before, and as so, a good model is not necessarily a model which will produce good results for the training data. For that reason, test and validation sets are used in order to measure the performance of a model. And for the same reason, regularization techniques are also commonly used. These allow preventing that the model only learns the training data and improve generalization performance. When training deep neural networks, regularization is any modification made to the model in order to allow it to reduce its generalization error but not the training error (Goodfellow et al., 2016). Without regularization techniques, the task of the model will be to essentially fit the training data the best possible, which in turn will create a very fluctuating line that perfectly fits to all the data points. This will be especially problematic if the dataset is small in size, or if the model has a large number of parameters. Common choices of regularization include, introducing an additional term in the loss function, L2 regularization, artificially augmenting the dataset by generating fake data, early stopping, or Dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012).

2.1.4 Optimization

In the previous sections, we have seen the goal of a network is to find parameters θ that minimize the loss function. The notion of derivative is useful in this context since the derivative of a function at one point is the slope of that function at that point. In other words, it tells us how the input can be changed in order to make a small change in the output. We do so by using the gradient of the loss function $\nabla_\theta L$, which constitutes the vector of partial derivatives that measure the change, or slope, in L as the each of the dimensions of θ change. Since the gradient represents the direction in which the function increases, the negative gradient points to the direction in which the function decreases, and as so this vector is used to search for the direction in which L decreases the fastest, and, the parameters θ can be adjusted by adding a small part of the negative gradient direction. In other words, the loss function can be minimized by iteratively taking small steps in the direction of the negative gradient –this algorithm is called **Gradient Descent (GD)**. As so, for a batch of n training examples we estimate the following gradient:

$$\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{n} \sum_{i=1}^{n} L\left(f_\theta\left(x_i\right), y_i\right) + R\left(f_\theta\right) \right], \tag{2.2}$$

where $R(f(\theta)$ is a regularization parameter. Once we have the gradients, we estimate the update direction:

$$\Delta\theta := -\epsilon \nabla_\theta g(\theta), \tag{2.3}$$

where ϵ is a small positive number known as the **step size or learning rate**. Finally, we update the gradient using:

$$\theta := \theta + \Delta\theta. \tag{2.4}$$

The learning rate determines how large is each step size in the negative direction of the gradient, or in other words, how much the parameters will be adjusted with respect to the loss gradient in each iteration. The choice of learning rate is important since if this value is too low, convergence will be very slow –as we will be taking very small steps–, and if it is too high, the algorithm may overshoot the minimum and diverge (Bengio, 2012). In practice, a good strategy is, for instance, to choose the largest learning rate possible that does not cause the gradient to diverge, observe the training, and if GD diverges start over with a learning rate 3 times smaller (Bengio, 2012). Another example is to start with a learning rate that does not cause the gradient to diverge, and as training progresses, gradually decay this parameter in order to avoid overshooting the minimum (Welling, 2011).

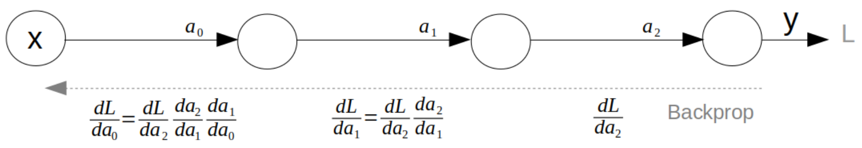
Note that applying GD to the entire training dataset is computationally expensive since datasets tend to be very large in size. Therefore, in practice, **Stochastic Gradient Descent (SGD)** (Bottou, 2010) algorithm is used, which uses mini-batches of data (usually about 100 examples) randomly sampled from the training set at each iteration. Iteratively, for each minibatch, it evaluates the loss and the gradient and computes the parameter update, and it performs these steps over and over until convergence. Note that the estimate of the gradient over the entire training tends to be noisy since the algorithm uses only a few examples at a time, where GD is smoother since it uses the entire training set. Moreover, SGD tends to have slow convergence, and in practice, different methods of optimization are used to accelerate the process. One of these methods is the **Momentum update** (Polyak, 1964), which, tends to be used with SGD. This method introduces a velocity parameter v which smooths out the oscillations steps of SGD by combining previous gradients with opposite signs. Specifically, it works by using the gradient to update a velocity parameter instead of the position, by accumulating the exponentially weighted sum of all the previous gradients to update the weights, given by $v = \alpha v + \nabla_\theta g(\theta)$, where α is a control parameter used to regulate how much previous gradients impact the current gradient. The v value is afterwards used to control the direction of the current step with $\theta = \theta - \epsilon v$. In most cases, the Momentum algorithm is, therefore, able to achieve faster convergence and mitigate some of the problems related to the learning rate, by introducing a new parameter v. However, for other cases, it is somewhat ineffective to a global learning rate for every weight, since different layers may have different gradient magnitudes. Therefore, new optimization algorithms were proposed that are able to achieve good results by using adaptive learning rates - different step sizes for different parameters. These include methods such as AdaGrad (Duchi, Hazan, & Singer, 2011), RMSProp (Hinton, Srivastava, & Swersky, 2012) or Adam (Kingma & Ba, 2014).

2.1.5 Backpropagation

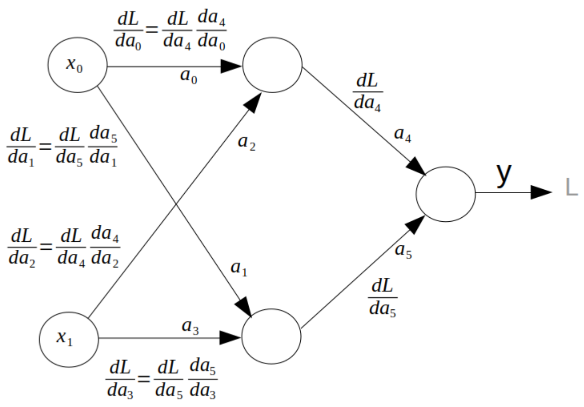
In the previous section we have seen that in order to minimize the loss function, L, its gradient is evaluated at each iteration. This evaluation is performed to obtain the rate of change of each of the parameters θ so that they can be updated in order to achieve a minimum loss. These gradients are computed with the back-propagation (Rumelhart, Hinton, & Williams, 1986) algorithm. This algorithm tells us how to quickly compute the gradients of the loss function with respect to every parameter (weight and bias) –how to find the value of $\nabla_\theta g(\theta)$. These values are needed as all the hidden nodes contribute to the loss value to some degree, and so, in order to get the individual contributions of each node, the loss value is sent backward through the network, from output to input, so that the individual gradients can be computed. Simply put, backpropagation represents how the gradient of the loss flows backward through the network.

Backpropagation relies on chain rule of calculus, which is recursively applied to calculate the gradient of the composite functions. As as example, if we have a function $f = (3a + 1) \times 3$ and want the derivative of f with respect to a, $\frac{\partial f}{\partial a}$, then chain rule can be used. Firstly, note that f can be seen as a sequence of intermediate variables dependent of each other, $x = 3a$, $y = x + 1$ and $f = y \times 3$. Chain rule tells us that the derivative of f with respect to a is given by: $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial a}$. The same can be applied to any other composite function: by multiplying the derivatives of intermediate functions the result will be the derivative of the output with respect to the input.

Similarly, if a neural network is considered a sequence of functions that transform intermediate variables until a loss value is produced, then chain rule can be used to understand the effect of each of the parameters θ (weights and biases) on the loss function and then apply SGD so that the network *learns* better parameters at each iteration, producing smaller and smaller losses until convergence. Two examples of the application of backpropagation in a neural network are described in Figure 2.3. Note that in practice, for efficiency reasons, both the forward pass and the backward pass are performed in the form if matrix or tensor multiplication.


x
a_0
a_1
a_2
y
L
$\frac{dL}{da_0}=\frac{dL}{da_2}\frac{da_2}{da_1}\frac{da_1}{da_0}$
$\frac{dL}{da_1}=\frac{dL}{da_2}\frac{da_2}{da_1}$
$\frac{dL}{da_2}$
Backprop

(a)


$\frac{dL}{da_0}=\frac{dL}{da_4}\frac{da_4}{da_0}$
x_0
a_0
$\frac{dL}{da_4}$
a_4
$\frac{dL}{da_1}=\frac{dL}{da_5}\frac{da_5}{da_1}$
a_2
y
L
$\frac{dL}{da_2}=\frac{dL}{da_4}\frac{da_4}{da_2}$
a_5
a_1
$\frac{dL}{da_5}$
x_1
a_3
$\frac{dL}{da_3}=\frac{dL}{da_5}\frac{da_5}{da_3}$

(b)

Figure 2.3: (a) Backpropagation in a network with only one node per layer. (b) Backpropagation in a network with multiple neurons per layer. During the forward pass, the input is propagated forwards and activations of each neuron are computed: a_0 to a_n. Eventually, the network produces an output, y, which is then compared to the expected output in order to compute the loss value, L. Following that, in order to find the individual contributions of all the activations –the gradient of the loss with respect to each individual activation– the loss value is sent backward through the network, and at each node, the gradient of the loss with respect to each activation is computed. By chain rule, this simply consists of recursively multiplying the local gradient at each node, by the value of the gradient produced in the previous layer. Note that, for efficiency reasons, the local gradient at each node can be computed at the forward pass as this value is known straight away. As so, the only task the network should perform during the backward pass is to multiply these values in order to get the final contributions.

The vector of partial derivatives of a node is simply a Jacobian matrix (LeCun et al., 2015) – a matrix that contains all the partial derivatives describing how one layer depends on the previous. As so, Backpropagation can be interpreted as the Jacobian-gradient product of all the nodes in the network.

2.2 DNN Types

Although the previous sections focused on the most basic type of DNN –vanilla neural networks–, today there are many different types of DNNs. This section covers some of the most prominent architectures.

Convolutional Neural Networks (CNNs or ConvNets) (LeCun et al., 1989; Krizhevsky et al., 2012): This is the most popular type of DNN as it is the current standard for image processing and visual recognition techniques. The main difference from deep feedforward neural networks to CNNs is that the latter operate over volumes, by taking volumes of activations as inputs and producing volumes of activations as outputs, making the intermediates volumes with spacial dimensions of width, height and depth, and not vectors as in DFFs. The three main operations in this type of network are convolutions, pooling, and activation. The core building block of a CNN is a **convolutional layer**, which operates by performing convolutions of the input volume with different filters, usually smaller spatially, by sliding the filter through all spatial locations of the volume. This convolution will produce an activation map of responses of the used filters –the depth of the activation map will depend on the number of used filters. As the network gets deeper, the features in the activation maps will become more specific. It is also common to use **pooling layer** in between convolutional layers in order to avoid overfitting. These layers reduce the dimensionality of the feature maps by with fixed transformations, and effectively reduce noise and save the most important information.

Recurrent Neural Networks (RNNs): The main idea with RNNs (Elman, 1990) is that they are able to process and produce sequential information, operate over variable sized inputs and learn dependencies between training examples, in contrast to CNNs where all training examples are fully independent. This is possible as RNNs have an inner hidden state, which is created by feedback connections and updated at every time step as a function of the current input and the previous state. These can be used to predict the next output based on all previous inputs and states, which has proved useful in different domain tasks such as:

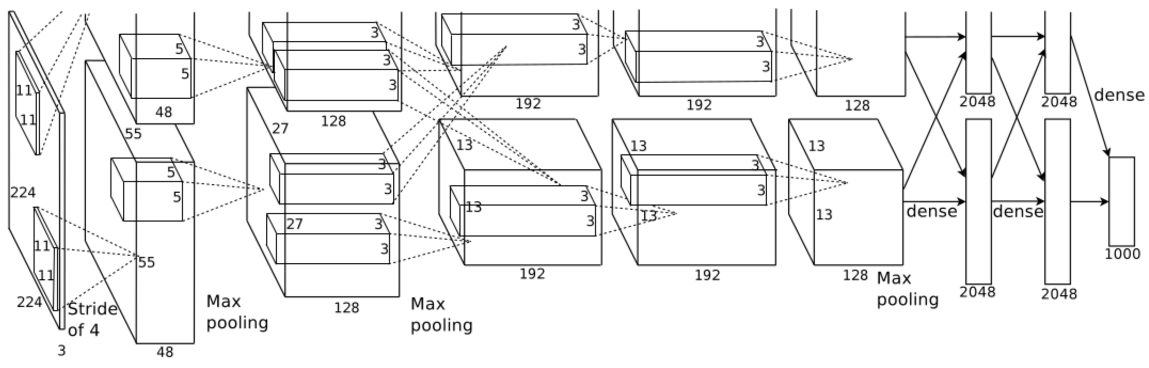


Figure 2.4: CNN model: architecture of the AlexNet model (retrieved from (Krizhevsky et al., 2012)). In an image classification problem, in order to get the final classification, one approach is to attach fully-connected layers and a final soft-max layer at the end of the network, which will produce, for each input, the probability distribution over all classes (Krizhevsky et al., 2012).

- Machine translation, where the input is a sentence of words in one language and the output is a set of words in another language (B. Zhang, Xiong, Su, & Duan, 2017);
- Video classification, where the task is to classify every frame in a video (Yue-Hei Ng, Joe and Hausknecht, Matthew and Vijayanarasimhan, Sudheendra and Vinyals, Oriol and Monga, Rajat and Toderici, 2015);
- Image captioning, where the main task is to take an input image and generate describing sequences of words. The work in (Karpathy & Fei-Fei, 2017) used two connected models: the first was a CNN, which did the processing of the input images and generated features, and the second was an RNN generative model, which received the features produced by the CNN as input (it was conditioned by the output of the CNN) and was responsible for generating descriptions for those features.

Long-Short Term Memory Networks (LSTMs): LSTMs (Hochreiter & Urgen Schmidhuber, 1997) are a specific more complex type of RNNs, which allow a better gradient flow through the network, thus making the training more efficient. This improvement in efficiency is achieved by not completely transforming the hidden state vector from time step to time step, but rather change the cell by

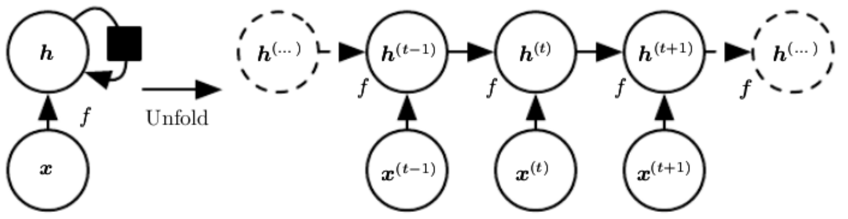


Figure 2.5: RNN model: on the left the representation of the unfolded computational graph, each input node corresponds to a time step. On the right the same graph, the black square represents one time step delay (retrieved from (Goodfellow et al., 2016)).

additive interaction, *i.e.* additively changing the cell state instead of completely transforming it. These models have been successfully applied in different domains, such as speech recognition (Graves, Mohamed, & Hinton, 2013) and handwriting generation (Graves, 2013);

2.3 DNNs in Visual Recognition Problems

In the last few sections, we have seen how a neural network operates and what types of networks exist. The present section covers how these networks are used in practice and what they can achieve in visual recognition problem. It also covers the basic problems in computer vision such as image classification and object detection, and what are the current standards to solve them.

2.3.1 Image Classification

The goal of image classification is to assign a class label to an input image from a fixed number of classes. The outputs are the objects present in that image in descending order of confidence –score. The deep learning approaches to image classification can be traced back to 1989 with LeNet-5 (LeCun et al., 1998), a CNN that forms the base of many of the more recent architectures. However, in the 1980s due to the unexistence of the resources we have today, such as large training sets (e.g., ImageNet (Jia Deng et al., 2009)), good GPU implementations, distributed

Model	top-5 test error
AlexNet (ILSVRC 2012) (Krizhevsky et al., 2012)	15.3%
ZFNet (ILSVRC 2013) (Zeiler & Fergus, 2013)	11.7%
VGG (ILSVRC 2014) (Simonyan & Zisserman, 2014)	7.3%
GoogLeNet (ILSVRC 2014) (Szegedy, Liu, et al., 2015)	6.7%
ResNet (ILSVRC 2015) (He et al., 2016)	3.6%
Inception-v3 (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2015)	3.58%

Table 2.1: Evolution of the classification top-5 test error rate.

clusters able to train large models (Dean et al., 2012), good initialization strategies, good regularization strategies (Hinton, Srivastava, Krizhevsky, et al., 2012) or normalization strategies (Ioffe & Szegedy, 2015) lead to few advancements in this field in the flowing years.

In 2012, when Krizhevsky famously won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a CNN codenamed AlexNet (Krizhevsky et al., 2012) the interest in this field was restored. AlexNet was able to achieve a top-5 classification test error rate of 15.3% and win by a big margin compared to 26.2% achieved by the second-best entry. In ILSVRC, the measure of error for the classification task is captured by the top-5 test error rate, which is a percentage that states that the correct label is among the top 5 classes predicted by the model for every example except that percentage. After 2012, ILSVRC has consistently been won by deep convolutional networks. AlexNet was followed by other architectures, which consistently lowered the top-5 classification test error rate (Table 2.1). These CNNs usually follow the same structure rules: convolutional layers, optionally followed by max-polling or normalization. Some examples are:

ZFNet (Zeiler & Fergus, 2013), which build on top of the architectures specified in (Krizhevsky et al., 2012) and (LeCun et al., 1989) and improved these models, using smaller receptive windows and smaller stride in the first convolutional layer, achieving an 11.7% top-5 classification test error which led to the win of the ILSVRC 2013. This architecture also focused on feature visualization and on mapping the feature activity in intermediate layers of the network back to the input

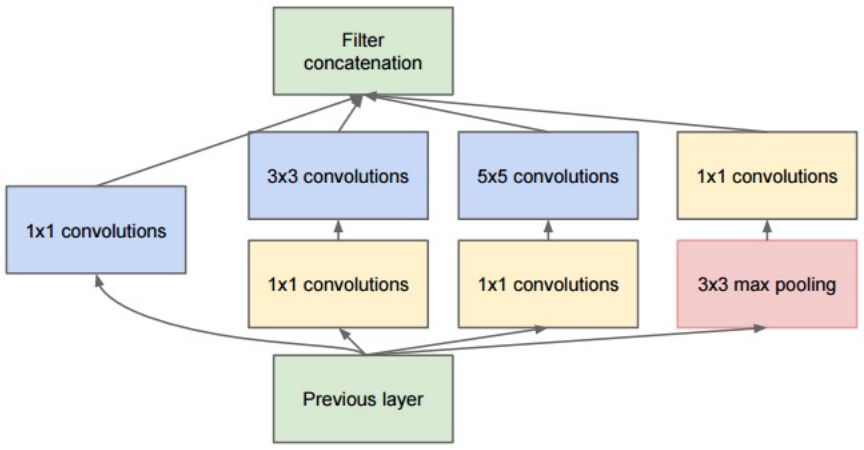


Figure 2.6: Inception module (Szegedy, Liu, et al., 2015). These modules are constituted by sub-modules which perform multiple convolution and max-pooling operations concurrently.

pixel space using Deconvolutional Networks, first introduced in (Zeiler, Krishnan, Taylor, & Fergus, 2010).

VGGNet (Simonyan & Zisserman, 2014) adjusted two parameters: the number of convolutional layers was increased and the size of the convolution filter was decreased to 3×3 in all layers. As a result, this CNN achieved state-of-the-art accuracy in the localization and classification challenges from ILSVRC 2014 with a top-5 test error rate of, respectively 25.3% and 7.3% which secured first and second place in those tasks.

GoogLeNet (Szegedy, Liu, et al., 2015), introduced in 2014, is the first iteration of the Inception architecture (Szegedy, Ioffe, Vanhoucke, & Alemi, 2016). By using $12\times$ fewer parameters than the model described in (Krizhevsky et al., 2012) and introducing the inception modules (Figure 2.6) this model achieved state-of-the-art results in the classification task in the ILSVRC 2014. Instead of convolutional layers, this model uses inception layers, composed by modules which perform multiple convolution and max-pooling operations in parallel and concatenate the resulting feature maps before the next layer. These layers constituted a 22-layer deep model which achieved a classification top-5 test error rate of 6.7%.

ResNet (He et al., 2016) is a 152 layer deep CNN introduced in 2015 by Microsoft which won first place in all five tasks in ILSVRC 2015. The focus of this architecture was to deal with the difficulties of training very deep convolutional networks such as the degradation problem, which states that when the depth of the network increases the accuracy gets saturated (He, 2015). In other words, after a certain depth, adding extra layers results in higher training error and higher validation error even when batch normalization (Ioffe & Szegedy, 2015) is used, which normalizes the output of each hidden layer. To deal with this problem, ResNets employ residual blocks which force the network to learn an identity mapping, by learning the residual of input and output of some layers (Figure 2.7), resulting in better gradient flows in the backward pass, similarly to what happens in LSTM models.

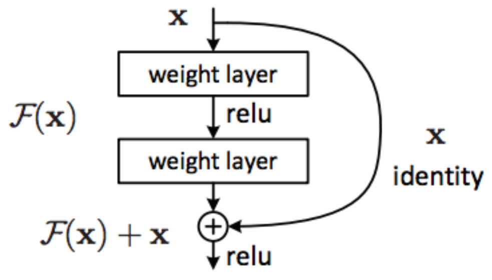

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.7: Residual block (He et al., 2016). These blocks force the network to learn network to learn an identity mapping, resulting in better gradient flow.

The most recent, best-performing models to solve the image classification problem are Inception (Szegedy, Liu, et al., 2015) and ResNet (He et al., 2016) architectures. Since they were first introduced, these architectures have been improved upon, and these improvements can be found in models such as Inception-v2 and Inception-v3 (Szegedy, Vanhoucke, et al., 2015), Inception-v4 and Inception-ResNet (Szegedy et al., 2016) which is a combination of Inception modules (Figure 2.6) present in the Inception architectures and residual blocks present in the ResNet architectures (Figure 2.7). Other recent models include Mobilenets (Howard et al., 2017), which are designed for mobile and embedded vision applications, FraccalNets (Larsson, Maire, & Shakhnarovich, 2016), which state that residual representations may not be necessary and that the key to training a good classifier is transitioning efficiently from shallow to deep architectures, DenseNets (G. Huang, Liu, Weinberger, & van der Maaten, 2017) which connect every layer to every other layer and CapsueNets (Hinton, Sabour, & Frosst, 2017) which try to preserve spatial hierarchies between simple and complex objects by taking into account the relative orientation of objects parts in an image.

2.3.2 Object Detection

In an object detection problem, given an input image and a set of classes, the goal is to find all instances of those classes (variable length output) and produce bounding boxes around all objects of interest. Over the years there have been many different approaches to detection and not all of them using deep learning models. In this section, we describe the most prominent approaches to solve object detection.

2.3.2.1 Classical Approaches

The classical approaches frequently make use of linear classifiers, such as Support Vector Machines (SVMs), on top of feature descriptors such as Scale Invariant Feature Transform (SIFT) (Lowe, 2004) and Histograms of Oriented Gradients (HOG) (Dalal & Triggs, 2005) with exhaustive searches, usually dense sliding window approaches. These types of approaches work well for images with obvious contours such as pedestrians, however, they tend to have worse performance when the objects of interest are more deformable. The part-based object detection method in (Felzenszwalb, Girshick, Mcallester, & Ramanan, 2009) fixed this problem by combining HOG features with Deformable Part Models (DPMs) and achieved state-of-the-art results on benchmarks such as (Everingham et al., 2014).

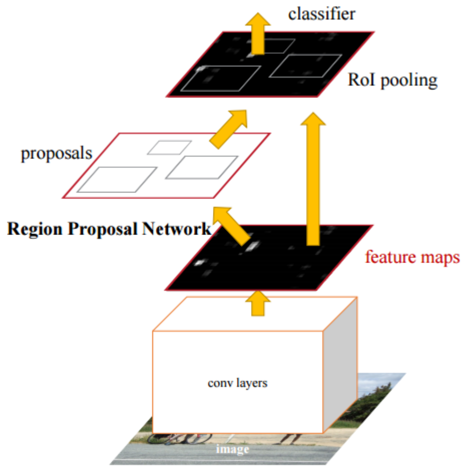


Figure 2.8: Faster R-CNN architecture (Ren et al., 2017). This architecture is composed of two main modules: a CNN which proposes the initial regions, and a Fast-RCNN detector which generates the final detection predictions.

2.3.2.2 Region Proposal Approaches

Many of the modern detection algorithms rely on a sliding window approach to hypothesize bounding box locations and then evaluate a classifier on these boxes. The brute force sliding window framework is, however, computationally expensive due to the need to run the algorithm on every possible location and scale. To solve this, several methods to generate region proposals were proposed (e.g., edge boxes (Zitnick & Dollár, 2014)). One of the most popular is Selective Search (Uijlings, Van De Sande, Gevers, & Smeulders, 2013) which uses segmentation techniques to predict object locations and generate region proposals instead of a blind exhaustive search. These region proposal methods are used by popular region-based convolutional neural networks object detection models (R-CNN) such as R-CNN (Girshick et al., 2014), Fast R-CNN (Girshick, 2015) and Faster R-CNN (Ren et al., 2017) [Figure 2.8].

2.3.2.3 Single CNN Approaches

Although region proposal approaches work well, they tend to be slower in speed when compared to other approaches (J. Huang et al., 2017) due to the region proposal step, which takes up much of the total running time. To overcome this speed limitation, a new set of models based on a single feed-forward convolutional network emerged such as You Only Look Once (YOLO) (Redmon et al., 2016) and Single Shot MultiBox Detector (SSD) (Liu et al., 2016)(Figure 2.9). The key idea is that each of the last few layers of the network is responsible for predictions of progressively smaller bounding boxes, and the final prediction is the union of all these predictions. These models eliminate the region proposal step altogether and are able to achieve a good improvement in speed. In contrast to Faster R-CNN, which runs at 7 frames per second (FPS) with 73.2% mean average precision (mAP), SSD operates at 59 FPS with 74.3% mAP (Liu et al., 2016).

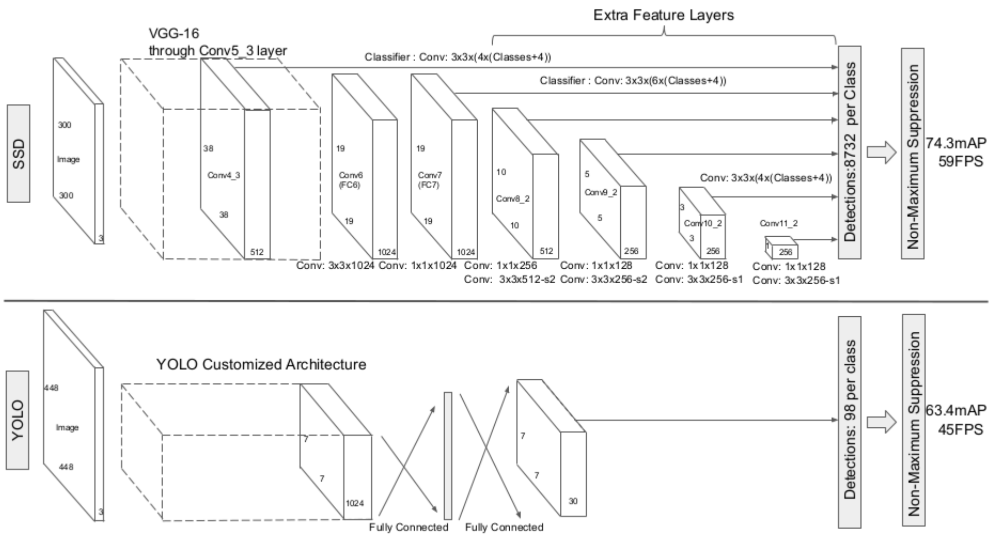


Figure 2.9: Comparisson between SSD (Liu et al., 2016) vs YOLO (Redmon et al., 2016) architectures. In the SSD model, each of the last layers in the network is responsible for smaller predictions with different scales and aspect ratios. The final bounding box prediction is the union of all these predictions. Retrieved from (Liu et al., 2016).

2.3.2.4 Detection Accuracy vs Speed Trade-of

The work in (J. Huang et al., 2017) performed an extensive analysis on the trade-off between speed and accuracy in three main deep learning detection architectures (called *meta-architectures* in this work): Faster R-CNN, R-FCN, and SSD models. Furthermore, for each meta-architecture, the authors alternate between different

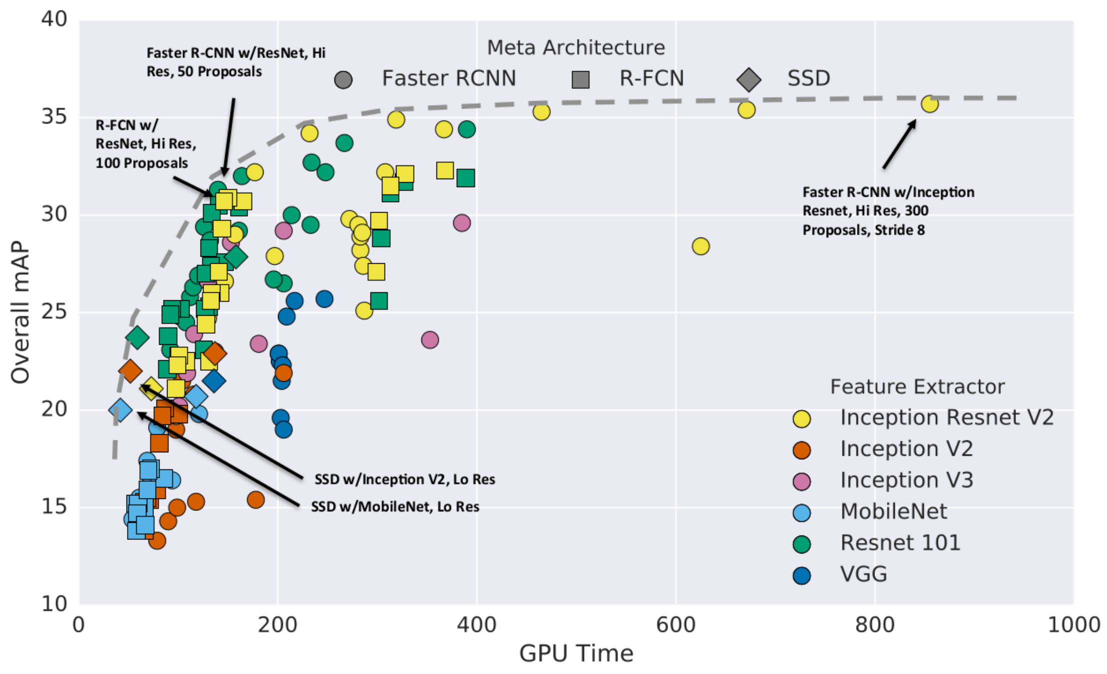


Figure 2.10: Accuracy vs time of different network configurations (Retrived from (J. Huang et al., 2017).

feature extractors and network parameters. This work shows that SSD and R-FCN models tend to be faster while Faster-RCNN is usually slower but more accurate, as represented in Figure 2.10. From this Figure, we can also observe that the most accurate of the If we add the true positives and the false positives we get all the computed detections; e fastest models are SSD with Inception-v2 or MobileNet feature extractors. Even though Faster-RCNN models tend to be slower, the authors note that if the number of proposals is lower (50 instead of 300 as in the original Faster R-CNN paper (Ren et al., 2017)), a significant speedup in inference can be achieved, making this model very close to the fastest models without sacrificing much accuracy. The authors also note that feature extractors have a strong impact on detection accuracy in Faster R-CNN and R-FCN models: ResNet and Inception-ResNet are the best defaults for these meta-architectures, while for SSD meta-architectures these are still the best but the effect is less visible.

Chapter 3

Related Work

Currently, there are two main fields of research in object tracking: Visual Object Tracking (VOT) and Multiple Object Tracking (MOT). In the VOT context, the goal is to track one single object of interest through a set of video frames, and as so, the main purpose is to discriminate the tracked object from the background. This presents challenges such as variations in appearance, illumination or scale. The tasks in Multiple Object Tracking (MOT), as the name suggests, are to detect and maintain multiple object identities through a set of video frames, which presents additional challenges: detection failures, interaction between objects, higher occlusion rates, and similarity in the appearance of different objects can all lead to identity switches and fragmented trajectories. This chapter reviews relevant solutions to tackle both VOT and MOT problems. We consider VOT methods as they form the base for more challenging MOT problems. Furthermore, despite the difference in definition, VOT methods can be scaled to solve MOT problems by applying the tracker multiple times for each of the objects in the frame. This chapter presents the relevant state-of-the-art work in this field that supports the present work.

3.1 Overview

The work in (Fan et al., 2016) splits the task of MOT in two components: observation model and the tracking process itself. The observation model is composed of algorithms that extract a set of attributes, proprieties, and features that describe each of the tracked objects. These features can then be employed to extract information about the state of the target, and to discriminate between different targets. On the other hand, the tracking process is responsible for associating each new observation to the identity of an object. Recently, the paradigm of *tracking-by-detection* became a popular approach in order to solve this problem (Okuma, Taleghani, de Freitas, Little, & Lowe, 2004; Ess, Leibe, Schindler, & Van Gool, 2009; Kuo, Huang, & Nevatia, 2010; Berclaz, Fleuret, Türetken, & Fua, 2011; Andriyenko et al., 2011). Following this paradigm, firstly, for each frame, objects of interest are detected. With the advent of deep learning detection methods, the detection process is generally addressed using a highly efficient deep learning detector. Secondly, those detections have to be linked together with the information from previous frames in order to form the trajectory of each object –this step is generally called the data association step. Here, the challenge is grouping the detected objects in adjacent frames in order to represent the targets by their trajectory over all frames. Different techniques can be employed to successfully execute this step, which are further discussed in the sections below. Additionally, one important distinction to establish right away is online vs offline object tracking: online trackers use information originated only from the current frame and previous frames, in contrast to offline trackers, which require detection information from past and future frames to solve the data association step. As the present work has the constraint of tracking in real-time, we focus on online trackers.

3.2 Observation Model

In the context of object tracking, the goal of an observation model is to extract attributes from objects that accurately describe those objects, and that can therefore be used to differentiate one object from another. These attributes include object appearance, velocity, or location. Two main types of observation models exist: motion models –also known by dynamic models–, which focus on how objects move through space, and appearance models, which strive at creating a feature representation of the objects.

3.2.1 Motion Models

For scenarios in which the type of movement of the target object is known or can be inferred, it is helpful to use a model to make location estimates more accurate and reduce search space. This task can be done by motion models: they help explain the state progression of the target object through time and help predict possible future states. The simplest and most used example of a motion model in the context of object tracking is the **linear motion model** (Shafique, Mun, & Haering, 2008; Breitenstein, Reichlin, Leibe, Koller-Meier, & Van Gool, 2009; Q. Yu & Medioni, 2009) which uses constant velocity and constant acceleration to impose constraints on the movement of targets. In (Andriyenko et al., 2011; Milan, Roth, & Schindler, 2014) a constant velocity model is used in order to predict the velocity of each object, favor straight trajectories, make trajectories smother and therefore improve tracking accuracy prevent identity switches.

However, the linear model is not suitable for cases in which target objects move in a more uncertain manner. In these cases, we can use a **non-linear motion model**. For instance, in (B. Yang & Nevatia, 2012) a non-linear motion map is learned online, which is then used in order to explain gaps in tracklets and link fragmented trajectories, as shown in Figure 3.1. Another example is the work in (Dicle, Camps, & Sznaier, 2013): here, without any appearance information, tracklets of different

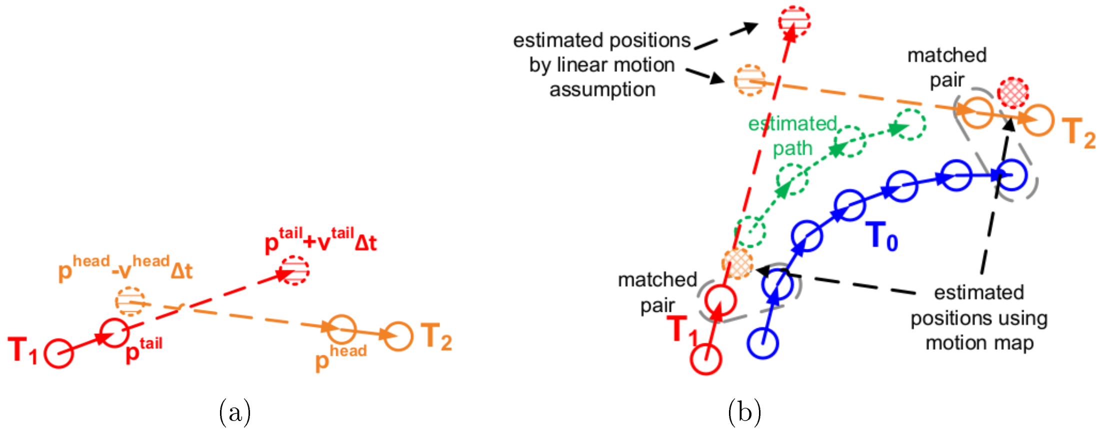


Figure 3.1: (a) Linear motion estimation: the connection between tracklets T1 and T2 cannot be explained by a linear motion estimation. (b) Motion estimation using a non-linear motion map. Using a motion map, this method automatically learns dynamic non-linear motion patterns, which help explain the non-linear gaps between tracklets with high confidence scores. This enables the authors to estimate high affinities between tracklets that would have low affinity following a linear motion estimation. Using this method, the pattern tracklet T0 helps explain a non-linear connection between tracklets T1 and T2. (retrieved from (B. Yang & Nevatia, 2012)).

lengths are associated to form trajectories using motion dynamics information and the Hankel matrix.

3.2.2 Appearance Models

The purpose of an appearance model is to discriminate between the identities of different objects by using object color, texture, shape or appearance information. This information can then be used to compute affinity between new observations and previous objects, providing yet another strategy to associate detections to object identities. Historically, a popular approach to compute such information is to use color histograms (Bradski, 1998; Okuma et al., 2004; L. Zhang, Li, & Nevatia, 2008; Mitzel & Leibe, 2011; S. I. Yu, Yang, & Hauptmann, 2013; Riahi & Bilodeau, 2015) and then employ a distance metric to compare the similarity between these histograms (Okuma et al., 2004). Additionally, gradient-based features such as HOG (Histograms of Oriented Gradients) (Dalal & Triggs, 2005) and SIFT (Scale-Invariant Feature Transform) (Lowe, 2004) features have been widely adopted to build appearance models in object trackers (Kuo et al., 2010; Tang & Tao, 2008). More recently, as deep learning techniques were shown to produce powerful image feature representations, they started to be used in order to generate target-specific representations in object tracking appearance models. Specifically, in VOT problems, neural networks have seen an increased usage in recent years in order to distinguish objects from the background. For instance, the tracker in (Zhai, Roshtkhari, & Mori, 2016) employs a neural network to learn object appearance probability densities online, which adapts itself to the changes in appearance as the object moves. Similarly, the visual object trackers in (Ma, Huang, Yang, & Yang, 2015) and (Qi et al., 2016) use convolutional features from multiple layers and target appearance is encoded using correlation filters on those layers. Other deep learning alternatives employ siamese network architectures to distinguish one object from another. As these architecture are not solely used for appearance modeling but also to track objects end-to-end, we describe them in more detail in section 3.4.1.

Although less common than in VOT problems, CNN based appearance models have also started being adopted in the context of MOT. In (L. Chen, Ai, Shang, Zhuang, & Bai, 2017) in order to build and appearance model, object features are extracted from multiple CNN layers, as different layers encode different features: the deeper the layer, the more specific the extracted features. As so, the first layers are useful to extract information that helps distinguish objects from the background, and lower layers provide more detailed features that help deal with occlusions and interacting objects. At writing time, this method currently holds the first place in the 2D MOT 2015 Challenge (Leal-Taixé et al., 2015)[1] with a MOTA value (Multiple Object Tracking Accuracy) of 38.5%.

Another field of research closely related with MOT is person re-identification. In this context, given an image of an object, we would like to re-identify that object in a large pool of images using solely the appearance of the object. Additionally, re-id problems often deal with fields of view from different cameras, and consequently, the algorithms have to recognize objects from different perspectives and

[1]`https://motchallenge.net/results/2D_MOT_2015/`

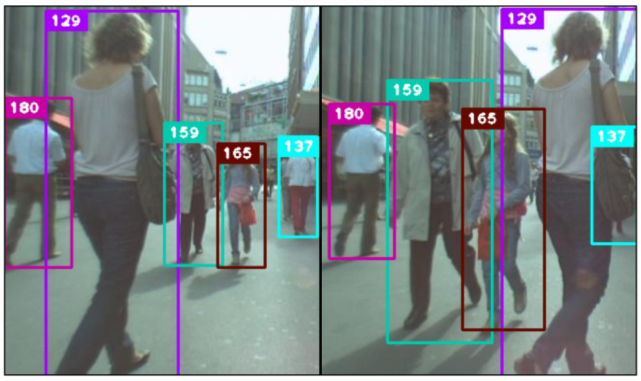


Figure 3.2: Method presented in (Wojke et al., 2018) handling object occlusion in a sample from the MOT dataset (Milan, Leal-Taixe, et al., 2016). Notice that the identities are maintained between different frames.

points of view. Re-id solutions are quite useful in the MOT context, as we have multiple interacting objects prone to occlusion, which, in many cases, have to be re-identified in order for the trackers to recover from occlusions or fix broken trajectories. The multiple object tracker proposed in (Wojke et al., 2018) employs this strategy. The authors build upon the tracker presented in (Bewley, Ge, Ott, Ramos, & Upcroft, 2016), which originally uses Kalman filtering to predict object locations and the Hungarian method to perform data association for new detections, and add a deep association metric to build an appearance model (Figure 3.2). The used appearance model is presented in (Wojke & Bewley, 2018): a CNN is trained on the MARS dataset (Zheng et al., 2016) in order to learn to represent objects by their features and discriminate between different identities. The training process works as a regular classification task, expect that the goal is to have the network distinguish between different people– each person has its own class– and, for this end, a softmax classifier encodes metric learning straight into the classification task.

An additional popular approach to encode object appearance is to employ correlation filters. These filters can be trained to model the appearance of the target object, which can be adapted online, as the object moves. For this reason, these algorithms have been used in object tracking in order to distinguish between an image patch and the surrounding regions in the image (Bolme, Beveridge, Draper, & Lui, 2010).

3.3 Object Tracking

3.3.1 Probabilistic Location Prediction Methods

Generally, probabilistic approaches to object tracking attempt to estimate the current state –or location– of an object based on the history of previous states. One of these methods is the **Kalman Filer**, which has been extensively applied to object tracking (Reid, 1979; Black, Ellis, & Rosin, 2002; Rodriguez, 2011; Wojke et al., 2018).

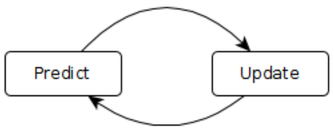


Figure 3.3: Kalman Filter predict-update cycle.

This algorithm is formed by two recursive steps: predict and update (Figure 3.3). In the predict step, the Kalman Filter estimates the new state of a tracked object by using the previous state information. In the update step, the filter reads the new noisy observations in order to update the probability distribution of states, before the next predict step. In this way, the Kalman Filter builds a model of the system that maximizes the probability of previous measurements. The prior assumptions are that the object states, and the noise these are subjected to, follow a normal distribution and that the system has a linear nature, i.e. each state can be represented by a matrix multiplied by the previous state.

However, when the linearity of the system or the prior assumptions do not hold, for instance, when object motion does not follow the assumed linear motion model, Kalman Filters do not provide a powerful representation. In these cases, one approach is to use a **Extended Kalman Filter**, for instance as in (Mitzel & Leibe, 2011). This algorithm is an extension to the Kalman Filter which approximates the system to a linear one by using a Taylor series expansion. Although extended Kalman Filters can be used in non-linear systems, they still fail when the systems are highly non-linear or non-Gaussian. In these cases, a good alternative is to use **Particle Filters**, also known by Sequential Monte-Carlo methods. Particle Filters work by using a set of weighted particles to approximate the probability distribution online. Each particle represents a sample of the distribution, and therefore, the greater the number of considered particles, the greater the accuracy of the distribution representation. For this reason, prior assumptions about the distribution are no longer needed. Similarly to a Kalman Filter, a Particle Filter also consists of successive predict-update steps. In the predict step particles are updated, and in the update step, a weight is attributed to each particle. The weights depend on the consistency of the particle with the new observation. Finally, a set of particles is sampled, using the weights to represent the probability of a particle chosen. Particle Filters are widely adopted in object tracking methods (Z. Khan, Balch, & Dellaert, 2004; Okuma et al., 2004; C. Yang, Duraiswami, & Davis, 2005; Breitenstein et al., 2009).

3.3.2 Data Association

When considering a tracking-by-detection framework, an important decision to make is how to cluster detections over time. This process of clustering the detections is called the data association problem. In other words, this problem regards how to associate detections to object identities in order to initialize, maintain, and finish object tracks.

Basic approach: The most basic method for data association is to calculate the intersection-over-union (IOU) overlap between the new detection and each of the last bounding boxes of already identified objects. This is the approach is used in (Bochinski, Eiselein, & Sikora, 2017): a R-CNN detector (Girshick et al., 2014) is applied in each frame and detections are associated to tracks by calculating the IOU overlap of the bounding boxes in consecutive frames (Figure 3.4). However, this work has two prior assumptions: detection accuracy is perfect, and objects move slowly, i.e., detections of the same object in consecutive frames have a high IOU overlap. Furthermore, MOT videos frequently feature interacting/overlapping objects and occlusions, and thus, the bounding box overlap in consecutive frames may be higher for objects of different ground-truth identities, leading to identity switches. Despite this, this type of approach forms a base for many of the more complex existing methods that better address these problems.

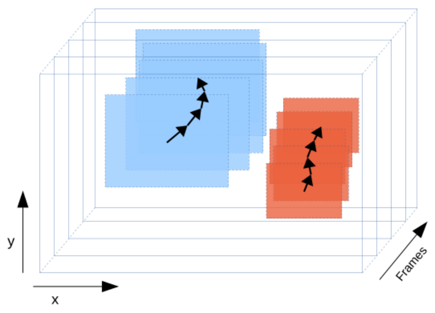


Figure 3.4: Tracking-by-detection approach of (Bochinski et al., 2017): tracklets are created using the IOU between detections in consecutive frames.

Multiple Hypothesis Tracking: Traditional methods to perform data association include Multiple Hypothesis Tracking (MHT) (Reid, 1979) and are a part of different MOT implementations (Han, Xu, Tao, & Gong, 2004; Blackman, 2004; Kim, Chanho; Li, Fuxin; Ciptadi, Arridhana; Rehg, 2015). In this method, for each target, a history of hypothetical decisions is built, maintained and pruned, forming a breadth-first search hypothesis tree, containing all possible combinations of new detection observations and previous targets (Figure 3.5). This method attempts to delay important decisions until sufficient information arrives and no ambiguities exist. When this information arrives, the most probable detection-target association is selected using Bayesian theory. MHT is, therefore, able to manage tracked objects, from beginning to end, including possible (severe) occlusions or false detections.

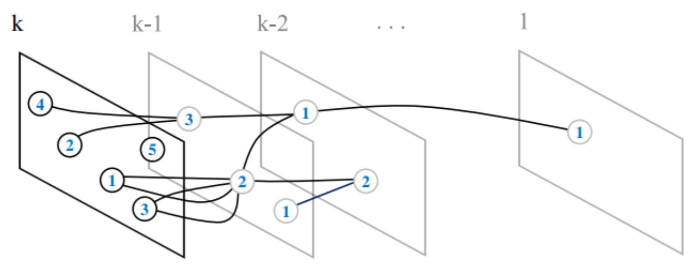


Figure 3.5: Visualization of the Multiple Hypothesis Tracking Tree –the potential connections between observations in adjacent frames– for frame *k* (Kim, Chanho; Li, Fuxin; Ciptadi, Arridhana; Rehg, 2015).

Hungarian Algorithm: We can think of data association as an assignment problem in a bipartite graph, in which the goal is to find the optimal match between detections and tracked objects. One of the most broadly used algorithms in MOT to solve the assignment problem is the Hungarian method (Kuhn, 1955). This algorithm takes an affinity matrix as input and finds the optimal detection assignment. The affinity matrix can be computed in different ways. In (Bewley et al., 2016) a simple approach is used: new object locations are predicted using a Kalman Filter, and then the Hungarian Algorithm is employed to find the optimal match between predicted bounding boxes and newly detected bounding boxes. In this work, the affinity matrix is build using only geometric cues (IOU bounding box overlap). On the other hand, in (Geiger, Lauer, Wojek, Stiller, & Urtasun, 2014) a more complex combination of measures –geometry and appearance cues– is used: the geometry cues take into account the IOU overlap between bounding boxes, and a cross-correlation is used for appearance cues. These cues are then combined into the final affinity entry in the matrix:

$$M(i,j) = \begin{cases} \Gamma(d_i,d_j), & \text{if } \Gamma(d_i,d_j) <= \tau \\ \infty, & \text{if } \Gamma(d_i,d_j) > \tau \end{cases}, \tag{3.1}$$

$$\Gamma(d_i,d_j) = \left(1 - \frac{bbox\_d_i \cap bbox\_d_j}{bbox\_d_i \cup bbox\_d_j}\right) \times (1 - corr(d_i,d_j)),$$

where d_i and d_j represent the detections, $bbox\_d_i$ and $bbox\_d_j$ represent the detection bounding boxes, $corr()$ returns the maximum value of of the normalized cross-correlation between two detections and τ is a pre-specified threshold. Using this newly computed affinity matrix, the Hungarian algorithm computes the optimal associations between detections to form the first tracklets. The second step in this work consists of using the same algorithm to associate tracklets to each other, in order to fix broken trajectories due to occlusion.

The work in (Wojke et al., 2018) also employs a combination of motion and appearance cues. For motion, as a Kalman Filter is used in order to predict the next location of each object, the authors use the squared Mahalanobis distance between the predicted locations and the new detections. For appearance, the neural network of (Wojke & Bewley, 2018) is used to extract features, and then the cosine distance between the produced features for each object is calculated. The final affinity is calculated using a weighted sum of both appearance and motion cues.

Other examples of cue usage to calculate affinity between objects include the Euclidean Distance (Sanchez-Matilla, Poiesi, & Cavallaro, 2016), the work in (Stauffer, 2003) takes accounts tracklet initialization and ending, and in (Riahi & Bilodeau, 2015) a variety of different features are employed, including color histograms to model appearance, optical flow histograms to model motion, and Euclidean distance between detections and candidates to model spatial information.

3.4 Deep Learning Object Tracking Methods

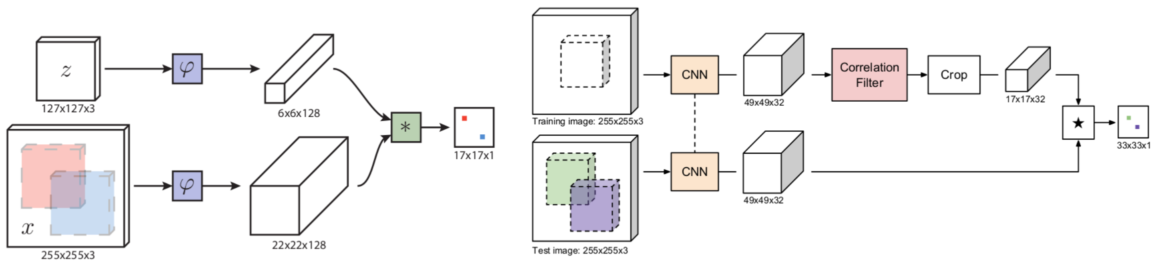
With the advent of deep learning in various visual recognition tasks (Krizhevsky et al., 2012; Girshick et al., 2014), these techniques naturally found their way into object tracking problems. This section reviews the most prominent work, exploring current state-of-the-art deep learning VOT and MOT alternatives.

3.4.1 Simese Neural Networks

The siamese architecture for neural networks recently gained popularity and is used today in different object tracking applications. The name comes from having two sub-networks with similar configuration, which are concatenated at the end to create one single output. Variations of this architecture include triplet and quadruplet neural networks, in which, respectively, there and four sub-networks are used. Each of the sub-networks takes its own input and produces a representation. Those representations are then combined in the third module of the network, in which the final operation occurs, which can be, for instance, the application of a similarity function to each of the outputs of the sub-networks. Due to these proprieties, siamese architectures are quite popular to model appearance.

Siamese Architectures in VOT: A popular approach in visual object tracking is to use this architecture in order to *find* a target object in a search area (Tao, Gavves, & Smeulders, 2016; Leal-Taixe, Canton-Ferrer, & Schindler, 2016; K. Chen & Tao, 2016; Held et al., 2016; Bertinetto et al., 2016). To do this, these networks usually take two inputs: a query image and a search image. The query image is crop of the target object in frame *t - 1* and the search image is a crop of the next frame –frame at time *t*–, centered on the same location as the was target in frame *t - 1* –this corresponds to the area where the network will be trying to find the target object. This area is usually also scaled by a factor to ensure that the query object is present in the target image.

The network in (Bertinetto et al., 2016) employs a fully convolutional siamese network to learn a function $f(x, y)$ which is able to compare the similarity between two the objects in two images, and yields a high similarity value if the two objects in frames belong to the same target (Figure 3.6). Once features are computed, the output feature maps of the two sub-networks are combined with a cross-correlation, which produces a sore-map that can be translated into the displacement of the target object between frames, by multiplying the location of the highest score value by the stride of the network. In (Valmadre, Bertinetto, Henriques, Vedaldi, & Torr, 2017) this architecture is extended by adding a differentiable correlation



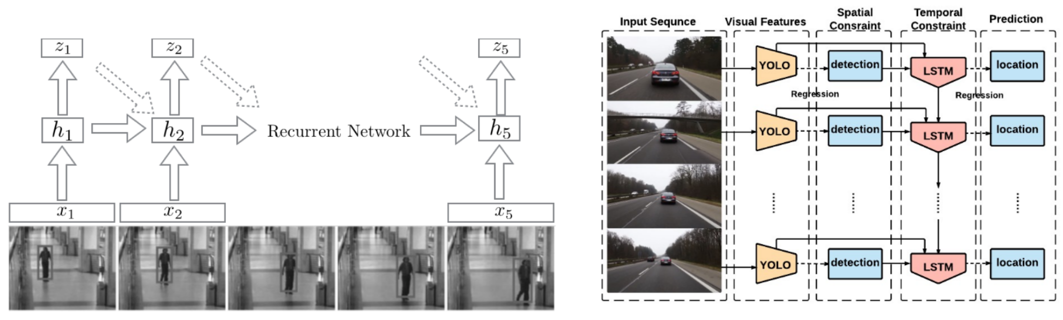
(a) (b)

Figure 3.6: (a) Siamese Neural Network Architecture in (Bertinetto et al., 2016): this network compares a target image z to a search area image x and returns a high score if the two images correspond to the same object. (b) Architecture of (Valmadre et al., 2017) which extends the previous by adding a correlation filter layer into the CNN.

filter layer directly in the CNN. Bu using such a filter, the network is able to learn and adapt to the object model.

The architecture proposed in (Held et al., 2016), code-named GOTURN (Generic Object Tracking Using Regression Networks), employs a similar approach. It performs generic object tracking –which is not trained for a specific class of objects– at very high speeds (100 FPS). In order to be able to recognize novel objects, the network is trained to learn a generic relationship between appearance and motion. The network is trained to directly regress the bounding box coordinates of the tracked object in the search area of the next frame. We further discuss this architecture in later chapters.

Siamese Architectures in MOT: A few variations of siamese convolutional architectures were proposed to track multiple objects. Mostly, they have been utilized in order to build appearance models to measure affinity between detections and help solve the data association problem. The work in (Leal-Taixe et al., 2016) uses this architecture to extract local spatio-temporal features, where the inputs are detection images and optical flow maps. Similarly, in (Wang et al., 2016) a siamese architecture is used jointly with temporally constrained metrics in order to build a tracklet affinity model. The authors in (Son, Baek, Cho, & Han, 2017) use a quadruplet network in order to model target appearance and motion. Here, via a muti-task loss, the network simultaneously learns to associate objects by their



(a) (b)

Figure 3.7: The internal state of the network is updated at each time step based on the previous information and the new feature vector in order to predict the new state (a) retrieved from (Gan et al., 2015). (b) retrieved from (Ning et al., 2017).

similarity and temporal distance, and additionally, it directly regresses bounding-boxes.

3.4.2 Recurrent Neural Networks Trackers

Recurrent Neural Networks (RNNs) have proprieties that allow them to process sequential information, updating a hidden state in order to predict a new state at each time step. These proprieties are useful in the context of object tracking, and therefore, they constitute another growing trend in this field (Gan et al., 2015; Ning et al., 2017; Kahou, Michalski, Memisevic, Pal, & Vincent, 2017; Gordon, Farhadi, & Fox, 2018). This approach is usually composed of two steps: firstly, at each time-step, a feature extractor is used to generate feature vectors. Secondly, the RNN, updates its internal state based on the extracted features and the previous states, and with this information, the network predicts the new object location.

RNNs in VOT: In (Gan et al., 2015), a RNN takes a video frame as input and returns the bounding box of the tracked object (Figure 3.7 (a)). Here, the whole frame is given as input to a CNN for features to be extracted. Secondly, three inputs are given to the RNN: the feature vector, the previous state, and the previous object location. With the newly updated state, the distribution over the location of the object is computed, and the mean of this distribution will correspond to the new predicted location of the object. Closely related to this work, is the network in (Ning et al., 2017), codenamed ROLO –recurrent YOLO– (Figure 3.7 (b)). Here, an LSTM is trained end-to-end and employed to directly regress bounding boxes. The authors firstly employ a YOLO detector (Redmon et al., 2016) network to collect visual features and detect the targets, which are fed into the LSTM to regress the object location. Another similar case is the work in (Gordon et al., 2018), which, contrary to previous work, is able to run on more challenging natural videos and achieve greater accuracy and speed by using both offline and online adaptation. Similarly to prior work, this work has three components: CNN layers to model target appearance, recurrent network layers to model motion and *remember* appearance and a regression layer to produce bounding boxes.

RNNs in MOT: Using RNNs in the MOT context is a more complex task than using RNNs for VOT problems as multiple states for multiple objects have to be considered simultaneously.

The multiple object tracker in (Milan, Rezatofighi, et al., 2016) introduces an RNN able to perform location prediction, data association, track initialization, and track ending. The authors divide the problem into two major stages: firstly, state prediction/update in conjunction with track management, and secondly data association. To solve the first stage, an RNN performs three tasks: predict states of all target objects based on the current state and the inner state of the network, update the states when information from the next frame is available, and establish which tracks were initialized or ended based on the states. To achieve this tasks the authors employ a four-component loss: firstly, in order to measure the loss of the state prediction and state update steps, the first two components are the mean squared error (MSE) between predicted values and ground-truth labels. Secondly, as the network should also learn to recognize which tracks are active, a binary cross entropy (BCE) loss is used to calculate the probability that each target exists. The second stage of this problem corresponds to data association: in this work, this problem is solved by having an LSTM predict which detection corresponds to each target. Here, at each time step, the feature vectors are formed

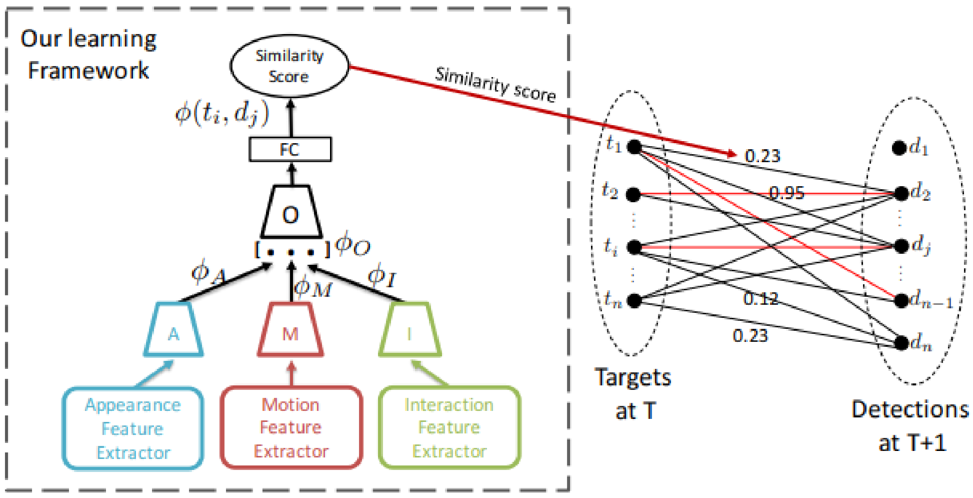


Figure 3.8: MOT tracking using RNNs (Sadeghian et al., 2017). In this work, a combination of four RNNs is employed in order calculate a similarity score between existing targets, *ti*, and detections, *dj*. The first three RNNs are employed in order to model target appearance, motion and interaction between targets, respectively. The feature outputs of each are then combined in the last RNN, which outputs the final feature vector, which is used to measure the similarity between targets, *ti*, and detections, *dj*.

by the input, the hidden state, and the cell state, and the network is trained on a negative log-likelihood loss. This approach does not reach high accuracy on MOT2D Benchmark challenge (Leal-Taixé et al., 2015) (19.0% MOTA), however, it is one of the fastest submissions with 165.2 FPS.

The work in (Sadeghian et al., 2017) is able to achieve better accuracy value on the MOT benchmark. This works employs a structure of RNNs to solve three tasks: appearance modeling, motion modeling, and target interaction modeling. Each of these tasks is solved by an RNN and the resulting features of each one are then combined and given as input to a fourth RNN, that will produce a final feature vector used to compute similarity scores between detections and targets. This architecture is able to achieve a 37.6% MOTA score, and currently holds the second place in the MOT2D Benchmark challenge (Leal-Taixé et al., 2015), being surpassed only by the work of (L. Chen et al., 2017) with a 38.5% MOTA value.

3.5 Summary

Despite its success in other visual recognition tasks, such as image classification (Krizhevsky et al., 2012) or object detection (Girshick et al., 2014), deep learning methods have only very recently begun to be utilized in object tracking problems and its application in this problem remains sparse.

This is particularly noticeable in the MOT context, in which surprisingly little work focuses on the application of deep learning techniques for the tracking problem itself (Milan, Rezatofighi, et al., 2016; Wang et al., 2016; Leal-Taixe et al., 2016; Son et al., 2017). In the MOT field, most of the approaches rely on heavy uses of more classical techniques such as probablistic methods (as Kalman Filtering (Rodriguez, 2011) or Particle Filtering (Breitenstein et al., 2009)), data association techniques such as Multiple Hypothesis tracking (Kim, Chanho; Li, Fuxin; Ciptadi, Arridhana; Rehg, 2015), or the Hungarian Algorithm (Geiger et al., 2014).

The lack of attempts to solve this problem using deep learning sets up the developments in the remaining chapters of the present work, in which we build on some of the existing deep learning approaches, and explore the cross-over between these newer techniques and more classical techniques, applying them to the object tracking problem.

Chapter 4

Single Camera Tracking

As our final goal of real-time multiple camera, multiple object tracking is a complex task, our approach consists of splitting the problem into two main stages: single camera person tracking (i), and multiple camera person tracking (ii). In this chapter, we present our solution and the proposed architecture for the single camera person tracking system. To solve this problem we divide this task into specific modules, each one dealing with different object tracking tasks, building up towards a scalable, modular architecture, that can afterward be scaled for multiple camera multiple object tracking.

4.1 Overview

Given a feed of video, the goal of multiple person tracking is to track and maintain an identity for each person in the video. As we are using one single camera, at this stage we only worry about solving multiple person tracking **for one single camera**. We tackle this problem in a modular manner: the tasks are divided by modules, which are chained together. Each of these modules is concerned with solving a specific problem and outputting its result to the next module.

Real-Time system: It is important to define what exactly constitutes a real-time system: we consider as real-time, any system able to process information as fast, or faster than the rate of the input that is given to it. For instance, if we feed our system with a 30 FPS video to the system, results should be computed in 30 seconds or less.

4.2 Single Camera Tracking Architecture

The proposed system is composed of four main modules represented in Figure 4.1. At a high level, this consists of reading frames from an IP camera in real time using the FFmpeg tool [1], sending those frames to the detection module, which, for each frame, will output the bounding boxes around the objects of interest and send those to the tracking module. The tracking module will link together the detections that belong to the same object in consecutive frames, and finally send the results to the visualization module, which will display the tracks to the user.

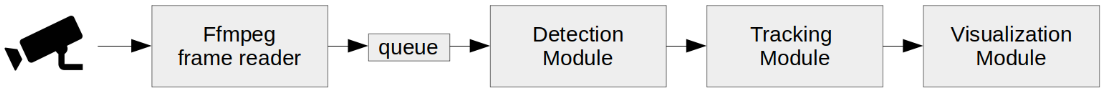


Figure 4.1: Single camera tracking architecture. The system is composed of four main modules, each responsible for a specific task. The frame reader module will read frames directly from an RTSP stream and send them to the detection module, which will produce bounding boxes around the objects of interest. The tracking module will take these detections as input, and its main task is to identify each one and maintain these IDs in subsequent frames. Optionally, the visualization module can be activated for the results to be displayed.

4.2.1 Frame Reading Module

This module is responsible for reading and sending frames to the detection module. Additionally, to the frames, it also sends the respective timestamp for each frame. In order for the program to be able to run in real-time, this module utilizes threading and the queue data structure to buffer frames, which also ensures the

[1] `http://ffmpeg.org/`

rest of the needed processing runs faster as we avoid using blocking I/O operations to read frames. Consequently, there is a significant speedup simply by creating a separate thread that is solely responsible for reading frames from the stream file and maintaining a queue.

The first iterations of this module made use of the OpenCV library (Itseez, 2014) in order to read frames. However, the OpenCV class responsible for reading RTSP streams does not currently support the functionality of retrieving frame timestamps, which is needed for many real-time applications. Using OpenCV for single camera tracking works fine as it ensures that frames arrive in the correct order, and thus, a timestamp is not needed. However, we need to have in mind that this system should be afterward scaled for multiple cameras, and in this scenario, the need for timestamps becomes more apparent as the different video streams have to be perfectly synchronized in time. Time synchronization is crucial to track through multiple cameras with overlapped fields of view since if they are out-of-synch, a person walking over the overlapped area in one camera, may not be present in the second camera while this information is being processed, and therefore, the system will fail to link both tracks and associate them to the same identity. Therefore we worry about timestamps at this stage, even though it is not necessarily needed for single camera tracking.

To achieve synchronization over multiple streams, presentation timestamps (PTS) are used, which represent the exact time that each frame has to be displayed. These are usually used to synchronize audio and video in one streams, however they can also be used to achieve video synchronization of different streams[2].

4.2.2 Person Detection Module

Having a well-performing detector is a key factor for the object trackers to also perform well. The main task of a detector is, given an input image and a set of classes, generate bounding boxes surrounding all objects belonging to those

[2]In order to read the PTS of each frame we use the FFmpeg library: we read frames through a pipe, and the PTS is retrieved by parsing the *stderr* output.

classes –detections–, and output the confidence scores for each of these detections. In order to achieve this, we train different CNNs to detect people –we consider one class "Person"– using the detection dataset described in the previous chapter. In order to do so, we use the Tensorflow Object Detection API (J. Huang et al., 2017), which provides a set of tools and pre-trained detection models, that can be easily trained on custom datasets.

Choice of detection model: With the results of the work of (J. Huang et al., 2017) in mind, which performed an analysis on the trade-off between speed and accuracy in deep learning detection architectures, we have chosen two main architectures to integrate in the detection module of the present work: a SSD detector with an Inception-V2 feature extractor and a Faster R-CNN detector with a ResNet101 feature extractor, available to download at the tensorflow Model Zoo page [3]. We also employ different variations of different parameters –namely non-maximum suppression (NMS) threshold and number of proposals– in the Faster R-CNN model in order to understand if, using our datasets, a Faster R-CNN detector can achieve SSD speeds without compromising accuracy. We further elaborate on this topic in Chapter 6.

Training pipeline: Using the Tensorflow Object Detection API, training the models is a fairly straightforward process. The detection dataset is split into train and test sets (80/20) and the resulting files are transformed into the TfRecord format, which is a standard format in Tensorflow. In order to reduce training time and avoid overfitting, we employ Transfer Learning (Caruana et al., 1997; Bengio, 2011), which consists of using a pre-trained model and re-training it with new data. Tensorflow provides pre-trained object detection models on different datasets. We employ the models pre-trained on the Microsoft COCO (Lin et al., 2014) dataset, as it is the closest to the custom data in the present project. Network hyperparameters are set in the configuration files provided by tensorflow. In order to monitor the network training process, the Tensorboard tool is used,

[3]`https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`

which allows to easily access to metrics such as loss and accuracy during training. This makes Tensorboard very useful for debugging the training process.

Tasks: This module is responsible for reading frames from the queue, previously put there by the frame reader, and generate bounding boxes around each person in the frame. As so, the main task in the main class of this module is to continuously read frames from the specified source, pass them through the specified detector, and forward the results for each frame –bounding boxes, detection scores, presentation timestamps and the frame itself– to the tracking module.

4.2.3 Person Tracking Module

The tracking module is responsible for receiving detections from the detection module, and for each frame, linking detections together in order to form the individual tracks of each person through a set of frames. Our goal is to design a tracking module that:

- is able to perform in real-time;
- is able to maintain the track identities even with multiple interacting people;
- is able to recover from occlusions and fix fragmented tracklets;
- is able to successfully start, maintain and end a track.

In order to get a better perception of what different types of tracking architectures are able to achieve, we selected 3 different trackers to implement, test and integrate in the tracking module: one basic MOT IOU tracker to use as a benchmark, one deep learning visual object tracker in order to investigate how it can be extended for the MOT problem, and one multiple object tracker, which we slightly extend to better fit our problem. For the most part, the trackers employ a **prediction-association framework** –the only exception to this is the IOU tracker, which only uses the association step. In this framework, for each new frame, in first place trackers predict the next locations for each of the existing tracked objects, and in second place, detections are compared with tracker predictions in order to be associated with the most similar prediction. At the most basic level, we can represent each tracked object by one object identifier, a sequence of detections in adjacent frames and a sequence of locations predicted by the trackers for each frame. The architectures of the used trackers are described below.

4.2.3.1 IOU Tracker

Firstly, we employ a simple IOU tracker similar to the one used in (Bochinski et al., 2017) mainly to form a baseline and be able to better evaluate the remaining more complex trackers. Out of all the implemented trackers, this is the only one that does not predict the next location of the tracked objects, only using detection information to establish object tracks. Here, at each input from the detection module –formed by the frame and respective detection bounding boxes– we measure the Intersection over Union (IOU) overlap between the newly arrived detection bounding boxes from the detection module and the last bounding box associated to each of the already established tracked objects. Recall that the IOU is defined by:

$$IOU(d_i, d_j) = \frac{Area(d_i) \cap Area(d_j)}{Area(d_i) \cup Area(d_j)}, \tag{4.1}$$

where d_i and d_j correspond to detection bounding boxes in adjacent frames. We empirically define a threshold value, and if this score is greater than this value we consider that the detection belongs to that object. Otherwise, if a new detection has no suitable candidate –an already existing tracked object– a new tracked object is created and initialized with that detection. At the end of each iteration, the list of the tracked object is cleaned of *dead* tracks, i.e, objects that have not had detections associated to in at least the passed K frames. As this approach is simple, we utilize this tracker in order to form a baseline for the remaining trackers. In other words, we investigate if using deep learning models and MOT strategies is worth the higher inference time and computational complexity, or if the current state-of-the-art object detectors have reached a point were it is feasible to track multiple objects using only detection information.

4.2.3.2 GOTURN Tracker

For the second tracker, we employ the deep learning model proposed in (Held et al., 2016) code-named GOTURN (Generic Object Tracking Using Regression Networks). Recall that this is a siamese CNN architecture (Figure 4.2) trained offline in order to learn a generic relationship between appearance and motion, and recognize new objects online. In order to learn this relationship, similarly to other siamese architectures (Bertinetto et al., 2016), the network is trained on pairs of frame crops: one crop centered on the object we would like to track in frame at time t –the target object–, and one crop centered on the same location, in frame at time $t + 1$ scaled by a factor of k –the search area. In other words, the network will try to localize the target object in the search area, and directly regress the bounding box of the new object location. Note that the scaling of the frame patch at time $t+1$ is done in order to ensure that the object will be present in the search area. Due to this cropping scheme, which avoids that too many patches are compared (as for instance in (Tao et al., 2016), which runs at 4 FPS), and as no learning is performed online, this model achieves very high speeds: 100 FPS on a Nvidia GTX 680 GPU and 165 FPS on a Nvidia GeForce GTX Titan X GPU (Held et al., 2016). To our knowledge, this is one of the fastest deep learning based visual object trackers. For these reasons, we chose this tracker to scale for the MOT problem.

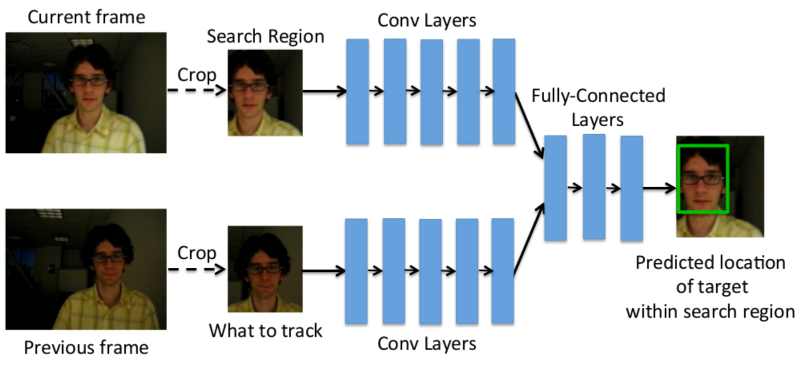


Figure 4.2: GOTURN network architecture (Retrieved from (Held et al., 2016)). This siamese neural network takes two inputs: a crop of the target object of the previous frame, and a search area of the current frame. It directly regresses the bounding box coordinates of the target object in the search area.

Scaling for MOT- Data Association: In a VOT problem, GOTURN would be given the initial location of the object to track –a bounding box–, and in subsequent frames, it would run exactly one time per each frame –one time to predict the new location of the target object in the new frame. Each predicted location would serve as one of the inputs to the network in the next iteration, along with the search area in the new frame. When scaling this to an MOT context, as we have multiple objects we would like to track per frame, the tracker has to run one time for each object in the frame in order to predict the new location for all objects. At each iteration, the tracker uses the previously predicted frame crop at time *t - 1*, and the search area in the current frame at time *t* as an input in order to predict the next position of each of the tracked objects. Furthermore, as in this context objects can interact and possibly become occluded, we do not rely solely on the predictions of the tracker: instead, we complement those predictions with the detections produced by the detection module. Therefore, we employ GOTURN as a prediction mechanism: for each new frame and for each object in the frame, we have GOTURN predict the new location, and then we validate those predictions using the detection bounding boxes. In order to perform this validation, we compute the affinity value between each of the new unmatched detection and each of the tracker predictions for each of the existing tracked objects. we investigate and implement three different hypothesis in order to compute the affinity value:

- Firstly, we employ a simple IOU overlap score between the detection bounding box and the network prediction bounding box;
- We employ a deep learning appearance model, and compute the distance between the detection features and the prediction features;
- We employ a weighted sum of both IOU overlap and appearance score.

Once affinity values between targets and detections are computed, we proceed to order all the target-detection pairs by their ascending affinity value, and associate detections to the existing objects by this order. Additionally, we consider a lower-bound empirically defined affinity threshold, as target-detection pairs that are too far away or dissimilar will never correspond to the same object. Therefore, all the target-detection pairs in which the affinity value is lower than this threshold will not be associated.

Track Initialization: This refers to the problem of knowing how and when should a track be initialized, and is a fairly straightforward process: an object track is initialized when the tracker receives the first detection belonging to that object from the detection module. In the section above we have seen that affinity between detections and tracker predictions are computed in order to associate detections to tracked objects. Following this logic, we know that the detection belongs to a new object if it does not match with any of the predictions for the previous objects. In other words, if the tracker location predictions for the existing objects and the new detection an affinity lower than the empirically defined threshold, then a new tracked object is initialized with that detection. This detection constitutes the first input fed to the tracker (target) and defines the features of the object that the tracker will be looking for during its runtime.

Track Ending: When an object leaves the camera field of view its track should be removed. One way to know when an object is no longer present in the field of view is to confirm if a detection exists for that object in the current time step. However, considering the track finished when no detection exists is a naive strategy, as there may be detection errors. Specifically, consider that a detection error occurs for one particular tracked object and this object is not detected in the current frame. If we remove that object straight away and consider that track as finished, and the object is afterward re-detected in the next frame, a new track ID will be created. This type of identity switch is undesirable for a multiple object tracker. As so, we attenuate this problem via a parameter: the maximum age of the tracked objects. The track of an object is only considered finished when no detection is matched with that object in MAX_AGE frames. Furthermore, when an object fails to be detected, we initialize a counter AGE for that object and increment it by one at each frame that the object is not detected. In other words, even if there is no detection for a particular tracked object, the tracker will keep predicting its location and maintaining the track alive for MAX_AGE

frames, while incrementing the AGE counter. If during this period, the object is re-detected and successfully matched to the tracked object, the age counter is reset to 0. Otherwise, if after this period the object is still not associated with any detection the track is considered finished and removed from the active tracks list.

4.2.3.3 DeepSORT Tracker

For the MOT tracker, we employ the work of (Wojke et al., 2018), which extends the SORT tracker (Simple Online and Realtime Tracking) in (Bewley et al., 2016) in order to integrate appearance information. This work can be decomposed in two main stages: firstly, Kalman Filtering is performed in order to predict the next location of each target object, and secondly, detections are associated to tracked objects using the Hungarian Algorithm (Kuhn, 1955). Recall that the Hungarian Algorithm computes an affinity matrix between tracked objects and new detections. In this work, the affinity matrix is computed using the weighted sum of two metrics: the Mahalanobis distance between the locations predicted by the Kalman Filter and the new detections, and cosine distance between appearance features of tracked objects and new detections. Therefore, each entry in the affinity matrix is given by:

$$M(i,j) = \lambda d_1(i,j) + (1-\lambda) d_2(i,j), \tag{4.2}$$

where $d_1(i,j)$ corresponds to the squared Mahalanobis distance between locations predicted by the Kalman filter and new detections, and $d_2(i,j)$ corresponds to the cosine distance between the appearance features of the i-th tracked object and the appearance features of the j-th detection. $d_1(i,j)$ is given by:

$$d_1(i,j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i), \tag{4.3}$$

were d_j is the j-th detection bounding box, d_i is the i-th tracked object bounding box, y_i is the arithmetic mean vector and S is the sample covariance matrix. As

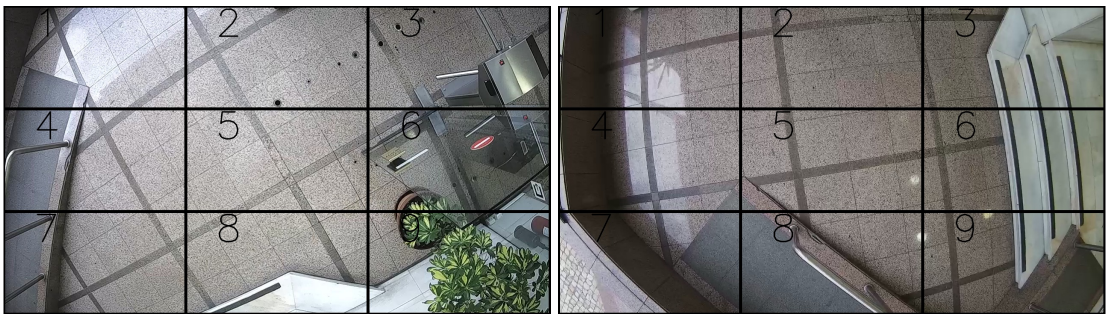


Figure 4.3: Grids drawn over camera fields of view. These grids enable us to define static rules about which trajectories are possible and which trajectories are not, which is useful for the system to recover from diverse tracking errors.

the Mahalanobis distance is usually employed to find unusual combinations of two or more variables, it is useful in this context to find abnormal track-detection associations. The second metric, $d_2(i,j)$ is given by:

$$d_2(i,j) = min(1 - r_j^T r_k^{(i)}, r_k^{(i)}), \tag{4.4}$$

were r_j is the feature vector for detection j, $r_k^{(i)}$ is the vector of the last k feature vectors of tracked object i. We chose this MOT tracker as we wanted a tracker that employs classical MOT algorithms –as Kalman Filters and Hungarian Algorithm– and combines it with more recent deep learning techniques aimed at feature extraction –the neural network in (Wojke & Bewley, 2018). Additionally, we needed a fast method: this method achieves high speeds –40 FPS– still maintaining high MOTA values –61.4% on the MOT16 benchmark challenge (Milan, Leal-Taixe, et al., 2016)–, as opposed to, for instance, the tracker in (F. Yu et al., 2016) which runs at 10 FPS and which achieves 66.1% MOTA on MOT16 benchmark.

4.2.3.4 The Grid Algorithm: Handling Occlusions and Error correction

Any of the trackers described above will yield reasonable results. However, in our specific scenario, cameras are static, and this enables us to make additional assumptions about the space in the field of view in order to further refine tracking results. We propose to split the field of view into a grid-like structure as represented in Figure 4.3. As the space is static, the grid enables us to create static rules to describe situations that are plausible and situations are impossible. Specifically, we can define which cells correspond to entrances/exits and which cell transitions are possible. We consider the cells that correspond to entrances/exits to be tracker start cells. In other words, we consider that a tracked target can only be initialized in one of these cells, as these are the only locations where new objects can enter the field of view. All the cells have coordinates which can be used to check in which cell a tracked object is located at the current time. Additionally, at construction time, all the cells are informed of what cells are their neighbors, and therefore we can define which cell transitions are possible. We further build upon this concept and manually define the cell transition sequences that correspond to accepted end states: in other words, each accepted end state corresponds to a sequence of cell transitions that an object can make from an entrance cell to an exit cell. As so, firstly, we begin by defining which grid cells correspond to start states –start cells. Secondly, we define what transitions from cell to cell are possible, as shown in Figure 4.4. Lastly, we manually define which sets of transitions correspond to accepted end states. Having these rules enables us to perform three validations:

Track initialization: As we use the grid to define start cells, we can add an additional constraint to the track initialization process, or in other words, to the process of assuming that a detection belongs to a new object: in addition to not having an affinity higher than the empirically defined threshold with any other tracked object, the detection only belongs to a new object if this detection is located in one of the pre-stipulated start cells.

Track ending: The grid also enables us to validate track ending. Recall from the sections above, that an object track is considered finished when no new detections are associated to this object in MAX_AGE frames. Using the grid we can extend this constraint by also defining what cell sequences are accepted track ending sequences. In Figure 4.4 one example of a finished track sequence is [8, 5, 2, 3]. Using this additional constraint, an object track will only be considered finished if it has not had a detection in MAX_AGE frames and if the grid cells in the trajectory of that object constitute an accepted end state sequence.

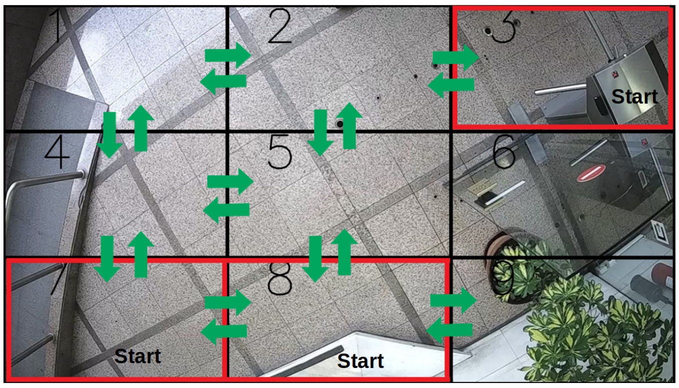


Figure 4.4: Possible state transitions having a grid-like space separation. We define start cells (7, 8, 3) which constitute the cells where track initialization is possible. Additionally, we define all the possible state transitions that describe how objects can move. Furthermore, we manually define sequences of transitions that constitute accepted end states, that describe the path that an object takes from the entrance to the exit, e.g., the sequence of transitions (8, 5, 2, 3) corresponds to an accepted end state, as all the cell transitions in this sequence are possible. This enables us to recover from occlusions, reduce identity switches and reason about lost tracks.

Recovery from errors: Due to detection and tracking errors, for instance, occlusions, tracked objects may suffer from identity switches. For instance, without using the grid algorithm, if an object fails to be detected in MAX_AGE frames, then a new tracked object will be created, leading to an identity switch. However we can employ the grid to reason about occlusions and lost tracks. We know that if a new object suddenly appears in a grid cell that is neither an entrance or an exit cell, it probably is not a new object, but a switched identity. Therefore, firstly, we temporarily store all the tracked objects that failed to be detected in MAX_AGE frames and did not have an accepted end state cell sequences in a *unfinished_tracks* list. Secondly, every time a new tracked object suddenly appears in a grid cell that is not an entrance or exit, we check the *unfinished_tracks* list for possible matches to previous undetected objects: if the grid cells match –if it is the same cell or a neighbor cell– the current detection will be associated to that object.

Chapter 5

Multiple Camera Tracking

In the previous chapter we have developed a system able to track objects in a video originated from one single IP camera. In this chapter, we discuss how to scale that system to multiple cameras with overlapping fields of view. Once more, we tackle this problem in a modular manner, in which, the components and algorithms of each module can be easily switched so that we are able to test multiple solutions in a scalable way.

5.1 Overview

Given a real-time feed of video, originated from multiple overlapping cameras, we would like to track each person that goes through each field of view and maintain these identities through all fields of view. The goal is to link tracks belonging to the same people between cameras in order to form the full path of each individual person in all cameras, as represented in Figure 5.2. As these cameras have overlapped fields of view, the same target may appear simultaneously in both. This task can be thought of as a person re-identification problem as we are attempting to decide whether or not two individuals, seen from different perspectives, are the same. The additional problem is that the individual is being captured from different viewpoints, and therefore, its appearance may vary.

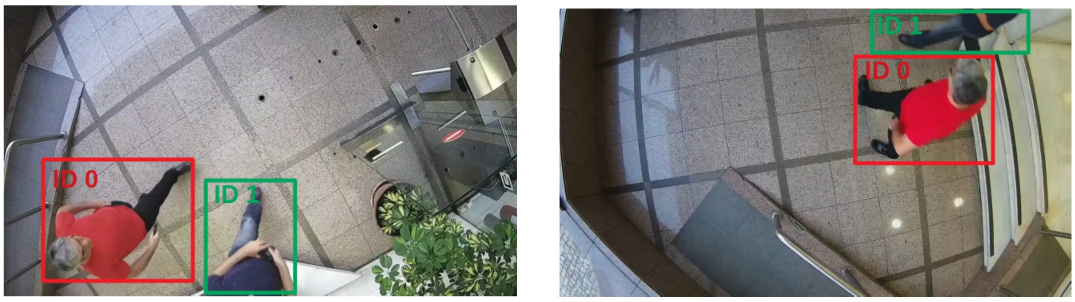


Figure 5.1: Multiple Camera Tracking: camera feeds should be perfectly synchronized in time and the same objects in the different feeds should be linked together in one single trajectory with one object identifier.

This process presents multiple sub-tasks:

Tracked object identities should be linked between cameras, even when multiple objects are present in the fields of view. As so, the algorithm should makes use of position and appearance cues in order to discriminate between different objects. This is a challenge as the representation of each object is different in each camera (see Figure 5.1).

Local identifiers should be mapped into global identifiers: Recall that each camera has its own object ID counter. We denominate these identifiers as local object identifiers. When the local camera information is processed by the multiple camera tracking modules, these identifiers should be translated and mapped into a global object identity referential, common to all cameras. These global identifiers extend through camera fields of view, as the objects move through space.

The architecture should be distributed: If we have a large number of cameras, different camera groups have to run on different machines as the tracking pipeline requires heavy computational power. Some cameras can run on the same machine. In this case, the camera feeds have to be processed in an iterative fashion, one feed at a time.

Synchronization of feeds: It is imperative that the outputs from each camera are perfectly time synchronized, otherwise, the algorithms will fail to link object identities through cameras.

With these goals in mind, firstly we propose an information synchronization system, in which the information from all cameras is aligned in time by PTS. Secondly, we propose two inter-camera target association systems: a one-shot association solution, and an adaptable cost matrix solution. We further describe these systems in the next sections.

5.2 Associating Objects in Different Cameras

Before describing the built systems, it is important to describe what information is available, and how can it be utilized in order to achieve the final goal: to recognize that the individual walking through the field of view in camera 1 is the same individual as the one walking through the field of view in camera 2. We make use of three main cues, which we connect and correlate in order to get the final inter-camera object association: object appearance, temporal, and location information.

5.2.1 Appearance Information

We utilize the appearance model in (Wojke & Bewley, 2018) to model the features of the target object, such as colors, textures, and shapes. Recall that this is a deep learning based model, which employs a deep association metric in order to discriminate between different objects by their features. This network is originally trained on the MARS dataset (Zheng et al., 2016), which contains over one million images for thousands of different identities taken from six different near-synchronized cameras. The difference to our dataset is that ours, besides having a much smaller scale, is taken from top-view cameras, while MARS is not. Even though datasets are quite different in nature, and since our dataset is relatively small in size, we use the pre-trained network on the MARS dataset, made available by the authors online [1] and re-train it on our own appearance data. Recall that this

[1] `https://github.com/nwojke/cosine_metric_learning`

dataset contains samples from both cameras for each object ID, and therefore, the network learns to produce similar features vectors for the same object in different viewpoints. Therefore, in order to associate two objects in different cameras, we calculate the similarity score between observations: we have the network produce feature vectors for both objects and compute the cosine similarity between these vectors. If this score is greater than a threshold, we assume the objects belong to the same identity.

5.2.2 Spatio-Temporal Information

As comparing feature vectors in every frame for every object in the field of view would be expensive, computationally and time-wise, we make use of spatio-temporal information in order to narrow down the search. Therefore, we only compute the features of objects that are spatio-temporally "close" to each other. We solve the temporal aspect using PTS timestamps for each frame. It is quite probable that the PTS of the closest frames in time from different cameras will not be exactly the same. However, a good approximation –to the millisecond– can be achieved by looking for the best possible fit and comparing information from pairs of frames that are the closest possible to each other in time. The spatial aspect is solved using the grids described in previous sections. Firstly, we perform a fully manual step which consists of attributing to each cell in the first camera one or multiple corresponding grid cells in the second camera and vice-versa. In other words, we perform a mapping operation from camera to camera using grid cells. Note that performing a complete mapping of every pixel from camera 1 to camera 2 would be a much more difficult task, as the fields of view of both cameras largely vary in scale. This grid mapping strategy is only an approximation, as one grid cell in one camera will never perfectly fit a grid cell in the other camera, but it still enables us to greatly reduce the search space and therefore improve efficiency. Once we have these spatial and temporal cues, we are able to eliminate unnecessary computation by comparing the similarity of only those objects that meet the constraints of being in roughly corresponding grid cells in both cameras at about the same time.

5.3 Multiple Camera Tracking Architecture

Multiple camera tracking has four main ideas: firstly, each camera tracks objects present in its field of view independently of the other cameras. Secondly, single camera tracking information –object appearance and spacio-temporal information– from all cameras is synchronized, in order to be aligned in time. Thirdly, information from different cameras is crossed and connected, having local camera object identifiers mapped into global identifiers. Lastly, the outputs of all the previous modules are displayed for the user –the global tracking IDs and respective bounding boxes are drawn directly on the frame. As so, we propose a system that can be decomposed in these four main components:

- single camera tracking pipeline;
- synchronization of information pipeline;
- global track assembly pipeline;
- visualization pipeline.

Each of these pipelines is composed of different modules in charge of different tasks, and information flows from module to module. The high-level system architecture is represented in Figure 5.2. Recall from Section 5.1 that we propose two different variants in order to connect information from all cameras. These strategies are held in the global track association pipeline, and are interchangeable with each other, while the rest of the modules are maintained.

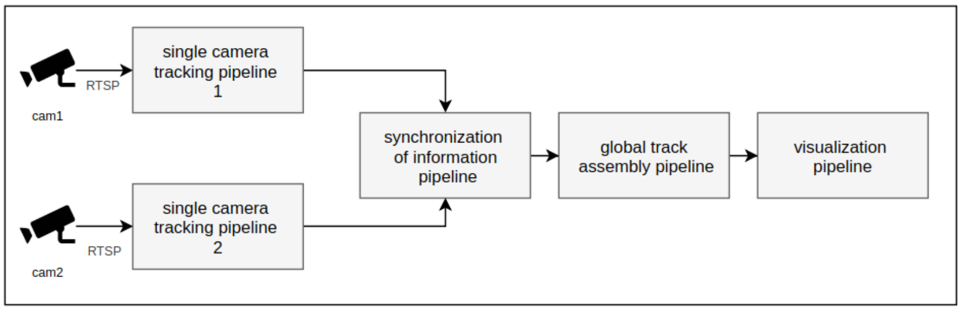

cam1
RTSP
single camera tracking pipeline 1
cam2
RTSP
single camera tracking pipeline 2
synchronization of information pipeline
global track assembly pipeline
visualization pipeline

Figure 5.2: Multiple Camera Tracking Architecture: The system is composed of modules, where the output of one is the input to the next.

5.3.1 Single Camera Tracking Pipeline

Each camera has its own tracking pipeline which is used to track objects independently from the other cameras (Figure 5.3). This pipeline is mostly identical to the one described in the previous chapter, with the exception that in this case, multiple tracking pipelines can run on the same machine. In this case, each camera has its own ffmpeg stream reader and tracking module. To save computational power, the entirety of the detection module (models and logic), and the tracking model (model only) are shared. For each frame, tracking information is written to a Redis queue, from which the synchronization module will subsequently read and process this information. Using a Redis queue to write this information enables us to easily distribute the system between different machines. Tracking information is composed by the camera identifier, the frame presentation timestamp and the list of current tracked objects, composed of an object identifier, object position in the grid –which we call *state*–, object bounding boxes and object features, computed using the appearance model. Note that we do not send the frame itself into the Redis queue: as the frame is only needed to compute appearance features we compute them in this pipeline straight away and avoid sending the entire frame, or frame crop to the Redis queue. If results visualization is needed, this module also writes frames identified by PTS into a remote file system, in order for this information to be used later by the visualization module.

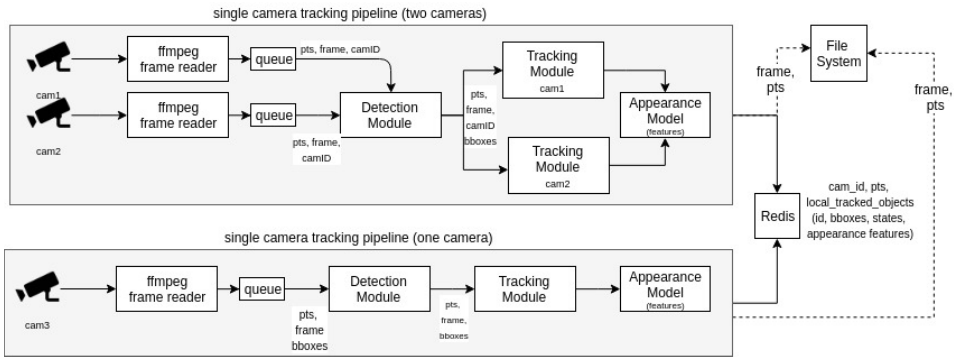


Figure 5.3: Single Camera Pipeline. This module is almost identical to the single camera tracking pipeline described in the previous chapter. The exceptions are the here we extract the features of each tracked object right away and send those features the next module instead of sending images. Additionally, here we account for information from multiple cameras being processed on the same machine.

5.3.2 Information Synchronization Pipeline

The purpose of this set of modules (Figure 5.4) is to take the information from each of the individual single camera tracking pipelines and synchronize the messages in time so that the next module receives information in the correct order.

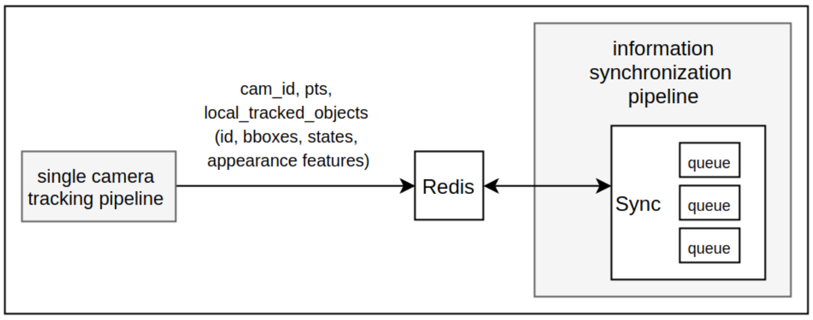

single camera tracking pipeline
cam_id, pts, local_tracked_objects (id, bboxes, states, appearance features)
Redis
information synchronization pipeline
Sync
queue
queue
queue

Figure 5.4: Information synchronization pipeline. In this module, a threaded class constantly reads single camera tracking information from the Redis queue and divides it by camera ID into different FIFO queues. Subsequently, we apply draining operations to these FIFO queues in order to have the information from different cameras perfectly aligned in time.

The synchronization module is in charge of reading different camera tracking information messages from the Redis queue and aligning this information in time to the best match possible. In order to align the information, we employ a set of FIFO queues, in which each camera has a queue assigned to it. Additionally, we employ a threaded class that constantly reads tracking information from the Redis queue and transfers this information to camera specific FIFO queues using the camera ID in each message. The stream synchronization operation is performed once at the beginning, and subsequently, it is performed periodically during the runtime of the program. This operation involves draining the FIFO queues in which tracking information has lower PTS timestamps until the difference between timestamps in all FIFO queues is small enough to be insignificant, as represented in Figure 6.5. This process ensures that the next module receives all tracking information synchronized in time in a near-perfect manner.

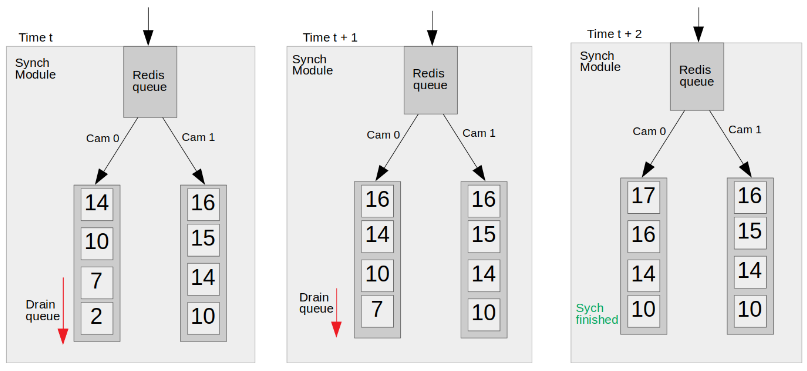


Figure 5.5: The synchronization operation consists of draining camera specific FIFO queues using timestamps until the difference between timestamps of messages from different cameras is minimal or none.

5.3.3 Global Track Assembly Pipeline

The task of the set of modules (Figure 5.6) is to link objects IDs from one camera to object IDs from another camera, which belong to the same actual objects. It is in charge of using the tracking information messages from each individual camera, already synchronized in time, in order to build the overall object tracks, and global identity referential between all cameras. We propose two solutions for this problem: a one-shot association variant, and an adaptable distance matrix variant. Both are described in the sections below. The final output of these set of modules is a list of the global objects to all cameras, the camera ID and the PTS, which is sent to another Redis so that the visualization module can read and display this information on the frames.

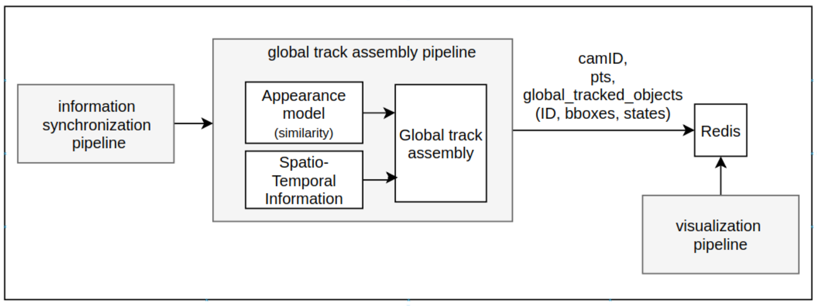

information synchronization pipeline
global track assembly pipeline
Appearance model (similarity)
Spatio-Temporal Information
Global track assembly
camID, pts, global_tracked_objects (ID, bboxes, states)
Redis
visualization pipeline

Figure 5.6: Global Track Assembly Pipeline. This module takes already synchronized tracking information as input and employs spacial and appearance similarity operations in order to assemble the global paths of the target objects through all cameras.

One Shot Association

This strategy links the object identifiers only in the camera transition area, essentially, transitioning object identifiers through cameras as they cross the different fields of view. In order for this to take place, we have to successfully propagate object identifiers through cameras. As so, we have a global track ID counter. We increment this ID every time an object enters a camera viewpoint and propagate this ID to the next camera when the objects transitions into the overlapped area. In other words, we link object IDs through cameras using the overlapped area between the fields of view. In order to do so, we employ the grids and object appearance features. Recall that the grid cells are manually mapped between cameras: for instance, grid cell 7 in camera 1 roughly corresponds to grid cell 6 and grid cell 9 in camera 2. As so, we perform a prediction of positions for the next iteration from one camera to another at each time step: if object A is in grid cell in grid cell 7 in camera 1, it should probably appear in grid cell 6 or 9 in camera 2 in the next iteration or next few iterations. When a corresponding object is detected in camera 2, in the expected grid cell, we compare the feature vectors of both objects. If these vectors match, i.e, if the cosine similarity between vectors is greater than the considered threshold, we propagate the ID from object in camera 1 to the new object in camera 2. The following steps are followed when a new message with track information is read from the queues, for each track:

- If it is the very first object, i.e. the tracked objects list is empty, initialize that ID by adding it to the list;
- If it is not the first object, we are going to try to match it to a previous object in the same camera. We perform this solely by checking the previous object IDs in this camera. If the IDs match, it is the same object. We do not do anything else for this object in the current iteration;
- If no match is found, it means one of two things: the object is new to this camera and came to this position from another camera, or, the object is new to all cameras, i.e, it is the very first time this object appears anywhere in

the fields of view. In order to check if this object came from another camera, we are going to attempt to match position and appearance features. If the position of the object in the current camera is consistent with the position predicted for the objects in the other camera at the previous time step, we proceed to check the features. To test feature similarity we empirically established a lower bound similarity threshold. In other words, if the feature similarity is greater than this threshold, we propagate the object ID to the new camera. If one of these conditions does not hold, we assume it is a new object for all cameras and initialize a new ID;

- We remove dead tracks: tracks that have not been detected in at least MAX_AGE frames and whose last position was registered in an exit grid cell;
- We update inter-camera position expectations for the next time step.

Adaptable Distance Matrix Association

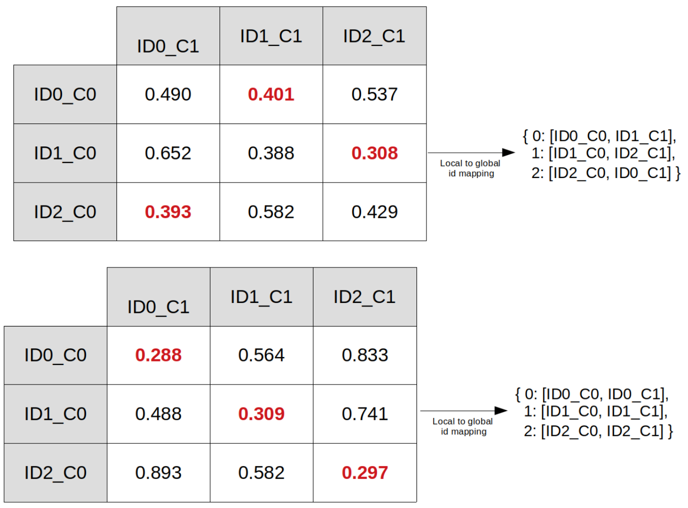
As the name implies, this system is based on using an adaptable distance matrix in order to associate objects from different cameras, and integrate them into a global identity referential for all cameras. The biggest difference from this algorithm to the previous, is that, here, once an ID is transitioned between cameras, it can still be switched in the future, while in the previous case it would be fixed forever –while the object is present in the field of view of the camera where it was transitioned to. Note that we only want to compute the correspondence between objects from one field of view to another when these cross the overlapped area since this is when objects have representations in both views. This can be decomposed in the following sub-problems:

The first problem to solve is the following: **how to know if an object has representations in both fields of view?** In order to solve this problem, we utilize the grids, similarly to the one-shot association algorithm. Recall that we do not map the space from one field of view to another in every point of the frame since these are taken by different fish-eye cameras, and largely vary in scale and aspect ratios which makes the problem of mapping every pixel in one view to a pixel in another view a more difficult challenge. As so, we employ the grids in each camera in order to roughly map the cells that are in the overlapped area. This mechanism is afterward used to check if one object in one view has a candidate object in another view, in order to perform further computations that will decide if the objects are the same –e.g., check the similarity between feature averages of both objects.

The second step to solve is to **compute the distance matrix,** M, in each iteration. Recall that as we only want to compute this matrix to find corresponding objects in overlapped areas, we compute one matrix for each of the camera pairs that share overlapped areas. In these matrices, each column corresponds to objects in one camera view and each row to objects in another camera view. Once we decide which objects should be compared –by checking if their current location is overlapped with another view–, we compute cosine distance of the feature average of each object and all the remaining objects in the other view –all row/column associations. In other words, each position (i, j) in the matrix is given by $M(i, j) = d(F_i, F_j)$, were $d()$ is the cosine distance function and F_i and F_J are the averages of all the feature vectors of objects i and j until the current iteration.

The following step is to **find the optimal object mapping** or to decide which objects are most similar– or less distant. In order to do so, we employ a mechanism that iteratively finds the most similar pair (i, j), associates object i to object j and deletes the i-the row and the j-th column from the matrix from the next iteration to take place. This process ends when there are no rows and columns left in the matrix.

The last step is to **update the correspondences between objects and global IDs**. Every object crossing the field of view, besides its local ID, has to have a global ID at all times –even when it is not in the overlapped area. This is necessary as we are tracking the objects in real-time, and need to compute the global IDs right when the object enters the field of view. The challenge is transferring these global IDs from one camera view to another camera view and switching these global IDs when the matrix correspondence changes (Figure 5.7), without any information from the future.



	ID0_C1	ID1_C1	ID2_C1
ID0_C0	0.490	**0.401**	0.537
ID1_C0	0.652	0.388	**0.308**
ID2_C0	**0.393**	0.582	0.429

Local to global id mapping

{ 0: [ID0_C0, ID1_C1],
1: [ID1_C0, ID2_C1],
2: [ID2_C0, ID0_C1] }

	ID0_C1	ID1_C1	ID2_C1
ID0_C0	**0.288**	0.564	0.833
ID1_C0	0.488	**0.309**	0.741
ID2_C0	0.893	0.582	**0.297**

Local to global id mapping

{ 0: [ID0_C0, ID0_C1],
1: [ID1_C0, ID1_C1],
2: [ID2_C0, ID2_C1] }

Figure 5.7: Adaptable distance matrix data association: the cosine similarity scores between features of objects in different cameras may change from time step to time-step.

This apparently straightforward process is non-trivial when analyzed with more depth. This is difficult as every time there is a switch –a different object association from the previous iteration in the matrix– we need to re-assign global IDs and decide which associations inherit which IDs. There are different possibilities to handle different types of object association switches. Roughly, we can decompose all the possibilities in the following cases:

- **Situation 1**: Neither of the objects in the new association (i, j) was previously in another association. We need to create the association, increment the global ID counter, assign this ID to the new association, append it to the association list and continue to the next association. This situation is represented in Figure 5.8;
- **Situation 2**: The simplest scenario occurs when both objects in association (i, j) were already associated to each other in the previous iteration, and, as so, we do not need to update the global ID. We only update the information

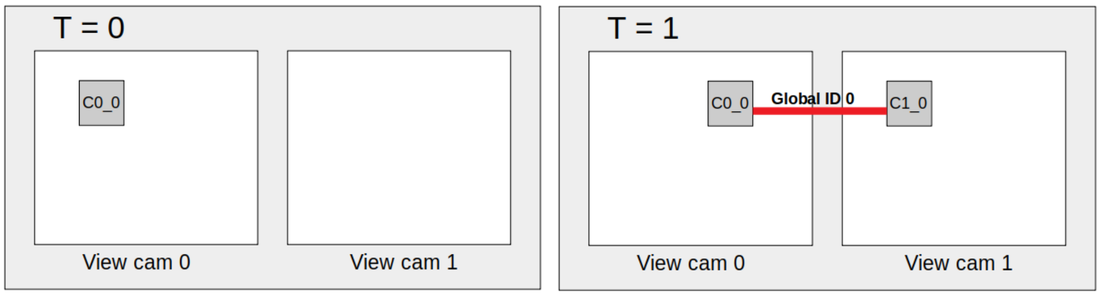


Figure 5.8: Global ID switches (1). At T=0 we have the first object $C0\_0$. We assign a new global ID 0 to this object, and continue. At T=1, a new object, $C1\_0$, appears in the camera 1 view. We compute the distance matrix and find out that $C0\_0$ and $C1\_0$ are the same object. As so, we append $C1\_0$ to the global object association $(C0\_0, C1\_0)$. This is straight forward, since, as $C0\_0$ was alone in the previous association (T=0), we only have to append $C1\_0$ to that association, which maintains the global ID 0.

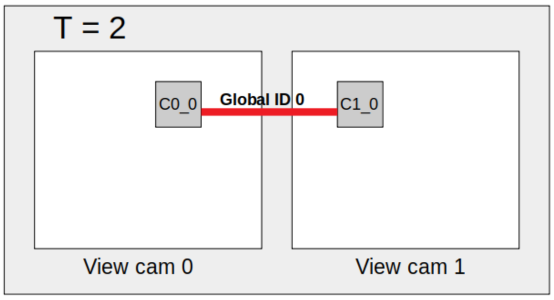


Figure 5.9: Global ID switches (2). As the association $(C0\_0, C1\_0)$ from T=1 is maintained at T=2, we only update the information of each local object and continue.

for each object –bounding box, camera IDs, frame IDs– and continue to the next object association. This situation is represented in Figure 5.9;

- **Situation 3**: Object j from the new association (i, j) is in a another previous association with two objects (i, k), while object i is alone in an association (i). In this case, we have to decide which global ID is going to be assigned to the new association (i, j). For this to be correct, we perform two operations: check the association (i, k), in which object i was previously and determine which local object, i, or k is oldest, or in other words, which object is present in the field of view for the most time. This is done as the oldest object in any previous association should always inherit the global ID that previous association in order to maintain coherence. This situation is represented in Figure 5.10;

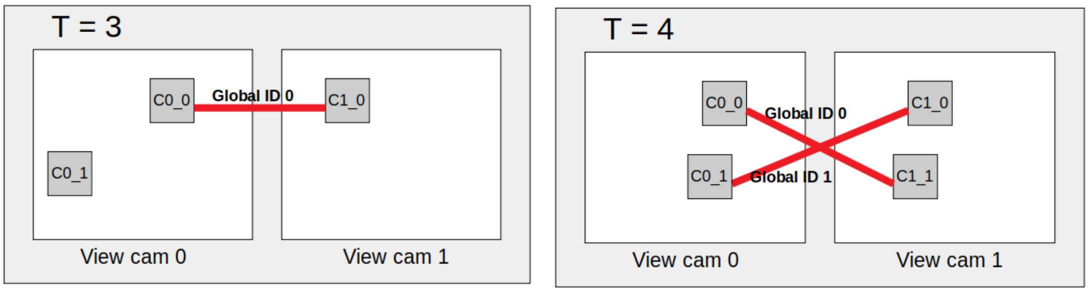


Figure 5.10: Global ID switches (3). **At T=3** we have a new local object in camera 0, $C0\_1$. As this object is at the entrance area and has no representation in the other camera, we attribute global ID 1 to it and increment the global ID counter. Also at T=3, association $(C0\_0, C1\_0)$ is maintained, as so, we update the local object information and continue. At **T=4** we have two new associations, $(C0\_0, C1\_1)$ and $(C0\_1, C1\_0)$. Notice that the association switched objects. We begin by ordering the new association list by the age of the oldest object in the associations and iterate over each one. As so, we begin by the association $(C0\_0, C1\_1)$, which contains the oldest object. Here, the oldest object is $C0\_0$, and we can see that this object, in the previous iteration was in association $(C0\_0, C1\_0)$. As object $(C0\_0)$ is the oldest in the previous association (between object $C0\_0$ and $C1\_0$), we remove $C1\_0$ from this association and append it to a removed object list, leaving $C0\_0$ alone, and proceed to append the $C1\_1$ to this association. This means that now, the new association $(C0\_0, C1\_1)$ at T=4 will have global ID 0, inherited from object $C0\_0$. We then proceed to the other association at T=4, $(C0\_1, C1\_0)$, where the oldest object is $C1\_0$. However, this object is also in the removed objects list, which means that it will not maintain its previous global ID –as it was inherited by its pair of the previous association. As the newer pair in the new association, $C0\_1$, has global ID 1 associated to it and is in an association alone, we append $C1\_0$ to $C0\_1$. As so, the association ($C1\_0$, $C0\_1$) maintains global ID 1. We clear the removed object list and continue.

- **Situation 4**: Association (i, j) and (k, l) were association (i, k) and (j, l) in the previous iteration. Effectively, this situation corresponds to a total pair object switch from the previous association. This situation is represented in Figure 5.11.

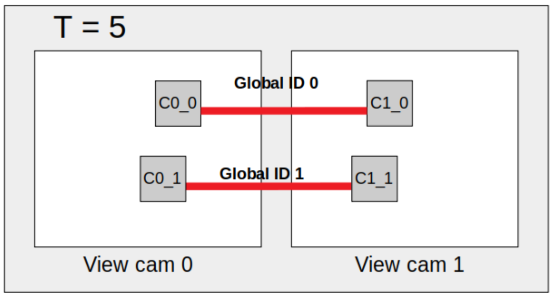


Figure 5.11: Global ID switches (4). The logic in this situation is the same as in the previous. We begin by ordering the new association by the oldest object in each pair and proceed to iterate over that list. First new association to be checked is $(C0\_0, C1\_0)$ as object $C0\_0$ is the oldest. We check its previous association $(C0\_0, C1\_1)$, remove the newest object in that association –$C1\_1$– and append it to the removed objects list. We then check if the new $C0\_0$ pair, $C1\_0$, was in any previous association, and if it is, we remove it. We do so in order to avoid duplicate local objects in the association mappings. We can then append the new $C0\_0$ pair, $C1\_0$, to the to the new association $(C0\_0, C1\_0)$. As so, this association inherits global ID 0. We then proceed to the second new association $(C0\_1, C1\_1)$. The oldest object in this association is $C0\_1$, which previously had global ID 1 assigned to it, and is now associated to object $C1\_1$, which is now in the removed objects list. As so, we remove $C1\_1$ from the removed object list, append it to to the association where $C0\_1$ is. Therefore this second association inherits global ID 1.

We can summarize the object association update with Algorithm 1.

5.3.4 Scaling to More Cameras

Over the course of this chapter we have developed and tested a multiple camera tracking system using two cameras with overlapping fields of view. As our goal for this system is for it to be deployed in large areas we have to consider how it scales to more than two cameras. Two of the pipelines automatically scale without further alterations: the single-camera tracking pipeline and the information synchronization pipeline. However, the global track assembly pipeline requires additional considerations in order to be scaled. The modifications are simpler for the one-shot association algorithm, firstly introduced in section 5.3.3: the only adjustment needed is to manually perform the mapping for all the grids of the cameras with adjacent fields of view. However, when scaling to large amounts

Algorithm 1 Object Association Update

```
 1: function UPDATE_OBJECT_ASSOCIATION(association_list)
            ▷ Sort the new association list by the age of the oldest object in each association
 2:     association_list.sort()
 3:     removed_objects = []
 4:     for obj1, obj2 in objects do
 5:         if association_already_exists(obj1, obj2) then
 6:             update_object_info(obj1, obj2)
 7:             continue
 8:         end if
 9:         if obj1 in removed_objects  obj2 not in removed_objects  then
10:             previous_association = get_association_(obj2)
11:             append_to_previous_association(obj1)
12:             update_object_info(obj1, obj2)
13:             continue
14:         end if
15:         if obj2 in removed_objects  obj1 not in removed_objects then
16:             previous_association = get_association_(obj1)
17:             append_to_previous_association(obj2)
18:             update_object_info(obj1, obj2)
19:             continue
20:         end if
21:         if obj2 in removed_objects  obj1 in removed_objects then
22:             new_association(obj1, obj2)
23:             global_id += 1
24:             continue
25:         end if
26:         older, newer = order_by_age(obj1, obj2)
27:         if already_in_association(older) then      ▷ get the newer pair of the previous association in which older was
28:             obj_to_remove = get_previous_newer_pair(older)
29:             if object_to_remove is not None then  ▷ remove the newest pair from the previous association
30:                 removed_objects.append(object_to_remove)
31:                 if already_in_association(newest) then
32:                     remove_object_from_previous_association(newer)
33:                 end if
34:                 append_to_previous_association(newer)
35:                 update_object_info(obj1, obj2)
36:             end if
37:         end if
38:     end for
39: end function
```

of cameras this manual mapping becomes unpractical. A good alternative is to employ planar homography techniques in order to align the fields of view, as made in (S. M. Khan, Yan, & Shah, 2007; Eshel, Moses, & Arazi, 2008; Detone & Rabinovich, 2016). For the adaptable distance matrix algorithm of section 5.3.3 the alterations are more noticeable. Here we need one distance matrix for each pair of cameras with adjacent fields of view. Therefore, if we have three cameras, and all of them have overlapping views with each other, we will have three distinct matrices. Additionally, in order to correctly update the correspondence between local IDs and global IDs for all cameras, we need to consider information from all the fields of view simultaneously and perform the updates based on information from all cameras. As the present work is ongoing, adding more cameras is the next step in order to further develop a system capable of handling multiple object tracking in large areas.

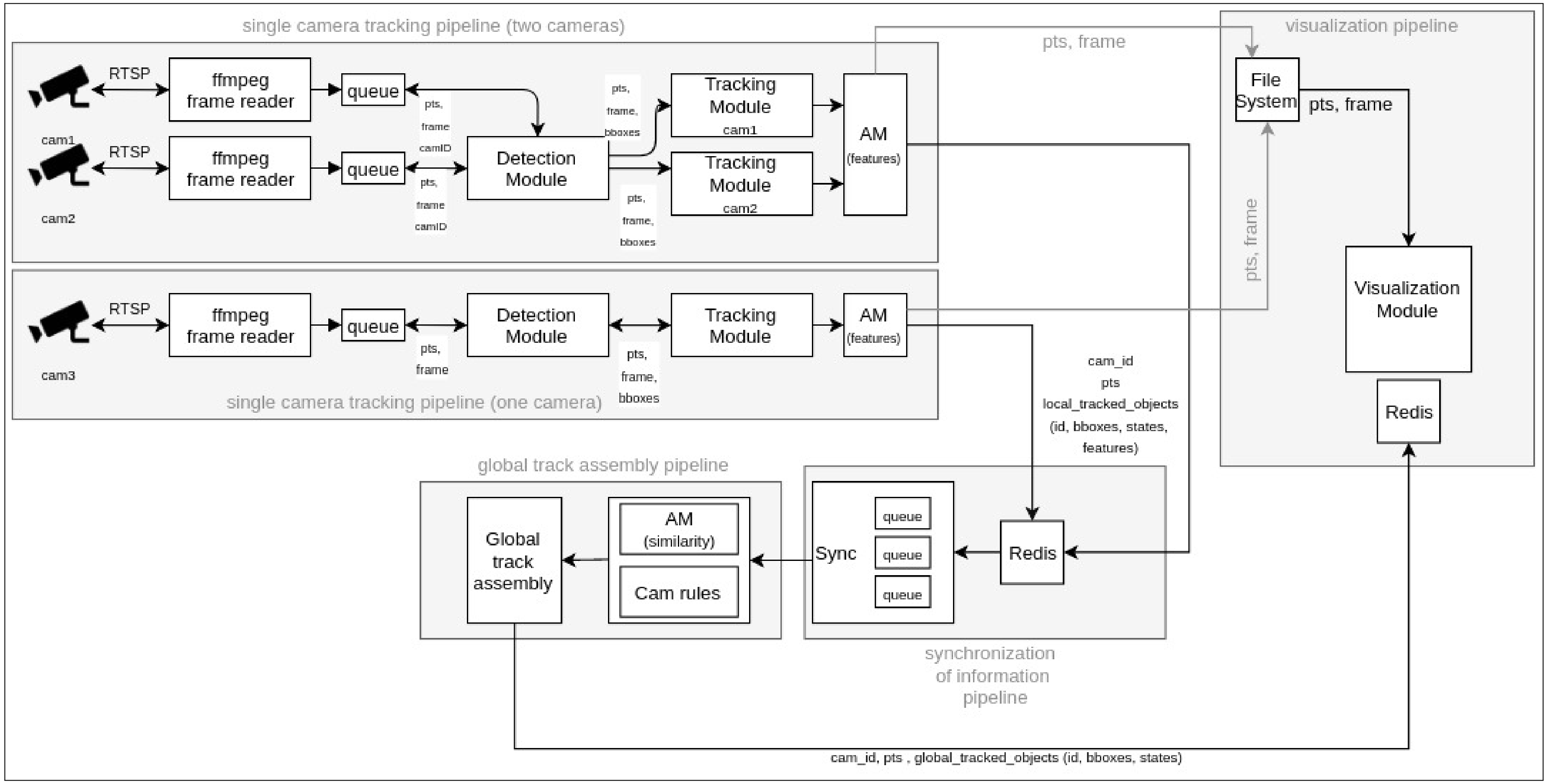


Figure 5.12: Multiple Camera Tracking Full Architecture

Chapter 6

Results

We split this chapter into two main sections: firstly, we present the process of building the datasets which were later used in order to train and evaluate the models. We discuss how the data was gathered, labeled and the final structure for each of the types of data annotations. Secondly, we present the results of the evaluation of the algorithms used in the present work. We evaluate all the components -detection, appearance and tracking models- separately. As we have no benchmark we use our own custom data test sets for each of the model types.

6.1 Datasets

The datasets are a crucial factor for the success of the present work, which is composed, for the most part, of data-driven models, namely the used detectors, trackers and appearance models. The datasets provide a way to train and evaluate these models. The sections below describe how we built them, namely the pipeline for data gathering and data labeling, how this process evolved over time, how the models were trained and the challenges regarding these tasks. There are currently quite a few tracking datasets available, namely the Visual Object Tracking (VOT) dataset (Kristan et al., 2017), and Object Tracking Benchmark (OTB) (Wu, Lim, & Yang, 2015) for visual object tracking, and the Stanford Aerial Pedestrian Dataset (Robicquet, Sadeghian, Alahi, & Savarese, 2016), the Multiple Object Tracking (MOT) dataset (Milan, Leal-Taixe, et al., 2016), DukeMTMC dataset (Ristani, Solera, Zou, Cucchiara, & Tomasi, 2016) or the KITTI dataset (Geiger, Lenz, Stiller, & Urtasun, 2013) for multiple object tracking. Furthermore, for the tasks of image classification and object detection, there are even more datasets available, such as the Pascal dataset (Everingham et al., 2014), the ImageNet dataset (Russakovsky et al., 2015), or the COCO dataset (Lin et al., 2014), and many others. Additionally, there are also multiple person re-identification datasets available: the VIPeR dataset (Beeck, Engeland, Vennekens, & Goedem, 2017), the Market1501 dataset (Liang Zheng, 2016), or the MARS dataset (Zheng et al., 2016).

However, to the best of our knowledge, there are currently no top-view multiple camera tracking dataset publically released, that also satisfy the requirements of having overlapping fields of view between cameras, and that are compliant with other requirements of the present work (e.g., cameras should not be placed too high up).

For this reason, one of the key points of the present work was to build the necessary datasets. The process of building a dataset from scratch is a challenging and iterative one: the final datasets used for the present work were continuously built upon over the period of one year. Furthermore, for the models to produce the desired results, the data not only has to be large in size, but also be varied and clean, and that takes a lot of time and effort. A good dataset can only be achieved by continuously accumulating, cleaning, and balancing data in an iterative process: feeding data to a model, observing its behavior and how it reacts to that data, isolate failure cases, create new data to neutralize those failures, possibly go back to the previous data and clean it further, forming one big data creation loop.

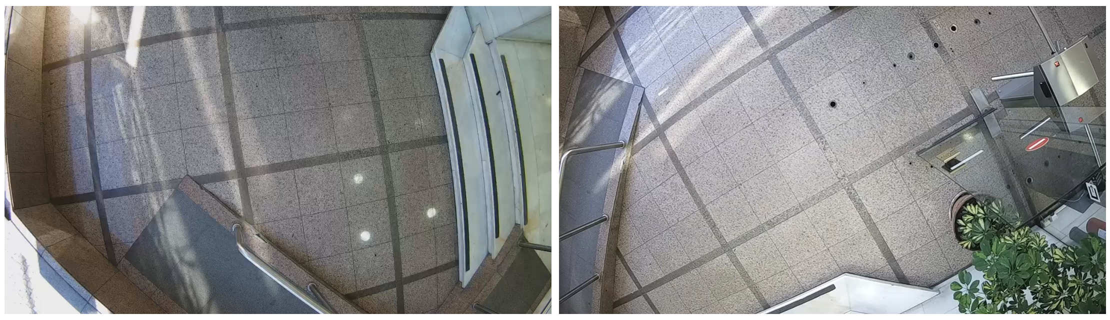


Figure 6.1: Camera fields of view.

6.1.1 Environment Setup

The first step in order to collect datasets, develop algorithms, and test models was to assemble a controlled test environment. This setup was assembled at the INOV-INESC entrance area, which is situated in a Lisbon building where hundreds of different people go through daily. The setup consists of two top view fish-eye cameras which provide live video streams, accessible in real time via RTSP protocol. Both cameras are Dahua, model DH-IPC-HDBW4431R-ZS and model DH-IPC-HDW5431R-Z, and both have 4 Megapixels and provide H.264 support. The cameras are set to a height of about 3 meters and, between both cameras, we achieve a coverage of about $15\,\mathrm{m}^2$ of the area –basically the entirety entrance area. The cameras were assembled in a way so that their fields of view are overlapped as shown in Figure 6.1.

In order to harvest data to label, we employed a script which ran three times a day, during rush hours, for the period of one hour, read frames from the live RTSP streams and stored them in a storage system.

6.1.2 Data Formats

Before data can be labeled, it is important to consider that all three types of models –detection, tracking, and appearance models– require specific input file data formats in order to be trained.

Out of the three, the simplest scenario is a **object detection** dataset: in order to train an object detection model we need a file that maps the frame location of objects which we need the algorithm to detect –the coordinates of bounding boxes enclosing object of interest $(x\_min, y\_min, x\_max, y\_max)$– to the image files –the frames themselves–, and frame order does not necessarily matter.

The **tracking models** require extra information: Additionally, to the mapping of the bounding box to the image file, each bounding box enclosing each object must be associated to an object identifier, which will be used to discriminate between object identities. Here, frame order matters, as some of the used trackers learn to track objects from the movement of these objects –i.e. how each object changes position and appearance from frame $t - 1$ to frame t. As so, the final tracking dataset is composed of a file that links frames to tracklets, i.e. sequences of bounding boxes of the same object ID in multiple adjacent frames, or in other words, fragments of tracks. Note that the track of one object begins when an object enters the field of view and ends when this object exits the field of view. If after leaving the field of view, an object re-enters, it is given a new tracking ID.

Building datasets for the **appearance models** is more challenging than for both detection and tracking models. These models require thousands of identities and hundreds of examples for each identity. Additionally, each identity should have examples from multiple camera viewpoints so that the models can learn different representations for the same object. Each identifier is not only composed by one tracklet as in the tracking dataset case but by every tracklet belonging to that object –in contrast to the tracking dataset, when an object re-enters the field of view after it has left, it should be given the same ID it had previously. So in order to build this dataset, one must continually re-identify objects that enter the field of view and try to match them with previous objects already labeled, and this process should be repeated for multiple camera viewpoints. Similarly to the tracking dataset, the final dataset for the appearance models should be composed by a file that maps frames to bounding boxes with IDs, but to a much greater scale since one ID is composed by all available tracklets of that object in one video sequence. Furthermore, for easy access, frame crops should be stored in a directory

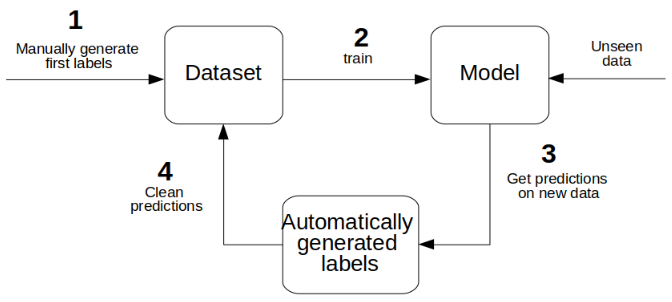

1
Manually generate first labels
Dataset
2
train
Model
Unseen data
3
Get predictions on new data
4
Clean predictions
Automatically generated labels

Figure 6.2: Data labeling pipeline. The first task consist of manually generating the first labels, as a completely untrained model will not be able to make accurate predictions. Secondly, labeled data is fed to the model so that it can be trained. The third step is to have the trained model generate predictions for new, previously unseen data, and finally, those predictions have to be manually cleaned and filtered, in order to be concatenated with the previous dataset.

tree: each object identifier has its own directory and using the generated bounding boxes, frames are cropped and those crops are saved into the respective directory.

6.1.3 Labeling Data

In order to label large amounts of images retrieved from the RTSP streams, and make this process the less human labor intensive possible, there was the need to create a data labeling pipeline, which is represented in Figure 6.2. At a high level, this pipeline is applied to all three types of models –detection tracking and appearance models. Overall, for all the model types, this process is composed by 4 sub-tasks: manually generating the first labels, training the model with that data, having the model generate predictions on previously unseen data and finally cleaning and filtering those predictions so that they can be concatenated with the original dataset. However, it is necessary to consider that specific implementation requisites vary from model to model, essentially creating sub-pipelines for each model. These are described in the sections below.

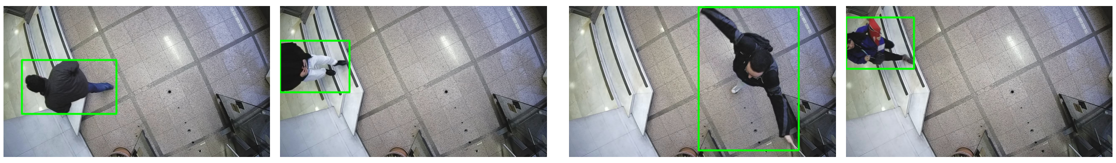


Figure 6.3: Labeled detection frames.

6.1.4 Manually generating labels

This step is necessary as completely untrained models or models pre-trained with datasets such as ImageNet (Russakovsky et al., 2015) or COCO (Lin et al., 2014), will not be able to make predictions on our custom data that are accurate enough to easily be concatenated with previous data. In other words, it is simpler and quicker to hand-draw the first labels than to extensively clean and correct wrong predictions. As so, this step was applied for the detection and tracking models, but with a slight difference.

Detection Models: For the detection models, the first labels were generated in a fully manual manner: 2000 frames were selected, and bounding boxes were manually drawn in each frame, frame to frame. There are diverse image labeling programs available, however, for convenience reasons, a custom script with a labeling interface was created. This was done so that the output is created in the correct format, and there is no need to run any post-processing. This script enables the user to list and browse frames and draw bounding boxes over these. As each frame is labeled, labeling information is written to the output file, containing the image path and the respective bounding box coordinates. A few examples of labeled detection model data are represented in Figure 6.3.

Tracking Models: Generating the first labels for the tracking models is a slightly different process. Recall that the output file format for these models consists of, for each frame, bounding box coordinates as well as the object identifiers and some sort of timestamp, which specifies a reference of order in the frames. Having a trained detection model, there was no need to generate bounding boxes for the tracking data in a fully manual manner. Instead, a trained detection model is employed to generate detections for the tracking data. The labor-intensive task

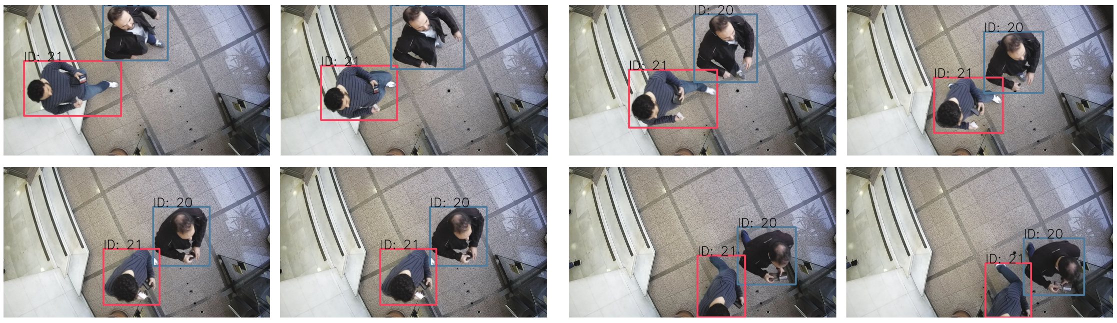


Figure 6.4: Labeled tracking sequence.

resides in labeling each bounding box in every frame with an object identifier, in order to create the tracklets, as represented in Figure 6.4. Similarly to the interface used to crate data for the object detection models, another interface was implemented in order to link bounding boxes, object identifiers and frame identifiers for the tracking models. This program allows to list frames and input tracking identifiers for each bounding box in an image and writes this information to the output file in the correct format.

Appearance models: The appearance model data labeling process builds upon both detection and tracking labeling processes. Recall that the objective is to link tracklets that belong to the same object: as so, the first step is to run the detection and tracking models, which output these tracklets. The remaining steps are to link together tracklets that belong to the same object, link these tracklets through multiple camera viewpoints, crop all the frames using the bounding boxes, and store these crops in a directory structure. The final data looks similar to the example represented in Figure 6.5. In order to perform these steps in the fastest way possible, another interface was created: this interface shows crops of frames containing objects from each tracklet to the user and asks for the final object identifier for that object. At the same time, in order to facilitate this task, it allows the user to browse previous object identities. Once the user inputs the information –frame filenames, frame IDs, bounding box coordinates, and object IDs–, it is written to the output file. Furthermore, frame crops are written to the directory structure, in the directory respective to that object identifier.

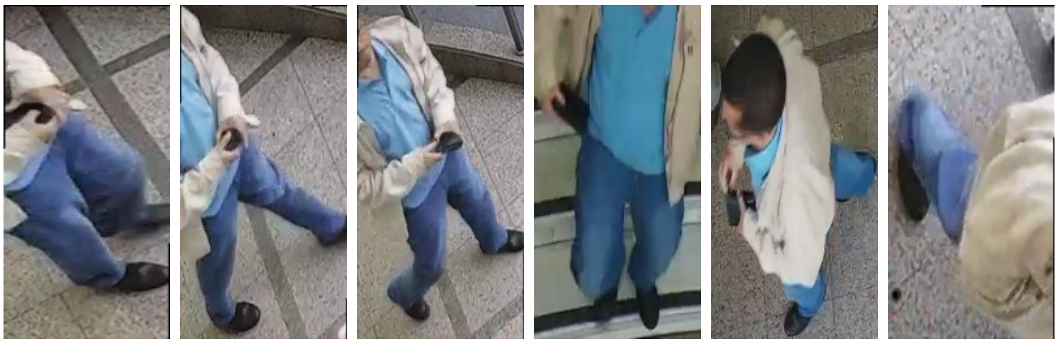


Figure 6.5: Appearance model dataset samples belonging to the same ID. The samples are taken from different cameras and were already resized in order to be fed to the networks.

6.1.5 Label generating loop

Once the initial labels are created, the remaining processes are a constant data enlargement loop: training the models, having them predict on new data, clean the predictions and concatenate clean predictions with existing data. This process can be considered a form of Semi-Supervised Learning (SSL) and more concretely Pseudo Labeling, in which, the purpose is to label unlabeled data using a model trained on some labeled data. There are a variety od different strategies to SSL, namely Co-training (Blum & Mitchell, 1998): in this work two different classifiers are trained on different views of the data, and used to generate new training data.

In the present work, a similar strategy was used: three different object detection models are trained on three different sets of data. These models are then shown new data and asked to create new predictions. The final label is a combination of all predictions: we check if all models *agree* on the predictions: to do this, as bounding boxes produced for the same object may vary in coordinates, we employ an IOU overlap between predictions, and if this overlap is greater than a pre-specified threshold we consider that the produced bounding boxes correspond to the same detected object. If all models agree with the bounding box prediction, the final bounding box will be the average of all the bounding box coordinates in each of the predictions. If two models agree, then the final bounding box will have as coordinates the average of the all respective bounding box coordinates in those two predictions. If only one model predicts a bounding box, it is assumed that no object exists in that location.

This strategy saves time and human labor, however, there still has to be fully manual cleaning stage, in order to make sure that data is valid and clean. To this end, a data validation interface is created: the user is shown all bounding box predictions, frame by frame, and can choose to accept them as valid, if the bounding box is correct, redo them if the bounding box is not in the right place, or reject them if the bounding box is prediction is wrong. This strategy is mainly used to generate detection data since detentions are the base for both object tracking and appearance model data, however, it can be extended to these tasks as well. In this way, some relief in human labor is achieved by making data enlargement process semi-automated.

6.1.6 Labeling Conventions

One of the challenges with the process of creating a dataset from scratch is to define conventions and standards in order to make sure that data is balanced, clean and consistent throughout. One of these decisions lies in **what labels are chosen to be in the data**. Note that this is important to consider since the images are taken by top view fish-eye cameras, and therefore if a person is in the peripheries of the fields of view, it is likely that only the legs or the arms are visible, as shown in Figure 6.6. In many cases there is not one really clear solution, and as so is necessary to analyze a few variants in order to decide what labels two use and how to use them.

One of the variants is to introduce *Leg* or *Arm* labels for the cases when only one leg and or one arm are visible. However, this introduces additional complexity as a new set of rules and conditions should be met when deciding if a person should be detected in the field of view or not: *if two legs and two arms then person.* Furthermore, if this process is done for legs or arms, then it should also probably be done for the class *Head* or *Torso* as well. As so, instead of designing a set

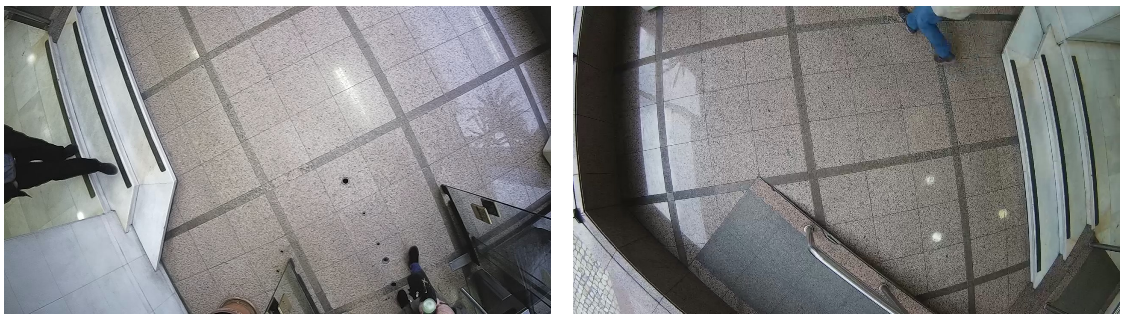


Figure 6.6: Examples of data with people only in the peripheries of the fields of view.

of rules and in order to avoid unnecessary complexity, as a neural network can be successfully trained for one class, *Person*, and if the examples in this class do not contain small fragments of people, for instance only one foot, it was decided the most viable solution was to have one single class *Person* in which examples of partially occluded people are present only if half or more of the person is visible. This excludes every case in which only shoes and legs are visible.

Another problem to consider is **data variety**. Most of the occurring situations are of people entering or exiting the building –recall that this is an entrance area–, and to do so they take the quickest way out or in, as in the sequence represented in Figure 6.7. As so, **these situations are inevitably going to be over-represented in the data** if we get it at random from the RTSP streams. In order to make sure data is varied when building the dataset, we need to be very particular with the chosen examples to make sure that one type of data is not over-represented and another is not underrepresented. With this in mind, many of the chosen examples are taken from rush hours, particularly the lunch hour: it is common for groups of people to wait by the entrance and to move more freely in the fields of view. Another example of this problem is **underrepresented situations**. These are important to keep in mind since if not enough examples are given to the network, it may struggle at test time. One of these problems is that it is fairly uncommon to have people very close together, for instance greeting each other or handshaking, as represented in Figure 6.8.

This particular type of situations consistently presented itself as a failure case. These are not only difficult to find due to being uncommon, but are difficult

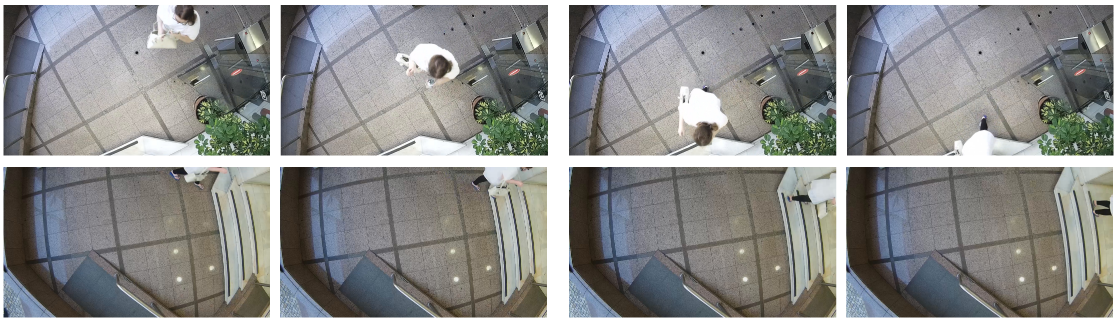


Figure 6.7: Common data sequence. People generally take the quickest way in from the turnstiles to the stairs and vice-versa.

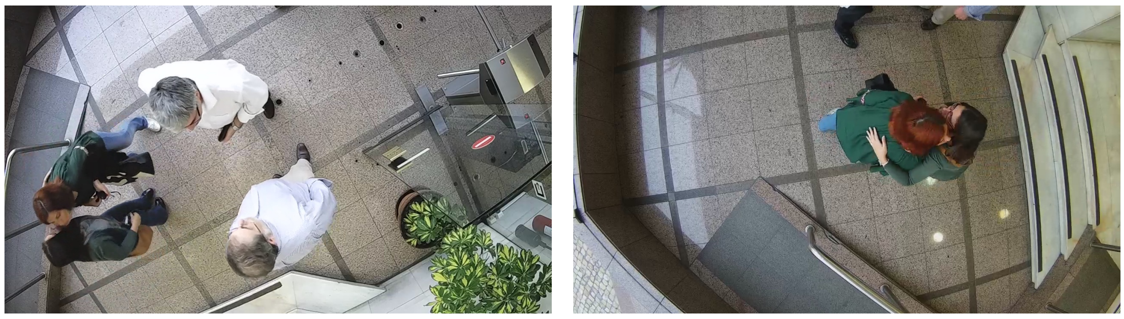


Figure 6.8: Example of data with people very close to each other from two different cameras taken at the same time.

to deal with label-wise, since if a bounding box is drawn around one person, it is going to end up having some fraction of the other person inside it –and this type of situation is challenging for the detectors. Firstly, to deal with the bounding boxes, two options are available: the first one is to draw the bounding box ignoring the fact that another person is in that bounding box as well, and the second is to try to draw the bounding boxes in a way that there is the minimum possible fraction of another person in that bounding box. The options that present fewer problems is the second one as if we start including two or more people in one bounding box, and if there are enough examples like this in the training data, the network may start to have problems discriminating between what is one person, and what is a set of people. As for the underrepresentation problem, since the data from the streams by nature has this uneven distribution, the answer lies in concisely trying look for the maximum possible number of these examples, possibly artificially staging them and employing data augmentation techniques (Perez & Wang, 2017) in order try to balance out the data to the maximum.

6.1.7 Final Datasets

After these processes, we now have three different datasets:

- A detection dataset, consistent of 41158 bounding box annotations, with an average of 1.55 annotations per frame;
- A tracking dataset, with 4660 bounding box annotations, 147 person IDS, an average of 40 annotations per ID and an average of 1.7 people in each frame;
- An appearance dataset, with a total of 294 person IDS, with an average of 114 frame crop samples per ID.

6.2 Performance Evaluation

Over the next sections, we present the results of the evaluation of the employed algorithms. We discuss the used methodologies and present the results for each of the different types of model separately.

6.2.1 Hardware Specifications

Since training CNNs is a computationally heavy process, witch resorts to many large-scale matrix operations, it is remarkably hard to complete the training using only a CPU. For this reason we built, trained and evaluated all the models using a NVIDIA GeForce GTX 1080 Ti GPU, 16GB of RAM and an Intel Core i5-7600K CPU provided by INOV-INESC.

6.2.2 Object Detection Performance

In order to evaluate the detectors, we employ the Mean Average Precision (mAP) metric which is widely accepted as a standard evaluation metric for object detectors (Everingham, Gool, Williams, & Winn, 2010; Russakovsky et al., 2015). This value is computed by calculating Average Precision for each class and then computing the mean across classes. As we only work with one class ("Person") the mAP values will be equal to the AP values. This value tells us how good are the detections produced by the models. Specifically, the predictions produced by the model on the test set are compared to the ground truth annotations, and intuitively, if a ground truth bounding box is highly overlapped with a prediction, this prediction is good. In order to decide whether the produced predictions are good or not, firstly, the following values are computed:

- True Positives (TP): This value is incremented every time a bounding box highly overlaps with the ground truth - generally an IOU value of 0.5 is considered;
- False Positives (FP): This value increments each time a produced bounding box has an IOU of less than 0.5 with its corresponding ground-truth, or if this the ground truth has already been associated to another detection;
- False Negatives (FN): This value increments every time the method fails to produce a bounding box that is in the ground truth.

With these values –model bounding box predictions, respective confidence scores, and correspondence TP, FP or FN classification– an ordered list is built which is used in order to compute precision and recall. Precision an recall give us good intuitions about the model behavior: if a model has high precision, then there is a high chance that bounding box predictions by that model are correct. If the recall score is high, on the other hand, then there is a high chance that all objects in the dataset are correctly detected. More specifically, we compute the raw precision-recall curve as well as the final interpolated precision-recall curves. The final mAP value is computed from these curves[1].

Recall from Chapter 4 that we employ and two main detection architectures: an SSD detector with an Inception-V2 feature extractor and a Faster R-CNN detector with a ResNet101 feature extractor. Following the work in (J. Huang et al., 2017), we know that the Faster R-CNN model is noticeably slower than the SSD model. As we would like to maintain the highest mAP possible with the highest possible speed, we proceed to instigate how the Faster R-CNN model can be modified in order to achieve speeds similar to the SSD model while maintaining accuracy. For this reason, the next tests involve adjusting the parameters in the Faster-RCNN Resnet model in order to examine how these impact the results. As the goal is to maintain accuracy, even when objects of interest are very close together in the field of view, as well as speed up the model inference process, the two main parameters that can be tweaked to modify the results, and that can, therefore, have a greater impact on the model behavior are the following:

Number of candidate region proposals: The Faster R-CNN model has a subcomponent responsible for generating region proposals where objects might be located. This is called the region proposal network (RPN), and it employs a combination of default anchor boxes at different scales and sizes in order to search the image for these objects of interest. Intuitively, the greater the number of proposals, the longer the network will need to compute them. Similarly, the greater the number of proposals, the more accurate the results will be, as we are covering a larger area in the image with the bounding box hypothesis. The original Faster R-CNN Resnet model employs 300 proposals. The work in (J. Huang et al., 2017) noted that this value can be lowered in order to speed up the models without compromising mAP results. As so, we experiment with a few different numbers of proposals in order to decide what works best for our dataset: we train models with 300, 100 and 20 candidate region proposals.

[1]`https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3`

Model Code	Num Proposals	First IOU Threshold	Second IOU Threshold
F-RCNN1	300	0.7	0.6
F-RCNN2	100	0.7	0.6
F-RCNN3	20	0.7	0.6
F-RCNN4	300	0.4	0.3
F-RCNN5	300	0.3	0.2

Table 6.1: Faster R-CNN Resnet 101 parameter variations.

NMS threshold: After candidate regions are proposed, the Non-Maximum Suppression (NMS) step is responsible for merging together all detections that belong to the same object, in order to suppress the lowest scoring detections and avoid False Positives. The NMS algorithm takes detections with the highest scores and suppresses the neighbor detections with lower scores. To do so, it calculates the IOU between the highest scoring detection and its lower scoring neighbors. If this IOU value is greater than the considered threshold, the lowest scoring neighbor detection is removed. As so, this step can heavily influence the model behavior on very close, overlapping objects, and yield only one bounding box where there are two objects close together. We want to understand what is the impact of lowering the NMS threshold value, as so, we used the best of the previous models –Faster R-CNN Resnet– and, in addition to the our base Faster R-CNN Resnet model, which uses an first-stage IOU threshold of 0.7 and a second-stage IOU threshold of 0.6, we trained two additional Faster R-CNN Resnet models: one with a first-stage NMS threshold of 0.4 and second-stage of 0.3, and the second with a first-stage NMS threshold of 0.3 and second-stage threshold of 0.2.

In sum, we evaluate 5 different variations of the Faster R-CNN Resnet model and compare it to the default SSD model to in order to investigate if any of these can still achieve better accuracy at a higher frame rate on our test datasets. These models are summarized in Table 6.1.

Detection Results

We begin by evaluating all the models on the **test set dataset**. The full detection dataset has a total of 41158 examples, which was split into train and test sets.

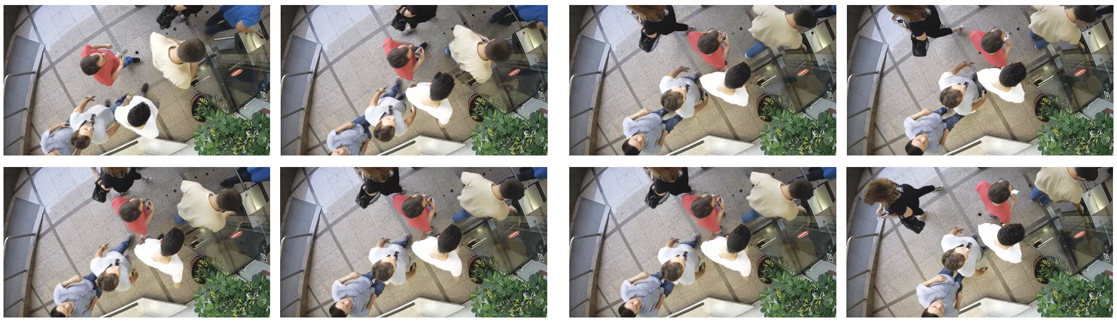


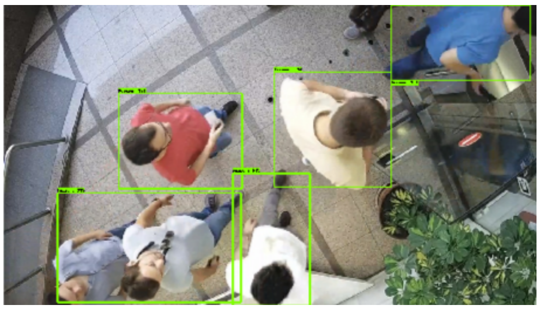
Figure 6.9: Additional sequence used to evaluate the detectors.

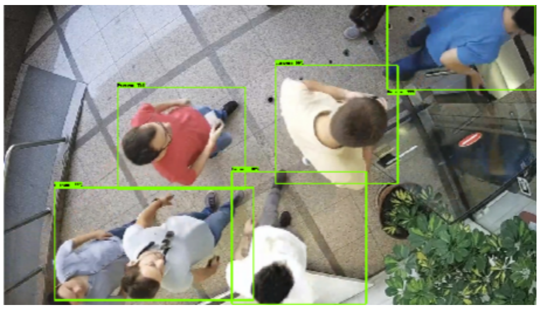
Model	step	mAP Test set	mAP sequence	FPS	Avg time Frame
SSD	187.1k	0.9856	0.9868	80.2482	0.0124
F-RCNN1	338.9k	0.9889	0.9910	9.6100	0.1040
F-RCNN2	1.07M	0.9263	0.8877	14.8107	0.0675
F-RCNN3	217.4k	0.9795	0.9771	30.0744	0.0523
F-RCNN4	386.7k	0.9832	0.9880	9.3418	0.1070
F-RCNN5	642.1k	0.9827	0.9904	8.9970	0.1111

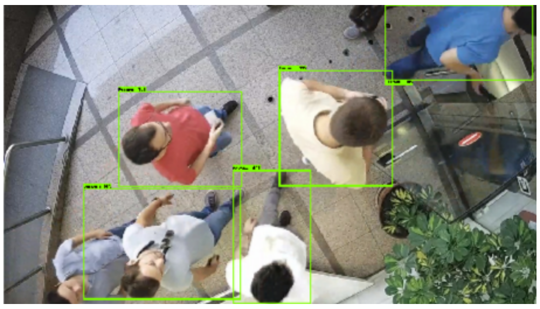
Table 6.2: Detection mAP results on the test set and additional sequence with multiple overlapping targets.

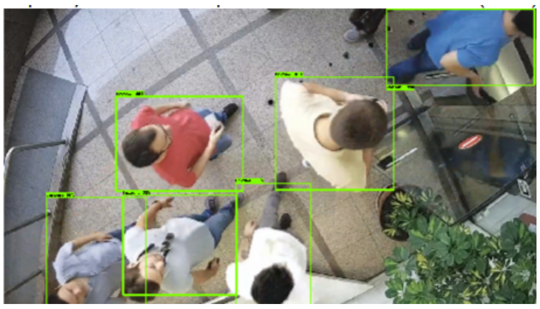
All the models were trained on the same train set of 32999 examples and are now evaluated on the test set of 8159, which was sampled at random from the full dataset. As we also needed to better evaluate how the models behave in the specific scenario when, multiple people, close to each, other cross the field of view, **we build an additional test sequence**. This sequence is formed by 797 annotated frames and is representative of one of the hardest cases for the detectors: the case where 8 people simultaneously cross the field of view, and therefore, multiple people are overlapped with each other. A fragment of this sequence is represented in Figure 6.9. For the object detectors used in the present work, these types of sequences are hard due to the Non-Maximum Suppression (NMS) step, which can heavily influence the model behavior on very close, overlapping objects. We present the results in Table 6.2.

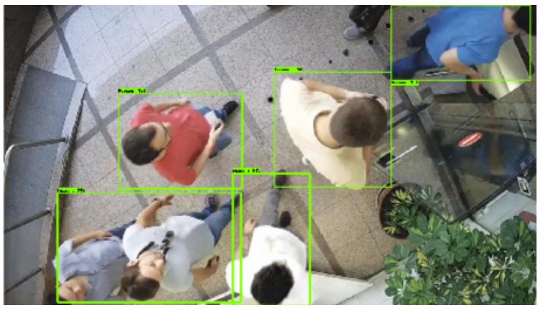
Analyzing the results, we conclude that the most accurate model, with 0.991% mAP, is still the F-RCNN1 –300 proposals and 0.7 and 0.6 IOU thresholds– however this is the second slowest model with 9.6 FPS. We are able to speed up this model from 9.6 FPS to 14.8 FPS and 30 FPS by using, respectively, 100 and 20


(a) F-RCNN1 (300 proposals, 0.7 & 0.6 IOU NMS thresholds.


(b) F-RCNN2 (100 proposals, 0.7 & 0.6 IOU NMS thresholds.


(c) F-RCNN3 (20 proposals, 0.7 & 0.6 IOU NMS thresholds.


(d) F-RCNN4 (300 proposals, 0.4 & 0.3 IOU NMS thresholds.


(e) F-RCNN5 (300 proposals, 0.3 & 0.2 IOU NMS thresholds.

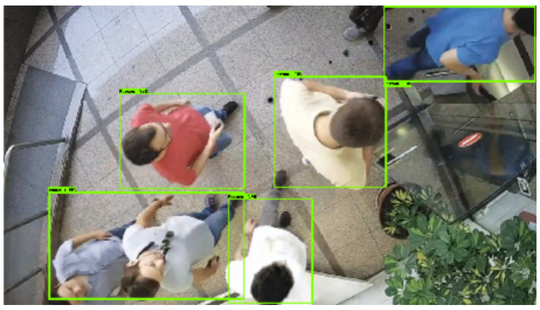

(f) SSD Inception.

Figure 6.10: Examples of different detector predictions on the same frame.

proposals in the F-RCNN2 and F-RCNN3 models, instead of the original 300 while maintaining the 0.7 and 0.6 IOU NMS thresholds. This, however, degrades mAP from 0.991 to 0.88 and 0.97 respectively. We also conclude that, while lowering the NMS IOU threshold in the models that use, respectively, 0.4 and 0.3, and 0.3, 0.2 NMS IOU thresholds (F-RCNN4 and F-RCNN5), achieves better mAP than lowering the number of proposals, it degrades speed, lowering it from the original 9.6 FPS to 9.3 and 8.9 FPS respectively. The fastest model is still the Inception SSD model, which achieves 80 FPS, and is the 4th most accurate with 0.986, surpassed by F-RCNN1 model with 0.991 mAP (300 proposals and 0.7 and 0.6 IOU thresholds), F-RCNN5 model with 0.990 mAP (300 proposals, 0.3 and 0.2 IOU thresholds), and the F-RCNN4 model with 0.988 mAP (300 proposals, 0.4 and 0.3 IOU thresholds). In sum, despite not being able to speed up the Faster R-CNN Resnet model to Inception SSD speeds and maintain mAP, we conclude that lowering the number of proposals speeds up the model and also lowers mAP: lowering the number of proposals improves the speed but degrades mAP. An example of an inference of each of the models is represented in Figure 6.10. In this Figure we can see, that, despite the models with higher IOU NMS thresholds achieving higher overall mAP, the models with lower NMS IOU thresholds do better with very close overlapping objects, specifically the F-RCNN4 model. Our conclusion with these tests is that the choice of model, and of model parameters should be aligned with the concrete examples of the real data the model will be used on. In the case of our datasets, despite having some overlapping objects, most cases are not like this. As so, using a model that does better specifically on these cases, which represent a smaller percentage, will damage model performance on the most cases with non-overlapping objects.

6.2.3 Appearance Model Performance

As we train the appearance model in two stages the model evaluation is also performed in two stages. The first stage corresponds pre-training the model on the MARS dataset (Zheng et al., 2016), which contains 1500 identities, as it would be less effective to train on our smaller dataset –316 identities– from scratch. The second corresponds to the fine-tuning of this model on our own appearance dataset.

We pre-train the model on the MARS dataset for about 160000 iterations, until the loss and classification accuracy converge. Then we proceed to re-train the same model on our dataset. The evolution of the classification accuracy and cross entropy loss during training is represented in Figure 6.11 and the final values achieved by the models are presented on Table 6.3.

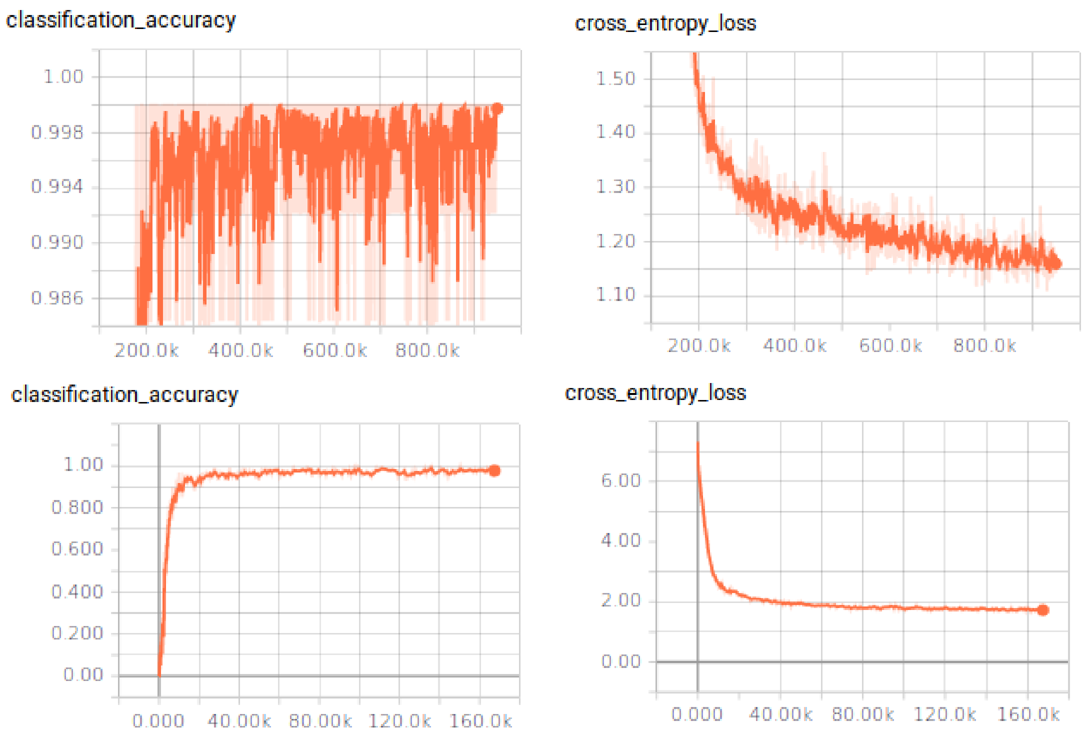


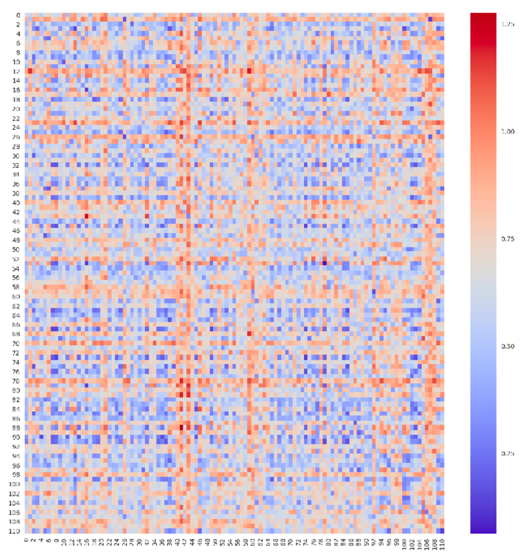
Figure 6.11: Appearance model training. The top row plots correspond to the classification accuracy and cross entropy loss of the process of re-training the model on the custom INOV dataset. Bottom row plots correspond to the classification accuracy and cross entropy loss of the process of pre-training the model on the MARS dataset.

Model	Test Classification Accuracy	Cross Entropy Loss
MARS pre-training	0.977	1.732
INOV re-training	0.999	1.143

Table 6.3: Final values of classification accuracy on the test set and cross entropy loss while training on the MARS and re-training on the INOV datasets.

The classification test set accuracy values are seemingly good. From these values, we could assume that the models are almost perfect. However, we want to further explore two intuitions: firstly, we want to evaluate the effect more training and more training data on the model performance, and secondly, we want to validate the intuition that one sample belonging to one identity taken from one camera, will always be more similar to another sample from the same identity in another camera, than any other sample from any other identity in the dataset. This is important as the appearance model is heavily used in order to link identities between cameras when performing inter-camera tracking. As so, the model should yield higher similarities for crops belonging to the same object, even if they are taken by different cameras. In order to get a deeper understanding of these issues, firstly, we created 110 new identities, with crops from both cameras to test on. Secondly, we select random pairs of crops belonging to the same identity from both cameras –one sample in camera 1 and another in camera 2– for every identity in the new dataset. Thirdly, we feed those crops to the network in order to produce the features. And finally, we compute the cosine distance between all possible pairs of features, which produces a similarity matrix, which we represent as a heat map. Ideally, the diagonal of this heat map should be more pronounced since the diagonal corresponds to the distance between crops from the same identity, and therefore we expect this distance to be considerably lower for the crops belonging to the same objects. We perform this test four times:

- Firstly we take the model trained solely on the MARS dataset and compare the cosine distance between the produced features for all identities taken from our cameras. We do so to analyze how well the model trained solely on the MARS dataset can generalize to data similar our own appearance dataset;
- Secondly, we take the fine-tuned model on our appearance dataset and re-run the same test. We perform this test in order to evaluate the effect of training data in the model predictions and to validate the need for a custom dataset to train the model on, instead of training on a pre-existing dataset such as MARS. Intuitively, the distance between features produced using the same identities should be smaller, and the distance between features from different identities should be higher;
- Next we perform the same test using a randomly sampled fraction from our train dataset instead of the new identities. Firstly we take the model trained solely on the MARS dataset and run this test;


(a)

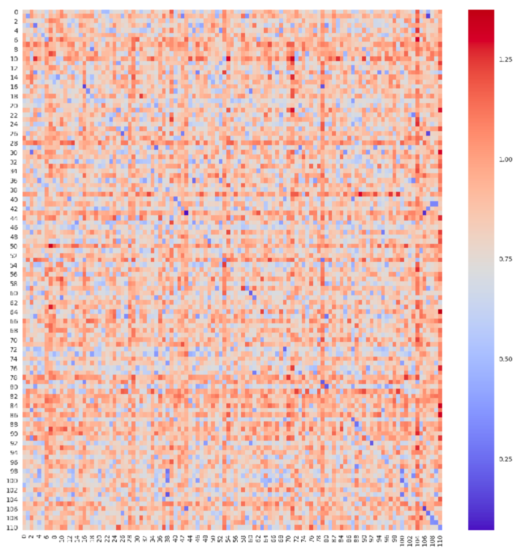
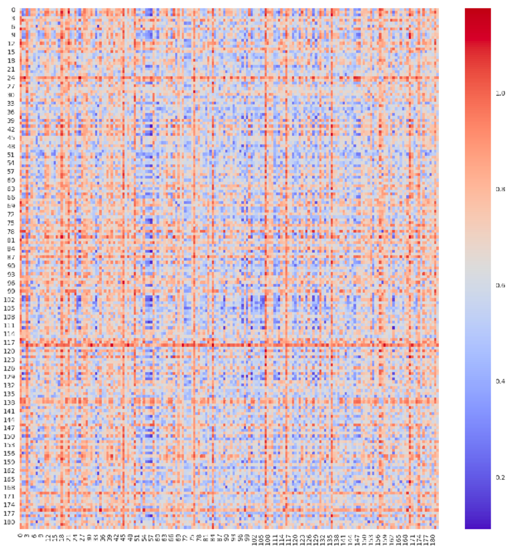

(b)

Figure 6.12: (a) Cosine similarity between features produced by the appearance model trained solely on the MARS dataset. (b) Cosine similarity between features produced by the appearance model pre-trained on the MARS dataset and re-trained on our dataset. The bluer cells correspond to more similar features, and red cells correspond to more dissimilar features.

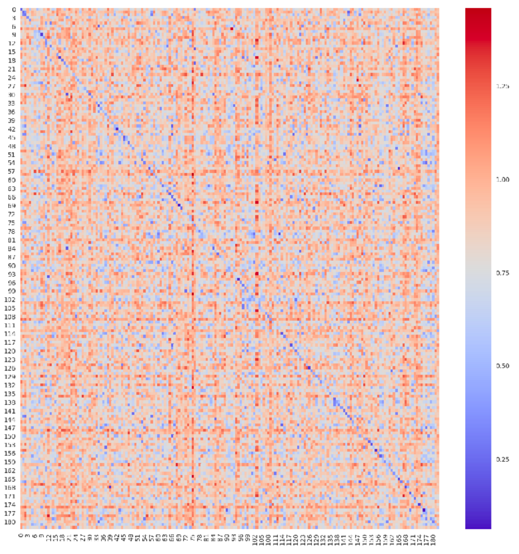
- Finally we validate how well the trained model learned the train data. As so, we perform the same test as the above using the trained model and a fraction of the train data.

The heat maps produced by the first two tests are represented in Figure 6.12. Analyzing both heatmaps, the results are close to what is expected. The model trained solely on the MARS dataset produces much more random results than the model that was re-trained on our dataset. The retrained model generally yields higher distance for different identities, and smaller distance for the same, as we can observe a more pronounced blue diagonal.

We represent the heat map produced by the trained model on the train data in Figure 6.13. Here, as expected, the heat map diagonal produced by the trained model on the train INOV data is much more pronounced and noticeable than for the previous heatmaps. As so, we can conclude that the model successfully learned the train data, and is adequate enough to generalize to previously unseen data. However, the diagonal produced by the trained model on unseen data (Figure 6.12



(a)



(b)

Figure 6.13: (a) Cosine similarity between features produced by the appearance model on the inov train set trained solely on the MARS dataset. (b) Cosine similarity between features produced on the inov train data by the appearance model pre-trained on the MARS dataset and re-trained on our dataset. The bluer cells correspond to more similar features, and red cells correspond to more dissimilar features.

(b)) could be much more pronounced. This leads us to conclude that, despite being on the right track, the model needs to be trained on more data in order to produce more accurate results.

6.2.4 Tracker Performance

MOT Evaluation Metrics

Objectively evaluating a Multiple Object Tracker, in comparison to other visual recognition tasks, is not straightforward for several reasons (Milan, Schindler, & Roth, 2013). Firstly, opinions may differ when accounting for what the ground truth should look like. As an example, imagine a scene with a great number of people, many of which are occluded. This type of scenario will inevitably lead to ambiguities in the ground truth annotations. For instance, the TUD-Stadtmitte dataset (Andriluka, Roth, & Schiele, 2010) has many different ground truth solutions available, and they are all different from one another (Milan et al., 2013). Another difficulty in evaluating and benchmarking trackers is that, in many tracking datasets, partially occluded people are simply ignored, as is the case with the TUD-Stadtmitte dataset (Andriluka et al., 2010) or the ETHMS dataset (Ess, Leibe, Schindler, & Van Gool, 2008). Therefore, when considering the ability of the tracker to track through, and recover from occlusions, the results on these datasets do not provide an accurate result.

In the present work we have an additional challenge. Recall that we developed the present work using our own custom dataset. As so, all the algorithms are trained on our own custom data, and therefore will not produce accurate results on popular benchmarks such as the MOT challenge (Milan, Leal-Taixe, et al., 2016), which provide convenient detailed instructions on how to evaluate the trackers. Therefore we have to create custom evaluation sequences and ground truths.

The main problem is finding the best alternative to compare the tracker output with the ground truth annotations. In the context of other visual recognition tasks, for instance, when evaluating an image classifier or object detector, we can measure the similarity of the model prediction and the ground truth in all frames independently, as the result in one frame does not depend on the result of other frames. In MOT, however, in order to accurately evaluate the tracker, this process has to be performed over all frames, as we are testing the ability that the trackers have to track through many frames. To do so, and in order to compute a basic evaluation, in addition to the ground truth annotations and the tracker predictions, we also need to define the distance used to compare annotations and predictions. The distance function may incorporate many components so that the evaluation is representative of different kinds of error that the tracker may yield.

Many different tracking evaluation approaches have been proposed over the years (Stiefelhagen et al., 2006; Schuhmacher, Vo, & Vo, 2008; Yuan, Huang, & Nevatia, 2009). In the present work, we use the popular and widely accepted MOT evaluation approach described in (Leal-Taixé et al., 2015) which is based on the CLEAR MOT evaluation described in (Bernardin & Stiefelhagen, 2008). There are two main metrics: MOTA (Multiple Object Tracking Accuracy) and MOTP (Mulitple

Object Tracking Precision). Both metrics encompass different error types and are therefore a good representation of the overall performance shown by the trackers:

MOTA: The Multiple Object Tracking Accuracy combines three types of tracking errors: false positives (FP), false negatives (FN) and identity switches. A false positive occurs when the tracker produces a result that has no match in the ground truth annotation. A false negative, on the other hand, happens when a tracker fails to produce a result for an existing annotation. In order to decide whether or not a output by the tracker corresponds to a ground truth annotation, a distance metric is used. The most widely accepted is the Intersection over Union (IOU) of the ground truth bounding box with the predicted bounding box. Usually, a threshold of 0.5 is employed: if the intersection is greater or equal to the threshold, then the correspondence is valid. An identity switch or a mismatch happens when the tracker switches the identity of a tracked object. This may also happen when an object is mistakenly given a new identifier: for instance, when two objects come close to each other and the tracker swaps their identities. In order to count these errors a mapping from ground truth annotations to tracker predictions is built, and every time this mapping is contradicted, a new identity switch error is counted. If this error happens, the mapping is updated for the next iterations. Then the final MOTA value computed over all ground truth objects in all frames for a sequence is given by:

$$MOTA = 1 - \frac{\sum_t (fn_t + fp_t + i_t)}{\sum_t gt_t}, \tag{6.1}$$

where fn_t, fp_t, m_t and gt_t are the number of false negatives, false positives, identity switches (mismatches) and ground truth objects at time t, respectively.

MOTP: The Multiple Object Tracking Precision describes the capacity that the tracker has to predict accurate object locations independently of its capability to maintain identifiers. This can be seen as the average distance over all matched ground truth/tracker prediction pairs, given by:

$$MOTP = \frac{\sum_{t,i} d(gt_i^t, p_i^t)}{\sum_t m_t}, \tag{6.2}$$

where $d(gt_i^t, p_i^t)$ represented the distance between the ground truth and the predicted location for that ground truth, and m_t represents the total number of matches at time t.

Tracker Quality Metrics: We employ additional ways to measure the quality of the trackers in comparison to the ground truth. Each track can be labeled as one of three categories: Mostly tracked (MT), Partially Tracked (PT), and Mostly Lost (ML) tracks. Additionally, we count the number of Fragmentations (FM). These metrics describe how close each ground truth trajectory is followed by the trackers.

- MT: The Mostly Tracked measure describes the number of ground-truth trajectories correctly covered by the tracker for at least 80% of their lifespan;
- ML: The Mostly Lost measure describes the number of ground-truth trajectories that are covered by the tracker for at most 20% of their lifespan;
- PT: The Partially Tracked measure describes the number ground-truth trajectories correctly covered by the tracker between 20% and 80% percent their lifespan (The number of tracks that are neither MT nor ML);
- FM: The number of Fragmentations is incremented each time the track of a ground-truth is interrupted. In other words, the number of gaps in the tracker prediction compared to the ground truth.

Precision and Recall: Additionally, we report precision and recall in the evaluation. Recall corresponds to the number of detected objects over the ground truth number of objects. Precision corresponds to the number of detected objects over the sum of the number of false positives and the number of detected objects. Since the object identifiers are important, we also report ID Precision and Recall metrics:

- IDP (ID-Precision): Corresponds to the ratio of tracker detection predictions that are correctly identified;
- IDR (ID-Recall): Corresponds to the ratio ground truth detections that are correctly identified;
- IDF1 (ID-F1 score): Corresponds to the ratio of correctly identified detections over the average of ground-truth and tracker-produced detections.

Evaluation method

We evaluate our trackers on 17 tracking sequences captured from two cameras at a 10 FPS frame rate – 34 sequences in total between both cameras. These sequences were captured over the course of two weeks, and hand-picked in order to include both simple and hard examples, that we consider are representative of most of the situations that occur daily. We manually generated tracking ground truth for each of the sequences in order for the evaluation to be the most correct possible. We run the evaluation on all 17 sequences simultaneously, which were concatenated in order to better understand how the trackers behave over time, and how errors can propagate from sequence to sequence. We run this evaluation for each camera separately and then calculate the average. Recall that we evaluate three different types of tracker:

- The IOU tracker, which uses a simple IOU association of detection bounding boxes and tracklet bounding boxes. We evaluate this tracker with two different IOU thresholds: 0.2 and 0.5;
- The GOTURN tracker, with there different variations for the data association step: IOU intersection threshold, appearance similarity score, and a weighted sum of both appearance and IOU overlap. For the GOTURN IOU overlap tracker, we evaluate two IOU thresholds: 0.2 and 0.5;
- The deep sort tracker, which employs a combination of Kalman Filter and appearance modeling to solve the data association step.

We do not provide the ground truth detection bounding boxes for the tracker, as is done in different tracking challenges (Leal-Taixé et al., 2015), since our goal is to evaluate the overall system performance. We want the detection to be a part of the pipeline as it provides a more realistic representation of the type of behavior that the system will demonstrate when it runs on real data: when the system runs on real data there will be no ground truth annotations, only the results from the detection. As so, instead of providing ground truth detection annotations, we have the detection model solve the detection step and provide the bounding boxes to the trackers. Recall from the detection evaluation section, that the Faster-RCNN Resnet 101 model with 300 proposals and 0.7 and 0.6 IOU thresholds achieves the best mAP value, while the Inception SSD model achieves the highest speed while maintaining a pretty high mAP value –not the best, however. We, therefore, run the tracker evaluation for all tracking models two times: one for the Faster R-CNN Resnet 101 model and another for the Inception SSD model. This will allow us to better understand if the mAP value achieved by the SSD model is high enough to produce good tracking results, or if we have to maintain the Faster R-CNN model, which has great mAP but slower speed.

In order to evaluate the trackers, we employ the py-motmetrics package [2], which provides a large variety of evaluation metrics, compatible with the metrics used in the MOTChallenge.

Tracking Results

Tracking evaluation results are represented in Table 6.4. Firstly, we analyze the most descriptive metrics: MOTA and MOTP. Both top scoring trackers with these metrics, utilize, non-surprisingly the Faster R-CNN Resnet 101 detector. The highest MOTA of 86.7% is achieved by the GOTURN tracker that employs an IOU overlap association metric of 0.2. The highest MOTP value belongs to the GOTURN tracker that employs a weighted sum of both appearance features and IOU bounding box overlap with 86.7%. However, examining the FPS value, neither

[2] `https://github.com/cheind/py-motmetrics`

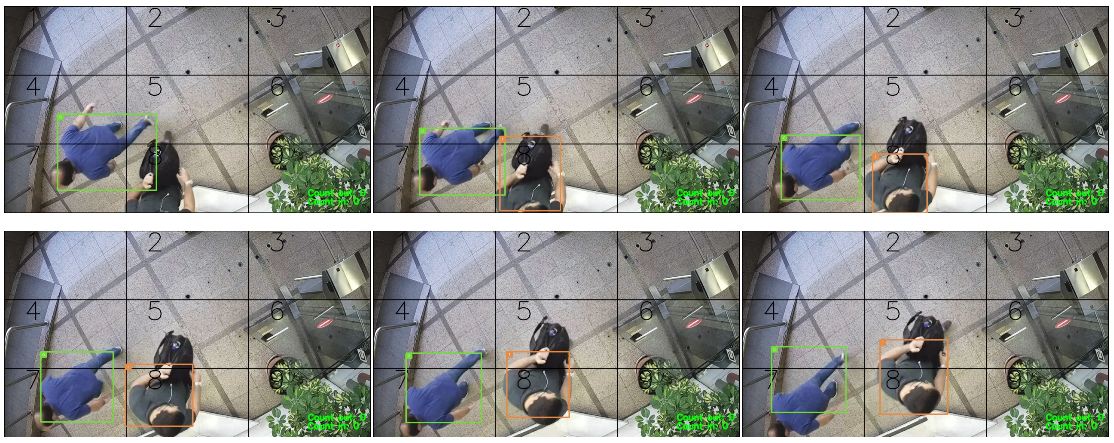


Figure 6.14: Tracking sequence example.

	IDF1	IDP	IDR	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDsw	FM	MOTA	MOTP	Hz
goturn-iou-02-ssd	84.6%	84.1%	84.7%	92.2%	91.8%	120	101	18	1	532	506	79	116	82.8%	86.018%	42.554
goturn-iou-05-ssd	54.2%	53.9%	54.3%	91.9%	91.7%	120	100	19	1	541	523	938	127	69.2%	86.015%	41.972
goturn-appearance-05-ssd	76.5%	77.2%	75.5%	89.9%	**92.2%**	120	91	27	2	**493**	657	163	170	79.8%	85.961%	38.519
goturn-ws-05-ssd	79.6%	79.2%	79.8%	92.2%	91.8%	120	102	17	1	532	505	184	118	81.2%	85.993%	39.388
deep-sort-ssd	85.2%	87.2%	83.1%	86.9%	91.6%	120	72	46	2	519	851	59	92	78.0%	80.131%	50.553
iou-05-ssd	57.5%	57.1%	57.8%	92.6%	91.7%	120	102	17	1	546	483	952	105	69.5%	85.997%	**55.694**
goturn-iou-02-faster-rcnn	89.3%	86.3%	**92.1%**	97.1%	91.3%	120	**115**	4	1	602	187	76	50	**86.7%**	88.475%	7.036
goturn-ws-05-faster-rcnn	82.8%	80.0%	85.4%	97.1%	91.3%	120	**115**	4	1	603	188	203	52	84.7%	**88.468%**	6.812
goturn-appearance-05-faster-rcnn	81.5%	79.7%	83.0%	95.1%	91.6%	120	106	13	1	564	317	160	96	84.0%	88.462%	6.734
deep-sort-faster-rcnn	**91.1%**	**89.9%**	92.0%	92.9%	91.1%	120	103	16	1	592	460	**37**	60	83.2%	80.881%	7.109
iou-02-faster-rcnn	53.2%	51.3%	55.0%	**97.4%**	91.2%	120	**115**	4	1	610	**167**	1057	**39**	71.7%	88.462%	7.519
iou-05-faster-rcnn	51.7%	49.9%	53.5%	**97.4%**	91.2%	120	**115**	4	1	610	**167**	1115	**39**	70.8%	88.462%	7.656

Table 6.4: Single camera tracking evaluation. The top half corresponds to the tracking results retried using an Inception V2 SSD detector. The bottom half corresponds to the tracking results retrieved using a Faster R-CNN Resnet 101, with 300 proposals, first IOU NMS threshold of 0.7 and second IOU threshold of 0.6.

of the trackers that employ the Faster R-CNN Resnet 101 detector achieve an FPS rate higher than 10 FPS, with the fastest tracker being the basic IOU overlap tracker with 7.656 FPS. Recall that the that the Faster R-CNN detector works at a 9.6 FPS rate, which means that most of the time is spent on the detection. As the input videos have a frame rate of 10 FPS, this automatically disqualifies all the Faster R-CNN Resnet 101 based trackers from being classified as real-time. A few examples of different tracker predictions on the same frame sequence are represented in Figures 6.16 to 6.18. The ground truth annotations for this sequence are represented in Figure 6.15. In these sequences, we can observe that the GOTURN tracker with an IOU affinity metric is the closest to the ground truth identities. The GOTURN tracker with an appearance affinity metric loses one of the tracked identities in later frames, while the deep sort tracker suffers from an identity switch during the sequence.

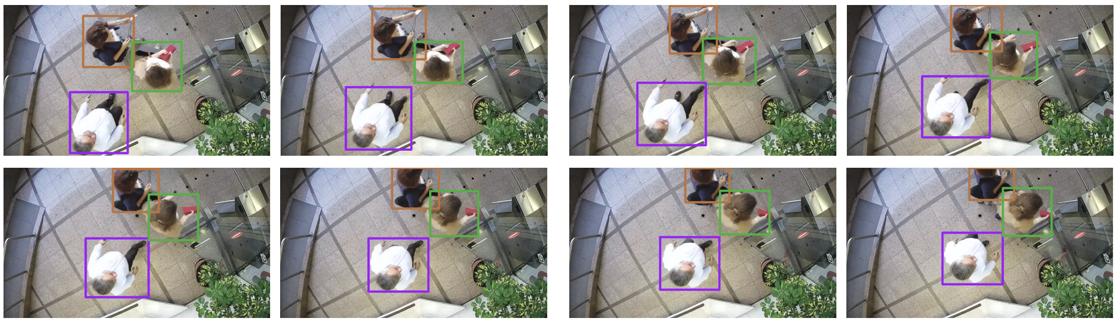


Figure 6.15: Ground truth identities.

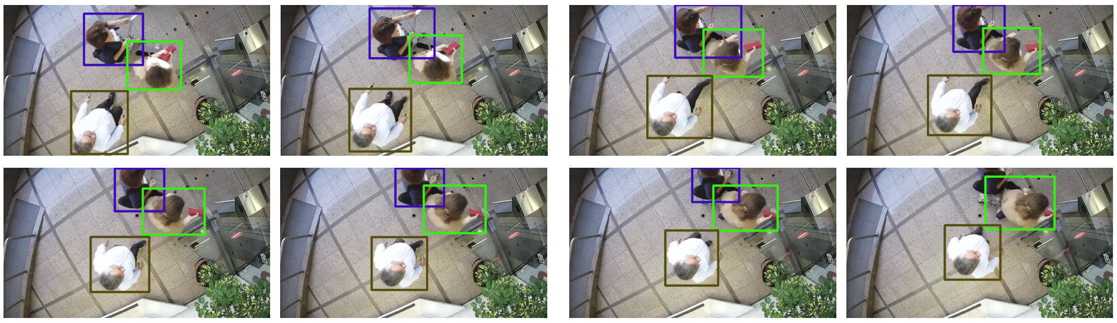


Figure 6.16: Tracking sequence example– Predictions made by the GOTURN tracker with an IOU Affinity Measure.

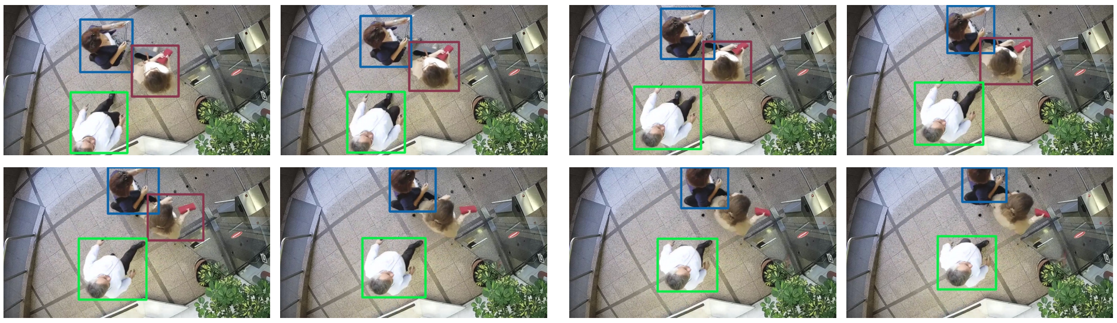


Figure 6.17: Tracking sequence example– Predictions made by the GOTURN tracker with an appearance feature distance affinity measure.

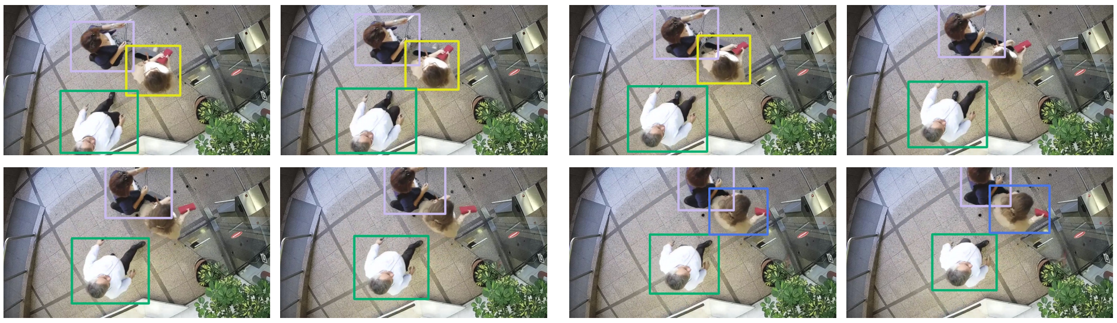


Figure 6.18: Tracking sequence example–Predictions made by the Deep Sort tracker.

6.2.5 Multi-Camera Tracking Performance

In this section, we evaluate the performance of the two multi-camera tracking algorithms. Recall that we have two algorithms to evaluate: the single shot association algorithm and the adaptable distance matrix algorithm. Our goal is to have the multi-camera tracking algorithms track objects smoothly through all of the camera views, similarly to the single camera tracking algorithms. however, in this scenario, the object identities should be extended through various fields of view. In order to evaluate the multi-camera algorithms, we employ an evaluation scheme similar to the single camera tracking algorithms.

Firstly, we employ the same 17 tracking sequences, which ground truth we adapted for the multi-camera context. In other words, to scale these sequences for multi-camera evaluation, we mapped every ID in one camera to every ID in another, which resulted in one ground truth file for both camera views. This is possible as at sequence capture time, we retrieved the PTS for each frame, and for this reason, we are able to synchronize the sequences of both cameras in time and map the IDs between cameras. The resulting ground truth is one single file for both cameras, which contains the ground truth object IDs and bounding boxes for each frame.

Secondly, we have the single camera tracking algorithms generate predictions and send it to the multi-camera tracking module. More specifically, we employ the detector/tracker combination which achieved the best results in real-time: we employ the SSD detector with the GOTURN Tracker with IOU association metric. We then have both multi-camera tracking algorithms generate predictions using these single camera tracking results, and write the predictions to a file –we use the same file for both cameras.

Lastly, we compare both files –ground truth and the generated predictions– using the py-motmetrics package.

The results of the evaluation of both algorithms are presented in Table 6.5. Firstly, notice that these results are much worse all-around when compared to the single

	IDF1	IDP	IDR	Rcll	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP
single shot	61.2%	62.0%	59.5%	85.3%	90.3%	65	46	18	1	595	954	880	297	62.6%	80.351%
matrix	63.2%	61.5%	65.4%	90.7%	84.6%	65	55	10	0	1069	605	1015	278	58.6%	80.395%

Table 6.5: Multiple camera tracking evaluation.

camera tracking results. This is expected, as these algorithms run on top of the previous and are given the already not perfect results of the single camera tracking modules to work with. Additionally, this task is one order of magnitude more complicated than the previous, as here the results of both cameras are considered simultaneously –we are not averaging the results of each camera–, and therefore ground truth tracks are longer and have a wider motion range, which means that the global IDs have to be sustained for a longer period of time.

The one shot association algorithm noticeably outperforms the adaptable distance matrix algorithm on most the metrics. Namely, the MOTA result is 4% higher for the one-shot association algorithm. We can explain this difference using other three metrics: recall that this metric encompasses the False Positive, False Negative and Identity switches. Note that the FP count for the adaptable distance matrix algorithm is nearly double the FP count of the one-shot association algorithm. Additionally, ID switches are also much higher for the adaptable distance matrix. We infer that this is due to the ability for the adaptable matrix to constantly correct itself and switch the global object IDs. This self-correction algorithm is ultimately detrimental for the evaluation, which benefits low false positive and identity switch counts. Interestingly, this self-correction mechanism is beneficial for other metrics such as IDR, which correspond to the ratio of ground truth detections that are correctly identified and is 5% greater for the distance matrix algorithm. As so, this leads us to conclude that despite the constant identity switches in order to perform corrections, once this process is stable and the feature vectors are more descriptive of each object, the adaptable distance matrix ultimately correctly identifies a larger amount of IDS than the single shot association algorithm. This is desirable since our goal is not only to have the trackers track the objects correctly, but also correctly identify them.

Chapter 7

Conclusions

The rapid advancements in computer vision techniques that we have witnessed in the past years, combined with continuous improvements in hardware and software, allowed us to develop a real-time, multiple camera person detection and tracking system prototype. Specifically, the objectives in the present work were to investigate, develop and evaluate solutions for the multiple person tracking problem, using static, overlapped, fish-eye top view cameras. In order to solve this problem, we made use of a variety of techniques. Namely, we used a combination of the representational power of deep neural networks and more classical tracking algorithms in order to represent and track the target objects. We experimented with different architectures and techniques which allowed us to get a good understanding of the current computer vision paradigms, as well as how the more classical solutions fit in these recent algorithms, and additionally to find solutions that better fit our specific problem.

7.1 Summary and Considerations

Firstly, we split our main problem –multiple camera, multiple person tracking– into two sub-problems: single camera tracking, which we tackle in Chapter 4, and multiple camera tracking which we discuss in Chapter 5. To solve the single camera tracking problem we employed the tracking-by-detection paradigm, which allowed us to track the target objects in each camera individually and independently of each other camera. In order to solve multiple camera tracking, the tracking results of each individual cameras are synchronized and connected in order to represent each object by their trajectory in all cameras.

As we have no benchmark, we wanted to implement various types of solutions in order to compare them and find which one best applied to our problem. We first researched and studied the current state-of-the-art multiple object tracking solutions in Chapter 3, from which we derived three main different solutions for the **single camera tracking** problem: a simple IOU tracker similar to (Bochinski et al., 2017), mainly to be used as a baseline, an adaptation of the deep learning based visual object tracker of (Held et al., 2016) which we scaled for multiple object tracking, and the tracker of (Wojke et al., 2018), which employs a combination of deep learning techniques to encode appearance information, Kalman filters to predict motion, and the Hungarian algorithm to solve the data association step. Additionally, as we employed the tracking-by-detection paradigm, we trained two different deep learning detection architectures: the Faster R-CNN detector (Ren et al., 2017), with a ResNet feature extractor (He et al., 2016) and a SSD detector (Liu et al., 2016) with a Inception-v2 feature extractor (Szegedy, Vanhoucke, et al., 2015). Furthermore, we proposed the grid algorithm, which helps the tracker correct itself and recover from errors such as occlusions or detection failures. Additionally, the work developed on some of the modules of Chapter 4 originated the work in (Baikova, Maia, Santos, Ferreira, & Oliveira, 2018), which, in addition to single camera multiple object tracking, also focused on a practical application of a system of this kind.

In order to solve the **multiple camera tracking** problem, we implemented processing a pipeline, from which we can highlight two steps. Firstly we performed the synchronization of the feeds from each individual camera in order for the single camera tracking information to be aligned in time. Secondly, we proposed two different algorithms in order to connect information between cameras: the **single shot association** algorithm, which connects objects between cameras only once when these objects transition to the overlapped area, and the **adaptable distance matrix** algorithm, which builds and updates a distance matrix where each element is the cosine distance between the features of objects in different cameras, at each iteration. For both algorithms we use both appearance and location cues: we employ the appearance model of (Wojke & Bewley, 2018), which we trained with our custom appearance dataset to produce appearance features, and we employ the grid originally used in the single camera tracking pipeline in order to solve occlusions and detection failures, and re-purpose it in order to roughly map the space from one camera to another, producing location information.

Finally, in as the majority of the developed methods were mostly based on data-driven algorithms, the existence of **large, varied, clean and balanced datasets** was one of the most important factors of success that could greatly impact the results. However, as we worked on data originated from top-view static camera footage, that had a field of view overlap, and since to our knowledge, there are no existing detection or tracking datasets that follow these requirements, we had to build all the datasets from scratch, using data from our test environment cameras. As so, we built three datasets: one person detection dataset, one person tracking dataset and one person appearance dataset. The process of building and refining these datasets is described in the first section of Chapter 6.

In the second section of Chapter 6 we evaluated every component. We started by evaluating the **detectors**: we compared the SSD detector with different variations of the Faster R-CNN detector. We performed these tests as we wanted to verify if the Faster R-CNN model could achieve SSD detection speeds without compromising accuracy. We conclude that this is still not possible solely by varying parameters such as NMS and number of proposals. Additionally, we conclude that if we want to use these detectors in real-time combined with our trackers, the Faster R-CNN is not an option as it is too slow.

Secondly, we evaluate the **appearance model** that was used in both single camera tracking and multiple camera tracking sub-problems. We conclude that, although the results are encouraging, the model needs to be trained with a greater quantity –and quality– of data.

In third place, we evaluate the **single camera trackers**, using detection bounding boxes produces by both SSD and Faster R-CNN detectors. Although we already have reached the conclusion that the Faster R-CNN detector could not be used in real-time, we still employed both detectors in order to understand the performance degradation of using a faster but slightly less accurate detector. Additionally, we tested different variations of the data association step for the GOTURN tracker: we employed an IOU threshold association, an appearance feature distance association and a weighted sum of both appearance and location. We evaluate these trackers on a custom build evaluation dataset of 17 different sequences, viewed from 2 different cameras. In conclusion, the GOTURN tracker with IOU threshold of 0.2 for the data association step outperforms every other tracker, followed by the GORURN variations with the weighted sum of appearance and IOU overlap, and appearance only, respectively. Furthermore, we conclude that using the SSD detector degrades the MOTA score by about 4.5%, however, it increases the overall system speed by about 500%. Overall, we conclude that the GOTURN tracker with an IOU overlap data association step that employs an Inception SSD detector is able to achieve a nice accuracy/speed balance, with an 86.018% MOTA score and 42.5 FPS on the evaluation sequences –which were captured at 10 FPS.

Finally, we proceeded to evaluate the **multiple camera trackers**. For this evaluation, we employed the previous evaluation sequences viewed from both cameras, in which we mapped the ground truth IDs to be consistent between cameras. We conclude that the single shot association outperforms the adaptable distance matrix on the majority of metrics, including MOTA score, which is 4% greater for the single shot association. However, the adaptable matrix outperforms the prior in metrics such as IDR, which corresponds to the ratio of ground truth bounding boxes correctly identified. As so, despite having a higher number of identity switches, due to constantly trying to correct its ID correspondence between cameras estimate, which greatly compromises the results of metrics such as MOTA, the adaptable distance matrix algorithm ultimately yields a higher number true identity hits over time. Additionally, we conclude that using a better appearance model, trained with greater amounts of data, will return better results, as most of the errors are caused by appearance mismatches.

In sum, analyzing the overall results, over the present work, we have developed a functional and scalable real-time multiple person detection and tracking system prototype. This prototype already yields promising results, and, as this is an ongoing work, which will continue on beyond the present work, overtime should improve as we create and feed the models with new and better data, and as we continuously single out and fix possible future error situations.

7.2 Limitations

Despite the encouraging results, there are also limitations to the trackers. To begin with, the single-camera tracking frequently suffers from a set of errors. In order to better understand these errors, and how we can correct them in the future, we manually analyze each frame of each of the validation sequences and report the most prominent and frequent below:

Tracker drifts: This is the most frequent set of errors and it occurs when the tracker starts yielding progressively more incorrect predictions as the objects move through space. For situations in which these drifts occur in a severe manner, and we are employing an IOU data association method, the outcome is the creation of new wrong IDs. This happens since the drifted prediction will cease to have a high enough overlap with the detection corresponding to the original target object, and hence, the tracker will assume this is a new object and create a new ID. This situation is represented in Figure 7.1.

Occlusions followed by tracker drifts: This is a more specific situation of the previous. Here, the tracker drifts are caused by occlusions. More specifically, by the detectors poor ability to handle partial occlusions, namely, due to the greedy NMS step that is employed in both detectors. Therefore this problem is originated

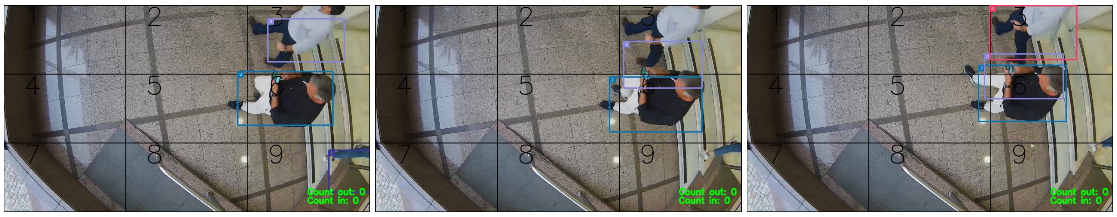


Figure 7.1: Tracker drift: in the first frame (left) we have two IDs, 2 and 3. In next frame, the tracker prediction drifts from its original target. In the last frame, as the prediction for ID 3 drifted completely from its original target, prediction and bounding box have not reached the overlap threshold to be associated to each other and hence a new ID 4 is created.

from the detector, which then propagates it to the tracker. This error can also be due to lack of training examples with partial occlusions. An example of this situation is represented in Figure 7.2.

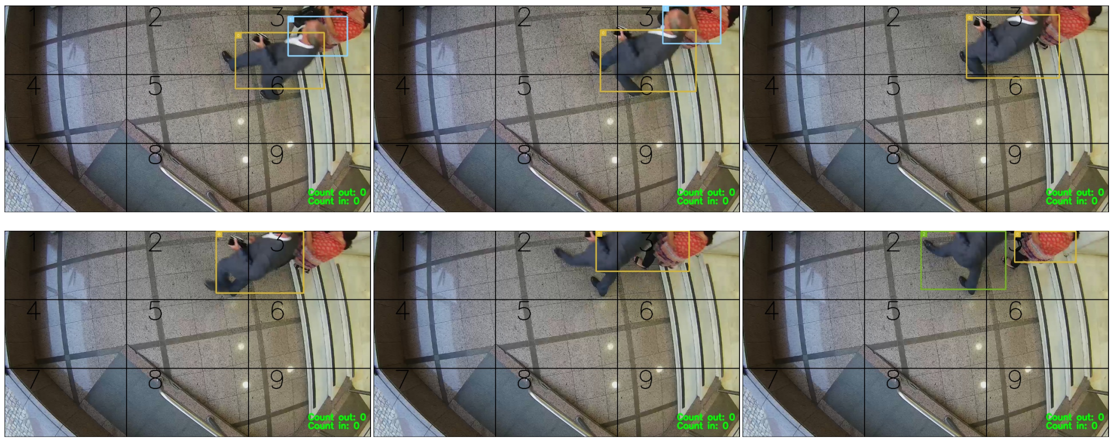


Figure 7.2: Occlusion followed by drift. Initially, we have two objects: ID0 and ID1 (first two frames). In the subsequent frames ID1 becomes partially occluded by ID0, and therefore the detection fails. In the next frames, the detection is wrongly generated for both objects, and ID0 starts drifting. When the object that originally had ID1 is re-detected, the identities are switched. ID1 is now ID0 and ID0 has a newly created ID 2.

Missed Detections in Entrance/Exit areas: When a detection is missed in a grid cell that does not correspond to an entrance or exit area, usually the tracker is able to recover using the grid algorithm, and therefore this does not constitute a problem. However, when we have an equivalent situation in an entrance/exit area, where we purposely set a low tolerance threshold for missed detections, the situation is problematic. Recall that we do this in order to smoothly terminate tracks of objects that exit the field of view. Without this setting, the tracker would continue predicting new locations for the object even after this object exited the field of view, and it would do this for MAX_AGE frames. As so, when we have a true missed detection in the entrance and exit areas, and the object is re-detected in subsequent frames, two things happen. Firstly, we considered the lifetime of the object that was being tracked as finished, removing it from the list of tracked objects. Secondly, after a few frames with no detection, when this object is finally re-detected it is considered a new object, and therefore a new ID is attributed to it. This problem is represented in Figure 7.3.

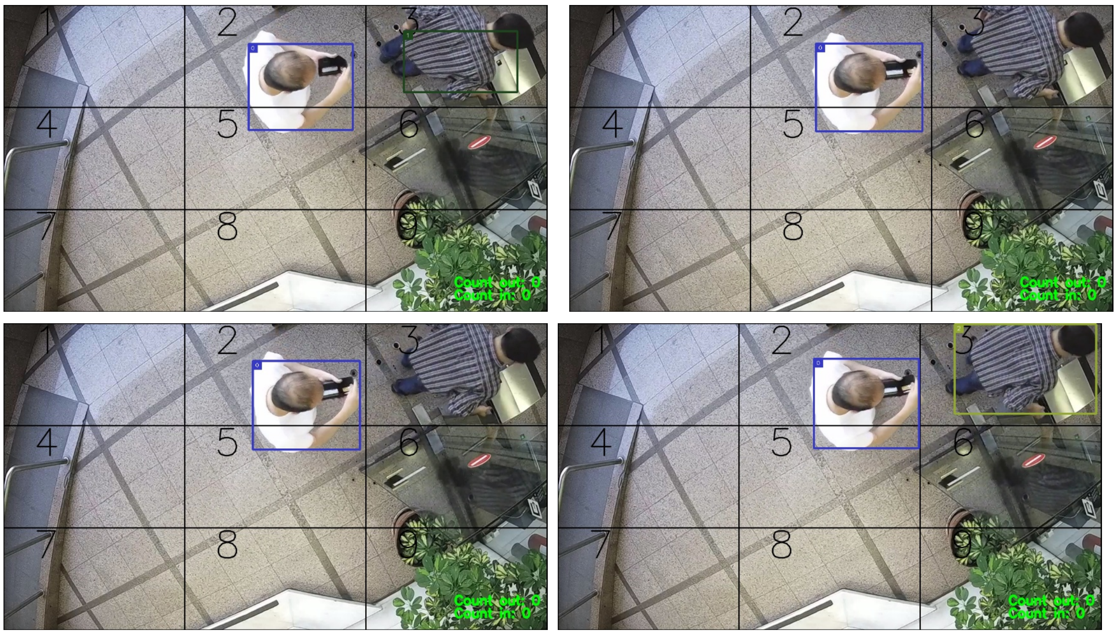


Figure 7.3: Missed Detections. In the first frame (top left) we have IDs 0 and 1. Note that ID 1 is located in an exit grid cell. In the following two frames, the detection for object 1 is lost (top right and bottom left). In the last frame (bottom right) the object is re-detected and a new ID (2) is created for the object that previously had ID 1.

7.3 Future Work

Given our current limitations, future work can be summarized as follows:

Dataset enlargement: The availability of large and varied datasets is one of the main challenges and also one of the main limitations of the present work. Despite having built and continuously incremented three different datasets, these are still very small compared to datasets such as ImageNet (Russakovsky et al., 2015) or COCO (Lin et al., 2014). As so, building upon the existing datasets and striving at making them better and more representative of real scenarios is a fundamental point of improvement for the future. One possible solution to generate data in a more efficient and less painful manner is to resort to crowdsourcing tools such as the Amazon Mechanical Turk[1].

More robust detection: In the previous section we discussed that the current detection process yields limited performance under occlusion situations, namely partial and total occlusions. Since detection quality greatly impacts tracker performance, occlusion handling is another crucial point for future work. Luckily, some works have proposed fully end-to-end architectures which deal specifically with detection occlusions. Namely, the work in (Stewart, Andriluka, & Ng, 2016) proposes a fully end-to-end detection architecture which fully avoids the NMS step, which is one of the main sources of errors due to occlusion. Another approach is to employ a part-based detection system, such as the work in (Tian, Luo, Wang, & Tang, 2015), which employs several part detectors in order to be able to correctly deal with partial occlusions. Inspired by these approaches, we already started to work on a similar part-based solution, which, additionally to the person, also detects the head of the person. Recall that, as our dataset is top-view, the heads are almost never occluded, and as so, are a good indicator of whether a person is in the field of view or not. As so, we plan to use person head detections in order to further refine and reason about full body detections, and therefore alleviate occlusion errors.

Solve tracker drifts: As seen in the previous section, tracker drifts are also a main source of errors. Some of these drifts are caused by detection occlusion errors, which propagate to the trackers, others by similar object appearances, while others are caused by abrupt motion changes in the target. Nevertheless, currently in the present work, we have no method that specifically deals with drifts and tries to correct the trajectories. This theme of handling drift is already explored in some works. For instance, the work in (Bae & Yoon, 2014) attributes a confidence

[1] `https://www.mturk.com/`

score to each tracklet, which is then used in order to associate tracklets to each other. The confidence score is based on factors such as tracklet length –since older tracklets are usually more reliable–, occlusion score –occluded targets should have lower confidence– and affinity between tracklets and detections. We leave the exploration of this sort of solution for future work.

Final Reflections: In summary, although the developed trackers achieve good performance, a great amount of work and improvements remain to be made in order for this system to be fully reliable. If successful, the enhancements described in the section above have great potential to allow us to significantly improve the performance. Once the performance is stable, in the future, this type of multiple camera person tracking system could be deployed and used in a variety of different commercial applications, which will be a step towards intelligent machines, able to understand and interact with the visual world around them.

References

Andriluka, M., Roth, S., & Schiele, B. (2010). Monocular 3D Pose Estimation and Tracking by Detection. *IEEE Conference on Computer Vision and Pattern Recognition*(2), 623–630. doi: 10.1109/CVPR.2010.5540156

Andriyenko, A., Schindler, K., & Group, R. S. (2011). Multi-target tracking by continuous energy minimization. *IEEE Conference on Computer Vision and Pattern Recognition*. doi: 10.1109/CVPR.2011.5995311

Bae, S.-H., & Yoon, K.-J. (2014). Robust Online Multi-object Tracking Based on Tracklet Confidence and Online Discriminative Appearance Learning. *IEEE Conference on Computer Vision and Pattern Recognition*, 1218–1225. Retrieved from `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909555` doi: 10.1109/CVPR.2014.159

Baikova, D., Maia, R., Santos, P., Ferreira, J., & Oliveira, J. (2018). Real Time Object Detection and Tracking. *Ambient Intelligence – Software and Applications –, 9th International Symposium on Ambient Intelligence*. doi: 10.1007/978-3-319-67774-3_19

Beeck, K. V., Engeland, K. V., Vennekens, J., & Goedem, T. (2017). Abnormal Behavior Detection in LWIR Surveillance of Railway Platforms. *Proceedings of the 14th IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS)*(August). doi: 10.1109/AVSS.2017.8078540

Bengio, Y. (2011). Deep Learning of Representations for Unsupervised and Transfer Learning. *JMLR: Workshop and Conference Proceedings 7*, *7*, 1–20. doi: 10.1109/IJCNN.2011.6033302

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. *Springer Berlin Heidelberg*, 437–478.

Berclaz, J., Fleuret, F., Türetken, E., & Fua, P. (2011). Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(9), 1806–1819. doi: 10.1109/TPAMI.2011.21

Bernardin, K., & Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: The CLEAR MOT metrics. *Eurasip Journal on Image and*

Video Processing, *2008*. doi: 10.1155/2008/246309

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., & Torr, P. H. (2016). Fully-convolutional siamese networks for object tracking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9914 LNCS*, 850–865. doi: 10.1007/978-3-319-48881-3_56

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *Proceedings - International Conference on Image Processing, ICIP*, *2016-Augus*, 3464–3468. doi: 10.1109/ICIP.2016.7533003

Black, J., Ellis, T., & Rosin, P. (2002). Multi view image surveillance and tracking. *Proceedings - Workshop on Motion and Video Computing, MOTION 2002*, 169–174. doi: 10.1109/MOTION.2002.1182230

Blackman, S. S. (2004). Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, *19*(1), 5–18. Retrieved from `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1263228` doi: 10.1109/MAES.2004.1263228

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proceedings of the eleventh annual conference on Computational learning theory - COLT' 98*, 92–100. Retrieved from `http://portal.acm.org/citation.cfm?doid=279943.279962` doi: 10.1145/279943.279962

Bochinski, E., Eiselein, V., & Sikora, T. (2017). High-Speed Tracking-by-Detection Without Using Image Information. *In Proceedings of the IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*(August). doi: 10.1109/AVSS.2017.8078516

Bolme, D., Beveridge, J. R., Draper, B. a., & Lui, Y. M. (2010). Visual object tracking using adaptive correlation filters. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2544–2550. Retrieved from `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5539960` doi: 10.1109/CVPR.2010.5539960

Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. *Proceedings of COMPSTAT'2010*, 177–186. doi: 10.1007/978-3-7908-2604-3_16

Bradski, G. R. (1998). Computer Vision Face Tracking For Use in a Perceptual User Interface. *Intel Technology Journal*, *2*(2), 12–21. Retrieved from `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.7673` doi: 10.1.1.14.7673

Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., & Van Gool, L. (2009). Robust tracking-by-detection using a detector confidence particle filter. *Proceedings of the IEEE International Conference on Computer Vision*, *i*, 1515–1522. doi: 10.1109/ICCV.2009.5459278

Caruana, R., Pratt, L., Thrun, S., Mitchell, T., Pomerleau, D., Dietterich, T., & State, O. (1997). Multitask Learning. , *75*, 41–75. doi: https://doi.org/10.1023/A:1007379606734

Chen, K., & Tao, W. (2016). Once for All: a Two-flow Convolutional Neural Network for Visual Tracking. , 1–15. Retrieved from `http://arxiv.org/abs/1604.07507` doi: 10.1109/TCSVT.2017.2757061

Chen, L., Ai, H., Shang, C., Zhuang, Z., & Bai, B. (2017). Online Multi-Object Tracking with Convolutional Neural Networks. *IEEE International Conference on Image Processing (ICIP), 2017.*, *2*, 645–649. doi: 10.1109/ICIP.2017.8296360

Choi, W. (2015). Near-online multi-target tracking with aggregated local flow descriptor. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 3029–3037. doi: 10.1109/ICCV.2015.347

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05)* (Vol. 1, pp. 886–893 vol. 1). doi: 10.1109/CVPR.2005.177

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., … Le, Q. V. (2012). Large scale distributed deep networks. *Advances in Neural Information Processing Systems*, 1223–1231. doi: 10.1109/ICDAR.2011.95

Detone, D., & Rabinovich, A. (2016). Deep Image Homography Estimation.

Dicle, C., Camps, O. I., & Sznaier, M. (2013). The way they move: Tracking multiple targets with similar appearance. *Proceedings of the IEEE International Conference on Computer Vision*, 2304–2311. doi: 10.1109/ICCV.2013.286

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*, 2121–2159. Retrieved from `http://jmlr.org/papers/v12/duchi11a.html` doi: 10.1109/CDC.2012.6426698

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14* (2), 179–211. doi: 10.1016/0364-0213(90)90002-E

Eshel, R., Moses, Y., & Arazi, E. (2008). Homography Based Multiple Camera Detection and Tracking of People in a Dense Crowd.
doi: 10.1109/CVPR.2008.4587539

Ess, a., Leibe, B., Schindler, K., & Van Gool, L. (2008). A mobile vision system for robust multi-person tracking. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 1–8. Retrieved from `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4587581` doi: 10.1109/CVPR.2008.4587581

Ess, a., Leibe, B., Schindler, K., & Van Gool, L. (2009). Robust multiperson tracking from a mobile platform. *Pattern Analysis and Machine Intelligence*, *31*, 1–14. doi: 10.1109/TPAMI.2009.109

Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2014). The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, *111* (1), 98–136. doi: 10.1007/s11263-014-0733-5

Everingham, M., Gool, L. V., Williams, C. K. I., & Winn, J. (2010). The PASCAL Visual Object Classes (VOC) Challenge. *International Journal*, 303–338. doi: 10.1007/s11263-009-0275-4

Fan, L., Wang, Z., Cail, B., Tao, C., Zhang, Z., Wang, Y., . . . Zhang, F. (2016). A survey on multiple object tracking algorithm. *2016 IEEE International Conference on Information and Automation (ICIA)*(61573057), 1855–1862.

Retrieved from http://ieeexplore.ieee.org/document/7832121/ doi: 10.1109/ICInfA.2016.7832121

Felzenszwalb, P. F., Girshick, R. B., Mcallester, D., & Ramanan, D. (2009). Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(9), 1–20. doi: 10.1109/TPAMI.2009.167

Gan, Q., Guo, Q., Zhang, Z., & Cho, K. (2015). First Step toward Model-Free, Anonymous Object Tracking with Recurrent Neural Networks. , 1–13. doi: 10.3389/fpsyg.2013.00124

Geiger, A., Lauer, M., Wojek, C., Stiller, C., & Urtasun, R. (2014). 3D traffic scene understanding from movable platforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(5), 1012–1025. doi: 10.1109/TPAMI .2013.185

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, *32*(11), 1231–1237. doi: 10.1177/0278364913491297

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1440–1448. doi: 10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 580–587. doi: 10.1109/CVPR.2014.81

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. *Adaptive Computation and Machine Learning Series*.

Gordon, D., Farhadi, A., & Fox, D. (2018). Re3 : Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects. *IEEE Robotics and Automation Letters*, *3*(2), 788–795. doi: 10.1109/LRA.2018.2792152

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. , 1–43. Retrieved from http://arxiv.org/abs/1308.0850 doi: 10.1145/2661829.2661935

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech Recognition with Deep Recurrent Neural Networks. (3). Retrieved from `http://arxiv.org/abs/1303.5778` doi: 10.1109/ICASSP.2013.6638947

Han, M., Xu, W., Tao, H., & Gong, Y. (2004). An Algorithm for Multiple Object Trajectory Tracking. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR.*, *1*, 864–871. Retrieved from `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1315122` doi: 10.1109/CVPR.2004.1315122

He, K. (2015). Convolutional Neural Networks at Constrained Time Cost. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5353–5360. doi: 10.1109/CVPR.2015.7299173

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1026–1034. doi: 10.1109/ICCV.2015.123

He, K., Zhang, X., Ren, S., & Sun, J. (2016, dec). Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* doi: 10.1109/CVPR.2016.90

Held, D., Thrun, S., & Savarese, S. (2016). Learning to track at 100 FPS with deep regression networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9905 LNCS*, 749–765. doi: 10.1007/978-3-319-46448-0_45

Hinton, G. E., Sabour, S., & Frosst, N. (2017). Dynamic Routing Between Capsules. (Nips). Retrieved from `http://arxiv.org/abs/1710.09829` doi: 10.1371/journal.pone.0035195

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. , 1–18. Retrieved from `http://arxiv.org/abs/1207.0580`

Hinton, G. E., Srivastava, N., & Swersky, K. (2012). Lecture 6a- overview of mini-batch gradient descent. *COURSERA: Neural Networks for Machine Learning*, 31.

Hochreiter, S., & Urgen Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

Huang, G., Liu, Z., Weinberger, K. Q., & van der Maaten, L. (2017). Densely Connected Convolutional Networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: 10.1109/CVPR.2017.243

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. *IEEE Conference on Computer Vision and Pattern Recognition*, *4*. doi: 10.1109/CVPR.2017.351

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 448–456. doi: 10.1007/s13398-014-0173-7.2

Itseez. (2014). *The OpenCV Reference Manual.* Retrieved from `http://opencv.org/`

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *Proceedings of the IEEE International Conference on Computer Vision*, 2146–2153. doi: 10.1109/ICCV.2009.5459469

Jia Deng, Wei Dong, Socher, R., Li-Jia Li, Kai Li, & Li Fei-Fei. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*(June), 248–255. doi: 10.1109/CVPRW.2009.5206848

Kahou, S. E., Michalski, V., Memisevic, R., Pal, C., & Vincent, P. (2017). RATM: Recurrent Attentive Tracking Model. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, *2017-July*, 1613–1622. doi: 10.1109/CVPRW.2017.206

Karpathy, A., & Fei-Fei, L. (2017). Deep Visual-Semantic Alignments for Generating Image Descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 664–676. doi: 10.1109/TPAMI.2016.2598339

Khan, S. M., Yan, P., & Shah, M. (2007). A homographic framework for the fusion of multi-view silhouettes. *Proceedings of the IEEE International Conference on Computer Vision*. doi: 10.1016/j.ast.2017.07.021

Khan, Z., Balch, T., & Dellaert, F. (2004). An MCMC-Based Particle Filter for Tracking Multiple Interacting Targets. *Eccv*, 279–290. doi: 10.1109/TPAMI.2005.223

Kim, Chanho; Li, Fuxin; Ciptadi, Arridhana; Rehg, J. M. (2015). Multiple Hypothesis Tracking Revisisted. *Iccv*, *19*(1). doi: 10.1074/JBC.274.42.30033.(51)

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. , 1–15. Retrieved from `http://arxiv.org/abs/1412.6980` doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503

Kristan, M., Leonardis, A., Matas, J., Felsberg, M., Pflugfelder, R., & Cehovin, L. (2017). The Visual Object Tracking VOT2017 challenge results. *IEEE International Conference on Computer Vision Workshops (ICCVW)*. doi: 10.1109/ICCVW.2017.230

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th international conference on neural information processing systems - volume 1* (pp. 1097–1105). USA: Curran Associates Inc.

Kuhn, H. (1955). The Hungarian Method For The Assignment Problem. *Naval Research Logistics*, *2*(1), 83–97. doi: 10.1002/nav.20053

Kuo, C. H., Huang, C., & Nevatia, R. (2010). Multi-target tracking by online learned discriminative appearance models. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 685–692. doi: 10.1109/CVPR.2010.5540148

Larsson, G., Maire, M., & Shakhnarovich, G. (2016). FractalNet: Ultra-Deep Neural Networks without Residuals. , 1–11. Retrieved from `http://arxiv`

.org/abs/1605.07648

Leal-Taixe, L., Canton-Ferrer, C., & Schindler, K. (2016). Learning by Tracking: Siamese CNN for Robust Target Association. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 418–425. doi: 10.1109/CVPRW.2016.59

Leal-Taixé, L., Milan, A., Reid, I., Roth, S., & Schindler, K. (2015). MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking. , 1–15. Retrieved from http://arxiv.org/abs/1504.01942

LeCun, Y., Bengio, Y., Hinton, G., Jordan, M. I., Berkeley, U. C., & Hinton, G. (2015). Deep Learning. , *444*(May), 436–444. doi: 10.1038/nature14539

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). *Backpropagation Applied to Handwritten Zip Code Recognition* (Vol. 1) (Nos. 4 OP - Neural Computation; Winter 1989, Vol. 1 Issue: 4 p541-551, 11p).

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. doi: 10.1109/5.726791

Liang Zheng. (2016). Scalable Person Re-Identification : A Benchmark Person Re-Identification. , 1–7. doi: 10.1109/ICCV.2015.133

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., … Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8693 LNCS*(PART 5), 740–755. doi: 10.1007/978-3-319-10602-1_48

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. *Computer Vision – ECCV 2016*. doi: https://doi.org/10.1007/978-3-319-46448-0_2

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, *60*(2), 91. doi: https://doi.org/10.1023/B:VISI.0000029664.99615.94

Ma, C., Huang, J. B., Yang, X., & Yang, M. H. (2015). Hierarchical convolutional features for visual tracking. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 3074–3082. doi: 10.1109/ICCV.2015.352

Milan, A., Leal-Taixe, L., Reid, I., Roth, S., & Schindler, K. (2016). MOT16: A Benchmark for Multi-Object Tracking. , 1–12. Retrieved from `http://arxiv.org/abs/1603.00831`

Milan, A., Rezatofighi, S. H., Dick, A., Reid, I., & Schindler, K. (2016). Online Multi-Target Tracking Using Recurrent Neural Networks. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 4225–4232.

Milan, A., Roth, S., & Schindler, K. (2014). Continuous energy minimization for multitarget tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(1), 58–72. doi: 10.1109/TPAMI.2013.103

Milan, A., Schindler, K., & Roth, S. (2013). Challenges of ground truth evaluation of multi-target tracking. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*(iii), 735–742. doi: 10.1109/CVPRW.2013.111

Mitzel, D., & Leibe, B. (2011). Real-time multi-person tracking with detector assisted structure propagation. *Proceedings of the IEEE International Conference on Computer Vision*, 974–981. doi: 10.1109/ICCVW.2011.6130357

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*(3), 807–814. doi: 10.1.1.165.6419

Nielsen, M. A. (2015). *Neural Networks and Deep Learning.* Determination Press. Retrieved from `http://neuralnetworksanddeeplearning.com/`

Ning, G., Zhang, Z., Huang, C., He, Z., Ren, X., & Wang, H. (2017). Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. *IEEE International Symposium on Circuits and Systems (ISCAS).* doi: 10.1109/ISCAS.2017.8050867

Okuma, K., Taleghani, A., de Freitas, N., Little, J. J., & Lowe, D. G. (2004). A

Boosted Particle Filter: Multitarget Detection and Tracking. , 28–39. doi: 10.1007/978-3-540-24670-1_3

Perez, L., & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1–17. doi: 10.1016/0041-5553(64)90137-5

Qi, Y., Zhang, S., Qin, L., Yao, H., Huang, Q., Lim, J., & Yang, M.-H. (2016). Hedged Deep Tracking. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4303–4311. doi: 10.1109/CVPR.2016.466

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788. doi: 10.1109/CVPR.2016.91

Reid, D. (1979). An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, *24*(6), 843–854. doi: 10.1109/TAC.1979.1102177

Ren, S., He, K., Sun, J., & Girshick, R. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. , *39*(6), 1137–1149. doi: 10.1109/TPAMI.2016.2577031

Riahi, D., & Bilodeau, G.-A. (2015). Multiple object tracking based on sparse generative appearance modeling. *2015 IEEE International Conference on Image Processing (ICIP)*, 4017–4021. doi: 10.1109/ICIP.2015.7351560

Ristani, E., Solera, F., Zou, R., Cucchiara, R., & Tomasi, C. (2016). Performance measures and a data set for multi-target, multi-camera tracking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9914 LNCS*(c), 17–35. doi: 10.1007/978-3-319-48881-3_2

Robicquet, A., Sadeghian, A., Alahi, A., & Savarese, S. (2016). Learning social etiquette: Human trajectory understanding in crowded scenes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9912 LNCS*, 549–565. doi:

10.1007/978-3-319-46484-8_33

Rodriguez, M. (2011). Data-driven Crowd Analysis in Videos. , 1235–1242.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. doi: 10.1038/323533a0

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. , *115*(3 OP - International Journal of Computer Vision. Dec 2015, Vol. 115 Issue 3, p211, 42 p.), 211. doi: 10.1007/s11263-015-0816-y

Sadeghian, A., Alahi, A., & Savarese, S. (2017). Tracking the Untrackable: Learning to Track Multiple Cues with Long-Term Dependencies. *Proceedings of the IEEE International Conference on Computer Vision*, *2017-Octob*, 300–311. doi: 10.1109/ICCV.2017.41

Sanchez-Matilla, R., Poiesi, F., & Cavallaro, A. (2016). Online multi-target tracking with strong and weak detections. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9914 LNCS*, 84–99. doi: 10.1007/978-3-319-48881-3_7

Schuhmacher, D., Vo, B. T., & Vo, B. N. (2008). A consistent metric for performance evaluation of multi-object filters. *IEEE Transactions on Signal Processing*, *56*(8 I), 3447–3457. doi: 10.1109/TSP.2008.920469

Shafique, K., Mun, W. L., & Haering, N. (2008). A rank constrained continuous formulation of multi-frame multi-target tracking problem. *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR.* doi: 10.1109/CVPR.2008.4587577

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. , 1–14. doi: 10.1016/j.infsof.2008.09.005

Socher, R., Perelygin, A., & Wu, J. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the …*, 1631–1642. doi: 10.1371/journal.pone.0073791

Son, J., Baek, M., Cho, M., & Han, B. (2017). Multi-Object Tracking with Quadruplet Convolutional Neural Networks. *Cvpr*, 5620–5629. doi: 10.1109/

CVPR.2017.403

Stauffer, C. (2003). Estimating Tracking Sources and Sinks The Tracking Correspondence Model. *Proceedings of the Second IEEE Workshop on Event Mining*, 1–8. doi: 10.1109/CVPRW.2003.10036

Stewart, R., Andriluka, M., & Ng, A. Y. (2016). End-to-end people detection in crowded scenes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2325–2333. doi: 10.1109/CVPR.2016.255

Stiefelhagen, R., Bernardin, K., Bowers, R., Garofolo, J., Mostefa, D., & Soundararajan, P. (2006). The CLEAR 2006 Evaluation. *Multimodal Technologies for Perception of Humans*(April), 1–44. doi: 10.1007/978-3-540-69568-4_1

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. doi: 10.1016/j.patrec.2014.01.008

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June*, 1–9. doi: 10.1109/CVPR.2015.7298594

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. doi: 10.1109/CVPR.2016.308

Tang, F., & Tao, H. (2008). Probabilistic object tracking with dynamic attributed relational feature graph. *IEEE Transactions on Circuits and Systems for Video Technology*, *18*(8), 1064–1074. doi: 10.1109/TCSVT.2008.927106

Tao, R., Gavves, E., & Smeulders, A. W. M. (2016). Siamese Instance Search for Tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. doi: 10.1109/CVPR.2016.158

Tian, Y., Luo, P., Wang, X., & Tang, X. (2015). Deep learning strong parts for pedestrian detection. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1904–1912. doi: 10.1109/ICCV.2015.221

Uijlings, J. R. R., Van De Sande, K. E. A., Gevers, T., & Smeulders, A. W. M.

(2013). Selective Search for Object Recognition. *International Journal of Computer Vision*, *104*(2), 154–171. doi: 10.1007/s11263-013-0620-5

Valmadre, J., Bertinetto, L., Henriques, J. F., Vedaldi, A., & Torr, P. H. S. (2017). End-to-end representation learning for Correlation Filter based tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2805–2813. Retrieved from `http://arxiv.org/abs/1704.06036` doi: 10.1109/CVPR.2017.531

Wang, B., Wang, L., Shuai, B., Zuo, Z., Liu, T., Chan, K. L., & Wang, G. (2016). Joint Learning of Convolutional Neural Networks and Temporally Constrained Metrics for Tracklet Association. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 386–393. doi: 10.1109/CVPRW.2016.55

Welling, M. (2011). Bayesian Learning via Stochastic Gradient Langevin Dynamics. *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 681–688.

Wojke, N., & Bewley, A. (2018). Deep Cosine Metric Learning for Person Re-Identification. *IEEE Winter Conference on Applications of Computer Vision (WACV)*. doi: 10.1109/WACV.2018.00087

Wojke, N., Bewley, A., & Paulus, D. (2018). Simple online and realtime tracking with a deep association metric. *Proceedings - International Conference on Image Processing, ICIP*, *2017-Septe*, 3645–3649. doi: 10.1109/ICIP.2017.8296962

Wu, Y., Lim, J., & Yang, M. H. (2015). Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *37*(9), 1834–1848. doi: 10.1109/TPAMI.2014.2388226

Yang, B., & Nevatia, R. (2012). Multi-target tracking by online learning of non-linear motion patterns and robust appearance models. *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, *1*, 1918–1925. doi: 10.1109/CVPR.2012.6247892

Yang, C., Duraiswami, R., & Davis, L. (2005). Fast multiple object tracking via a hierarchical particle filter. *Proceedings of the IEEE International Conference* *on Computer Vision*, *I*, 212–219. doi: 10.1109/ICCV.2005.95

Yu, F., Li, W., Li, Q., Liu, Y., Shi, X., & Yan, J. (2016). POI: Multiple object tracking with high performance detection and appearance feature. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9914 LNCS*, 36–42. doi: 10.1007/978-3-319-48881-3_3

Yu, Q., & Medioni, G. (2009). Multiple-target tracking by spatiotemporal Monte Carlo markov chain data association. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(12), 2196–2210. doi: 10.1109/TPAMI.2008.253

Yu, S. I., Yang, Y., & Hauptmann, A. (2013). Harry potter's marauder's map: Localizing and tracking multiple persons-of-interest by nonnegative discretization. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 3714–3720. doi: 10.1109/CVPR.2013.476

Yuan, L., Huang, C., & Nevatia, R. (2009). Learning to associate: HybridBoosted multi-target tracker for crowded scene. *IEEE Conference on Computer Vision and Pattern Recognition*, 2953–2960. doi: 10.1109/CVPR.2009.5206735

Yue-Hei Ng, Joe and Hausknecht, Matthew and Vijayanarasimhan, Sudheendra and Vinyals, Oriol and Monga, Rajat and Toderici, G. (2015). Beyond Short Snippets : Deep Networks for Video Classification. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4694—-4702. doi: 10.1109/CVPR.2015.7299101

Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Neural Networks. *Computer Vision – ECCV 2014*. doi: 10.1007/978-3-319-10590-1_53

Zeiler, M. D., Krishnan, D., Taylor, G. W., & Fergus, R. (2010). Deconvolutional networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2528–2535. doi: 10.1109/CVPR.2010.5539957

Zhai, M., Roshtkhari, M. J., & Mori, G. (2016). Deep Learning of Appearance

Models for Online Object Tracking. , 1–8. Retrieved from `http://arxiv.org/abs/1607.02568`

Zhang, B., Xiong, D., Su, J., & Duan, H. (2017). A Context-Aware Recurrent Encoder for Neural Machine Translation. *IEEE/ACM Transactions on Audio Speech and Language Processing*, *25*(12), 2424–2432. doi: 10.1109/TASLP.2017.2751420

Zhang, L., Li, Y., & Nevatia, R. (2008). Global data association for multi-object tracking using network flows. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8. doi: 10.1109/CVPR.2008.4587584

Zheng, L., Bie, Z., Sun, Y., Wang, J., Su, C., Wang, S., & Tian, Q. (2016). MARS: A Video Benchmark for Large-Scale Person Re-identification. *Computer Vision – ECCV 2016*. doi: 10.1007/978-3-319-46466-4_52

Zitnick, C. L., & Dollár, P. (2014). Edge boxes: Locating object proposals from edges. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8693 LNCS*(PART 5), 391–405. doi: 10.1007/978-3-319-10602-1_26