Aalto University
School of Science
Degree Programme of Computer, Communication, and Information Sciences

Tzu-Jui Wang

Multiple Object Tracking with Correlation Filters and Deep Features

Master's Thesis
Espoo, June 4th, 2018

Supervisors:	Professor Samuel Kaski, Aalto University
Instructor:	Jorma Laaksonen D.Sc. (Tech.), Aalto University

Aalto University
School of Science

Aalto University
School of Science
Degree Programme of Computer, Communication, and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Tzu-Jui Wang		
Title:	Multiple Object Tracking with Correlation Filters and Deep Features		
Date:	June 4th, 2018	**Pages:**	73
Supervisors:	Professor Samuel Kaski, Aalto University		
Instructor:	Jorma Laaksonen D.Sc. (Tech.), Aalto University		

This thesis studies on-line multiple object tracking (MOT) problem which has been developed in numerous real-world applications, such as emerging self-driving car agents or estimating a target's trajectory over time to identify its movement pattern. The challenges that an on-line MOT tracker always faces are: (1) being able to consistently and smoothly track the same target over time with the presence of occlusions, (2) being able to recover from fragmented tracks, (3) handling identity switches of the same target, and (4) being able to operate in real-time. This work aims to provide an efficient detect-and-track framework to address these challenges. To narrow down the classes of objects to be studied, but without losing the tracker's extendibility to a generic object, we pick *pedestrians* as the primary objects of interest.

The proposed framework consists of four building blocks, i.e. object detection, object tracking, data association, and object re-identification. While most of the MOT frameworks make the assumption of the availability of the detector in every frame, the proposed MOT tracker operates with the detector being triggered only periodically, e.g. in every three frames, leading to improved efficiency. As for each building block, the detection is performed by Single Shot Detector (SSD), which has proven efficiency and efficacy on generic object classes. When the detector is triggered and active tracks exist, data association module identifies the correspondence of the objects detected by the detector and tracked by the tracker. In cases where newly detected objects cannot be identified as any of current tracks, the re-identification module then attempts to find the correspondence for them in the history track.

The experiments show that the proposed framework is outperformed by the recently published on-line MOT trackers which are based on different object detectors. However, the results suggest that the proposed framework's performance does not degrade when the detector is partially unavailable and improves in certain conditions due to better temporal consistency. Based on these experiments, we are able to identify major shortcomings of the current framework, providing possible ways to improve it and directions for the future work.

Keywords:	correlation filters, multiple object tracking, object detector
Language:	English

Acknowledgements

The thesis was completed within the CBIR group in Computer Science Department at Aalto University and the Video Analytics team at CGI Finland under the supervision of Professor Samuel Kaski, Dr. Jorma Laaksonen, and M.Sc. Vesa Kuoppala. The work was financially supported by CGI Finland.

I would like to firstly thank Professor Samuel Kaski for your support to this work. Next, I would like to thank Jorma for being my principal advisor, and for the informative and knowledgeable feedbacks you have given to my academic writing, and how rigorous research is conducted. Also, thank you for being the bridge between the academic and industry world, so we can be always thinking how to apply latest research results to business cases. In addition, I would like to thank Dr. Hamed Rezazadegan Tavakoli for your instructions in my critical thinking and academic writing. Although we did not cooperate in this work, your advice in our collaborative research had been helpful for me conducting this project.

Numerous people to thank for in CGI Finland (previously Affecto Finland before it was acquired by CGI), but I would especially like to thank Vesa for your constant support for any technical difficulties I have encountered. I also thank the VA team as a whole because it allows me to study and work at the same time so that I could grow as fast as I can. Lastly, I would like to thank CGI Finland for financially supporting this thesis project. I have been delightful on my way from initiating to finishing the project.

The completion of the thesis marks the end of my journey as a Master's student in Aalto University, an employee in CGI Finland, and a normal person who enjoys the Finnish culture. Thank you who have been always with me there, it is my hearty pleasure to work with you and have you around.

Espoo, June 4th, 2018

Tzujui (Julius) Wang

Abbreviations and Acronyms

CNN	Convolutional Neural Network
Conv.	Convolutional layer
DSST	Discriminative Scale Space Tracker
FM	Fragmentation
fps	frames per second
GT	Number of Ground-Truth Tracks
HoG	Histogram of Oriented Gradients
IDFT	Inverse Discrete Fourier Transform
IDs	Identity Switch
KCF	Kernelized Correlation Filtering Tracker
mAP	mean Average Precision
MARS	Motion Analysis and Re-identification Set
ML	Mostly Lost
MOSSE	Minimum Output Sum of Squared Error
MOT	Multiple Object Tracking
MOTA	Multiple Object Tracking Accuracy
MOTP	Multiple Object Tracking Precision
MT	Mostly Tracked
NUC	Next Unit of Computing
PT	Partly Tracked
R-CNN	Region-based Convolutional Neural Networks
RPN	Region Proposal Network
RoI	Region of Interest
SORT	Simple Online and Realtime Tracking
SSD	Single Shot Detector
VOC	Visual Object Classes

Contents

Chapter 1

Introduction

This work is initiated by the Video Analytics (VA) team in Affecto Finland[1] and the Content-Based Image Retrieval (CBIR) Group in the Computer Science Department at Aalto University. The VA team is specialized in delivering analytics regarding the information presented in images and videos. The primary driving force of this work is from the urgent requests and needs from the industry, especially the corporates who would like to analyze every possible aspect in multimedia. To consolidate the foundation of video analytics capability within the VA team, we pilot in developing an object detection and tracking framework as generic and efficient as possible. In addition, to be more specific on the topic, we select *pedestrians* as our primary objects of interest in multimedia files as several requests from other collaborators are interested in analyzing people flow in an open public space. As a result, the problem boils down to object detection and multiple object tracking problems, which have been studied in the academia. The following sections will present the readers the motivation, scope and the structure of the thesis. The main contributions of the thesis will be presented briefly.

1.1 Motivation

We study the multiple object tracking (MOT) problem which lies in the domains of computer vision and machine learning. MOT has been a popular problem in the academia and industry given its practicality and usefulness in the real world. One such useful MOT case is the development of self-driving cars. A robot driver needs to be constantly tracking the surrounding objects, such as pedestrians, cyclists, and vehicles to avoid collision of any kind. Another application is the surveillance case where the system should be

[1]Affecto Finland was acquired by CGI in 2018, it is now part of CGI Finland.

tracking people or objects so as to detect anything suspicious and abnormal. As for the business cases, the VA team receives myriad of requests in which stakeholders are interested in understanding how people are moving in the city, what is their reaction towards advertisements, do they move in groups, etc.

Despite numerous interesting MOT applications and published works on it, MOT is still worth research efforts due to several difficulties. First, the system should be running in real-time, i.e. beyond 15 frames per second (fps), to be capable of responding to the real world and making analytics accordingly and spontaneously. Second, severe occlusions of objects are often the case in the context of MOT, e.g. pedestrians blocking each other from the camera's perspective. Third, following the second problem, a tracker can easily be confused by two visually similar objects, suffering from either starting to track the same object or mistaking the object's identity for another object's. Last but not least, some of MOT trackers are operated partially on-line or off-line, requiring efforts in post-inference and post-processing. This can cause some difficulties in carrying out the real-time analytics in the cases where the video feeds are streaming 24/7. Thereby, in this thesis, we aim to design an efficient on-line MOT algorithm focusing on *pedestrians* while taking into account the addressed problems.

1.2 Scope of the Thesis

The focus of the thesis is to design and implement an efficient on-line MOT algorithm, i.e. the algorithm can not peep into the future frames and the efforts in post-processing the video frames should be minimized. The on-line setting is more tailored to applications where the video frames are constantly streaming in and the analytics should be made in real-time. However, it could make a MOT tracker more difficult to consistently track a single object, leading to a number of fragmented tracks of the same object. In the case of a tracker starting drifting away from an object, it should be able to re-identify it and recover from the tracking failure. In addition, MOT algorithms require an object detector, which always has to be triggered in every frame, to initialize the tracker with the target's location. However, performing object detection in every frame can drag down the running speed as it is often more computationally demanding than the tracking part, even though it can possibly reduce the localization errors of the targets. We address the issue and do not assume the availability of the object detector triggered in every frame, but, triggered periodically. We further examine the tradeoff between the number of frames where the detector is triggered and the tracking accuracy to reason how frequently the system should incorporate the detector.

The development is mostly based on the Python programming language and open-source frameworks that can easily be deployed across the platforms with or without GPU support, e.g. Amazon Web Service, mini PC like Intel Next Unit of Computing (NUC) [1], etc. In particular, Tensorflow [4] and Dlib [23] come as the primary choice due to their great support in Python and ease of deployment. Besides, alternative detector models, in terms of different levels of inference speeds and accuracies, are provided within the Tensorflow framework [3], allowing us for easy replacement of the models if needed.

In a nutshell, the main contributions of this work are summarized as follows:

- Provide detailed reviews on modern techniques for object detection and tracking
- Design and implement a detect-and-track framework which can be extended to generic object classes
- Study the efficacy of the proposed framework on the MOT'16 challenge dataset
- Provide suggestions of possible improvements and future research directions based on the results from the evaluation

1.3 Structure of The Thesis

The following chapters are organized as follows:

Chapter 2 introduces necessary background knowledege, which covers the state-of-the-art object detectors (i.e. Faster R-CNN [28] and SSD detector [26]), the on-line single object tracker [9, 12], and the on-line multiple object tracker (e.g. Deep SORT [36]).

Chapter 3 describes the proposed MOT framework, including the architecture and the detailed description of each constituent in the framework.

Chapter 4 starts with the benchmark datasets and the evaluation metrics. Next, it provides ablation studies of the parameters in the proposed framework, a comparison with other on-line trackers and discussions on the experimental results. Finally, it discusses the possible improvements based on the experiment results.

Chapter 5 concludes the work and discusses the future work.

Chapter 2

Background

2.1 Object Detection

Despite its long history of development since 90's [29, 35], object detection has experienced major breakthrough since 2012 after AlexNet [24] was popularized. Several pivotal works, such as Overfeat [30], SPPNet [18], Fast R-CNN [15], more recently Faster R-CNN [28] and Single Shot Detector (SSD) [26] have advanced the object detection approaches in terms of both speed and accuracy. In the following sections, the latter two works are introduced due to the fact that their designs well preserve the advantages but also compensate the shortcomings of the previous object detectors.

2.1.1 Faster R-CNN

Faster R-CNN is an object detector comprising of an object proposal generator and a detection network serving as classifiers classifying the generated object proposals as shown in Figure 2.1. Unlike Fast R-CNN [15] relying on an external object proposal generator, e.g. Selective Search [34], Faster R-CNN introduces Region Proposal Network (RPN) which learns to generate the object proposals during the network training phase. The major contribution of this architecture is that it shares the convolutional features not only among the object proposals (as Fast R-CNN does) but also among the object proposals and detection networks, contributing to less wasted computation and faster inference and, in addition, higher mean Average Precision (mAP) than Fast R-CNN on PASCAL VOC 2007 and 2012 benchmark datasets. In Section 2.1.1.1 and 2.1.1.2 we introduce the RPN architecture and the loss functions deviced to train RPN, respectively.

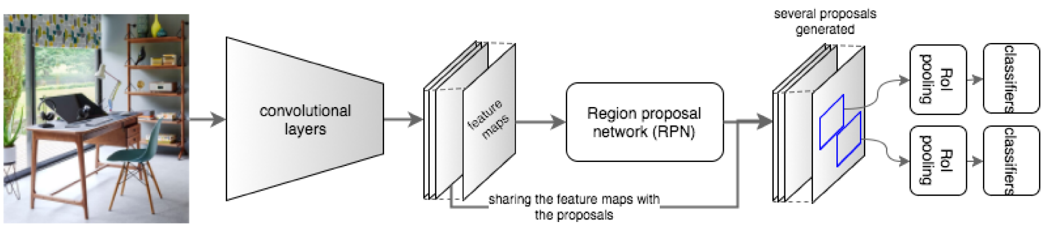

convolutional layers
feature maps
Region proposal network (RPN)
sharing the feature maps with the proposals
several proposals generated
RoI pooling
classifiers
RoI pooling
classifiers

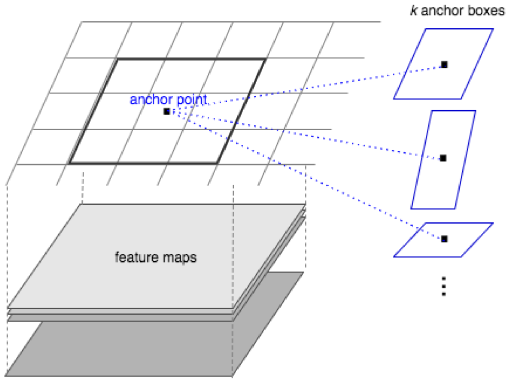
Figure 2.1: Faster R-CNN network structure [28].

2.1.1.1 Region Proposal Networks

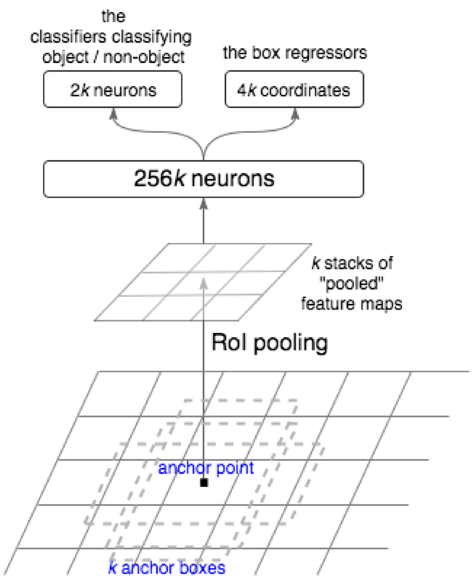
Figure 2.2 illustrates the architecture of RPN. The inputs of RPN are the feature maps produced by the last shared convolutional layer as shown in Figure 2.2b. To generate the proposals of different sizes and aspect ratios and to be traslation-invariant, sliding windows of size $n \times n$ are applied over the feature maps. In addition, at the center of each window, RPN looks at the fixed set of multi-scale k *anchors*, which are of different sizes and aspect ratios. If the width and height of a feature map are of W and H, respectively, the total amount of anchors that RPN produces is $W \times H \times k$. Staring from sampling k anchor boxes with pre-defined scales and aspect ratios, each anchor box is pooled within a RoI pooling layer, resulting in the pooled feature maps of fixed size, e.g., 3×3. RoI pooling layers enable Faster R-CNN to take the images of arbitrary size as inputs. The pooled feature maps of k anchor boxes are later on fed into a fully-connected layer of $256k$ neurons as shown in Figure 2.2b. The final layers of RPN consists of the classifiers and box regressors which produce $2k$ values indicating objects versus non-objects and $4k$ *encoded* objects' coordinates, respectively. It is noted that RPN learns k sub-networks specifically on k anchors across all locations of the sliding windows.

2.1.1.2 The Loss Function for Training RPN

Among all of the anchor boxes, those having *Intersection over Union* (IoU) overlap with ground-truth bounding boxes larger than 0.7 are treated as the positives. In case that there is no single anchor whose IoU is larger than 0.7, the anchor with the highest IoU value is also treated as a positive. The anchor whose IoU is lower than 0.3 is treated as a negative. Those boxes which do not meet any of the previously mentioned conditions are not included in the training. The loss function of RPN evaluated on the anchors (can be positive and negative) indexed by i in a mini-batch is defined as



(a) "Anchor boxes" idea deviced in RPN.



(b) Illustration of RPN

Figure 2.2: Illustration of Region Proposal Network (RPN) architecture and its "anchor boxes" concept.

$$\begin{aligned} L(\{\mathbf{p}_i, \mathbf{t}_i\}) &= \texttt{classification loss} + \texttt{box regression loss} \\ &= \frac{1}{N_{cls}} \sum_i L_{cls}(\mathbf{p}_i, \mathbf{p}_i^*) + \frac{\lambda}{N_{reg}} \sum_i p_i^* L_{reg}(\mathbf{t}_i, \mathbf{t}_i^*). \end{aligned} \tag{2.1}$$

The first loss term is for measuring classifiation loss while the second is for anchor box regression loss. The two losses are balanced by λ and normalized by N_{cls} and N_{reg}, respectively. N_{cls} is the size of a mini-batch of images while N_{reg} is the number of the anchor locations. $L_{cls}(\cdot)$ is the negative log loss defined over predicted probabilities $\mathbf{p}_i$ and true probabilities $\mathbf{p}_i^*$ of the box being an object or a non-object, where

$$L_{cls}(\mathbf{p}_i, \mathbf{p}_i^*) = -p_i^{\text{obj}*} \cdot \log p_i^{\text{obj}} - p_i^{\text{non-obj}*} \cdot \log p_i^{\text{non-obj}}, \tag{2.2}$$

and

$$\mathbf{p}_i = (p_i^{\text{obj}}, p_i^{\text{non-obj}}),\ \mathbf{p}_i^* = (p_i^{\text{obj}*}, p_i^{\text{non-obj}*}),$$

where p_i^{obj} and $p_i^{\text{obj}*}$ denote the probability of the corresponding anchors being an object and $p_i^{\text{non-obj}}$ and $p_i^{\text{non-obj}*}$ denote that not being an object. $(p_i^{\text{obj}*}, p_i^{\text{non-obj}*}) = (1,0)$ if the corresponding i^{th} anchor box is positive and $(0,1)$ if it is negative.

In the second term of (2.1), the regression loss $L_{reg}(\mathbf{t}_i, \mathbf{t}_i^*)$ is defined over $\mathbf{t}_i$ and $\mathbf{t}_i^*$, where $\mathbf{t}_i$ and $\mathbf{t}_i^*$ are the predicted and ground-truth "encoded" bounding box coordinates, respectively, and parameterized by

$$\begin{aligned}
&\mathbf{t}_i = (t_x, t_y, t_w, t_h),\ \mathbf{t}_i^* = (t_x^*, t_y^*, t_w^*, t_h^*),\\
&t_x = (x - x_a)/w_a,\ t_y = (y - y_a)/h_a,\\
&t_w = \log(w/w_a),\ t_h = \log(h/h_a),\\
&t_x^* = (x^* - x_a)/w_a,\ t_y^* = (y^* - y_a)/h_a,\\
&t_w^* = \log(w^*/w_a),\ t_h^* = \log(h^*/h_a),
\end{aligned} \tag{2.3}$$

where x, y, w, h denote the predicted anchor box's center coordinates and its width and height. x_a, y_a, w_a, h_a are the anchor box's center oordinates and its width and height. Likewise, $x_a^*, y_a^*, w_a^*, h_a^*$ are for the ground-truth anchor box. t_x and t_y are parametrized to be scale-invariant to (w, h) and (w_a, h_a), respectively. The losses contributed by t_w and t_h are measured by the shift of w versus w_a and h versus h_a, respectively, and likewise for t_w^* and t_h^*. Then, $L_{reg}(\mathbf{t}_i, \mathbf{t}_i^*)$ is defined as

$$L_{reg}(\mathbf{t}_i, \mathbf{t}_i^*) = \sum_{j \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_j - t_j^*), \tag{2.4}$$

where $\text{smooth}_{L_1}(\cdot)$ is a smooth L_1 loss function,

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1, \\ |x| - 0.5, & \text{otherwise.} \end{cases} \tag{2.5}$$

2.1.1.3 Combining RPN and Fast R-CNN

To classify the object proposals generated by RPN, Faster R-CNN simply incorporates Fast R-CNN as the detection network while sharing the convolutional layers with RPN. The combining scheme of RPN and fast R-CNN is illustrated in Figure 2.3. As shown, the feature maps of each proposal generated by RPN are extracted from the projected RoI on the feature maps produced by the last shared convolutional layer. Afterwards, each proposal's feature maps are processed by a RoI pooling layer to produce feature maps of fixed size. The pooled feature maps are processed by sequence of fully-connected layers and branched to the K-way softmax layer to estimate the class probability and the box regressor to predict the coordinates of the bounding box, where K is the number of the classes. Combining RPN and Fast R-CNN in this manner allows sharing the features not only among the proposals, but among the proposals and the detection network. This could enforce the whole network to learn more generalized feature extractors that extract the features estimating both the objectness and distinctiveness of each object.

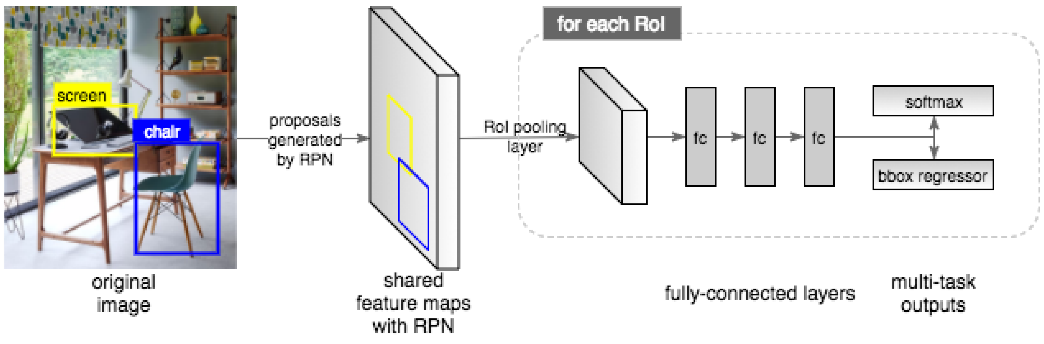


Figure 2.3: Illustration of the combining scheme of RPN and Fast R-CNN.

2.1.2 Single Shot Detector

Single Shot Detector (SSD) attempts to cope with object detection problem using a single neural network. As opposed to Faster R-CNN, SSD skips generating the object proposals (with one network) which are then classified (with another network), and predicts the presence of each object category straight from the pre-defined *default boxes*, similiar to the anchor boxes in Faster R-CNN, on the feature maps. Furthermore, instead of relying on feature maps the single scale to make the predictions (such as e.g. Fast R-CNN and Faster R-CNN), SSD aggregates the predictions made at multiple scales to produce the final detections, outperforming Faster R-CNN in mAP on VOC2007 `test` by 1.1% and inference speed by 8 times.

SSD architecture is illustrated in Figure 2.4. It comprises of a base network, e.g. VGG-16 net [26], followed by several extra feature layers which are fully convolutional. Each of these six feature layers, i.e. `Conv4_3`, `Conv7`, `Conv8_2`, `Conv9_2`, `Conv10_2`, and `Conv11_2`, has an auxiliary network consisting of a convolutional layer serving as the classifiers at the corresponding scale. We discuss SSD in details as follows.

2.1.2.1 Default Boxes with Different Aspect Ratios

Default boxes are tiled in the feature layers which are branched to an auxiliary classification network. For a feature layer of size $m \times n$ (e.g. `Conv4_3` is of 38×38), k default boxes with different aspect ratios are anchored at each of $m \times n$ locations across the feature layers. This allows the predictions to cover

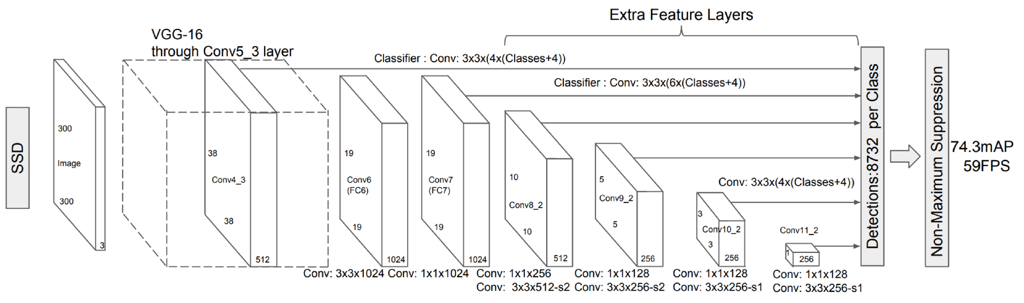


Figure 2.4: Single Shot Detector Architecture [26]. The figure is duplicated from [26].

a wide set of objects' locations, aspect ratios, and scales by exploiting the features at multiple scales. (We recommend the reader to read Section 2.2 in [26], especially how the set of scales and aspect ratios of those default boxes imposed in each layer are designed.) Specifically, one can obtain k predicted bounding boxes per object class at each location on the feature maps. The total number of predicted bounding boxes $N_{\texttt{all}}$ one can gather from all the feature layers is calculated as

$$N_{\texttt{all}} = \sum_{i \in \{\texttt{feature layers}\}} \texttt{feature\_map\_width} \times \texttt{feature\_map\_height} \times k_i, \tag{2.6}$$

where k_i is the number of different default boxes at i^{th} feature layer. For instance, in Figure 2.4, $N_{\texttt{all}}$ can be carried out by summing $38 \times 38 \times 4$ (`Conv4_3`), $19 \times 19 \times 6$ (`Conv7`), $10 \times 10 \times 6$ (`Conv8_2`), $5 \times 5 \times 6$ (`Conv9_2`), $3 \times 3 \times 4$ `Conv10_2`, $1 \times 1 \times 4$ (`Conv4_3`), amounting to 8732 default boxes per object class.

2.1.2.2 Convolutional Predictors at the Default Boxes

For k_i default boxes at i^{th} feature layer, each of them makes $(4 + P)$ predictions including the offsets $\Delta(c_x, c_y, w, h)$ of the center's coordinates (c_x, c_y), width w and height h relative to the default box, and the classification scores of P object classes, $(c_1, c_2, ..., c_P)$. To be more specific on how the predictions are made in each feature layer, for feature maps of size $m \times n$ and depth l, the predictions are made through applying $k \cdot (4 + P)$ filters at each location in the feature maps, yielding another feature maps of size $m \times n$ and depth $k \cdot (4 + P)$. Hence, each location of this resulting feature map contains $(4 + P)$ predictions (i.e. the class scores and the shape offsets) per default box. It is worth noting that although looked similarly, RPN in Faster

R-CNN and the auxiliary network in SSD play different roles. The former devices anchor boxes to generate generic object proposals, while the latter detects the objects within the default boxes. This can be seen by comparing the loss functions deviced in RPN, i.e. (2.2) and SSD, i.e. later in (2.8). As a result, SSD does not require a seperate object proposal network such as RPN in Faster R-CNN.

2.1.2.3 Training SSD

When training SSD, each default box is matched to a ground-truth box and considered to be positive if their IoU is higher than 0.5, otherwise, it is considered negative. SSD's loss function is defined over all default boxes as

$$\begin{aligned} L(\mathbf{x}, \mathbf{c}, \mathbf{l}, \mathbf{g}) &= \frac{1}{N}(\texttt{confidence loss} + \texttt{localization loss}) \\ &= \frac{1}{N}(L_{conf}(\mathbf{x}, \mathbf{c}) + \lambda L_{loc}(\mathbf{x}, \mathbf{l}, \mathbf{g})). \end{aligned} \tag{2.7}$$

The confidence loss is defined over the matching indicators $\mathbf{x}$ and confidences of the predictions for all object classes $\mathbf{c}$. $\mathbf{x} = \{x_{ij}^p\}$ that each $x_{ij}^p = \{1, 0\}$ is an indicator for i^{th} default box being matched to j^{th} grount-truth box of p object class. $\mathbf{c} = \{c_i^p\}$, where c_i^p is the confidence score of i^{th} default box being predicted as p object class. The localization loss is defined over the predicted bounding boxes, $\mathbf{l} = \{l_i^m | m \in \{c_x, c_y, w, h\}\}$, and the ground-truth bounding boxes, $\mathbf{g} = \{g_i^m | m \in \{c_x, c_y, w, h\}\}$, where (c_x, c_y) is the coordinates of the center of the given bounding box, and w, h are the width and height, respectively. λ is for balancing the two losses. The confidence loss is defined as the softmax loss over $(P + 1)$ classes including the negative class $(p = 0)$, i.e.

$$L_{conf}(\mathbf{x}, \mathbf{c}) = -\sum_{p}^{P} \sum_{i \in \texttt{positives}}^{N} x_{ij}^p \, log(\hat{c}_i^p) - \sum_{i \in \texttt{negatives}} \log(\hat{c}_i^0), \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}, \tag{2.8}$$

and the localization loss is defined similarly to the box regression loss in Faster R-CNN as,

$$L_{loc}(\mathbf{x}, \mathbf{l}, \mathbf{g}) = \sum_{i \in \texttt{positives}}^{N} \sum_{m \in \{c_x, c_y, w, h\}} x_{ij}^k \, \text{smooth}_{L_1}(l_i^m - \hat{g}_j^m), \tag{2.9}$$

where

$$\hat{g}_j^{c_x} = (g_j^{c_x} - d_i^{c_x})/d_i^w, \quad \hat{g}_j^{c_y} = (g_j^{c_y} - d_i^{c_y})/d_i^h,$$
$$\hat{g}_j^w = \log(\frac{g_j^w}{d_i^w}), \quad \hat{g}_j^h = \log(\frac{g_j^h}{d_i^h}).$$

$(d_i^{c_x}, d_i^{c_y}, d_i^w, d_i^h)$ are the shape parameters of a default bounding box.

2.2 Online Single Object Tracking

On-line single object tracking addresses the problem in which given the current frame and the initial state of an object, the tracker predicts the object's state in the next frame. The object's state in the context of this thesis project is the bounding box that well wraps around the object. Tracking-by-detection is an approach that started gaining its popularity since 2006. The works that applauds tracking-by-detection methodology include on-line boosting trackers [5, 16, 17], tracking-learning-detection (TLD) tracker [22], and later on the correlation filtering tracker, such as MOSSE tracker [9], DSST [12], kernelized correlation filtering tracker (KCF) [19], etc.

Due to their computational efficiency and effectiveness on modeling an object's appearance, correlation filtering based trackers have become the state-of-the-art where their results are always ranked top on different benchmark datasets. Hence, in the later sections, we introduce Minimum Output Sum of Squared Error (MOSSE) tracker, which is one of the very first approaches that apply correlation filters for the tracking problem, and Discriminative Scale Space Tracker (DSST), which had won the Multiple Object Tracking'14 Challenge and still is able to operate in real-time on CPU.

2.2.1 Minimum Output Sum of Squared Error Tracker

Minimum Output Sum of Squared Error (MOSSE) Tracker [9] tracks an object by first learning a filter h such that the mean square error of the desired outputs g and the correlation of h and object templates f is minimized. Once the learning is finished, the learned filter h is applied on the current frame to identify the location with maximal response as the predicted location of an object. Mathematically, assuming we have the target's templates $f_1, f_2, ..., f_N$ and the respective desired outputs $g_1, g_2, ..., g_N$, the learning objective is

$$h^* = \arg\min_h \sum_{i=1}^{N} ||f_i \star h - g_i||_2^2, \; h, f_i, g_i \in \mathcal{R}^{M \times K}, \tag{2.10}$$

where M and K is the width and height of the object templates. $\star$ denotes the correlation operator. By the correlation theorem, the correlation between two signals becomes element-wise matrix multiplication in the Fourier domain, specifically

$$f_i \star h = F_i \odot H,$$

where F_i and H denote f_i and h in the Fourier domain, respectively. This helps evaluate the objective in (2.10) faster by re-formulating it with

$$H^* = \arg\min_H \sum_{i=1}^{N} ||F_i \odot \bar{H} - G_i||_2^2, \text{ and } h^* = \mathcal{F}^{-1}(\bar{H}), \tag{2.11}$$

where $\mathcal{F}^{-1}(\cdot)$ is the inverse discrete Fourier transform (IDFT) by which the optimal h^* is obtained from H^*. The symbols with the bar over them denote the complex conjugates, e.g. $\bar{H}$.

To train a MOSSE tracker on-line, the object template f_i is built by taking the grayscale image patches around the object. The desired outputs g_i are generated by a Gaussian distribution whose mean is aligned with the center of the object. f_i and g_i are then transformed into the Fourier domain and followed by solving the objective in (2.11). (2.11) can be solved by equating its partial derivatives with respect to $\bar{H}$ to zero. The solution turns to be in closed-form, i.e.

$$H^0 = \frac{\sum_{i=1}^N G_i \odot \bar{F}_i}{\sum_{i=1}^N F_i \odot \bar{F}_i}. \tag{2.12}$$

The superscript zero in H^0 simply means that H^0 is the filter learned initially, (2.12) can be interpreted as the mean correlation between the inputs f_i and the desired outputs g_i in the Fourier domain normalized by the self-correlation of the inputs. A regularization term can be also added in (2.12) as

$$H^0 = \frac{A^0}{B^0} = \frac{\sum_{i=1}^N G_i \odot \bar{F}_i}{\sum_{i=1}^N F_i \odot \bar{F}_i + \epsilon}, \tag{2.13}$$

which is simply adding an ϵ in the denominator. To predict the target's location at frame t, the response map over a rectangle area $\mathcal{Y}$ is first computed as

$$R = \mathcal{F}^{-1}\Big\{F_{\mathcal{Y}} \odot H^{t-1}\Big\}, \tag{2.14}$$

where $F_{\mathcal{Y}}$ is the template of the rectangle area $\mathcal{Y}$. Then, the predicted target's location can be obtained by identifying the coordinates on $\mathcal{Y}$ that yields the maximal response value. As the tracking process continues with the coming new frames, in theory, one can learn the new filter on the target's previous templates and the templates from the coming frame. However, solving (2.13) grows in $O(N^2)$ when number of templates N increases. Alternatively, MOSSE updates the filter H^t after receiving the target's template F^t and the desired output G^t at frame t as:

$$\begin{aligned} H^t &= \frac{A^t}{B^t}, \\ A^t &= \eta G^t \odot \bar{F}^t + (1-\eta)A^{t-1}, \\ B^t &= \eta F^t \odot \bar{F}^t + (1-\eta)B^{t-1}, \end{aligned} \tag{2.15}$$

where η is the learning rate. Although the whole tracking process can be run at rather high speed (e.g. a hundred of frame per second), MOSSE does not address the case when the ground-truth bounding box of the target changes in size. A remedy for that is to construct a scale-space pyramid of search windows centered at the previous target's location. This strategy is adopted in DSST [12], which is introduced in the next section.

2.2.2 Discriminative Scale Space Tracker

Similar to MOSSE, which exploits the correlation filter to predict the target's location, Discriminative Scale Space Tracker (DSST) comes with two major improvements. Firstly, DSST estimates target's translation by learning a *translation filter* on multiple feature channels while MOSSE tracker can only be operated on a single channel. Second, it learns a *scale filter* that estimates the target's scale explicitly, which is not directly tackled by MOSSE tracker. In the following sections, we first in 2.2.2.1 describe how to incorporate multi-channel features into translation filter, and in 2.2.2.2 describe how to learn a scale filter. Finally, in 2.2.2.3 we put them all together and describe how those filters are involved in the tracking process.

2.2.2.1 Discriminative Correlation Filters for Multi-channel Templates

For incorporating a multi-channel target's template (i.e. multidimensional features), the objective that learns a single-channel filter in (2.10) is adapted to

$$\{h^{*^1}, ..., h^{*^D}\} = \underset{\{h^1,...,h^D\}}{\arg\min} \sum_{d=1}^{D}\sum_{i=1}^{N} ||f_i^d \star h^d - g_i||_2^2 + \lambda \sum_{i=1}^{N}\sum_{d=1}^{D} ||h_i^d||_2^2,\ h^d, f^d, g \in \mathcal{R}^{M\times K}, \tag{2.16}$$

where λ here denotes the regularization parameter, M and K denote the width and height of h_i^d, f^d, and g. d is the subscript that indicates the d^{th} feature channel of template f_i^d and filter h^d. With the objective in (2.16), D filters $h^{*^1}, ..., h^{*^D}$ are learned and they regress $f_i^1, f_i^2, ..., f_i^D$ to the same output g_i for all $i = 1, ..., N$. Likewise, filters can be learned in Fourier domain with the correlation theorem in (2.16), and their closed-form solutions are:

$$H_0^d = \frac{A_0^d}{B_0^d} = \frac{\sum_{i=1}^{N} G_i \odot \bar{F}_i^d}{\sum_{d=1}^{D}\sum_{i=1}^{N} F_i^d \odot \bar{F}_i^d + \lambda}, \forall d = 1, ..., D. \tag{2.17}$$

The subscript 0 and superscript d in H_0^d mean that H_0^d is the initial filter learned for the feature at d^{th} dimension. In frame t, the target's location can be estimated within an rectangle area $\mathcal{Y}$ through identifying the maximal value in the average response map R across the channels, where R is calcalated as

$$R = \mathcal{F}^{-1}\left\{\frac{\sum_{d=1}^{D} A_{t-1}^d \odot F_{\mathcal{Y}}^d}{B_{t-1}^d + \lambda}\right\}, \tag{2.18}$$

where $F_{\mathcal{Y}}^d$ denotes the d^{th} channel of the template $F_{\mathcal{Y}}$ from $\mathcal{Y}$. Once the target's location has been estimated, we extract the D-dimensional template centered around at its location in the Fourier domain, i.e. $F_t = \{F_t^1, F_t^2, ..., F_t^D | F_t^d \in \mathcal{R}^{M\times K}, \forall d = 1, ..., D\}$, and generate the desired output $G_t \in \mathcal{R}^{M\times K}$ to update the filters learned in frame $(t-1)$ through (2.19).

$$\begin{aligned} H_t^d &= \frac{A_t^d}{B_t^d}, \\ A_t^d &= \eta G_t \odot \bar{F}_t^d + (1-\eta) A_{t-1}^d, \\ B_t^d &= \eta F_t^d \odot \bar{F}_t^d + (1-\eta) B_{t-1}^d, d = 1, ..., D, \end{aligned} \tag{2.19}$$

where η is the filter's update rate. The new target's state can be estimated within an rectangle area $\mathcal{Y}$ through identifying the maximal value in the average response map, R, across the channels, where

$$R = \mathcal{F}^{-1}\left\{\frac{\sum_{d=1}^{D} A_t^d \odot F_{\mathcal{Y}}^d}{B_t^d + \lambda}\right\}. \tag{2.20}$$

So far, MOSSE has been extended to handle multi-channel features which allow DSST to fuse different features such as Histogram of Oriented Gradients (HoG) and grayscale pixel values to learn the filters which model the target patches. These filters are termed in [12] as the translation filters as they only estimates the translation of the target, but not scale. In the next section, we discuss how DSST estimates the change in the target's scale.

2.2.2.2 Scale Estimation

DSST constructs another correlation filter, i.e. the scale filter, to estimate the target's scale in the current frame. To achieve this, it constructs a scale-space pyramid of patches around the target on a fixed set of scaling ratios $r_1, r_2, ..., r_S$. Each patch (of different sizes) is converted to a fixed-length D-dimensional vector built on HoG features, where $s = 1, ..., S$ represents S scales. To construct the training samples for learning a scale filter, DSST forms a matrix $U \in \mathcal{R}^{S\times D}$ where each row comes from one D-dimensional HoG features it just calculated. Afterwards, it constructs D training samples $\mathbf{v}^1, \mathbf{v}^2, ..., \mathbf{v}^D$ for learning D scale filters where each sample $\mathbf{v}^d \in \mathcal{R}^S, d = 1, ..., D$, is one column extracted from U. The desired output, $\mathbf{g} \in \mathcal{R}^S$, is generated by a one-dimensional Gaussian peaked at the central position. Given the training samples and the desired output, D scale filters can be learned with (2.21), where G and V^d are the counterparts of g and $\mathbf{v}^d$ in the Fourier domain, respectively. Similar to (2.20) and (2.19), the filter response can be calculated via (2.22) and the scale filters can be updated with (2.23):

$$H_0^d = \frac{A_0^d}{B_0^d} = \frac{G \odot \bar{V}^d}{\sum_{d=1}^{D} V^d \odot \bar{V^d} + \lambda}, \forall d = 1, ..., D, \; H_0^d \in \mathcal{R}^S. \tag{2.21}$$

$$R = \mathcal{F}^{-1}\left\{ \frac{\sum_{d=1}^{D} A_{t-1}^d \odot F_{\mathcal{Y}}^d}{B_{t-1}^d + \lambda} \right\}, \; R \in \mathcal{R}^S. \tag{2.22}$$

$$\begin{aligned} H_t^s &= \frac{A_t^s}{B_t^s}, \\ A_t^s &= \eta G^s \odot \bar{V}_t^s + (1-\eta) A_{t-1}^s, \\ B_t^s &= \eta V_t^s \odot \bar{V}_t^s + (1-\eta) B_{t-1}^s, \forall s = 1, ..., S. \end{aligned} \tag{2.23}$$

2.2.2.3 Tracking with Translation and Scale Filters

Here we describe how DSST utilizes the translation filter (desrcibed in 2.2.2.1) and the scale filter (described in 2.2.2.2) to perform tracking. Given $\mathbf{p}_{t-1}$, the previous target location, and s_{t-1}, the previous estimated scale of the target, we would like to estimate $\mathbf{p}_t$ and s_t. Firstly, training samples $f_{t,\texttt{trans}}$ at $\mathbf{p}_{t-1}$ and at scale s_{t-1} are extracted for training the translation filter $h_{t,\texttt{trans}}$. The response $R_{t,\texttt{trans}} \in \mathcal{R}^{M\times K}$ can be calculated through (2.20) and $\mathbf{p}_t$ can be set to the location with maximal response on $R_{t,\texttt{trans}}$. Secondly, training samples $f_{t,\texttt{scale}}$ at $\mathbf{p}_t$ and at scale s_{t-1} are extracted for training the scale filter $h_{t,\texttt{scale}}$. Likewise in estimating the translation filter, the response $R_{t,\texttt{scale}} \in \mathcal{R}^S$ can be calculated through (2.22) and s_t can be estimated by seeking the location with maximal response on $R_{t,\texttt{scale}}$. Finally, both filters are updated with the templates at location $\mathbf{p}_t$ and scale s_t through (2.19) and (2.23), respectively.

2.3 Online Multiple Object Tracker

A multiple object tracker (MOT) typically has to handle a number of difficulties. Firstly, the identity switches, i.e. the situation in which the tracker mistakes another target for the one it is supposed to track, may happen due to the presence of other objects similar in their appearances. This situation happens frequently when the test videos are taken from the street view and other public spaces where the pedestrians wearing clothes with similar colors and styles and/or are highly occluded by each other. Secondly, because of the constraint of online methodology which cannot peek into future frames, the *tracklet*, i.e. the tracking trajectory on the same target, may be fragmented due to some tracking errors such as identity switches. Thirdly, if an object has been occluded for quite some time and re-appears, the tracker should be able to recognize and start tracking it again. To address these difficulties, Nicolai Wojke et al. [36] proposed an online multiple object tracking framework which incorporates an object detector, Kalman filter as the base tracker, and a data association method which is based on the features learned from a deep neural net to associate the results from detectors and trackers. We describe their algorithm in further details in the following section.

2.3.1 MOT with Kalman Filter and Deep Assoication Matrix

As the tracking-by-detection methodology has been shown promising in tracking single object over the past decade, the framework proposed in [36] applies a similar methodology, but adapts it for multiple object tracking. Their detection-tracking flow is illustrated in Figure 2.5. A data association method is required for associating the detection and tracking results. To better understand why and how data association is required, we take Figure 2.6 as an illustration. At each frame, the method runs the object detector to gather the bounding boxes of the objects, and the tracker to estimate the state of currently tracked objects. (Note: the tracker used in [36] is Kalman filter [8], which will not be introduced because we do not use it in this thesis.) The association comes in to find the correspondence of the detected and tracked objects as shown in Figure 2.6a.

Specifically to associate the detected and tracked objects, a popular approach has been suggested among recent MOT works [36]. Firstly, an association matrix $C = c_{i,j} \in \mathcal{R}^{I \times J}$, where each entry, $c_{i,j}$ defines the association cost of object i (from the tracker) and object j (from the detector), is firstly constructed. I and J denote the numbers of detected and tracked objects, respectively. Secondly, the set of pairs of objects $\mathcal{I} = \{(k,n) | x_{k,n} = 1; \forall k = 1, ..., I, n = 1, ..., J\}$ which minimizes the total association cost, i.e.

$$\texttt{association cost} = \sum_{i=1}^{I} \sum_{j=1}^{J} c_{i,j} x_{i,j}, \tag{2.24}$$

such that

$$\begin{aligned} &\sum_{j=1}^{J} x_{i,j} = 1, \quad (i = 1, ..., I) \\ &\sum_{i=1}^{I} x_{i,j} = 1, \quad (j = 1, ..., J) \\ &x_{i,j} \in \{0, 1\}, \forall i, j, \end{aligned} \tag{2.25}$$

can be found by the Hungarian algorithm [10]. It is worth noting that an association matrix can be designed based on different similarity measurements on heterogeneous information such as the motion and appearance of the objects. In [36], $c_{i,j}$ is calculated via

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1 - \lambda) d^{(2)}(i,j), \tag{2.26}$$

where $c_{i,j}$ is controlled by two terms, $d^{(1)}(i,j)$, $d^{(2)}(i,j)$, and λ is used to balance the two terms. The first term $d^{(1)}(i,j)$ is to measure the discrepancy between the predictions made by detector and the tracker (i.e. the Kalman filter). Assuming that the bounding box's state obtained from detector is $\mathbf{d}_j$ and the bounding box's state and corresponding covariance estimated by the tracker are $\mathbf{y}_i$ and $\mathbf{S}_i$, $d^{(1)}(i,j)$ is defined as

$$d^{(1)}(i,j) = (\mathbf{d}_j - \mathbf{y}_i)^T \mathbf{S}_i^{-1} (\mathbf{d}_j - \mathbf{y}_i). \tag{2.27}$$

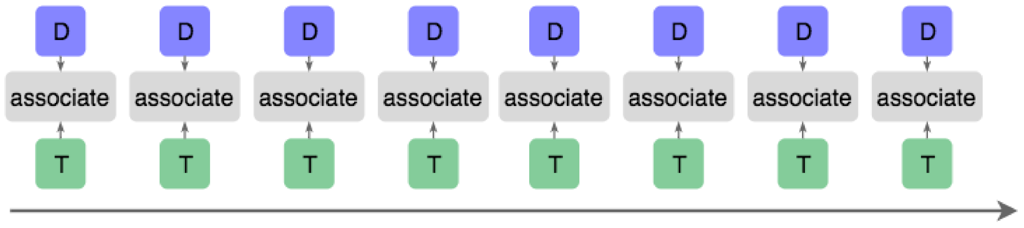


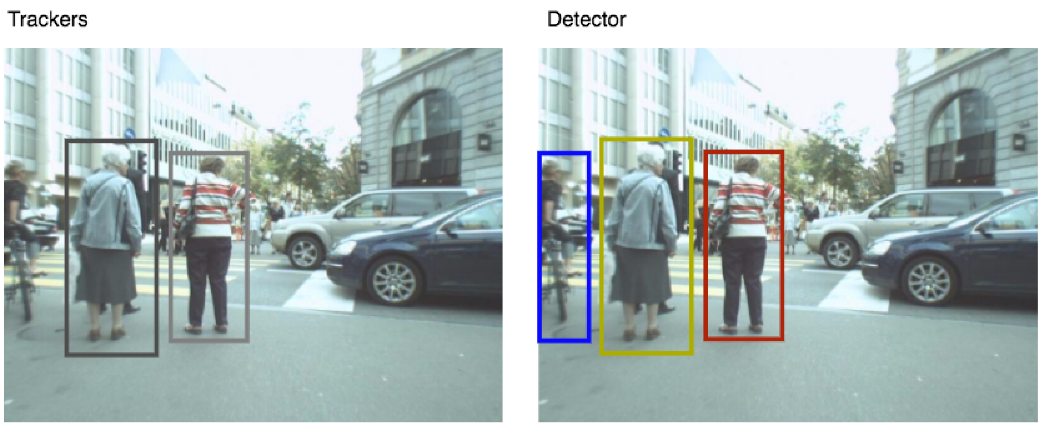
Figure 2.5: The detection-tracking flow proposed in [36]. "D" in the figure indicates that the detector is triggered at the time frame, and "T" indicates the tracker is triggered.

(2.27) is actually the squared Mahalanobis distance between the Kalman state of the i-th object and the state of the j-th object predicted by the detector. The term $d^{(2)}(i,j)$ is to measure the dissimilarity in appearance between the j-th detection and the history of the i-th track, i.e.

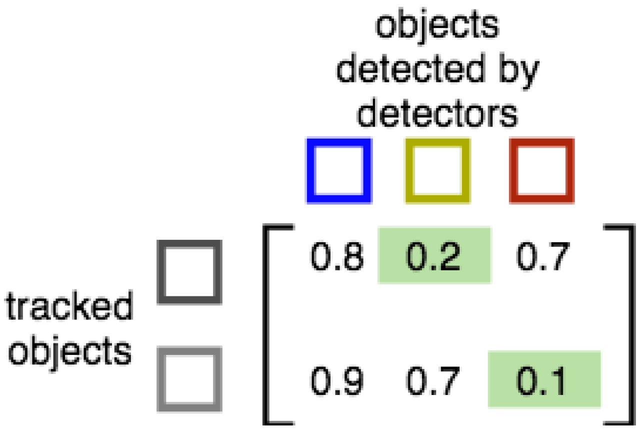
$$d^{(2)}(i,j) = \min\{1 - \mathbf{r}_j^T \mathbf{r}_k^{(i)} | \mathbf{r}_k^{(i)} \in \mathcal{R}_i\}, \tag{2.28}$$

where $\mathcal{R}_i = \{\mathbf{r}_k^{(i)}\}_{k=1}^{L}$ is the set of the past L (e.g., 100) visual descriptors $\mathbf{r}_k^{(i)}$ associated with track i along the frames, where $||\mathbf{r}_k^{(i)}|| = 1$. Proven to be a rich and informative feature descriptor, descriptor $\mathbf{r}$ is built with extracting the output values of a fully-connected layer in a convolutional neural network (CNN) [36].

One major assumption of the MOT described above is that it assumes the detector is always available in every frame. The framework proposed in this thesis and to be described in the next section does not strictly follow this assumption. We aim to propose a framework which utilizes the detector's output periodically, not assuming the availability of the object detector in every frame for accelerating the detect-and-track process.



(a) The data association scenario.



(b) The data association matrix.

Figure 2.6: Illustration of data association in action.

Chapter 3

The Proposed Detect-and-Track Framework

In this chapter, we describe the proposed framework for multiple object tracking in detail. Our primary object of interest in this work is *pedestrian*, and we do not impose any assumptions on the target object's size, aspect ratio, or appearance. Thereby, it is possible that the proposed framework can be extended to different object classes. In general, we follow the framework proposed in [36] but with few major adaptions. First, we replace the Kalman tracker in [36] with DSST tracker [12]. The rationale behind this is that when updating the Kalman state, one has to provide the measurement made in the current frame, which is, in their case, the measurement from object detector. Without the measurement from the object detector, the Kalman tracker would update the state merely with the pre-modeled linear motion [8]. As in our case, we do not assume the availability of the object detector in every frame, hence updating the Kalman tracker with only motion prediction may result in unsatisfatory result. Second, we do not train an object detector specifically as in [36], but we employ the object detector from [3] trained on MS COCO dataset [25] where it provides multiple detectors of different base networks (i.e. MobileNet V1 [20], InceptionV2 [33], RFCN [11], Faster RCNN [28]) that tradeoff the speed and accuracy [21]. Third, in order to monitor if a tracker starts to drift or has drifted, we measure the similarity between the patches of the tracked target in any two consecutive frames. Fourth, to enable the tracker recover from tracking failure, we employ a simple person re-identification method that is as well based on the same deep features. In the pursuit of a more efficient implementation, the similarity is measured based on the deep features extracted from the network that has been served as the base network in the object detector in use. These modifications enable the proposed framework to detect and track the object

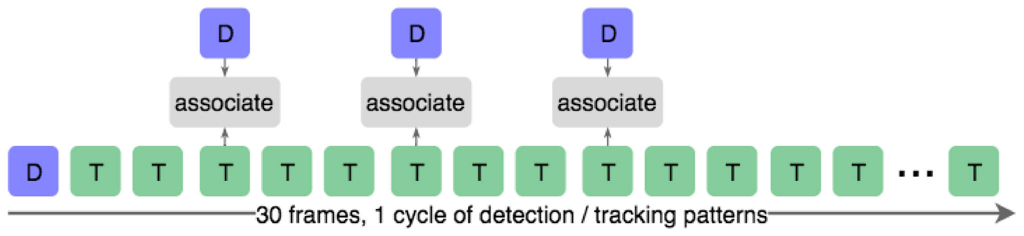


Figure 3.1: The proposed detection-tracking flow. "D" in the figure indicates that the detector is triggered at the time frame, and "T" indicates the tracker is triggered.

without triggering the object detector in every frame but only periodically.

3.1 Detect-and-Track Flow

Figure 3.1 provides a high-level detection-tracking flow of the proposed MOT framework. On the one hand, we would like to trigger the detector as infrequently as possible as running the detector and the tracker at the same time induces computational burden for the system. On the another hand, the detector should be triggered for enough times to provide sufficient measurement for calibrating the tracker. This also implies that it is critical to design a framework not sensitive to the change in the number of times that the detector is triggered within a certain number of frames.

3.2 Multiple Object Tracking with Correlation Tracker and Deep Features

In this section, we introduce the proposed detect-and-track framework by breaking it down into several constituents. Figure 3.2 provides a high-level description of the framework. The system maintains two types of tracks, the active tracks and the history tracks. To begin with, each DSST tracker is responsible for tracking a single target. In each frame, each DSST tracker updates its target's location while the object detector detects the presence of the target if the detector should be triggered according to the user configuration. Next, the association process takes as input the results returned from the DSST trackers and the object detector to associate the objects. The objects that are not associated can be considered as new objects if they are not re-identified as one of the historic objects. Finally, the poorly tracked tracks are removed and moved to the history tracks while the associated objects still remain in the active tracks. In the frames in which the detector is not triggered, the DSST trackers are updated alone and there is no association process. The procedures are detailed in the pseudocode provided in Algorithm 1. In the following sections, we introduce the key components of the framework.

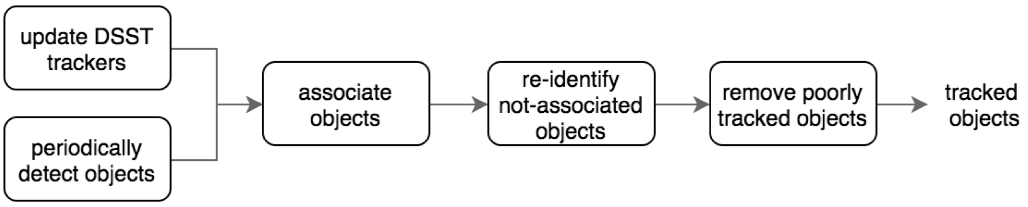

update DSST trackers
periodically detect objects
associate objects
re-identify not-associated objects
remove poorly tracked objects
tracked objects

Figure 3.2: High-level block diagram of the proposed framework.

3.2.1 Object Detector

Recently there have been releases of varing combinations of object detectors (i.e. Faster R-CNN [28], RFCN [11], SSD [26]) and the base networks imposed (i.e. MobileNetV1, InceptionV2, ResNet101, Inception-ResNet) [3, 21]. As we are pursuing the (near) real-time performance of the whole system, we select the SSD object detector with InceptionV2 as our base object detector. We extract *person* class from the detector and omit the rest of the classes. The network structure described in Table 3.1 replaces the VGG network in SSD. Later on, when it comes to updating the tracker, the framework re-uses InceptionV2 as the base network to extract the features from the targets as their *auxiliary templates*, which is introduced in Section 3.2.2.2. From the implementation point of view, repeatedly using the same base network avert us from reallocating the resources (i.e. CPU / GPU memories and the time spent on constructing symbolic graph in the deep learning framework such as Tensorflow) that have been allocated. Thereby, even though in principle, the choices of the base network for object detector and the generic feature extraction can be different, we opt to have a shared base network among the tasks to allow a more efficient implementation.

Typically the final layer of these object detectors is a soft-max layer where each neuron's output is non-negative and the neurons are normalized to be summed to 1.0. Thus, those outputs can be interpreted as the probability distribution of each object class of the object proposals. An object proposal is classified as one of the classes (i.e. in our case, *pedestrian*) if the probability of being that class is over a threshold τ^{det}. Note that different values of τ^{det} lead to varying recall and precision of the detector. This is discussed later in the next chapter.

Table 3.1: Base network structure of InceptionV2 [2]. The classification layers have been removed as we adopt the network as a generic feature extraction, hence the classification layers are not needed. Please note that the input size and the structure are different from what is described in [33]. Our implementation follows the one provided in [2]. Figure 3.3a to 3.3j can be seen on page 38 to 39.

name	layer	filter size / stride	input size
`conv2d_1a_7x7`	conv	7×7 / 2	$224 \times 224 \times 3$
`maxPool_2a_3x3`	max-pool	3×3 / 1	$112 \times 112 \times 64$
`conv2d_2b_1x1`	conv	1×1 / 1	$56 \times 56 \times 64$
`conv2d_2c_3x3`	conv	3×3 / 1	$56 \times 56 \times 64$
`maxpool_3a_3x3`	max-pool	3×3 / 1	$56 \times 56 \times 192$
`mixed_3b` (Fig. 3.3a)	Incep.	-	$28 \times 28 \times 192$
`mixed_3c` (Fig. 3.3b)	Incep.	-	$28 \times 28 \times 256$
`mixed_4a` (Fig. 3.3c)	Incep.	-	$28 \times 28 \times 320$
`mixed_4b` (Fig. 3.3d)	Incep.	-	$28 \times 28 \times 576$
`mixed_4c` (Fig. 3.3e)	Incep.	-	$14 \times 14 \times 576$
`mixed_4d` (Fig. 3.3f)	Incep.	-	$14 \times 14 \times 576$
`mixed_4e` (Fig. 3.3g)	Incep.	-	$14 \times 14 \times 576$
`mixed_5a` (Fig. 3.3h)	Incep.	-	$14 \times 14 \times 576$
`mixed_5b` (Fig. 3.3i)	Incep.	-	$7 \times 7 \times 1024$
`mixed_5c` (Fig. 3.3j)	Incep.	-	$7 \times 7 \times 1024$

3.2.2 Update of Trackers

Three major steps are involved in updating the trackers, (1) update the DSST tracker, (2) update the auxiliary templates of the targets to track, and (3) update the auxiliary templates with adaptive learning rate. As described in 2.2.2.1, DSST seeks the location with maximal response as the final prediction of the target location. The maximal response (or regression score) can, in the one hand, be interpreted as how confident the tracker is, but in the other hand, the scores are positive numbers which are not strictly bounded within a range, e.g. $[0, 100]$ or $[0, 1]$. This makes the response difficult to be interpreted and served as a reliable measurement.

3.2.2.1 Extraction of Auxiliary Templates and Similarity Measurement

We introduce *auxiliary templates* and the similarity measurement of those templates. Auxiliary templates do not interfere the update of the DSST tracker, but provides the observation to monitor whether the DSST tracker does lose its target or not. This is achieved by incorporating a similarity measurement (which ranges in $[0, 1]$) based on deep features extracted from a deep neural net as the auxiliary templates of the targets. As suggested in [13], features from lower layers, in which the resolution of corresponding feature maps are higher, are empirically found performing better in the tracking task.

Hence, to extract features that describe well the target object, we extract the feature maps $f(I^t)$ of size $73 \times 73 \times 64$ from the output of the max-pooling layer, `maxpool_3a_3x3`, where I^t denotes t-th frame. Next, we would like to extract the features for the i-th target that lies within the bounding box $\mathbf{b}_i^t = \{x_i^t, y_i^t, w_i^t, h_i^t\}$, where x_i^t and y_i^t denote the normalized coordinates (with respect to the image width and height) from the top left corner. w_i^t and h_i^t denote the normalized width and height (with respect to the image width and height) of the bounding box, respectively. As $\mathbf{b}_i^t$ is represented in the normalized coordinates, one can easily find the projected coordinates of $\mathbf{b}_i^t$ on $f(I^t)$ in which the feature maps $\mathbf{F}_i^t = \{F_{i,c}^t\}_{c=1}^{C_f}$ for $\mathbf{b}_i^t$ are obtained, where each $F_{i,c}^t$ is of size $W_f \times H_f$, C_f is the number of channels and $f(\cdot)$ denotes the feature map extration function. What is worth noting here is, as each bounding box $\mathbf{b}_i^t$ can be of different size, the feature maps $\mathbf{F}_i^t$ has been normalized to a fixed spatial resolution $W_f \times H_f$, i.e. $W_f = 24, H_f = 24$. Next, we define the similarity between $\mathbf{F}_i^t$ and $\mathbf{F}_i^{t-1}$ by averaging the cosine similarity channel-wise, i.e.

$$\begin{aligned}\texttt{sim}(\mathbf{F}_i^t, \mathbf{F}_i^{t-1}) &= \frac{1}{C_f}\sum_{c=1}^{C_f} \cos(v(\mathbf{F}_{i,c}^t), v(\mathbf{F}_{i,c}^{t-1})) \\ &= \frac{1}{C_f}\sum_{c=1}^{C_f} \frac{v(\mathbf{F}_{i,c}^t)\cdot v(\mathbf{F}_{i,c}^{t-1})}{||v(\mathbf{F}_{i,c}^t)||_2 ||v(\mathbf{F}_{i,c}^{t-1})||_2},\end{aligned} \tag{3.1}$$

where $v(\mathbf{X})$ denotes the flattened vector of any matrix $\mathbf{X}$. (3.1) captures how similar the two targets in the bounding boxes $\mathbf{b}_i^t$ and $\mathbf{b}_j^t$ are on their feature maps.

3.2.2.2 Adaptive Update of Auxiliary Templates

A common choice of updating the auxiliary templates is via a static update with respect to each channel in $\mathbf{F}_i^t$ with a fixed learning rate $\gamma \in [0, 1]$, i.e.

$$\mathbf{F}_i^t = \gamma \mathbf{F}_i^t + (1 - \gamma)\mathbf{F}_i^{t-1}. \tag{3.2}$$

However, the static update strategy may under or over update the target's template when a tracker starts to drift. For instance in Figure 3.4, the tracker starts to drift and no longer tracks the target with a satisfactory precision. This results in the target's auxiliary template $\mathbf{F}_i^t$ being updated with much background information instead of pixels from the target itself. Furthermore, if a tracker completely drifts to some static area in the background, but its auxiliary template is still updated with fixed learning rate, $\texttt{sim}(\mathbf{F}_i^t, \mathbf{F}_i^{t-1})$ can always retain a rather high value and is no longer a diagnostic to tracker's drifting. The described situation is illustrated in Figure 3.5, which clearly shows that the tracker is stuck at the background object.

To alleviate the over-learning issue, we provide an adaptive approach to update the learning rate according to the current similarity measurement as follows. To illustrate, as shown in Figure 3.6, $\texttt{sim}(\mathbf{F}_i^t, \mathbf{F}_i^{t-1})$ inclines to decrease if a tracker starts to drift away from the target. When $\texttt{sim}(\mathbf{F}_i^t, \mathbf{F}_i^{t-1})$ is of high value, although it could be that the tracker is still accurately on target, it could as well be that the tracker has drifted and started tracking the static object in the background from which it consistently updates the auxiliary templates to obtain the high similarity score. With these observations, we would like update $\mathbf{F}_i^t$ to keep $\texttt{sim}(\mathbf{F}_i^{t+1}, \mathbf{F}_i^t)$ high enough if the tracker is still on target, but to update $\mathbf{F}_i^t$ less aggressively if it is already high to avert over updating.

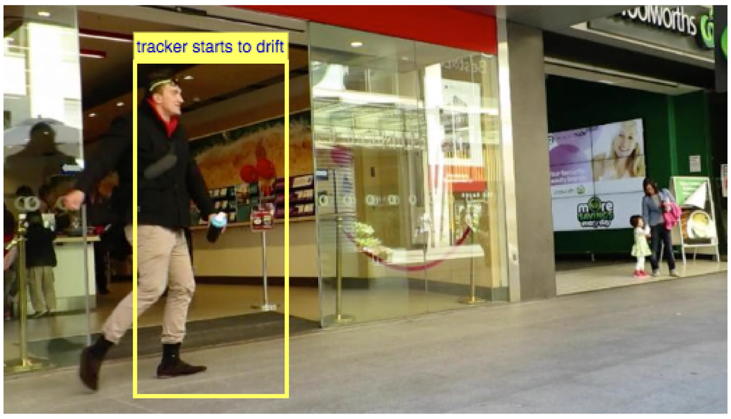


Figure 3.4: The target's auxliary template may be updated with wrong image content when a tracker starts to drift since much background information is included within the tracker's bounding box.

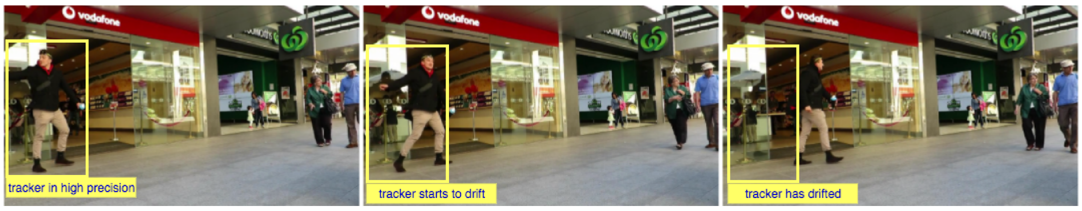


Figure 3.5: When a tracker starts drifting away from the target, it is likely to be stuck at a background object but still recognizes it as the target to track so the auxiliary template is kept updating with some learning rate. Under that case, $\mathtt{sim}(\mathbf{F}_i^t, \mathbf{F}_i^t)$ will retain high value and keep increasing until it saturates to a rather high value.

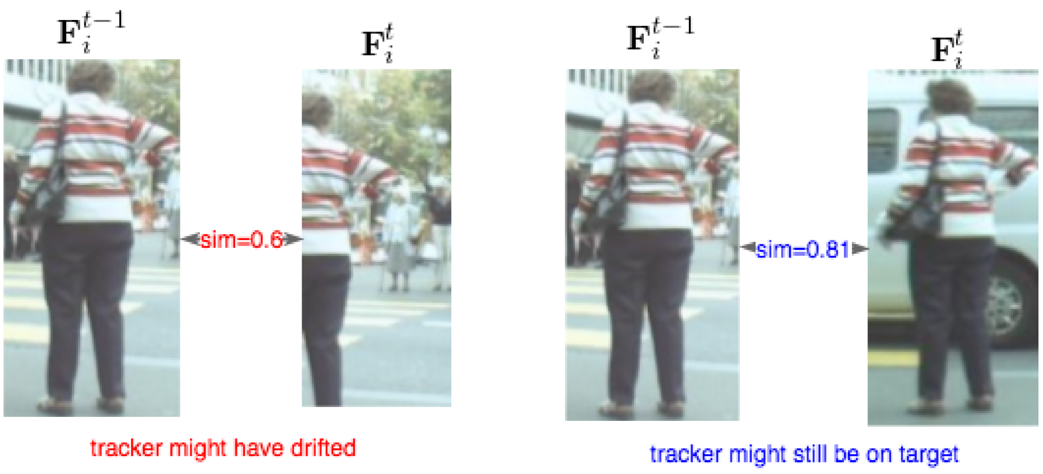


Figure 3.6: Adjusting the strategy of updating the target's auxliary template according to similarity measurement between $\mathbf{F}_i^t$ and $\mathbf{F}_i^{t-1}$.

To implement this idea, we generate an adjusting rate $\lambda \in [0, 1]$ that adjusts the base learning rate γ according to $s_i^t = \mathtt{sim}(\mathbf{F}_i^t, \mathbf{F}_i^t)$ before updating $\mathbf{F}_i^t$. Given a Gaussian function $\mathcal{N}(\cdot)$ parameterized by μ_x, σ_x^2, λ is calculated by

$$\lambda = \mathcal{N}(\mu = s_i^t|\mu_x, \sigma_x^2)/\mathcal{N}(\mu = \mu_x|\mu_x, \sigma_x^2), \tag{3.3}$$

where $\mathcal{N}(\mu = s_i^t|\mu_x, \sigma_x^2) = e^{-\frac{-||s_i^t - \mu_x||}{2\sigma_x^2}}$, $\mathcal{N}(\mu = \mu_x|\mu_x, \sigma_x^2) = e^{-\frac{-||\mu_x - \mu_x||}{2\sigma_x^2}}$, and the adaptive learning rate $\gamma_{\mathtt{adapt}}$ is given by

$$\gamma_{\mathtt{adapt}} = \lambda \cdot \gamma. \tag{3.4}$$

Note that the two Gaussian functions above are not probabilistic but deterministic. The update of $\mathbf{F}_i^t$ can be performed channel-wise by

$$\mathbf{F}_i^t = \gamma_{\mathtt{adapt}} \mathbf{F}_i^t + (1 - \gamma_{\mathtt{adapt}}) \mathbf{F}_i^{t-1}. \tag{3.5}$$

Figure 3.7 illustrates the case in which, given the pre-defined parameter $\mu_x = 0.8$, the adjusting rate λ is peaked when s_i^t is at $\mu_x = 0.8$. This leads to the highest value that $\gamma_{\mathtt{adapt}}$ can reach and most aggressive update of $\mathbf{F}_i^t$. When s_i^t is away from $\mu_x = 0.8$ (e.g. 0.7 or 0.9), indicating that the tracker might have lost the target (as the left image in Figure 3.6) or the tracker is accurately on target (as the right image in Figure 3.6), the corresponding λ is close to zero and so is $\gamma_{\mathtt{adapt}}$. One can observe, on one hand, the this strategy would cause the drop in s_i^{t+1} from s_i^t, however, if the tracker is still on the target sharply, s_i^{t+1} would not drop dramatically even $\mathbf{F}_i^t$ was not updated at the previous frame since the target's appearance would not adapt much. In the meanwhile, when s_i^{t+1} drops from s_i^t, e.g. 0.9 to 0.85, the adjusting rate λ gets increased accordingly, and the update becomes aggressive again. On the other hand, if s_i^t has been already low (e.g. 0.7), the tracker might have lost the target, resulting in λ close to zero and hence nearly no update on $\mathbf{F}_i^t$. As a result, if a tracker has been always on target, $\{s_i^t\}_t$ over time t are controlled within a range of values that would not go either too high or low.

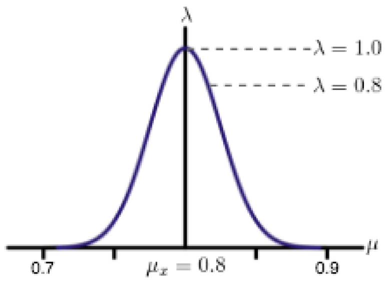


Figure 3.7: The illustration of the function that generates adjusting rate λ for the adaptive learning rate $\gamma_{\texttt{adapt}}$. Given the pre-defined parameter $\mu_x = 0.8$, the adjusting rate λ is peaked when s_i^t is at $\mu_x = 0.8$. When s_i^t is away from $\mu_x = 0.8$ (e.g. 0.7 or 0.9), indicating that the tracker might have lost the target (as the left image in Figure 3.6) or the tracker is accurately on target (as the right image in Figure 3.6), corresponding λ is close to zero and so is $\gamma_{\texttt{adapt}}$.

3.2.3 Data Association

Two major steps in the framework involves data association: (1) associating the detected and tracked objects, and (2) associating the detected objects and those in the history tracks for re-identifying the objects once in the active tracks. As described in Section 2.3.1, it requires a data association matrix defining the cost of two object being associated. Once a data association matrix is calculated, the Hungarian algorithm can be utilized to solve the linear assignment problem that minimizes the association cost.

The proposed data association matrix is based on three measurements: (a) the dissimilarity of two objects measured on the deep features, (b) the misalignment of the centers of the two objects, and (c) the IoU of the two objects' bounding boxes. We define each entry $\mathcal{M}[m,n]$ in the association matrix $\mathcal{M}$ as

$$\mathcal{M}[m,n] = (1-\lambda_M) \cdot \mathcal{M}_c[m,n] + \lambda_M \cdot \mathcal{M}_f[m,n], \tag{3.6}$$

where $\mathcal{M}_c[m,n]$ is defined based on the measurements (a) and (b), while $\mathcal{M}_f[m,n]$ is based on (c). Given the coordinates of two objects' bounding boxes $\mathbf{b}_m$, $\mathbf{b}_n$ and their centers (c^x_m, c^y_m), (c^x_n, c^y_n), respectively, λ_M controls the weighting between two costs. $\mathcal{M}_c[m,n]$ is calculated via

$$\mathcal{M}_c[m,n] = \frac{1}{2}(||(c^x_m, c^y_m) - (c^x_n, c^y_n)||_2 / l_{diag}) + \frac{1}{2}(1 - \texttt{IoU}(\mathbf{b}_m, \mathbf{b}_n)), \quad (3.7)$$

where l_{diag} denotes the diagonal length of the video frame. Next, given the feature maps of two objects $\mathbf{F}_m$ and $\mathbf{F}_n$, $\mathcal{M}_f[m,n]$ can be defined as,

$$\mathcal{M}_f[m,n] = 1 - \texttt{sim}(\mathbf{F}_m, \mathbf{F}_n), \quad (3.8)$$

where $\texttt{sim}(\cdot)$ is defined in (3.1). Likewise in [36], we define a threshold T_M to say the association between m-th and n-th objects is *admissible* if $\mathcal{M}[m,n] < T_M$. The association of two objects is not considered valid if not admissible even if they are assigned to each other by solving the linear assignment problem.

3.2.4 Re-Identification or Removal of Tracks

Once the object detector detects an object, we check in the following order whether the object should be (1) associated with a tracked object (described in Section 3.2.3), (2) associated with an object in the history tracks, or (3) treated as a new object in the active tracks. To check if the newly detected can be associated with any object in the history tracks and re-identify the corresponding track, we run the data asssociation process between the newly detected object and all the objects in the history tracks with a different data association matrix $\mathcal{M}_d$, where $\mathcal{M}_d = \mathcal{M}_f$. That is, for the re-identification purpose, we simply consider the dissimilarity between the feature maps of the objects since the displacements and scales of not actively tracked objects can vary much. Since $\mathcal{M}_f$ is constructed on the feature maps whose sizes are normalized to be fixed (see Section 3.2.2.1 for details), it can withstand a change in the scale to some certain extent. Likewise in other data association process, a threshold T_d is used to define if the association is admissible. If the newly detected object is not associated with any objects in any track, it will be added to the active track as a new object.

3.3 Implementation of The Framework

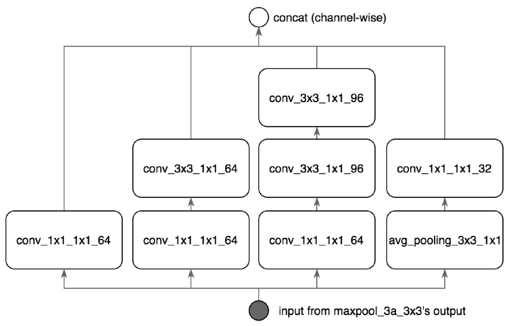
To ensure a reproducible implementation and provide all the bells and whistles in the proposed framework, we encapsule it in this section with the pseudocodes presented in Algorithm 1. The details of the main steps in Algorithm 1 are presented in Algorithm 2 to Algorithm 7.

The most outer loop of Algorithm 1 (`line 1`) loops over the video frames, and it is followed by mainly an if-else block which executes conditionally on whether the detector is triggered in the t-th frame. If there exists objects in the active tracks, i.e. $|\mathcal{O}^t| > 0$, and $t > 1$, every tracker is updated for every object in $\mathcal{O}^t$ firstly prior to the actions that follow (`line 2`, and see update details in Algorithm 2). If the detector is triggered (`lines 4-16`) and detects any object, the objects in $\mathcal{O}^t$ and $\mathcal{O}^{hist}$ (i.e. the history tracks) are checked whether they can be associated (`lines 5-6`, and see Algorithm 3 and Algorithm 4 for further details on data association). Those non-associated objects from the detector are denoted as $\mathcal{O}^{new} = \mathcal{O}^{det} \backslash \{\mathcal{O}^{det}_{k_j}\}_j$, $\forall j$ (`line 7`). As they are treated as the new and unseen objects so far, their p_i^{new} (how many times an object has been observed tracked or detected in the observation span, T_i^{new}) and its observation span T_i^{new} are initialized to be 1 (`line 8`).

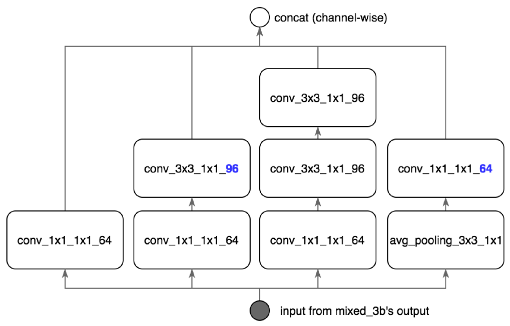
For objects in $\mathcal{O}^t$, we simply increment their observation span, T_i^t (`line 9`). For those associated active tracks (`lines 10-14`), their $p_{l_j}^t$ is incremented by 1 to account for being successfully associated (`line 11`), and their bounding boxes and feature maps can be updated with those provided by the detector, i.e. bounding box, $\mathbf{b}_{k_j}^{det}$, and feature maps of the object, $\mathbf{F}_{k_j}^{det}$ (`lines 12-13`). The updating rate, λ_{conf}^{det}, depends on how confident the detector is of the object.

The objects in $\mathcal{O}^{new}$ are required to re-identify themselves from the history tracks $\mathcal{O}^{hist}$ (`line 15` and Algorithm 6). In the case where any of them is re-identified, the corresponding track is recovered from the history and added back to the active tracks once again (`line 16`).

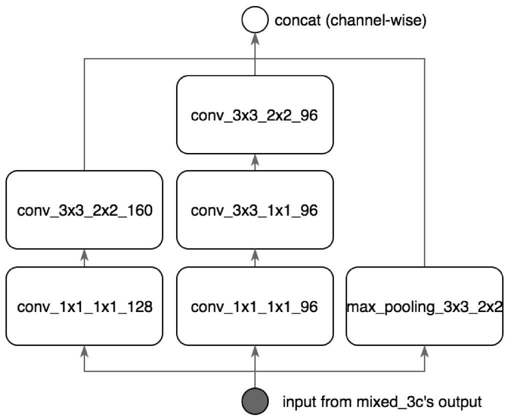
If the detector is not triggered at this frame, we simply increment p_i^t and T_i^t of i-th object in the active tracks (`line 18`). Finally, poorly tracked objects are moved into the history tracks (`line 20`, see details in Algorithm 7) while the objects that stay in the history tracks for over a pre-defined threshold are rooted out permanently from the history tracks (`line 21`).



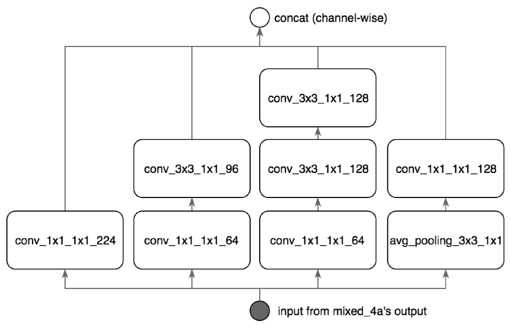
(a) mixed_3b structure in Table 3.1



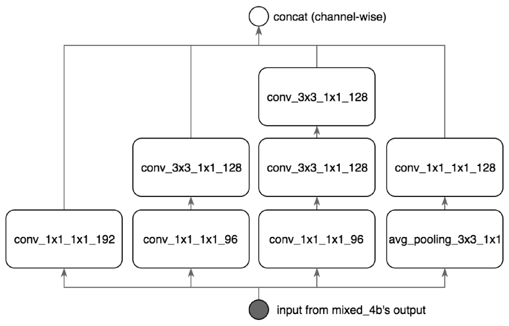
(b) mixed_3c structure in Table 3.1



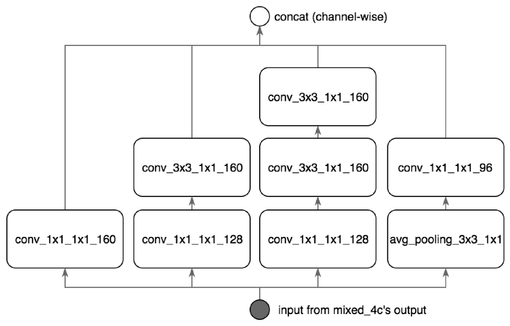
(c) mixed_4a structure in Table 3.1



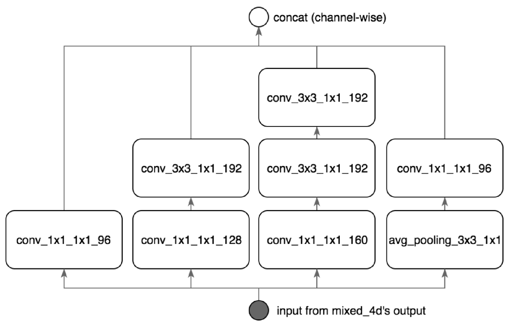
(d) mixed_4b structure in Table 3.1



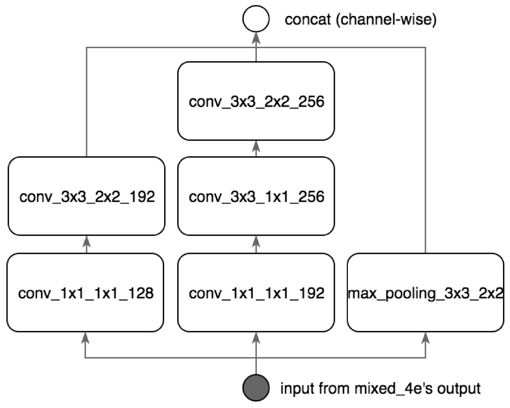
(e) mixed_4c structure in Table 3.1



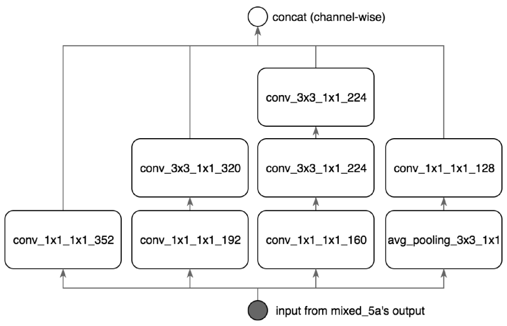
(f) mixed_4d structure in Table 3.1



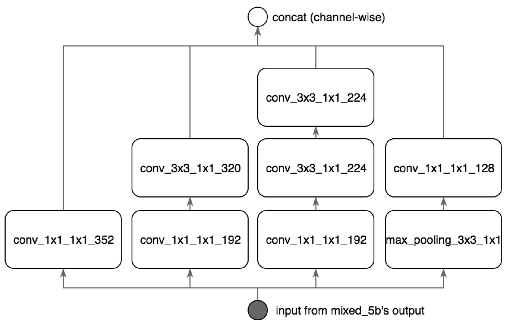
(g) mixed_4e structure in Table 3.1



(h) mixed_5a structure in Table 3.1



(i) mixed_5b structure in Table 3.1



(j) mixed_5c structure in Table 3.1

Figure 3.3: Illustration of "Inception modules" denoted as mixed_3b, mixed_3c, mixed_4a to mixed_4e, and mixed_5a to mixed_5c in Table 3.1. Those are defined in InceptionV2 implemented in [2]. Each module is labeled with the following format: [layer name]_[filter size]_[stride size]_[number of channels output], e.g. conv_3x3_1x1_128 represents a conolutional layer with filters of 3×3 in size, where the convolution makes strides of 2 in both row and column directions, and the number of output channels (or feature maps) is 128. max_pooling and avg_pooling are labeled only with the size of the filters and strides as those operate channel-wise and do not incur any change in the number of the channels from the input. A concatenation layer (i.e. concat) in the end of every module stacks channel-wise the output feature maps from every branch.

Algorithm 1: Multiple Object Tracking with DSST and CNN Features (Section 3.2)

Defined:

$\mathcal{O}$	: A data structure that stores the list of tracked objects, where $\mathcal{O} = \{\mathbf{O}_i \vert i = 1, ..., \vert\mathcal{O}\vert\}$ and $\mathbf{O}_i = \{o_i, \mathbf{b}_i, \mathbf{F}_i, s_i, p_i, T_i\}$
o_i	: object id
λ^{det}_{conf}	: detector's confidence
$\mathbf{b}_i$	: bounding box (x_i, y_i, w_i, h_i)
$\mathbf{F}_i$	: features extracted from $\mathbf{b}_i$
s_i	: similarity between current $\mathbf{F}_i$ and that calculated in previous frame
p_i	: number of frames of this object being tracked successfully
T_i	: number of frames of this object being observed

Given:

$\mathcal{O}^t = \varnothing$	: active tracks in t-th frame
$\mathcal{O}^{hist} = \varnothing$	: history tracks
$f(\cdot)$	: ConvNet feature extraction function (Refer to Section 3.2.2.1)
I^t	: t-th frame

Parameters:

τ^{det}	: detector threshold
s^{thres}	: the similarity threshold that defines whether an object is succesfully tracked
p^{thres}	: the successful rate of an object having been tracked
T^{thres}	: the count that defines the minimal observation span of whether a track should be removed
T_M	: threshold for checking admissibility for associating newly detected objects and objects in active tracks
T_d	: threshold for checking admissibility for re-identifying objects from the history tracks

```
1:  for frame $t = 1, ...$ do
2:      $s^t_i, \mathbf{b}^t_i, \mathbf{F}^t_i$ = update_tracker($f$, $I^t$, $\mathbf{b}^{t-1}_i$, $\mathbf{F}^{t-1}_i$), $i = 1, ..., |\mathcal{O}^t|$, $t > 1$      (Algorithm 2)
3:      if (detector is triggered) then
4:          $\mathcal{O}^{det}$ = detect_object($I^t$, $\tau^{det}$)      # detect objects
5:          $\mathbf{M}$ = compute_association_cost_matrix($\mathcal{O}^{det}$, $\mathcal{O}^t$)      (Algorithm 3)
6:          $\mathcal{I}$ = data_association($\mathbf{M}$, $T_M$), where $\mathcal{I} = \{(k_j, l_j) | k_j, l_j \in \mathbb{N}\}_j$.      (Algorithm 4)
7:          $\mathcal{O}^{new} = \mathcal{O}^{det} \backslash \{\mathcal{O}^{det}_{k_j}\}_j$, $\forall j$
8:          $p^{new}_i = T^{new}_i = 1$, $\forall i$, where $(p^{new}_i, T^{new}_i) \in \mathcal{O}^{new}$.
9:          $T^t_i := T^t_i + 1$, $\forall i$, where $T^t_i \in \mathcal{O}^t$.
10:         for each associated pair of objects in $\{\mathbf{O}^{det}_{k_j}, \mathbf{O}^t_{l_j}\}_j$ do
11:             $p^t_{l_j} := p^t_{l_j} + 1$
12:             $\mathbf{F}^t_{l_j} := (1 - \lambda^{det}_{conf})\mathbf{F}^t_{l_j} + \lambda^{det}_{conf}\mathbf{F}^{det}_{k_j}$
13:             $\mathbf{b}^t_{l_j} := (1 - \lambda^{det}_{conf})\mathbf{b}^t_{l_j} + \lambda^{det}_{conf}\mathbf{b}^{det}_{k_j}$
14:         end for
15:         $\mathcal{O}^{new}$, $\mathcal{O}^{hist}$ := re_identification($\mathcal{O}^{new}$, $\mathcal{O}^{hist}$, $T_d$)      (Algorithm 6)
16:         $\mathcal{O}^t := \mathcal{O}^t \cup \mathcal{O}^{new}$      # append new objects into $\mathcal{O}^t$
17:     else
18:         $p^t_i := p^t_i + 1, T^t_i := T^t_i + 1$, if $s^t_i > s^{thres}$, $\forall i = 1, ..., |\mathcal{O}^t|$
19:     end if
20:     $\mathcal{O}^t$, $\mathcal{O}^{hist}$ := relloc_bad_tracks($\mathcal{O}^t$)      (Algorithm 7)
21:     Remove object from $\mathcal{O}^{hist}$ if it has not been re-identified for some number of frames.
22: end for
    /* end tracking */
```

Algorithm 2: `update_tracker`$(f,\ I^t, \mathbf{b}_i^{t-1}, \mathbf{F}_i^{t-1})$ (Section 3.2.2.2)

```
1: $\mathbf{b}_i^t$ = DSST_tracker_update($I^t$, $\mathbf{b}_i^{t-1}$)
2: $\mathbf{F}_i^t$ = extract_bbox_CNN_feature($f(I^t)$, $\mathbf{b}_i^t$)
3: $s_i^t$ = calculate_cosine_simlarity($\mathbf{F}_i^t$, $\mathbf{F}_i^{t-1}$)
4: $\mathbf{F}_i^t$ := update_feature($\mathbf{F}_i^t$, $s_i^t$)          # update features with adaptive learning rate
5: return $s_i^t, \mathbf{b}_i^t, \mathbf{F}_i^t$
```

Algorithm 3: `compute_association_cost_matrix`$(\mathcal{O}^{det},\ \mathcal{O}^t)$ (Section 3.2.3)

```
Given:
        $\mathbf{M}_c = \mathbf{0}_{|\mathcal{O}^{det}|\times|\mathcal{O}^t|}$: cost matrix accounting for centers' and IoU misalignment
        $\mathbf{M}_f = \mathbf{0}_{|\mathcal{O}^{det}|\times|\mathcal{O}^t|}$: cost matrix accounting for features' similarity
        $l_{diag}$: diagonal length of the frame
1: for $m = 1, ..., |\mathcal{O}^{det}|$ do
2:     for $n = 1, ..., |\mathcal{O}^t|$ do
3:         $c_x^{det}, c_y^{det}, c_x^t, c_y^t$ = get_centers($\mathbf{b}_m^{det}$, $\mathbf{b}_n^t$)
4:             $\mathbf{M}_c[m,n] = \frac{1}{2}(||(c_x^{det}, c_y^{det}) - (c_x^t, c_y^t)||_2/l_{diag}) + \frac{1}{2}(1 - \texttt{IoU}(\mathbf{b}_m^{det}, \mathbf{b}_n^t))$
5:             $\mathbf{M}_f[m,n] = 1 - \texttt{sim}(\mathbf{F}_m^{det}, \mathbf{F}_n^t)$ (Refer to (3.1))
6:     end for
7: end for
8: return $\mathbf{M} = \mathbf{M}_c \cdot \mathbf{M}_f$, where $\mathbf{M} \in R^{|\mathcal{O}^{det}|\times|\mathcal{O}^t|}$
```

Algorithm 4: `data_association`$(\mathbf{M},\ T)$ (Section 3.2.3)

```
1: $\mathcal{I}$ := Hungarian_algorithm($\mathbf{M}$)
2: for each tuple $(k_j, l_j) \in \mathcal{I}$ do
3:     $\mathcal{I} := \mathcal{I} \backslash (k_j, l_j)$, if $M[k_j, l_j] > T$          # admissibility check
4: end for
5: return $\mathcal{I}$
```

Algorithm 5: `update_feature`$(\mathbf{F}_i^{t-1},\ \mathbf{F}_i^t,\ s_i^t)$ (Section 3.2.2.2)

```
Given:
        $\gamma$                          : base feature update rate
        $\mu_x$                          : a constant mean
        $\sigma_x$                          : a constant standard deviation
        $\mathcal{N}(\mu|\mu_x, \sigma_x^2)$   : a normal distribution PDF centered at $\mu_x$ and with variance $\sigma_x^2$
1: $\lambda = \mathcal{N}(\mu = s_i^t|\mu_x, \sigma_x^2)/\mathcal{N}(\mu = \mu_x|\mu_x, \sigma_x^2)$
2: $\gamma := \lambda \cdot \gamma$                                   # obtain adaptive feature update rate
3: $\mathbf{F}_i^t := (1-\gamma)\mathbf{F}_i^{t-1} + \gamma\mathbf{F}_i^t$                          # update features
4: return $\mathbf{F}_i^t$
```

Algorithm 6: `re_identification`$(\mathcal{O}^{new},\ \mathcal{O}^{hist},\ T_d)$ (Section 3.2.4)

```
1: $\mathbf{M}$ = compute_association_cost_matrix($\mathcal{O}^{new}$, $\mathcal{O}^{hist}$)
2: $\mathcal{I}$ := data_association($\mathbf{M}$, $T_d$), where $\mathcal{I} = \{(k_j, l_j)|k_j, l_j \in \mathbb{N}\}_j$.
3: *assign* the same object id as $\mathcal{O}_{l_j}^{hist}$ to $\mathcal{O}_{k_j}^{new}$, $\forall j$
4: *remove* $j$-th object from $\mathcal{O}_{l_j}^{hist}$, $\forall j$
5: *assign* new object id to the objects in $\mathcal{O}^{new}$ which are not associated.
```

Algorithm 7: `relloc_bad_tracks`($\mathcal{O}^t$, $\mathcal{O}^{hist}$)

```
1: for $i = 1, ..., |\mathcal{O}^t|$ do
2:     $T_i^t = \min(T_i^t, T^{thres})$
3:     if $T_i^t = T^{thres}$ and $\frac{p_i^t}{T_i^t} < p^{thres}$ then
4:         $\mathcal{O}^t := \mathcal{O}^t \setminus \mathbf{O}_i^t$        # remove poorly tracked object from active tracks
5:         $\mathcal{O}^{hist} := \mathcal{O}^{hist} \cup \mathbf{O}_i^t$        # add it into the history tracks
6:     end if
7: end for
8: return $\mathcal{O}^t$, $\mathcal{O}^{hist}$
```

Chapter 4

Experiments

This chapter aims to provide the empirical study on the proposed framework, analyzing its strengths and weaknesses in order to offer insight of how to improve it in the future. To begin with, we introduce the dataset used and explain the evaluation protocols used throughout the experiments in Section 4.1. Next in Section 4.2, we conduct the ablation studies on the parameters which are the most vital ones in the framework regarding the tracking performance. Several experiments are made in the way that one parameter or one set of parameters is treated as a variable while the rest are kept fixed to examine the effect of varying those parameters. This strategy enables us to understand how each parameter affects the performance as well as keeps the search space of the variables within reasonable size. Finally in Section 4.4, we provide a discussion and reasoning of the results.

4.1 Evaluation Dataset and Protocols

MOT Challenge 2016 (MOT'16) offers 14 video sequences evenly divided into seven training and seven testing sequences summarized in Table 4.1. The target class of the evaluation focus is *pedestrian*. Particularly, in MOT'16 challenge the pedestrians who are static (e.g. sitting or standing without moving), behind the glasses, in the reflection, or in the vehicles are omitted and not considered in the evaluation. Thus, constantly moving pedestrians are the only left. The videos are taken in unconstrained public spaces (e.g. open streets, shopping malls, squares, etc.) that are usually crowded. Some cameras are installed in driving vehicles, some are carried by a walking person, and some are stationary. Challenges including wide variety of sizes, orientations, walking speeds, and heavy occlusions make the dataset realistic and to highly correspond to the real-world applications.

Table 4.1: Summary of MOT'16 training and test sets.

training set	test set	frame rate (fps)	camera
MOT16-02	*MOT16-01*	30	static
MOT16-04	*MOT16-03*	30	static
MOT16-05	*MOT16-06*	14	dynamic
MOT16-09	*MOT16-07*	30	dynamic
MOT16-10	*MOT16-08*	30	static
MOT16-11	*MOT16-12*	30	dynamic
MOT16-13	*MOT16-14*	25	dynamic

Noted in [27], it is difficult to quantify a MOT tracker's performance or capture the charateristics of the tracker with a single metric. Among all the existing metrics that are designed for assessing MOT systems, CLEAR metrics [32] and the metrics introduced in [37] have been the most widely used. Please note that even these metrics are the most trendy in the recent MOT works and treated as the standard measures, but the research of standarizing the metrics for MOT problem is still ongoing [27]. In MOT'16, those metrics are used altogether to assess the overall performance while the trackers can be ranked by their average ranking calculated from the ranks with respect to each individual metric. In the following, we walk through the formal definition of every metric included in MOT'16.

True Positive (`TP`), False Positive (`FP`), False Negative (`FN`): These are the most common metrics quantifying the hypotheses made by the tracker. `TP` measures whether the hypotheses are matched to the annotations while `FP` measures if they are false alarms. `FN` measures the misses of the hypotheses with respect to the annotations. Either metric is counted when the IoU is less than 0.5 as suggested in [27].

Precision (`Precision`), Recall (`Recall`): `Precision` is defined in (4.1), reflecting how relevant the predicted bounding boxes are to the ground-truth bounding boxes. `Recall` is defined (4.2):

$$\texttt{Precision} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FP}} \tag{4.1}$$

$$\texttt{Recall} = \frac{\texttt{TP}}{\text{number of ground-truth bounding boxes}} \tag{4.2}$$

Identity Switch (`IDs`): `IDs` counts the mismatching error which happens when an annotated target x is matched to a track y in frame $t-1$ but matched to another track $z, z \neq y$ in frame t. Note that `IDs` alone may not inform the tracker's overall performance as it usually correlates with the number of annotated tracks. Hence, one can instead look at the ratio of `IDs` to the recall when needed. Note that throughout the experiments, we still report the raw `IDs` as `Recall` is also reported.

Fragmentation (`FM`): A fragmentation is counted when a track is interrupted for some frames and recovered either with or without ID switches.

Multiple Object Tracking Accuracy (`MOTA`): `MOTA` considers three sources of metrics to assess the overall accuracy of a MOT tracker across the frames. More formally, it is defined as

$$\texttt{MOTA} = 1 - \frac{\sum_t (\texttt{FN}_t + \texttt{FP}_t + \texttt{IDs}_t)}{\sum_t \texttt{GT}_t}, \tag{4.3}$$

where the subscript t denotes the frame index and $\texttt{GT}_t$ denotes the number of objects in frame t. Note that it is possible that `MOTA` value is below zero while its maximum is 1.

Multiple Object Tracking Precision (`MOTP`): `MOTP` measures in average across all frames how well do the tracker's outputs overlap with the annotations. More formally, it is defined as

$$\texttt{MOTP} = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}, \tag{4.4}$$

where c_t is the number of annotations in frame t and $d_{t,i}$ is the IoU value of the target i and its assigned annotation. In short, `MOTP` is used to measure the localization accuracy of a system where the detecter and the tracker work collaboratively with each other.

Number of Ground-Truth Tracks (`GT`), Mostly Tracked (`MT`), Partly Tracked (`PT`), Mostly Lost (`ML`): A track is said to be *mostly tracked* if over 80% of its annotations along the track are matched correctly to the tracker's outputs, while it is said to be *mostly lost* if under 20% of its annotations along the track is matched correctly, otherwise it is classified as *partly lost.* We define `MT`, `PT`, and `ML` as the percentage of each quantity (i.e. the numbers of mostly tracked, partly tracked, and mostly lost) to the number of grount-truth tracks, `GT`.

Tracker Ranking (`TR`): `TR` does not reflect the overall performance of a MOT tracker, but provides a relative figure that ranks the trackers by comparing the average ranking. The average ranking is calculated according to the rank made by each individual metric (`IDSW`, `MOTA`, `MOTP`, etc).

In the following section, we provide ablation studies on how do the different parameters in the proposed framework affect each metric.

4.2 Parameter Selections and Ablation Studies

In this section, we conduct parameter selection and ablation studies on MOT'16 training set which includes in total seven sequences. To avoid exhaustively searching over all possible combinations of parameters, we adopt the following strategy to search the parameter space defined by those of the highest impacts on the framework. The options of the parameter values are enlisted in Table 4.2.

As suggested in the literature [27, 36], the overall tracking performance is highly dependent on the detector's performance. Hence, we start exploring the parameter space by varying the values of the detection threshold τ^{det} while fixing other parameters. We select the value that results in the best `MOTA` score as `MOTA` encapsulates more measures together than other metrics introduced in Section 4.1, including three critical measures: false negative rate, false positive rate, and how many times has object identity been switched. Next, we vary the pair of values of (T^{thres}, p^{thres}) and keep the rest fixed, while the value for the detector threshold is set to be the best one picked previously. The same process continues with (μ_x, s^{thres}), T_M, and T_d, respectively in turn.

The selection process finally yields one set of values for all the parameters. During the process, one can observe the impact of each varying parameters as the other parameters stay unchanged. This can offer the insight on how significant each parameter is to the framework. Apart from the parameters in Table 4.2, we would also like to examine how sensitive is our framework to the availability of the detector, i.e. triggering the detector in every n frames, where $n = 1$ and 3, in the videos. Tables 4.3 and 4.4 summarize the experiments, parameters and the corresponding sets of values to be examined. In the following subsections, we report how the systems with $n = 1$ and 3 perform under varying (1) τ^{det}, (2) (T^{thres}, p^{thres}), (3) (μ_x, s^{thres}), (4) T_M, and (5) T_d, respectively.

Table 4.2: A table of parameters to be studied.

Parameter	Name	Used in
τ^{det}	Threshold on accepting detections from the detector	Algorithm 1: `line 4`
T^{thres} / p^{thres}	sliding observation span / threshold on the ratio of the span of an object being tracked and observed	Algorithm 7
μ_x / s^{thres}	mean of the Gaussian function used in adaptive feature update / threshold on the similarity that defines if two objects are the same one	Algorithm 1: `line 18`
T_M	threshold on association cost	Algorithm 1: `line 6`
T_d	threshold on re-identification cost	Algorithm 6

Table 4.3: Experiment strategy for conducting parameter selections and ablation studies. The dash (—) in each row means the variable(s) whose values are chosen from Table 4.4. Each of five sets of experiments, from A to E, finds optimal value(s) for a parameter or a set of parameters. For instance, experiment A finds τ^{det*}, the optimal values for variable τ^{det}. τ^{det*} is then brought to the experiment B and fixed along with (μ_x, s^{thres}), T_M, and T_d. The same process repeats for the experiments C, D, and E.

Exp.	τ^{det}	T^{thres}, p^{thres}	μ_x, s^{thres}	T_M	T_d	Output
A	—	(10, 0.75)	(0.86, 0.8)	0.3	0.2	τ^{det*}
B	τ^{det*}	—	(0.86, 0.8)	0.3	0.2	$(T^{thres}, p^{thres})^*$
C	τ^{det*}	$(T^{thres}, p^{thres})^*$	—	0.3	0.2	$(\mu_x, s^{thres})^*$
D	τ^{det*}	$(T^{thres}, p^{thres})^*$	$(\mu_x, s^{thres})^*$	—	0.2	T_M^*
E	τ^{det*}	$(T^{thres}, p^{thres})^*$	$(\mu_x, s^{thres})^*$	T_M^*	—	T_d^*

4.2.1 Experiment A: Varing Detector Thresholds τ^{det}

As known, SSD's classification layers are the layers that do softmax operations which yield the probability of each object proposal for each object class. We interpret the probability of each object class as the confidence score of the detector on the corresponding class. As the object of interest is *pedestrian*, we only extract the probability score of the pedestrian class, comparing it to

Table 4.4: Following Table 4.3, this table provides the values for the parameters to be selected in each experiment from A to E.

Exp.	Parameter	Values to be experimented
A	τ^{det}	0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
B	(T^{thres}, p^{thres})	(10, 0.5), (10, 0.75), (15, 0.5), (15, 0.75), (20, 0.5), (20, 0.75), (25, 0.5), (25, 0.75), (30, 0.5), (30, 0.75)
C	(μ_x, s^{thres})	(0.82, 0.74), (0.82, 0.76), (0.82, 0.78), (0.84, 0.76), (0.84, 0.78), (0.84, 0.80), (0.86, 0.78), (0.86, 0.80), (0.86, 0.82)
D	T_M	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4
E	T_d	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4

a pre-defined detector threshold, τ^{det}, and accept it as a pedestrian only if it is larger than the threshold. Small detector thresholds can lead to more true positives but as well more false positives. On the contrary, large τ^{det} can lead to fewer true positives but more precise detection results and less false positives. In the MOT'16 sequences, many pedestrians are rather small, i.e. the minimum height of the pedestrians' bounding boxes can be 19 pixels [27]. The tracking performance can be affected by either too many false positives (e.g., non-pedestrian objects or static background) or imprecise detections (i.e. bad localization of the object) that could get the tracker drifted easily. Thus, to select a sensible τ^{det} is the main focus in this subsection.

Figure 4.1 shows the results in experiment A listed in Table 4.3 and Table 4.4. The increase of τ^{det} leads to decrease of `FP` and increase of `FN`. In Figure 4.1a, `MOTA` score increases with the increase of τ^{det} as `FP` decreases faster than `FN` increases. `MOTA` peaks at 0.6 and start to saturate or descend when both `FP` and `FN` starts to saturate. As expected, while larger τ^{det} obtains larger `Precision`, it obtains poorer `Recall` as shown in Figure 4.1d, leading to a significant ascent in `ML` and descent in `PT` and `MT`. What is worth noting is that the drop of `IDs` and `FM` in Figure 4.1c with increased τ^{det} does not indicate that a larger τ^{det} would result in better tracker's performance. The drop is simply an effect of the fact that fewer present tracks can be correctly identified, hence resulting in a smaller number of cases when the object identity is switched or the tracks are fragmented along the predicted tracks.

Comparing the $n = 1$ and $n = 3$ case among all the metrics, the $n = 1$ case consistently shows improvement in `MOTA`, `MOTP`, `FP`, and `FN` although not significantly. The $n = 3$ case outperforms the $n = 1$ case in `Recall`, but falls short in `Precision`. In terms of `MT`, `PT`, and `ML`, there is no clear indication of which case is consistently better.

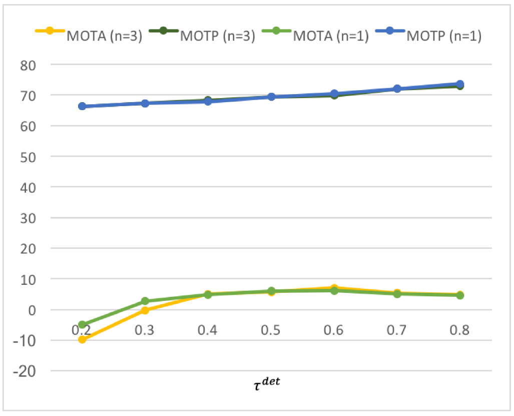
4.2.2 Experiment B: Varying T^{thres} and p^{thres}

Here we analyze the significance of T^{thres} and p^{thres}, where $T^{thres} \times (1 - p^{thres})$ defines the maximal number of frames that is allowed to not to be tracked or detected within a sliding window of T^{thres} frames. Hence the larger the p^{thres} is, the more easily a track will be removed from the active tracks. Having a reasonable amount of tolerance to noisy results from the detector and tracker can help keep the track active and at the same time get rid of false positive tracks. Specifically, if $T^{thres} \times (1 - p^{thres})$ is large, we may keep most of the tracks active, but possibly introduce more false positives, since each track is allowed to have more failures within a longer span.

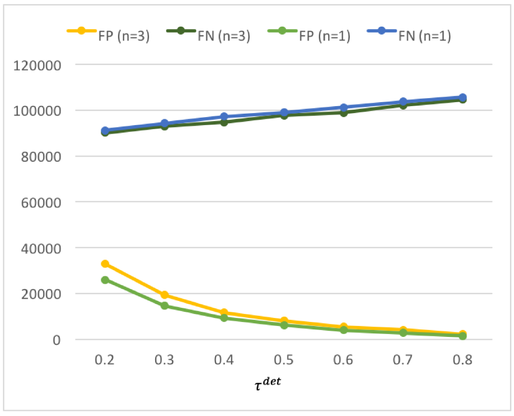
Figure 4.2 shows the results under varying T^{thres} and p^{thres}. In terms of `MOTA`, the $n = 3$ case is shown more sensitive to the varying T^{thres} and p^{thres} than the $n = 1$ case. Shown in Figure 4.2a, `MOTA` in both cases declines when (1) p^{thres} varies from 0.75 to 0.5 given any T^{thres}, and (2) T^{thres} grows. In Figure 4.2b, the $n = 1$ case shows stable in both `FP` and `FN` under varying variables while the $n = 3$ case appears more sensitive to p^{thres} and obtains a lower `FP` with p^{thres} being 0.75. `FN` in both cases appears to be insensitive to the variables. In Figure 4.2c, both cases show declination in `IDs` and `FM` when the sliding observation span T^{thres} grows with fixed p^{thres}. When T^{thres} is fixed, `IDs` and `FM` noticeably rise when p^{thres} goes from 0.5 to 0.75. Next in Figure 4.2d, `Recall` and `Precision` in $n = 3$ case appears to be sensitive to p^{thres} under fixed T^{thres}. Finally in Figure 4.2e, the $n = 3$ case slightly outperforms the $n = 1$ case in `MT`, `PT`, and `ML`. To recap the experiment results, the $n = 1$ case, in which the detector is triggered every frame, appears to be less sensitive to T^{thres} and p^{thres} and slightly outperforms the $n = 3$ case in most of the metrics.

4.2.3 Experiment C: Varying μ_x and s^{thres}

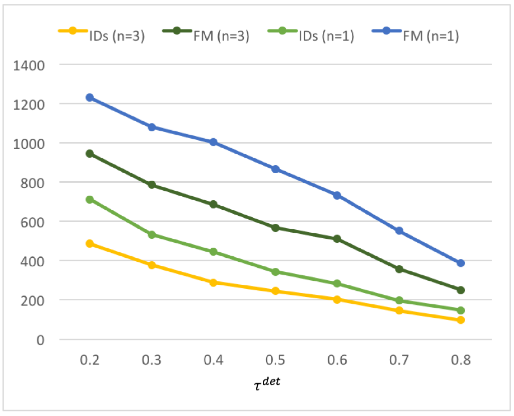
The role of μ_x is to prevent over-updating the auxiliary templates introduced in Chapter 3.2.2.1. The adaptive learning rate $\gamma_{\texttt{adapt}}$ is dependent on μ_x, and the update takes most of its effect only when the similarity measure between auxiliary target templates in the previous and current frames is close enough to μ_x (Please refer to Section 3.2.2.2 for more details). s^{thres} defines the threshold by which a target is accepted as tracked. These two parameters are selected together as the similarity measure between auxiliary templates of a successfully tracked target stays within a range from μ_x. If a tracker



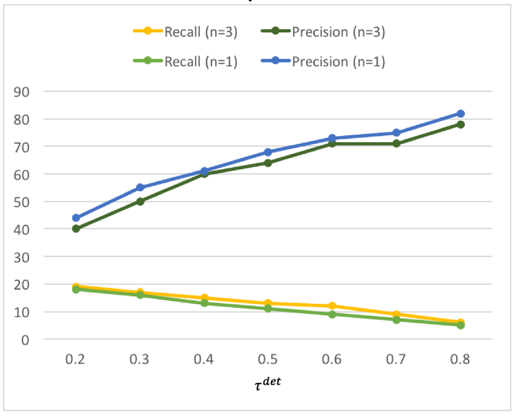
(a) MOTA and MOTP.



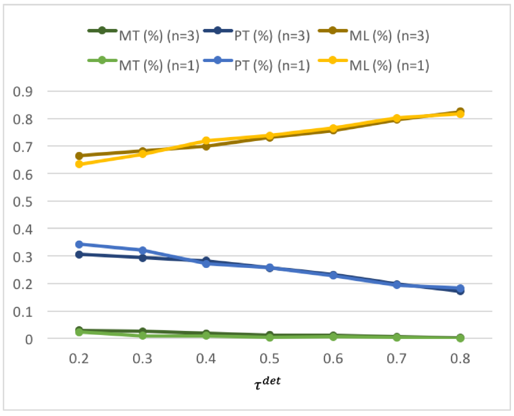
(b) FP and FN



(c) IDs and FM.

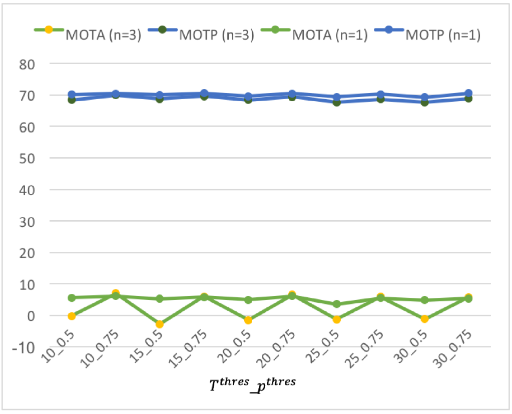


(d) Recall and Precision.

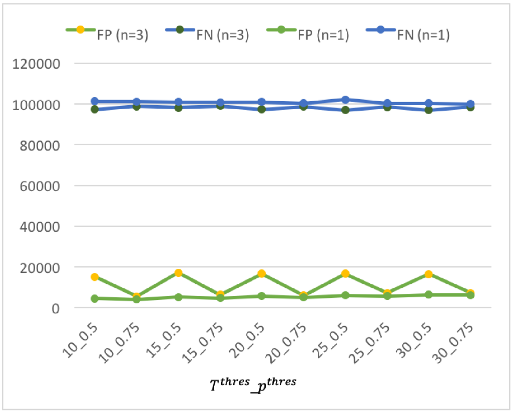


(e) MT, PT and ML.

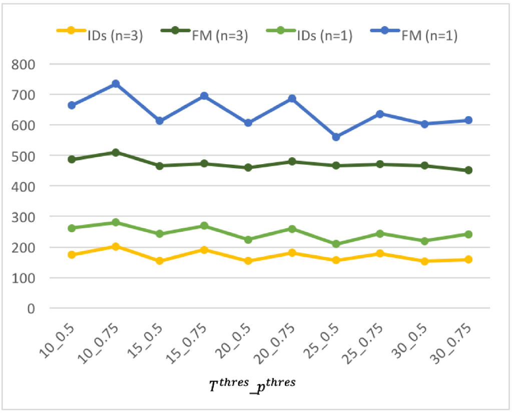
Figure 4.1: Experiment A: The proposed framework's performance under varying τ^{det}.



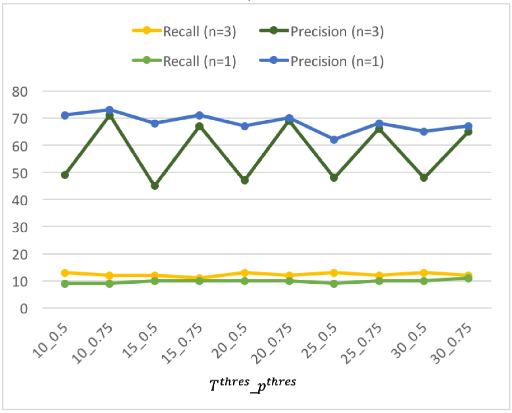
(a) `MOTA` and `MOTP`.



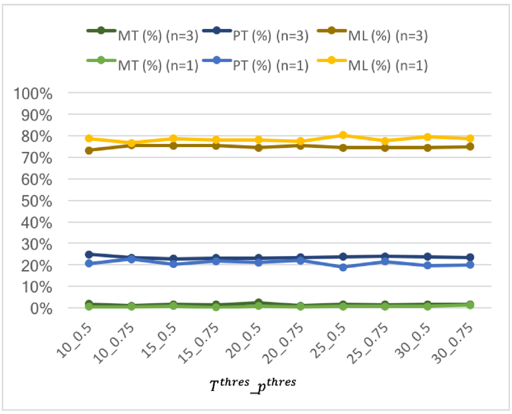
(b) `FP` and `FN`



(c) `IDs` and `FM`.



(d) `Recall` and `Precision`.



(e) `MT`, `PT` and `ML`.

Figure 4.2: Experiment B: The proposed framework's performance under varying T^{thres} and p^{thres}.

has drifted and the auxiliary templates are updated properly, its similarity measure can drop dramatically from μ_x. Hence we pick the values for s^{thres} which are below but close enough to μ_x in order to capture the drop of the similarity measure whenever a tracker starts to drift.

Figure 4.3 shows the results under varying μ_x and s^{thres}. Firstly, from Figure 4.3a, it shows that the $n = 3$ case slightly outperforms the $n = 1$ case in `MOTA` and achieves nearly equal performance in `MOTP`. Next, from Figure 4.1b, `FP` and `FN` vary only subtly with varying μ_x and s^{thres}. Figure 4.3c shows that the $n = 1$ case suffers from more ID switches and fragmentations than the $n = 3$ case. In addition, the $n = 1$ case, in most of the experiments under the same variable values, tends to obtain higher `Precision` and lower `Recall`. Last, shown in Figure 4.3e, the performance curves of the $n = 1$ and $n = 3$ cases appear to be much similar to each other except that at $(\mu_x, s^{thres}) = (0.88, 0.84)$, the $n = 3$ case obtains slightly lower `ML` and higher `PT`.

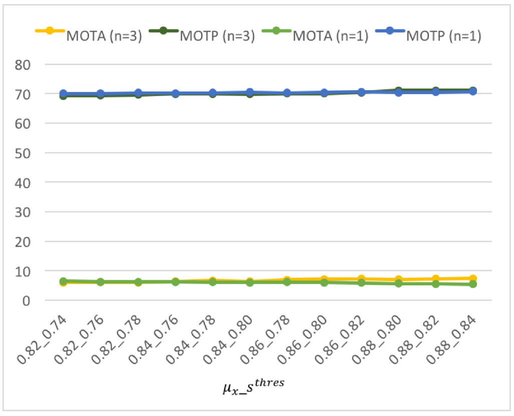
4.2.4 Experiment D: Varying T_M

The parameter T_M controls how strict it is to associate the detected objects to the objects in the active tracks. Allowing larger T_M may result in more wrong associations, causing more identity switches and fragmentations. Smaller T_M can make the association too strict, failing to associate the detected objects to any of the active tracks when needed, causing as well identity switches with new object IDs being created. This theory is also suggested in the experimental results shown in Figure 4.4. For instance, in Figure 4.4a, `MOTA` from both $n = 1$ and $n = 3$ cases peaks at 0.2 and decreases as T_M goes either smaller or larger.

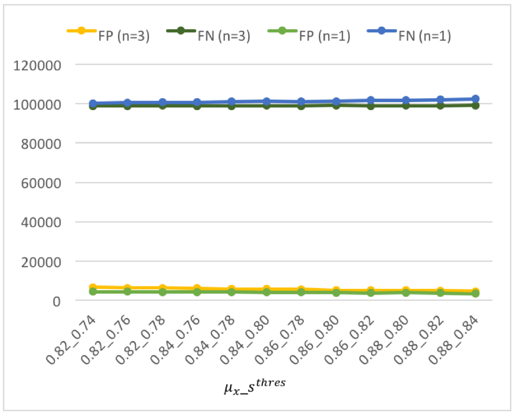
In Figure 4.4b, while the lowest `FN` and highest `FP` are located at where $T_M = 0.1$, `FN` starts to increase and `FP` starts to decrease when T_M rises. This indicates that one can find a tradeoff between `FN` and `FP` by adjusting T_M. Hence, as `FP` starts to saturate from $T_M = 0.25$ onwards while the `FN` still increases, the best performance in terms of `FN` and `FP` combined is suggested to be at around $T_M = 0.25$ in both the $n = 1$ and $n = 3$ cases.

Next, Figure 4.4c shows clearly that when $T_M = 0.25$, which is the halfway between 0.1 and 0.4, the tracks suffer the least ID switches. Although `FM` reaches the lowest at $T_M = 0.35$ instead of at around $T_M = 0.25$ in both cases, it is still suggested that in terms of `FM` and `IDs`, the best performance is reached at around $T_M = 0.25$ as `FM`'s curves appear to be less sensitive to T_M and thus makes less impact than `IDs`.

Next, in Figure 4.4d, while similar trends are shown in `Recall` in both $n = 1$ and $n = 3$, they behave differently in `Precision`. The `Precision`

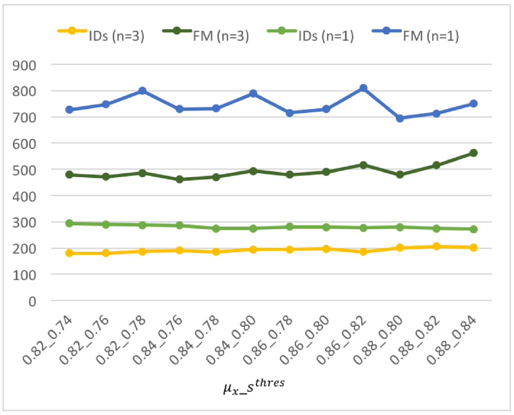


(a) `MOTA` and `MOTP`.

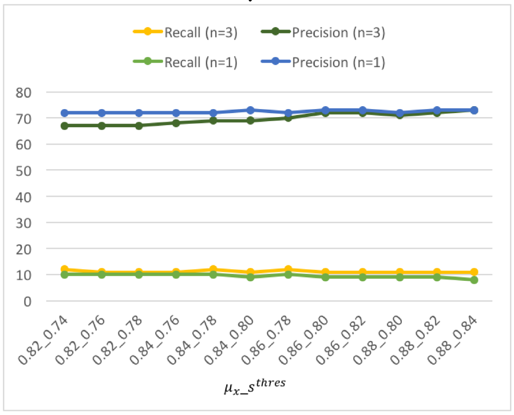


(b) `FP` and `FN`

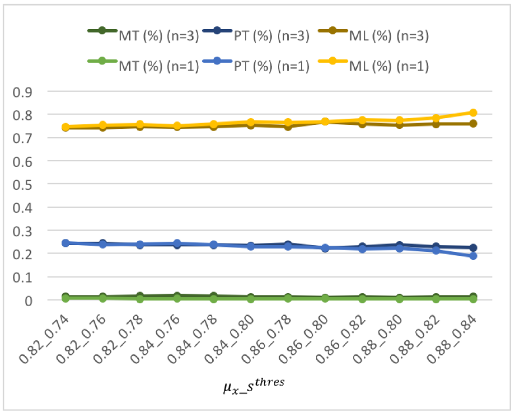
.



(c) `IDs` and `FM`.



(d) `Recall` and `Precision`.



(e) `MT`, `PT` and `ML`.

Figure 4.3: Experiment C: The proposed framework's performance under varying μ_x and s^{thres}.

curve of the $n = 3$ case stagnates around $0.2 \leq T_M \leq 0.3$ and rises ever since. The $n = 1$ case's `Precision` curve peaks at $T_M = 0.25$ as expected. Lastly, the trends of `MT`'s, `PT`'s, and `ML`'s curves appear similar for both $n = 1$ and $n = 3$ cases where `ML`'s curves climb with T_M increasing while other two curves descend accordingly.

4.2.5 Experiment E: Varying T_d

Here we study the associate cost threshold for target re-identification purpose. T_d is used as the threshold which governs the admissibility of association between newly detected objects and those in the history tracks. Large T_d can result in almost random association of the objects and the tracks, preventing the true correspondence of an object and its track being associated and incurring a large number of identity swicthes. Small T_d can result in passive re-identification, where less tracks in the history can be recovered and tracked actively again. Hence, `IDs` has the most noticeable change with varying T_d as shown in Figure 4.5c. The `IDs`' curves in $n = 1$ and $n = 3$ case obtain the lowest number at $T_d = 0.25$ and $T_d = 0.3$, respectively. The `IDs`'s curve in the $n = 3$ case starts to saturate and in the $n = 1$ case slightly ascends when $T_d > 0.3$, indicating that allowing more slack associations may not help recover more objects in the history tracks. We leave out other figures than Figure 4.5c from the discussion because nearly no effect or changes can be seen.

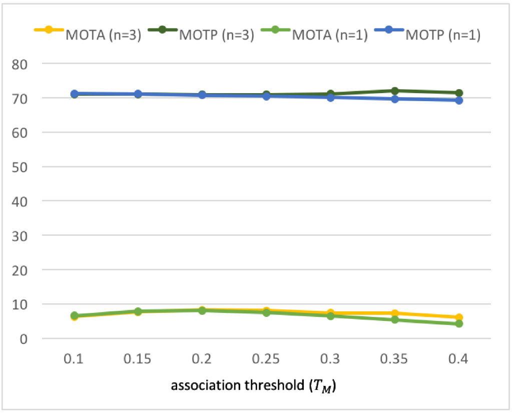
4.2.6 Detection Frequency

So far we have covered five experiments under different parameters as the variables and chosen the set of parameters separately for the $n = 1$ and $n = 3$ cases. The empirical chosen values for those parameters are summarized in Table 4.5.

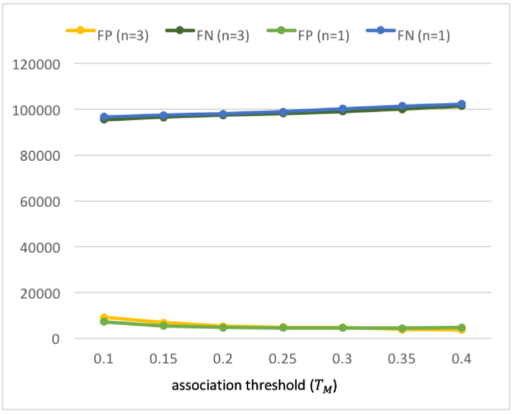
In the following, we present the average performance and individual performance of the proposed framework on each MOT'16 training set (including seven video sequences) with different detection frequencies, i.e. $n = 1$ and

Table 4.5: Summary of the values chosen for the parameters used in the framework aiming for optimizing `MOTA` score.

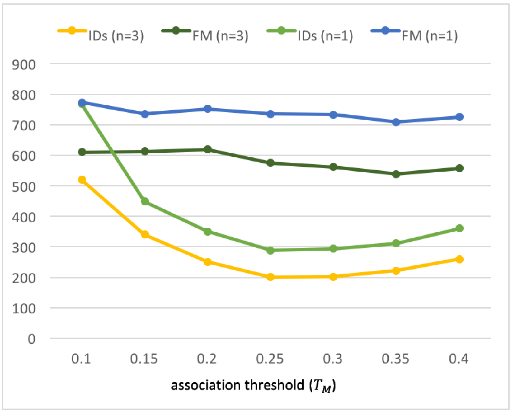
	τ^{det}	(T^{thres}, p^{thres})	(μ_x, s^{thres})	T_M	T_d
$n = 1$	0.6	(10, 0.75)	(0.88, 0.84)	0.2	0.25
$n = 3$	0.6	(10, 0.75)	(0.82, 0.74)	0.2	0.25



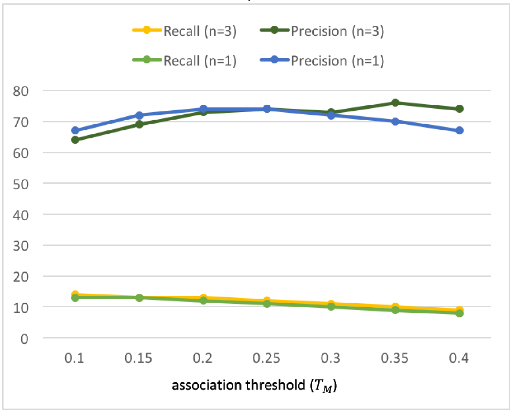
(a) MOTA and MOTP.



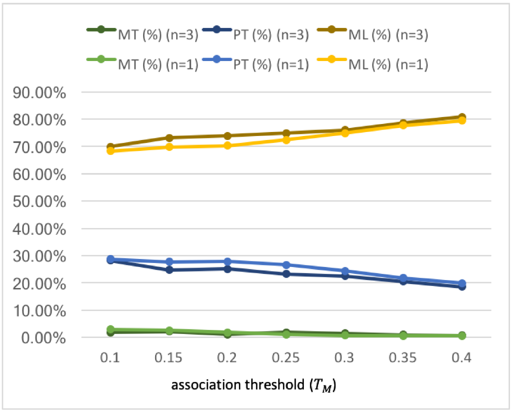
(b) FP and FN



(c) IDs and FM.

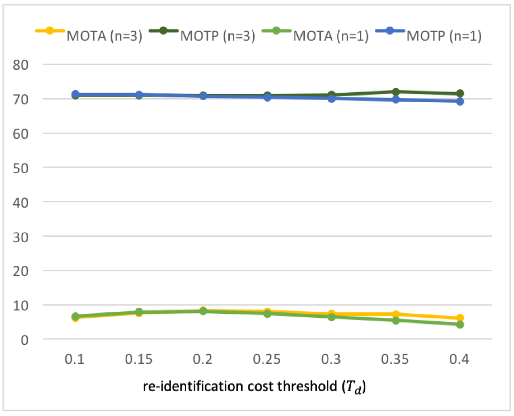


(d) Recall and Precision.

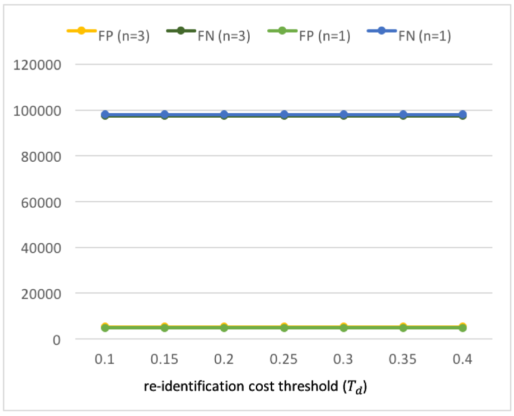


(e) MT, PT and ML.

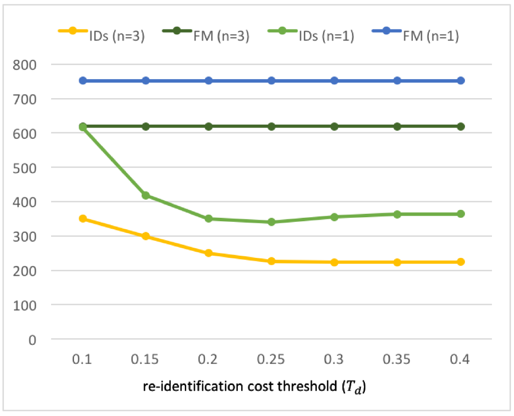
Figure 4.4: Experiment D: The proposed framework's performance under varying T_M.



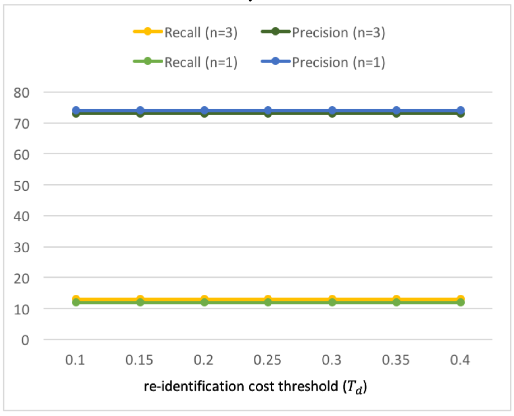
(a) MOTA and MOTP.



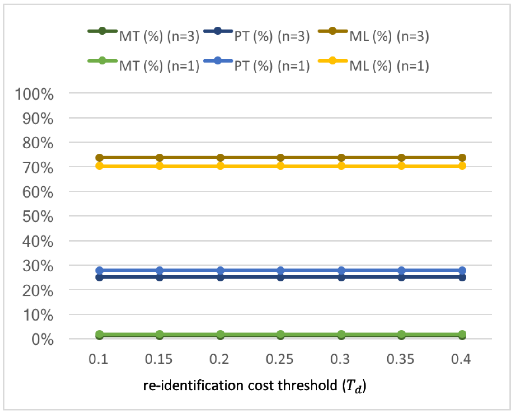
(b) FP and FN



(c) IDs and FM.



(d) Recall and Precision.



(e) MT, PT and ML.

Figure 4.5: Experiment D: The proposed framework's performance under varying T_M.

$n = 3$, to allow in-depth analysis. The $n = 1$ case triggers detection in every frame while the $n = 3$ case triggers detection in every three frames. Note that the inference time of detection in each frame is nearly the same, and hence the time spent on detection in the $n = 1$ case triples the computation time compared to that of the $n = 3$ case. It is thus worth analyzing what advantages the higher detection frequency can bring at the cost of additional computation time.

4.2.6.1 Results on MOT'16 Training Set

Table 4.6 shows the average results on all training sequences. While the $n = 1$ case slightly outperforms the $n = 3$ case in `FP`, `MT`, `PT`, and `ML`, $n = 3$ shows stronger in `MOTA`, `MOTP`, `FN`, `IDs`, and `FM`. However, the performance discrepancy between the two cases is not that significant. Next, we break it down to show the performance on each sequence in Tables 4.7 and 4.8. Note that here we present the results in two categories of video sequences: (1) the sequences captured by static cameras, and (2) the sequences captured by moving cameras. The former category is shown in Table 4.7 and the latter is shown in Table 4.8. The reason for the arrangement is to study if the frameworks based on different parameter settings would favor the different dynamics in the videos. Table 4.7 shows that the $n = 3$ case outperforms the other one consistently in `MOTA`, `MOTP`, `FN`, `IDs`, `FM`, and `Recall`. On two out of three sequences, the $n = 3$ case as well outperforms the another in `PT` and `ML`. As for the sequences presented in Table 4.8, one can see that on *MOT16-05*, the $n = 1$ case performs better in most of the metrics, i.e. `MOTA`, `FP`, `FN`, `IDs`, `FM`, `Recall`, `Precision`, `MT`, `PT`, and `ML`. On other dynamic sequences, the two cases perform rather similarly to each other. Later on, we study the results on the MOT'16 test set to examine if the proposed framework behaves similarly according to the dynamics in the videos.

Table 4.6: Comparison of average performance of the $n = 1$ and $n = 3$ cases on seven MOT'16 training sequences. Parameter selection is done by the strategy introduced in Section 4.2. Bold figures indicate the winner cases.

	`MOTA`	`MOTP`	`FP`	`FN`	`IDs`	`FM`
$n = 1$	8.1	70.7	**4809**	98067	340	752
$n = 3$	**8.3**	**70.9**	5344	**97403**	**226**	**619**
	`Recall`	`Precision`	`MT` (%)	`PT` (%)	`ML` (%)	
$n = 1$	12.0	**74.0**	**1.8**	**27.8**	**70.3**	
$n = 3$	**13.0**	73.0	1.1	25.1	73.8	

Table 4.7: The proposed framework on MOT'16 (training) static sequences (i.e. the camera is not moving). The bold figures indicate better performance.

MOT16-02						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	6.6	69.1	**571**	16734	46	92
$n = 3$	**7.3**	**69.6**	602	**16591**	**26**	**62**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	9.9	76.4	1.6	17.7	80.1	
$n = 3$	**10.7**	**76.8**	1.6	**21.0**	**77.4**	
MOT16-04						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	4.0	69.5	**539**	45087	39	116
$n = 3$	**5.0**	**70.3**	861	**44288**	**18**	**78**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	5.2	**82.1**	0.0	10.8	89.1	
$n = 3$	**6.9**	79.2	0.0	**13.3**	**86.7**	
MOT16-09						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	25.4	72.4	**303**	3622	49	87
$n = 3$	**26.3**	**73.5**	353	**3537**	**33**	**64**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	32.0	**84.9**	7.7	50.0	42.3	
$n = 3$	**33.6**	83.5	7.7	50.0	42.3	

4.2.6.2 Results on MOT'16 Test Set

The results on the MOT'16 test set are summarized in Tables 4.9 to 4.11. All in all, the $n = 1$ case slightly outperforms the $n = 3$ case in MOTA, MOTP, FP, Precision, MT, PT, and ML as shown in Table 4.9. Next, Table 4.10 shows the performance of the proposed framework on the static sequences in the MOT'16 test set. The $n = 1$ case shows stronger in MOTA in two out of three sequences, and consistently outperforms the $n = 3$ case in MOTP. The $n = 3$ case shows slightly better consistency in keeping the tracks not being fragmented, which reflects in higher MT on *MOT16-01*, higher PT on *MOT16-03*, and higher (MT + PT) combined on *MOT16-08*. As the performance on the dynamic sequences, the $n = 1$ case outperforms the another on every sequence in MOTA. This indicates the $n = 1$ setting is still favored if the dynamics in the videos is highly fluid.

Table 4.8: The proposed framework on MOT'16 (training) dynamic sequences (i.e. camera is moving). The bold figures indicate better performance.

MOT16-05						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	**30.9**	71.6	**743**	**3949**	**90**	**119**
$n = 3$	21.4	**72.3**	751	4619	67	173
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	**42.9**	**80.0**	**5.3**	**51.1**	**43.6**	
$n = 3$	33.2	75.4	0.6	40.6	58.7	
MOT16-10						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	0.7	66.2	**1568**	11123	61	159
$n = 3$	**2.3**	**66.4**	1609	**10884**	**48**	**133**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	13.4	52.3	0.0	**26.3**	**73.7**	
$n = 3$	15.2	54.9	0.0	24.6	75.4	
MOT16-11						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	24.6	**75.1**	**371**	6710	36	111
$n = 3$	**25.3**	74.2	406	**6617**	**29**	**78**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	28.9	**88**	0.0	**30.7**	69.3	
$n = 3$	**29.9**	87.4	**1.3**	29.3	69.3	
MOT16-13						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	**0.6**	65.6	**714**	**10842**	19	68
$n = 3$	0.1	**65.8**	762	10867	**5**	**31**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	**6.9**	**52.8**	0.0	**11.8**	**88.2**	
$n = 3$	6.7	50.4	**0.9**	9.1	90.0	

4.3 Comparison with Other On-line Trackers

In this section, we compare the proposed framework with other on-line trackers, SORT [7] and Deep SORT [36]. SORT and Deep SORT both use the Faster R-CNN object detector with VGG16 [31]. Likewise, they pass only the probability of the *person* class that is over 0.5 to the output as the detection result. The major difference between Deep SORT and SORT is that Deep

Table 4.9: Comparison of average performance of $n = 1$ and $n = 3$ cases on seven MOT'16 test sequences. Parameter selection is done by the strategy introduced in Section 4.2. Bold figures indicate the winner cases.

	`MOTA`	`MOTP`	`FP`	`FN`	`IDs`	`FM`
$n = 1$	**4.4**	**69.4**	**8179**	165691	515	1057
$n = 3$	4.1	68.7	10201	**164335**	**366**	**826**
	`Recall`	`Precision`	`MT` (%)	`PT` (%)	`ML` (%)	
$n = 1$	9.1	**67.0**	**2.1**	**27.4**	**70.5**	
$n = 3$	**9.9**	63.8	1.6	26.2	72.2	

Table 4.10: The proposed framework on MOT'16 (test) static sequences (i.e. the camera is not moving). The bold figures indicate better performance.

MOT16-01						
	`MOTA`	`MOTP`	`FP`	`FN`	`IDs`	`FM`
$n = 1$	**-11.4**	**64.1**	**1429**	**5661**	**31**	61
$n = 3$	-16.5	63.6	1820	5593	36	**57**
	`Recall`	`Precision`	`MT` (%)	`PT` (%)	`ML` (%)	
$n = 1$	11.5	**33.9**	0.0	**30.4**	69.6	
$n = 3$	**12.5**	30.6	**4.3**	26.1	69.6	
MOT16-03						
	`MOTA`	`MOTP`	`FP`	`FN`	`IDs`	`FM`
$n = 1$	1.5	**67.3**	**1056**	101930	52	122
$n = 3$	**2.5**	66.8	1480	**100449**	**28**	**79**
	`Recall`	`Precision`	`MT` (%)	`PT` (%)	`ML` (%)	
$n = 1$	2.5	71.3	0.0	3.4	96.7	
$n = 3$	**3.9**	**73.5**	0.0	**7.4**	**92.6**	
MOT16-08						
	`MOTA`	`MOTP`	`FP`	`FN`	`IDs`	`FM`
$n = 1$	**13.7**	**71.5**	**725**	13636	77	153
$n = 3$	12.7	71.2	1120	**13432**	**55**	**104**
	`Recall`	`Precision`	`MT` (%)	`PT` (%)	`ML` (%)	
$n = 1$	18.5	**81.1**	**4.8**	27	68.3	
$n = 3$	**19.7**	74.7	3.2	**31.7**	**65.1**	

SORT devises a new association cost matrix which is jointly defined by IoU and deep features instead of solely IoU. The deep features are extracted from another deep neural net specifically trained on a pedestrian dataset [39] used for person re-identification purposes. This modification increases `MOTA` by 1.6 and improves `IDs` by 45%. The comparison results are shown in Table 4.12.

Table 4.11: The proposed framework on MOT'16 (test) dynamic sequences (i.e. camera is moving). The bold figures indicate better performance.

MOT16-06						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	**18.4**	70.2	**2449**	**6757**	213	397
$n = 3$	12.6	**70.3**	2480	7436	**169**	**368**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	**41.4**	**66.1**	**4.5**	**54.8**	**40.7**	
$n = 3$	35.6	62.3	1.8	45.2	52.9	
MOT16-07						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	**3.1**	**67.8**	**1104**	14669	48	131
$n = 3$	1.7	67.2	1605	**14402**	**39**	**94**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	10.1	**60.0**	0.0	18.5	81.5	
$n = 3$	**11.8**	54.5	0.0	**24.1**	**75.9**	
MOT16-12						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	**24.0**	**73.2**	**404**	**5859**	43	92
$n = 3$	21.7	71.5	575	5900	**24**	**70**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	**29.4**	**85.8**	2.3	**32.6**	65.1	
$n = 3$	28.9	80.6	**4.7**	30.2	65.1	
MOT16-14						
	MOTA	MOTP	FP	FN	IDs	FM
$n = 1$	**1.3**	64.1	**1012**	17179	51	101
$n = 3$	1.2	**64.3**	1121	**17123**	**15**	**54**
	Recall	Precision	MT (%)	PT (%)	ML (%)	
$n = 1$	7.1	**56.3**	0.6	12.2	**87.2**	
$n = 3$	**7.4**	54.8	0.6	**14.0**	85.4	

It is clearly shown that the proposed framework is behind both SORT and Deep SORT in most of the metrics by a large margin. The proposed framework suffers from high `FN` across all the test sequences, thereby resulting in low `MOTA` at around only 4. We discuss the cause of this in more details in the coming section.

Table 4.12: Comparison of the average performance of the $n = 1$ and $n = 3$ cases with SORT [7] and Deep SORT [36] on seven MOT'16 test sequences. Bold figures indicate the winner cases.

	`MOTA`	`MOTP`	`FP`	`FN`	`IDs`	`FM`
Ours, $n = 1$	4.4	69.4	**8179**	165691	515	1057
Ours, $n = 3$	4.1	68.7	10201	164335	**366**	**826**
SORT [7]	59.8	**79.6**	8698	63245	1423	1835
Deep SORT [36]	**61.4**	79.1	12852	**56668**	781	2008
	`Recall`	`Precision`	`MT` (%)	`PT` (%)	`ML` (%)	
Ours, $n = 1$	9.1	67.0	2.1	27.4	70.5	
Ours, $n = 3$	9.9	63.8	1.6	26.2	72.2	
SORT [7]	-	-	25.4	**51.9**	22.7	
Deep SORT [36]	-	-	**32.8**	49	**18.2**	

4.4 Discussion

In this section, we first discuss the experiments conducted in the ablation studies. The discussion continues with the comparison made between the proposed framework and other on-line trackers. Finally, we discuss the possible improvements based on the observations made in the experiments.

4.4.1 On Ablation Studies

Firstly, we discuss the ablation studies presented from Section 4.2.1 to Section 4.2.5. In the studies, the detector threshold τ^{det} has shown to be most impactful to all of the metrics. Note that in these experiments we attempt to optimize `MOTA` score, and thus it is shown clearly (in Figures 4.1b and 4.1e) that their `FP`, `FN`, `MT`, and `PT` scores are not optimized along with `MOTA`. Hence, it is worth mentioning that if one would like to pursue tracking results which can cover as many as ground-truth tracks as possible, a lower detector threshold τ^{det} (than 0.6) should be applied for higher recall but in return of lower precision rate.

Secondly, we analyze the results obtained from the static video sequences in both training and test sets shown in Tables 4.7 and 4.10. It is observed that while only in few static sequences does the $n = 1$ case outscore the another case in `MOTA`, the $n = 3$ case achieves higher or equal `MT` and `PT` scores combined. On the one hand, the results indicate that the $n = 3$ case, which relies more on correlation filters that localize the targets, delivers more consistent and uninterrupted tracks. On the other hand, the $n = 1$

case, which fuses the information from trackers and detectors, does not offer significant improvements over these metrics. While it was our belief that the detections could calibrate the predicted target locations and prevent the tracker from drifting away, the association process could be itself noisy and instead provide interference to the trackers. This can happen if the cost matrix is not well-devised in the association process (refer to Section 3.2.3), thus providing noisy estimation of the association cost.

Thirdly, to analyze the results from dynamic video sequences, two observations are worth mentioning: (1) the $n = 1$ case draws significant improvement over the $n = 3$ case in almost all the metrics on the video *MOT16-05* in the training set and its counterpart test video *MOT16-06*, e.g. 31.5% and 30.74% boosts in `MOTA`, respectively. By watching the contents in *MOT16-05* and *MOT16-06*, one can observe that these are taken at 14 fps and possibly with a hand-held camera carried by a walking person. Large movement of the cameras is constantly presented and could create some difficulties for the trackers to track the target. (2) The $n = 1$ case draws significant improvement over the $n = 3$ case in `MOTA`, `MT`, `PT`, and `ML` on the test sequences. These oberservations develop the thought that if the contents in the sequences are changing rapidly due to high dynamics of the scene or low frame-rate production, triggering detectors more frequently is much demanding for consistent tracking results.

Fourthly, the $n = 1$ case performs worse in `IDs` and `FM` in almost every sequence than the other case. The cause of the unsatisfyingly many ID switches and fragmented tracks could be correlated with the point already discussed that the association may not be sufficiently reliable.

4.4.2 On Comparison with Other Online Trackers

Previously we have shown in Section 4.3 that the proposed framework is outperformed by SORT and Deep SORT by a large margin. For instance, the proposed framework incurs nearly three times `FN` and similar `FP` compared with that in Deep SORT. This indicates that the major difficulty is the proposed framework missing a huge number of detections, especially the misses on the pedestrians beyond some certain distances.

Figure 4.6 shows two cases where larger objects are always easier to detect than small and clutter ones. Hence, it could be the accuracy gap between the different detectors that leads to the performance gap between the proposed framework and Deep SORT (or SORT). As mentioned in [7, 36], a sophisticated object detector is much demanded in the tracking framework for high accuracy, e.g. simply replacing the base network used in Faster R-CNN from ZFNet [38] to VGG16 [31] can improve `MOTA` from 24 to 34.

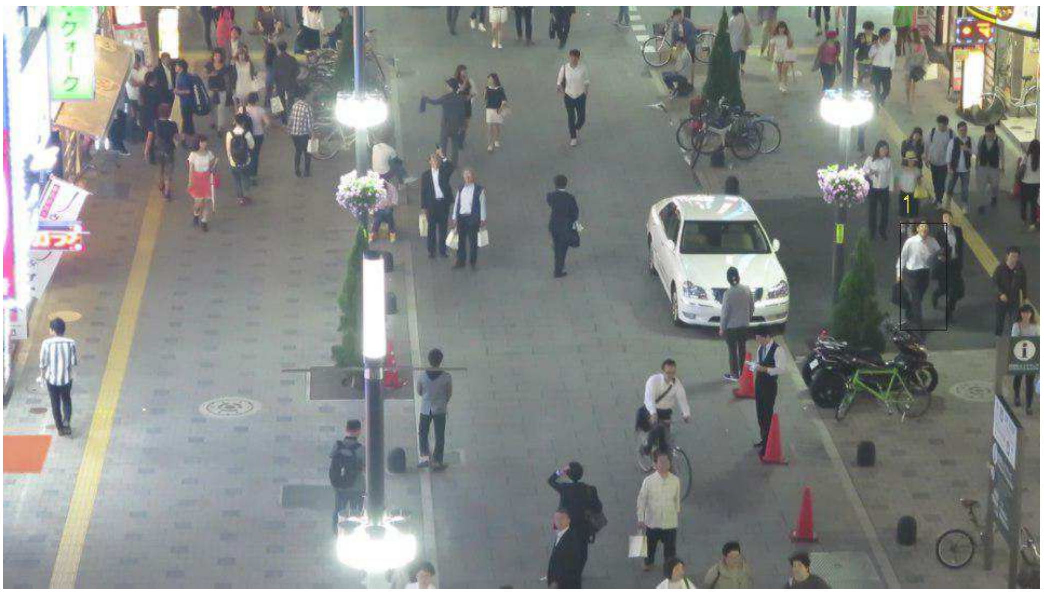
Despite the importance of selecting a highly accurate object detector, we emphasize that the aim of this work is to design a tracking framework being able to run at a reasonable speed without or with little modern GPU support. We cannot afford relying on the state-of-the-art object detector to boost the performance, instead we choose an object detector (introduced in Section 3.2.1) which runs at around 2.5fps on quad-core 2.6GHz CPU with no GPU support. In addition, we employ a more sophisticated single-target base tracker than the Kalman tracker used in SORT and Deep SORT, i.e. the correlation tracker that still ensures the real-time performance. By comparing $n = 1$ and $n = 3$ cases in the proposed framework shown in Table 4.12, the $n = 3$ case achieves comparable `MOTP` under similar `FN` with $n = 1$ case. This shows that the correlation tracker is able to localize the target accurately to some extent. However, the $n = 3$ case incurs higher `FP` which can be generated from the detector but can be also from the tracks that have been drifted to the wrong targets.

4.4.3 On Possible Improvements

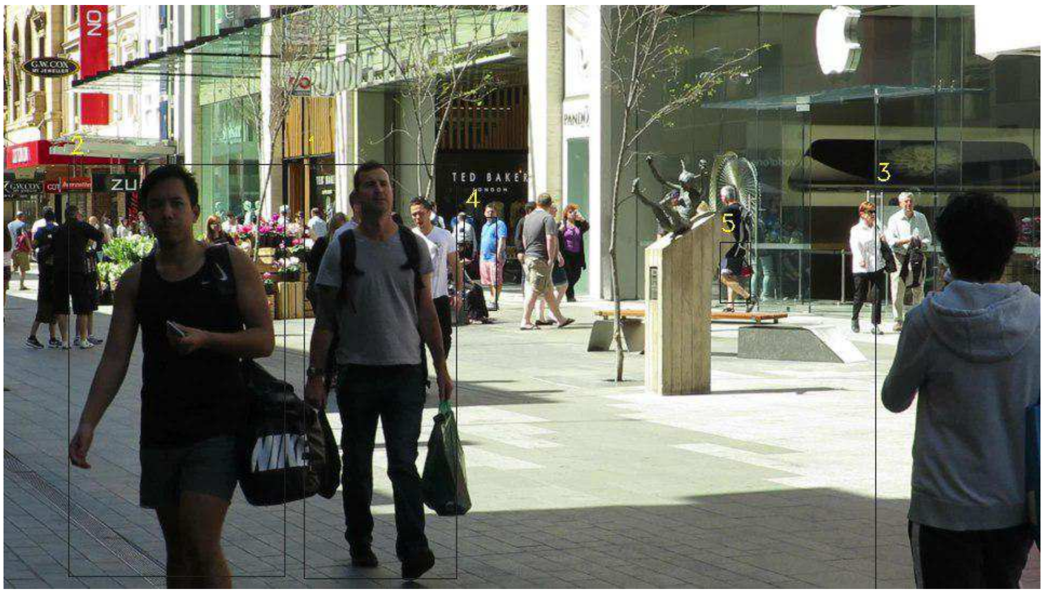
We provide a generic tracking framework in this work that does not require the availability of object detector in every frame. However, there is much room for improvement in the major constituents. We describe two possible improvements as follows, ordered by the suggested importance.

The pedestrian detector: It is of our knowledge that the detector's performance has the most impact on tracking performance. While we do not want to sacrifice the inference speed too much for the detection performance, alternatively one can train the same SSD network specifically on the large-scale pedestrian dataset (e.g. MARS [39] and KITTI [14]) initialized by the parameters in the SSD network used by this work. For improving the inference speed, one can remove all the neurons of classes in the classification layer except the *person* class. Thus, the computation will not be wasted on inferring other classes always ignored.

The deep features: Deep features incorporated in the framework plays an important role whenever data association takes place, e.g. data association for associating results from detector and trackers or for person re-identification. As shown in the experiment results triggering the detector more frequently does not show significant improvement over the metrics in general, despite that the detector we employ is of low recall rate, the association could possibly be improved by further reducing the false positive and false negative rates. For that, the discriminative power and descriptiveness



(a) A frame in *MOT16-03*. Only one detection is shown close to the right border.



(b) A frame in *MOT16-08*. Five detections are shown.

Figure 4.6: The top image shows when captured at far-range distance, the detector employed in the proposed framework often misses many small detections. The bottom image shows a relatively easier case for the detector as the pedestrians are close enough to the camera.

of the deep features should be improved. In addition, recall that we calculate the similarity between two auxiliary templates by averaging their channel-wise similarities where each template is of $24 \times 24 \times 64$ dimensions. However, empirically we observe that the features within some of the channels are sparse, i.e. many features are of zero values. On the one hand, this could lead to the curse of dimensionality where the distance measure becomes less meaningful. On the other hand, it may also indicate that the features we extract may not be sufficiently condensed and informative. Hence, to improve the deep features, it is suggested that one can train a network specifically on pedestrian data and use it for feature extraction [36]. What is more, to mitigate the curse of dimensionality, the features can be extracted from a fully-connected layer near the end of the network, thus the features would be of single channel and lower dimensionality (e.g. 128-D) [36].

Chapter 5

Conclusions and Future Work

5.1 Conclusions

We presented in this thesis an on-line detect-and-track framework which aims to be operated in real-time without or with minimal GPU support. Unlike most of the multiple object tracking systems, the proposed framework does not assume the detector's availability in every frame as object detection is usually the largest computational burden in such systems. Overall, the proposed system works as follows. The tracker localizes the targets itself or with the information (e.g. estimated location of a target) periodically provided by the detector. During the course of tracking, a track of a target is constructed based on continually receiving above-the-threshold similarity measure between the target's auxiliary templates in the two consecutive frames. In other words, a track is interrupted if the auxiliary templates are dissimilar to some extent, and a track is removed from the active tracks if there are too many interrupts. However, the removed tracks are moved to history tracks in which the tracks still have chances to be recovered in the future frames. The proposed system devises Single Shot Detector and correlation filter as the object detector and tracker, respectively. All relevant similarity measurements are based on the distance between the features in the Euclidean space extracted from the deep neural net.

We conducted experiments on the MOT'16 challenge dataset to demonstrate how the framework performs under full and partial availability of the detector, i.e. the detector is triggered in every frame versus in every three frames. The main findings in the experiments are: (1) in static sequences (where the camera is not moving) the case with partial availability of the detector achieves comparable or slightly better performance than the other case, however, (2) in dynamic sequences with a moving camera or when the dynamics in the video are high (i.e. when people are moving faster), the case with full availability of the detector tends to outperform that with partial availability of the detector, and (3) comparing the proposed framework with two other recently published on-line trackers, SORT and Deep SORT trackers, the proposed framework is underperformed in the MOTA scores. These trackers, however, leverage a more sophisticated object detector which we cannot afford due to the excessive computational burden.

5.2 Future Work

Besides the improvements already suggested in Section 4.4.3, it is also important to investigate how to share the features among the detector, tracker, and data association stages. Currently, in the proposed framework, the detector uses its own network model to do the inference while the tracker utilizes pixel values and a histogram of oriented gradients as the features. Sharing the features in similar tasks may bring several benefits, such as a higher level of generalization and less wasted computations, as suggested in [26, 28].

A recent publication proposed a two-way Siamese-like networks (i.e. two network streams fed with the frames at time t and $(t + 1)$ as inputs respectively) to allow the system learn object representation and localization end-to-end [6]. Hence, it would be interesting to extend their work to one that learns object representation, detection, and localization given two consecutive frames. In addition, while the state-of-the-art object detectors predominantly consider only spatial information from a single frame, in the context of object tracking, temporal information can be considered and possibly used to enhance the detection accuracy and consistency over frames if the detection and tracking are performed within a unified network. We leave the aforementioned as some thoughts for the future development of the project.

Bibliography

[1] `https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits.html`.

[2] `https://github.com/tensorflow/models/blob/master/research/slim/nets/inception_v2.py`.

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

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] Babenko, B., Yang, M.-H., and Belongie, S. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 983–990.

[6] Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. Fully-convolutional siamese networks for object tracking. In *European Conference on Computer Vision* (2016), Springer, pp. 850–865.

[7] Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. Simple online and realtime tracking. In *Image Processing (ICIP), 2016 IEEE International Conference on* (2016), IEEE, pp. 3464–3468.

[8] Bishop, G., and Welch, G. An introduction to the kalman filter. *Proc of SIGGRAPH, Course 8*, 27599-23175 (2001), 41.

[9] Bolme, D. S., Beveridge, J. R., Draper, B. A., and Lui, Y. M. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (2010), IEEE, pp. 2544–2550.

[10] Bruff, D. The assignment problem and the hungarian method. *Notes for Math 20* (2005), 29–47.

[11] Dai, J., Li, Y., He, K., and Sun, J. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems* (2016), pp. 379–387.

[12] Danelljan, M., Häger, G., Khan, F., and Felsberg, M. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014* (2014), BMVA Press.

[13] Danelljan, M., Hager, G., Shahbaz Khan, F., and Felsberg, M. Convolutional features for correlation filter based visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2015), pp. 58–66.

[14] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)* (2013).

[15] Girshick, R. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1440–1448.

[16] Grabner, H., and Bischof, H. On-line boosting and vision. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* (2006), vol. 1, IEEE, pp. 260–267.

[17] Grabner, H., Leistner, C., and Bischof, H. Semi-supervised on-line boosting for robust tracking. *Computer Vision–ECCV 2008* (2008), 234–247.

[18] He, K., Zhang, X., Ren, S., and Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision* (2014), Springer, pp. 346–361.

[19] Henriques, J. F., Caseiro, R., Martins, P., and Batista, J. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 37*, 3 (2015), 583–596.

[20] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[21] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., and Murphy, K. Speed/accuracy trade-offs for modern convolutional object detectors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[22] Kalal, Z., Mikolajczyk, K., and Matas, J. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence 34*, 7 (2012), 1409–1422.

[23] King, D. E. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research 10* (2009), 1755–1758.

[24] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[25] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision* (2014), Springer, pp. 740–755.

[26] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. In *European conference on computer vision* (2016), Springer, pp. 21–37.

[27] Milan, A., Leal-Taixé, L., Reid, I., Roth, S., and Schindler, K. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831* (2016).

[28] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99.

[29] Rowley, H. A., Baluja, S., and Kanade, T. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence 20*, 1 (1998), 23–38.

[30] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).

[31] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[32] Stiefelhagen, R., Bernardin, K., Bowers, R., Garofolo, J., Mostefa, D., and Soundararajan, P. The clear 2006 evaluation. In *International Evaluation Workshop on Classification of Events, Activities and Relationships* (2006), Springer, pp. 1–44.

[33] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *CoRR abs/1512.00567* (2015).

[34] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. Selective search for object recognition. *International journal of computer vision 104*, 2 (2013), 154–171.

[35] Viola, P., and Jones, M. J. Robust real-time face detection. *International journal of computer vision 57*, 2 (2004), 137–154.

[36] Wojke, N., Bewley, A., and Paulus, D. Simple online and realtime tracking with a deep association metric. *CoRR abs/1703.07402* (2017).

[37] Wu, B., and Nevatia, R. Tracking of multiple, partially occluded humans based on static body part detection. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* (2006), vol. 1, IEEE, pp. 951–958.

[38] Zeiler, M. D., and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision* (2014), Springer, pp. 818–833.

[39] Zheng, L., Bie, Z., Sun, Y., Wang, J., Su, C., Wang, S., and Tian, Q. Mars: A video benchmark for large-scale person re-identification. In *European Conference on Computer Vision* (2016), Springer, pp. 868–884.