Exploiting Motion and Visibility Cues for Tackling Occlusions in Multiple Object Tracking

Student: Tianji Liu
Supervisor: Prof. Gabriel J. Brostow
MSc Data Science and Machine Learning
August 2020

This report is submitted as part requirement for the MSc Degree in Data Science and Machine Learning at University College London. It is substantially the result of my own work except where explicitly indicated in the text.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Department of Computer Science
University College London

Abstract

Multiple object tracking is a visual recognition task combining detection and tracking, which considerably reflects conditions in real-world applications. Recent researches in this field focus on improving detection or localization precision of objects but neglect preserving their identity information during occlusions. In this thesis we propose a novel neural motion model for motion prediction that is able to adapt to scenes with or without occlusions and obtain relatively precise predictions in both cases. Our motion model consists of a motion regressor that works in a residual learning fashion and an instance-aware visibility estimator that is crucial to our occlusion handling mechanism. We evaluate our method against the well-optimized tracker Tracktor++. The one-step motion prediction loss of our motion model is a third of the loss of Tracktor++ on validation split of MOT17 dataset, and our model performs even better on the occluded examples. In tracking settings our method has fewer id-switches. We show that there is still room for improvement in terms of localization ability and track management of our method, which causes its failure against Tracktor++ on some of the MOT metrics. Our method serves as a possible direction for designing occlusion-aware multiple object trackers.

Acknowledgements

I would like to express sincere gratitude to my supervisor Professor Gabriel Brostow, for his time donated in supporting this project, helpful comments and invaluable advice, as well as providing computational resources and a LaTeX thesis template. I would like to thank Mohamed Sayed and other PhD students in Prism group for their useful suggestions on practical and technical issues. I am also grateful to Wan Jing Song and Dongdong Chen for their encouragements.

Contents

1 Introduction **1**

1.1 Organization of the Thesis 2

2 Background and Related Work **3**

2.1 Object Detection 3

2.1.1 Two-stage Detectors 4

2.1.2 One-stage Detectors 5

2.1.3 Anchor-free Detectors 5

2.2 The Stage-wise Paradigm of Multiple Object Tracking 6

2.2.1 Similarity Representation 7

2.2.2 Association 8

2.3 Single-stage Multiple Object Tracking 8

2.4 Motion Model 9

2.4.1 Models for Rigid Motion 9

2.4.2 Simple Non-rigid Motion Models 9

2.4.3 Kalman Filter 10

2.4.4 Optical Flow Motion Models 10

2.4.5 Historical Features and Recurrent Motion Models 11

3 Methods **13**

3.1 Neural Motion Model 13

3.1.1 Overview 13

3.1.2 Appearance Feature Extractor 14

3.1.3 Identity Branch 16

3.1.4 Visibility Estimator 17

3.1.5 Motion Residual Regressor 17

3.1.6 Overall Forward Algorithm of Neural Motion Model 19

3.1.7 Loss 19

3.2 Tracking Algorithm 21

3.2.1 Image Alignment 22

3.2.2 Association 22

3.2.3 Track Management 22

4 Experiments and Results **25**

4.1 Dataset 25

4.2 Evaluation Metrics 25

4.2.1 Motion Prediction Loss 25

4.2.2 Multiple Object Tracking Metrics 26

4.3 Training 28

4.4 Motion Prediction Evaluation 29

4.5 Multiple Object Tracking Evaluation 30

4.5.1 Qualitative Results 30

4.5.2 Quantitative Results 32

4.6 Ablation Experiments 34

5 Discussion **37**

5.1 Localization 37

5.2 Track Management 38

5.3 Occlusion Handling 38

6 Conclusion **40**

6.1 Practical Issues and Future Work 40

Appendices **43**

A Supplemental Experiments **43**

Bibliography **44**

List of Figures

2.1 The architecture of Fast R-CNN and Faster R-CNN. 4

3.1 The structure of Neural Motion Model. 14

3.2 The structure of appearance feature extractor. Multi-scale outputs of FPN and feature map assignments for RoI pooling are omitted in the figure for simplicity. 15

3.3 The structure of identity branch and visibility estimator. 17

3.4 The structure of the main branch of Neural Motion Model, including motion residual regressor, gating operation and the calculation for output motion. 19

3.5 The overall framework of our tracking method. 21

4.1 An example illustrating the heuristic of CLEAR MOT, adopted from [3] and rearranged. Identity A represents a ground truth trajectory, and identities 1 and 2 are the corresponding predicted trajectories. The idea of CLEAR MOT is that if predicted trajectories are assigned to ground truth trajectories globally and mismatches are accumulated for each frame, then the penalization for (a) and (b) should be different, but essentially the error made by the tracker in this two scenarios are the same (*i.e.,* one id-switch) and it should not be considered relevant to the time. 26

4.2 An example illustrating the heuristic of ID metrics, adopted from [39] and rearranged. The tracker in (a) has higher CLEAR MOT scores than tracker in (b) since it has less id-switches. However, tracker in (b) has higher IDF1 score because trajectory 1 is assigned to be the "correct" one by the algorithm of ID metrics, and the correspondence remains longer in (b) than in (a). Please compare this figure with Figure 4.1. 28

4.3 Two examples showing that Neural Motion Model is able to preserve identities when occlusion between two interested objects happens. In each sub-figure three frames selected before, during and after an occlusion are shown. The two objects of interest involved in the occlusion are highlighted for clarity. When generating this visualization result, we removed non-maximum suppression for terminating overlapping tracks in the track management algorithm of Tracktor++, otherwise the potential id-switches can not be observed. 31

4.4 Visualization results of ablation experiments. In this example all the methods except the original Neural Motion Model failed to preserve the identity of the occluded pedestrian. Please compare this example with Figure 4.3(c) and 4.3(d). 36

5.1 An example of localization failure. Neural Motion Model is unable to precisely localize the two person in the middle of the image for about 150 frames. 38

5.2 An failure example of occlusion handling by Neural Motion Model. In the beginning of the occlusion (between the first and second frame shown above), the model tries to predict a precise bounding box for the partly occluded object, and thus its velocity slows down (second frame). Since the model relies greatly on previous motion during the following period when the object is almost fully occluded, the predicted track lags behind the occluded object afterwards (third frame) and is then terminated. 39

List of Tables

4.1 Motion prediction evaluation results of Neural Motion Model compared with Tracktor++ and constant velocity motion model on MOT17 Validation Dataset. "low vis" refers to visibility values less than 0.3, and "high vis" refers to visibility values greater than 0.7. Lower loss is preferred. Neural Motion Model outperforms Tracktor++ on average and for low visibility samples, while Tracktor++ gives better results for high visibility samples. Although constant velocity motion model seems to have great advantages over the other two in this experiment, it can not handle the dynamics in long-term tracking. . 29

4.2 Multiple object tracking evaluation results of our method (Neural Motion Model) compared with Tracktor++ on MOT17 Validation Dataset. Direction of an arrow indicates either a high or low value is preferred for the corresponding metric. Our method have less id-switches but failed to outperform Tracktor++ on the other three metrics. 32

4.3 Motion prediction evaluation results of ablation experiments on MOT17 Validation Dataset. Details of which component/input is enabled in the model is also specified in the table. "low vis" refers to samples whose visibility values are less than 0.3, and "all" refers to all samples. Lower loss is preferred. The motion prediction performance worsens in all the three experiments, compared with the original method. 34

4.4 Multiple object tracking evaluation results of ablation experiments on MOT17 Validation Dataset. Direction of an arrow indicates either a high or low value is preferred for the corresponding metric. All the metrics worsen in all the three experiments. Please also refer to Table 4.3 for details of model components/inputs enabled in each experiment. . 35

A.1 Comparing the performance of Tracktor++ with or without non-maximum suppression (NMS) on multiple object tracking metrics, evaluated on MOT17 Validation Dataset. Direction of an arrow indicates either a high or low value is preferred for the corresponding metric. After NMS being disabled, all the metrics worsen sharply. Note that our method do not apply NMS when object-object overlap happens. 43

A.2 Ablation experiment on visibility prediction of disabling identity branch of Neural Motion Model. Please refer to Equation 3.3 for the definition of visibility prediction loss. The loss increases after identity branch being disabled. 43

List of Algorithms

3.1 Forward algorithm of Neural Motion Model 20
3.2 Tracking algorithm 24

Chapter 1

Introduction

In the field of visual recognition, detection and tracking have always been cornerstones of higher level visual tasks as well as key components in real-world applications. With the boom of deep learning, researches in detection and tracking witnessed rapid and remarkable improvements of benchmark scores [24, 32] in recent years. Despite of that, high performance on individual benchmarks does not necessarily mean reliability in real-world applications, where the system is required to make decisions based on multimodal information in complex scenes. Multiple object tracking (MOT) is a visual task that closely relates to real applications such as autonomous driving and surveillance. Technically, MOT can be regarded as the combination of detection and tracking, but it remains to be a challenging task where a tracker has to handle object interactions, occlusions, deformable objects, etc.

Early researches usually formalize MOT as a stage-wise problem, and the two key components in this pipeline are detection and data association. The core idea of this paradigm is to associate newly detected bounding boxes to the existing old tracks, and researches mainly focused on the representation of similarity scores/costs between tracks and detections that are used in association [4, 40, 36]. Research progress was slow, however, and the performance of multiple object tracker reached a bottleneck. It had stayed in a plateau for a long time, until last year a tracker named Tracktor++ [2] was proposed which handles MOT problem in a single stage, as opposed to the former multi-stage pipeline. This method that emphasizes on predicting track states is favored by following works and the research focus has changed to how to precisely provide such predictions [51, 34]. Particularly, ideas borrowed from state-of-the-art object detectors [52, 31] are adapted to trackers, which significantly improves tracking accuracy.

Although this new wave reveals the importance of exploiting appearance information, it might also lead researches into a detour. In fact, recent researches of MOT emphasize too much on localization accuracy but omit preserving object identities during occlusions. In some real applications, positions of objects during occlusion is also of interest. For example, in autonomous driving and surveillance, positions of occluded objects can be helpful in terms of warning the system of potential dangers so that the system can make preparation in advance. On the other hand, there are some researches that aim to provide possible positions of occluded objects [1, 41], but to our knowledge, all of such trackers leverage geometric information (*i.e.,* bounding box coordinates or velocity) only and thus their prediction accuracy is limited, since none of them relies on appearance information which has been proved to be

beneficial to tracker performance.

These gaps in previous works naturally lead us to the question that we try to discuss in this thesis: is it possible for a tracker to directly tackle occlusions by predicting relative precise locations for occluded objects? We show that the answer to this question is yes. Specifically, we propose a trainable deep neural network motion model that utilize not only appearance but also geometric information for predicting future states of existing tracks. In addition, we leverage provided visibility labels in the MOTChallenge [35] dataset to train a visibility estimator as a part of the motion model. The visibility estimator acts as an important role in our automatic occlusion handling mechanism which does not require any rule-based hard-coding when making predictions. For the overall tracking algorithm, we follow the previously mentioned single-stage paradigm. Our motion model is the core component of the tracking algorithm.

We evaluate our motion model and tracking method, comparing them with the highly optimized appearance-based tracker Tracktor++. Our model shows high performance on predicting positions for occluded objects, and yields encouraging results on MOT metrics. We hope that our work could provide a new direction for designing multiple object trackers in terms of tackling occlusions. Our code is made publicly available. [1]

Our main contributions can be summarized as follows:

- We propose a novel Neural Motion Model that successfully tackles occlusions in a direct and automatic way, yielding relatively precise predictions for occluded object states.
- We present a new way of leveraging visibility information in designing multiple object trackers.
- We evaluate our tracking algorithm and provide a comprehensive analysis of the results, showing that our method can be further improved to achieve state-of-the-art performance on MOT metrics.

1.1 Organization of the Thesis

This thesis is organized into the following chapters. In Chapter 2 we review the literature in the field of multiple object tracking and related areas. Next, in Chapter 3 we describe our proposed model and method in detail. Then, in Chapter 4 we evaluate our model and method, as well as analyzing the results. In Chapter 5 we further go deep into some failure examples, aiming to provide possible directions for future improvements. Finally we conclude in Chapter 6, and briefly discuss some practical issues and further modifications of the model.

[1] https://github.com/tefantasy/MSc_project

Chapter 2

Background and Related Work

In this chapter we give an overview of basic concepts and recent research progress on multiple object tracking and motion models, as well as other highly related areas. Specifically, we will first briefly introduce the field of object detection, which plays an very important role in recent multiple object trackers. Next we will give an introduction on the commonly used stage-wise paradigm of multiple object tracking problems, known as *tracking-by-detection*. We also mention single-stage MOT. Finally, we will review a few motion models in detail that are favored in recent MOT researches.

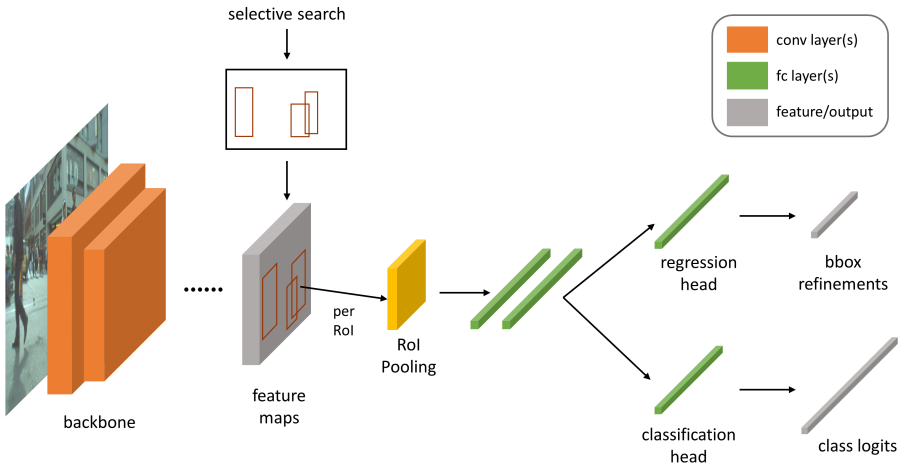
In particular, for object detection and MOT we mainly focus on the deep learning based methods. There is also rich literature in both of these fields before the era of deep learning, but it would be difficult to make a comprehensive review in this chapter.

2.1 Object Detection

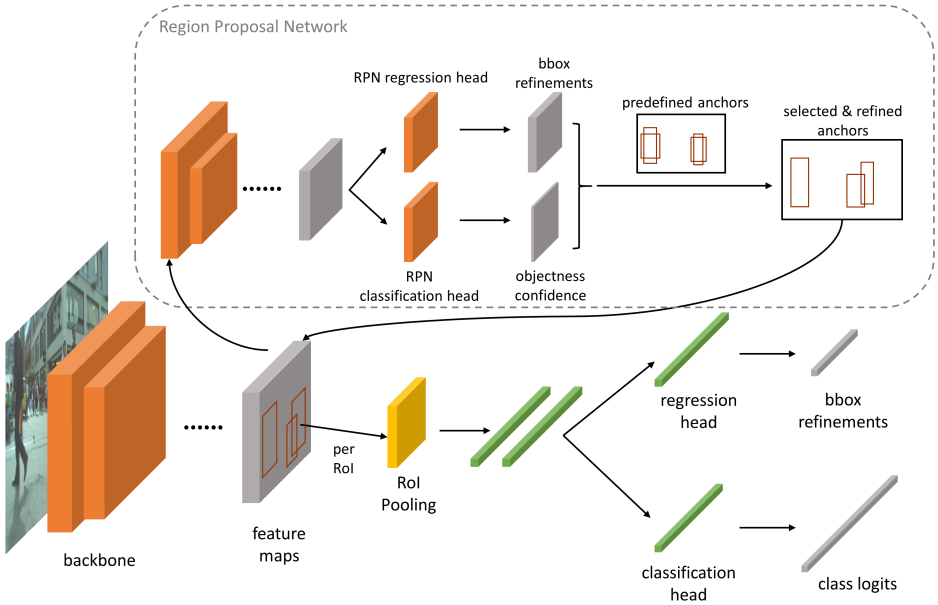
Object detection is a visual recognition task that has a wide range of applications in real world scenarios and serves as basic tools in more complex computer vision related tasks, *e.g.,* multiple object tracking, image captioning. The goal of object detection is to identify all recognizable objects and their locations in a given image, and assign a certain class to each object. Formally, given an input image, we aim to obtain a set of objects $\{(\mathbf{c}_i, \mathbf{b}_i)\}_{i=1}^{N}$. For each object i, $\mathbf{c}_i = \left[c_i^{(1)}, \ldots, c_i^{(C)}\right]^\top$ is the confidence score distribution over C classes, where $c_i^{(k)}$ denotes the probability of i being an object of class k. $\mathbf{b}_i = [x_i, y_i, w_i, h_i]^\top$ represents a bounding box of object i, where (x_i, y_i) denotes coordinates of the top-left corner (or the center) of the bounding box in the image, and w_i, h_i denote the width and height of the bounding box, respectively.

Object detection can be viewed as a multi-task learning problem that combines two fundamental tasks: localization and classification. Traditionally these two tasks are often tackled separately, by using search based methods (*e.g.,* sliding windows) for localization, and hand-crafted features and traditional machine learning classifiers (*e.g.,* SVM) for classification. In recent years deep convolutional neural networks (CNNs) have shown significant success on image classification [25, 17], and the translation equivariance property of CNNs can potentially contribute to object localization. Therefore, in recent studies CNNs are often used as basic frameworks for object detection.

In the remaining part of this section, we mainly review two-stage detectors, particularly the region-



(a) Fast R-CNN



(b) Faster R-CNN

Figure 2.1: The architecture of Fast R-CNN and Faster R-CNN.

based convolutional neural networks (R-CNNs) [14, 13, 38]. Also, we briefly introduce other types of deep object detectors, namely one-stage and anchor-free detectors.

2.1.1 Two-stage Detectors

The first two-stage deep object detector, R-CNN, was proposed by Girshick et al. [14], which also influenced the design of many recent detectors. The algorithm of R-CNN is quite simple. Firstly, a certain number of bounding boxes of object proposals (potential objects) are obtained by selective search [45], which is based on traditional image segmentation methods. Next the corresponding image part of each bounding box is cropped and resized to fit the input size of a CNN, and is then fed into the CNN to extract high-level semantic features. The final classification of each object proposal is performed by SVMs, and non-maximum suppression is performed to eliminate overlapping bounding boxes that may correspond

to the same object. R-CNN exploits the power of CNNs on automatically extracting image features, and thus has gained a significant boost on accuracy compared to the previous state-of-the-art detector. However, the algorithm is quite low-efficient since the feature extraction of object proposals can only be performed one by one, and many SVMs have to be trained because they are binary classifiers.

To deal with these problems, Girshick proposed Fast R-CNN [13], whose structure is shown as Figure 2.1(a). Object proposals are given by selective search, same as R-CNN. Next, the whole input image is fed into several fully convolutional layers (including pooling layers), or the "backbone" of Fast R-CNN, to extract features of the whole image in a single pass. Then, instead of cropping directly on the input image, the extracted features are cropped by the object proposals, whose positions are mapped from image coordinates to feature coordinates. This exploited the translation equivariance property of CNNs. The cropped features are further pooled to some fixed size representations (RoIPool). Finally, a two-branch fully-connected network predicts classification probabilities, and bounding box offsets for refining positions, scales and aspect ratios of the proposals.

Faster R-CNN, proposed by Ren et al. [38], aims to eliminate another speed bottleneck of Fast R-CNN, namely selective search, by introducing anchors and Region Proposal Network (RPN) for object proposing, shown as Figure 2.1(b). Anchors refer to a set pre-defined windows at each location of the feature maps extracted by the backbone network, which are of different scales and aspect ratios. RPN, on the other hand, takes those feature maps as input and for each anchor, it regresses offsets for refining positions of the anchor, and a class-agnostic confidence score indicating whether this anchor corresponds to an object or not, through a few fully convolutional layers. The remaining part of the network is essentially Fast R-CNN. Object proposals are provided by refined anchors with high confidence scores, and the aforementioned feature maps are used for RoI pooling.

Two-stage detectors follow the idea of "from coarse to fine". In the first stage, the proposals are class-agnostic and their positions are roughly predicted. Then in the second stage, the network begins to rely on fine-grained features and the positions are further refined. Empirically two-stage detectors have higher accuracy than their one-stage counterparts. However, speed is the major drawback of two-stage detectors.

2.1.2 One-stage Detectors

One-stage detectors can inference much more efficiently than two-stage detectors, but their accuracies are slightly lower. These detectors aim to predict both the class probablities and positions jointly. Liu et al. [33] exploit a strategy that is very similar to RPN [38] in their network SSD. Instead of predicting a single class-agnostic confidence score, SSD directly regresses C class scores and 4 bounding box coordinates. The network also utilizes feature maps generated by layers of different depths in the backbone. Since the spatial resolution of these feature maps are not the same, the pre-defined anchors are set to have same scales (but different aspect ratios) in the feature coordinates.

2.1.3 Anchor-free Detectors

Empirically, anchors are able to increase performance of detectors [37], while they also have potential drawbacks. Firstly, the performance of a detector is strongly related to the hyperparameters of anchors

(*i.e.,* scales, aspect ratios). Secondly, most of the anchors do not correspond to foreground objects, which introduces class imbalance problem in training. Recently anchor-free detectors become popular since they tackle these problems and detect objects in more direct ways.

CenterNet [52] formalize object detection as a keypoint detection problem. The network generates three kinds of maps whose resolutions are comparable to the input image: C heatmaps as class-aware confidence scores, 2 size maps as scales and aspect ratios of potential bounding boxes, and 2 maps as local position offsets. Points are selected if the confidence at these locations are local maxima, *i.e.,* its score is greater than the scores of nearest eight locations, and bounding boxes are built using the values of the size maps and offset maps at these locations.

CornerNet [27] aims to detect objects by predicting the top-left and bottom-right corner of each bounding box from two generated heatmaps. Since top-left and bottom-right corners need to be associated, for each corner an embedding vector is learned, and the similarity of embeddings of a pair of corners is expected to be high if the two corners belongs to the same bounding box.

2.2 The Stage-wise Paradigm of Multiple Object Tracking

Multiple object tracking (MOT) is a computer vision task that can be applied in the fields of autonomous driving, surveillance, etc. The goal of MOT is to detect or localize certain classes of objects in a video, and associate them across frames. That is, we would like to obtain a set of annotations $\{\mathcal{D}_t\}_{t=1}^{T}$. $\mathcal{D}_t = \left\{\left(\mathbf{b}_t^{(j)}, id_t^{(j)}\right)\right\}_{j=1}^{N_t}$ is the set of objects in frame t. $\mathbf{b}_t^{(j)}$ denotes the parameters of bounding box j, and $id_t^{(j)}$ is the assigned object id of bounding box j. For each object we would like the ids assigned to the corresponding bounding boxes to be the same across all frames. In practice, detections may be given as input in some benchmarks (*e.g.,* MOTChallenges [35]), and the models developed for this benchmark are required to generate data based on these detections for fair comparisons. This is referred to as *public detection* in MOTChallenges. On the other hand, *private detection* allows models to use their own object detectors to generate detections.

In this thesis we focus on *online* multiple object tracking, where trackers could only leverage the data in previous time steps for making decisions and generating results. The common approach of online MOT algorithms is *tracking-by-detection* that consists of four stages. For each time step, firstly, detection results in the new frame are extracted by detectors. Secondly, a motion model predicts the possible locations of previous obtained tracklets in this new frame. Thirdly, detections are associated to tracklets according to certain representations of similarity. Fourthly, based on the association results, tracklets are extended or terminated, and new tracklets may be initialized.

It should be pointed out that not all online MOT algorithms follow this paradigm, and in fact the algorithm we propose in this thesis follows a single-stage paradigm (see Section 2.3). However, similarity representation plays another important role (ReID) in our algorithm, so it is worth giving a survey of similarity representation as well as association and other issues that may need to be considered in MOT algorithms, *i.e.,* the third stage of *tracking-by-detection* paradigm. Motion models, or strategies of motion prediction will be comprehensively reviewed in Section 2.4, since it is the part that relates to our

work most closely.

2.2.1 Similarity Representation

Associating predicted bounding boxes of tracklets and direct detections is a common step in MOT algorithms. This is because these two types of information are usually complementary to each other. For example, if an object in the new frame in partially or fully occluded, the direct detection is thus not so reliable or even not provided, and one may want to depend more on the prediction results. By contrast, if the motion model is relatively weak (*e.g.,* constant velocity), then it can hardly predict precise locations of objects and the direct detections are more reliable. In order to make such associations, a scoring function that characterizes similarity or difference between candidates is needed.

Intersection over Union The most intuitive and commonly used similarity representation is Intersection over Union (IoU) between two candidate boxes, which is defined as

$$IoU(a, b) = \frac{|A(a) \cap A(b)|}{|A(a) \cup A(b)|}, \tag{2.1}$$

where $A(\cdot)$ denotes the area of a bounding box. Although IoU only considers geometric information, it is straightforward and fast to compute, so it can be applied in scenarios with sparse objects and few occlusions.

Bochinski et al. [5] proposed an IoU Tracker that does not apply any motion model (*i.e.,* assume static objects) and for each tracklet, a detection in the new frame that has largest IoU with the tracklet is greedily assigned to it. It achieves comparable accuracy with other trackers and is able to run at very high speed on a vehicle tracking dataset.

Similarity learning In more complicated scenarios, however, IoU is not sufficient for measuring similarity, and other types of information such as appearance and motion may also be considered. In these cases, scoring functions are usually modeled and learned by neural networks. Specifically, the aim is to obtain a mapping $f(\mathbf{x})$ to generate latent vectors for any input $\mathbf{x}$, such that the distance between latent vectors is a good representation of similarity of the two corresponding inputs. In the context of MOT, we would want the similarity representation to encode identity information, *i.e.,* for all the appearances and/or other information of an object (*e.g.,* a specific person), their representation in the latent space should be clustered. In our proposed algorithm, similarity representation is used for re-identifying (ReID) previously terminated tracks.

All kinds of losses has been proposed for similarity learning, and the most commonly used losses among them are contrastive loss [15]

$$L(\mathbf{x}_1, \mathbf{x}_2) = y^* \left\| f(\mathbf{x}_1) - f(\mathbf{x}_2) \right\|_2^2 + (1 - y^*) \left\{ \max\left(0, m - \left\| f(\mathbf{x}_1) - f(\mathbf{x}_2) \right\|_2 \right) \right\}^2, \tag{2.2}$$

where y^* denotes whether $\mathbf{x}_1$ and $\mathbf{x}_2$ represents the same object ($y^* = 1$ if same), m is a margin that prevents further contributions to the loss if the distance between representations are large enough when $y^* = 0$; or triplet loss [43]

$$L(\mathbf{x}_p, \mathbf{x}_n, \mathbf{x}_a) = \max\left(0, \left\| f(\mathbf{x}_a) - f(\mathbf{x}_p) \right\|_2^2 - \left\| f(\mathbf{x}_a) - f(\mathbf{x}_n) \right\|_2^2 + m\right), \tag{2.3}$$

where $\mathbf{x}_a$ is an anchor, $\mathbf{x}_p$ is an positive sample which has the same identity as the anchor, and $\mathbf{x}_n$ is an negative sample which corresponds to a different object.

Similarity learning has been widely applied in MOT. Voigtlaender et al. [46] utilize extracted appearance features of objects from a CNN to further learn association embeddings. This is performed in a multi-task fashion along with the learning of classification, localization and segmentation heads. Wojke et al. [47] design a similarity score that considers appearance and motion information at the same time. For appearance, embedding features are learned, and for motion they utilize Mahalanobis distance between new detections and the predicted states of their Kalman filter motion model. Sadeghian et al. [40] exploit appearance, motion and interaction information jointly. Interaction features are designed to be occupancy grids that models locations of other nearby objects. For each type of information, a Recurrent Neural Network (RNN) is used to encode all the historical information that belongs to a tracklet. Another RNN fuses all the three scores to form the final embedding vector.

2.2.2 Association

Hungarian algorithm [26] is most commonly used for association. The original association problem is formalized as finding maximum matching in a bipartite graph. The two disjoint sets of vertices of the bipartite graph are the new detections and tracklets respectively, and the weights of edges denote similarity scores. The goal is to find a subset of edges (associations) with maximum sum of weights (similarity scores) such that each vertex is linked to at most one vertex in the other set. While Hungarian algorithm is able to solve the association problem optimally, its computational complexity is quite high and it can not be accelerated by hardware such as GPU. Therefore, greedy [5] or heuristic [47] association is sometimes applied in MOT algorithms in order to trade accuracy for speed and simplicity, and in some cases [51] it is shown that there is no difference between the performance of greedy association and Hungarian algorithm in terms of tracking accuracy.

There has also been attempts on resolving the non-differentiability of the Hungarian algorithm. Xu et al. [49] proposed a RNN-based Deep Hungarian Net that is trained to yield approximated association results of the Hungarian algorithm. Therefore, the association process becomes differentiable as well and the MOT metrics can be directly optimized via back propagation. It is shown that the performance of trackers improved when trained with Deep Hungarian Net modules.

2.3 Single-stage Multiple Object Tracking

Although the multi-stage paradigm has been widely applied in previous works of MOT, in such settings the structures of trackers are usually loosely organized, which impedes smooth collaboration between individual components as well as improvement of tracking performance. Therefore, researches that try to jointly tackle some or all stages have also emerged. For instance, Feichtenhofer et al. [11] combine detection with motion prediction. Bergmann et al. [2] and Zhou et al. [51] integrate detection, motion prediction and association, so their trackers work in a *single-stage* fashion. Our proposed method also follows this paradigm.

Among them, Tracktor++ [2] is most closely related to our method. Tracktor++ is a tracker that

only consists of a Faster R-CNN [38] detector. The core idea is that at each time step, the bounding boxes coordinates of tracks are obtained by taking the bounding boxes at previous time step as object proposals, and directly refining them using the bounding box regressor of Faster R-CNN. More details about Tracktor++ are introduced in Section 3.2.

2.4 Motion Model

In this thesis, the module that predicts motion or future locations of objects using all kinds of known information (*e.g.,* historical appearances, previous motion or location, crowd information) is referred to as *motion model*. Motion model is one of the most important part in a multiple object tracker, and a variety of designs have been proposed, from traditional probabilistic models to recent deep learning based ones. We would like to emphasize that none of these motion models are equivalently effective for all multiple object tracking scenarios. It is not surprising that simple motion models can be competitive with delicately designed models in some cases. The key is that when designing a motion model, one should take consideration of data patterns and distributions, as well as coordination with other parts of the tracker.

2.4.1 Models for Rigid Motion

In many cases, rigid motion, or camera motion, is a non-negligible factor that may greatly affects the performance of motion models. For instance, in a very common application of panning or moving camera to follow a certain object, the object is generally static in the frame. If appearance information is used for modeling object motion, it can be misleading since the action of the object shows that it is moving but it actually does not in the frame coordinates. Meanwhile, the background is also moving in the direction against camera. Whilst high-capacity deep learning models might be able to handle these difficulties along with modeling individual object motions, separating rigid motion from non-rigid (object) motions and modeling them explicitly can be more effective.

The common approach for modeling rigid motion is to estimate a transformation between two frames [2, 18, 42]. Firstly, a set of pixels or keypoints are detected in the first frame using certain algorithms or heuristics. Next, the corresponding points in the second frame are found or tracked by methods such as exhaustive search (sliding window) or optical flow. Finally, a transformation matrix (euclidean, affine or homography) is estimated to represent a warping operation from the first frame to the second. In our proposed algorithm we also applied such methods for alignment of adjacent frames.

2.4.2 Simple Non-rigid Motion Models

In some scenarios where occlusions are rare and objects are not dense, simple models for objects are sufficient for predicting their motions. Constant velocity is a commonly used motion model which assumes the velocity of objects change very slowly. The motion between frame $t-2$ and $t-1$ is simply treated as the predicted motion between frame $t-1$ and t. Note that the motion of an object is not actually constant in a trajectory, because the association process follows motion prediction, which could alter the predicted locations. In order to reduce the variance of estimates, sometimes averaged motion of previous k time steps can also be used for prediction. Furthermore, if the absolute displacement of

objects is very small (*e.g.,* in high frame rate videos), the objects can be assumed to be static [2], *i.e.,* no motion model is applied to them.

2.4.3 Kalman Filter

Kalman filter [20] is a generative model based on first-order Markov process. It consists of two iterative updates, namely time update and measurement update. Time update, or prediction, is modeled as a state-to-state transition with Gaussian noise:

$$P(\mathbf{w}_t|\mathbf{w}_{t-1}) = \mathcal{N}\left(f_p(\mathbf{w}_{t-1}), \mathbf{\Sigma}_p\right), \tag{2.4}$$

where f_p denotes a linear mapping, $\mathbf{w}$ is the world state and $\mathbf{\Sigma}_p$ is the prediction noise. Similarly, measurement update, or correction, is also modeled as a linear mapping with Gaussian noise:

$$P(\mathbf{x}_t|\mathbf{w}_t) = \mathcal{N}\left(f_m(\mathbf{w}_t), \mathbf{\Sigma}_m\right), \tag{2.5}$$

where $\mathbf{x}$ denotes observation. At each time step, firstly world state distribution $P(\mathbf{w}_t|\mathbf{x}_{1\ldots t-1})$ is predicted using previous information $P(\mathbf{w}_{t-1}|\mathbf{x}_{1\ldots t-1})$. After having observed $\mathbf{x}_t$, the predicted distribution is corrected and $P(\mathbf{w}_t|\mathbf{x}_{1\ldots t})$ is obtained.

We then discuss the counterparts of these variables in MOT, taking SORT [4] as an example. World state $\mathbf{w}$, observation $\mathbf{x}$ and two linear mappings f_p, f_m are defined as

$$\mathbf{w} = \begin{pmatrix} \mathbf{b} \\ \dot{\mathbf{b}} \end{pmatrix}, \tag{2.6}$$

$$\mathbf{x} = \mathbf{b}, \tag{2.7}$$

$$f_p(\mathbf{w}) = \mathbf{F}\mathbf{w} = \begin{bmatrix} \mathbf{I} & \mathbf{I}dt \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{pmatrix} \mathbf{b} \\ \dot{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \mathbf{b} + \dot{\mathbf{b}}dt \\ \dot{\mathbf{b}} \end{pmatrix}, \tag{2.8}$$

$$f_m(\mathbf{w}) = \mathbf{H}\mathbf{w} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{pmatrix} \mathbf{b} \\ \dot{\mathbf{b}} \end{pmatrix} = \mathbf{b}, \tag{2.9}$$

where $\mathbf{b}$ denotes the four coordinates of a bounding box, $\dot{\mathbf{b}}$ represents motion, and dt is a scalar representing the update step size.

Kalman Filter is able to model the uncertainty of object states, which could be useful for long-term tracking with occasionally missing observations (*e.g.,* occlusions). However, it lacks the ability of modeling complex phenomena because of linear time and measurement update. There has been research that tries to integrate neural networks into Kalman Filter [23], but it would require using variation methods which makes learning and inference much more complicated.

2.4.4 Optical Flow Motion Models

Optical flow is an approach aiming to estimate the displacement (or motion) between the positions of a certain pixel or image patch in two images. In practice, the amount of pixels to be estimated can be arbitrary. If only the motions of a few pixels are estimated, then the results are called "sparse" optical flow. In applications of object tracking, it can be used to track points of interests without costly computations. Sanchez-Matilla and Cavallaro [42] use homography for rigid and non-rigid motion representation

in their work, and they estimate the positions of points in the current frame that matches the detected keypoints in the previous frame by sparse optical flow methods.

In the deep learning context, however, CNNs are good at generating two-dimensional vector fields, and hardware accelerators can largely mitigate the cost of computations. Therefore, it is common to directly predict optical flows for all pixels (which is called "dense" optical flow) using deep neural networks [9, 44]. We do not further review the methods of estimating dense optical flow since this is yet another rich research field, but rather focus on their application in MOT settings.

In MOT, optical flow estimator can be used as an off-the-shelf motion model that do not requires training on the tracking data, and several ways are proposed to utilize the generated dense vector field. Li et al. [29] directly take the motion vector at top-left and bottom-right corners of bounding boxes and translate them into new frames. Kang et al. [21] use the mean vector within a bounding box as the motion of corresponding object. These approaches are easy to implement but are not able to utilize most of the rich data. Besides, their accuracy is not guaranteed when there are camera motions, because the top-left and bottom-right corners of bounding boxes are mostly background points, and taking the mean vector may also fails in the "camera-follows-object" scenario (discussed in Section 2.4.1). To tackle this problem, Bullinger et al. [7] generate segmentation masks for the objects to be tracked (people, in their case) by using another off-the-shelf segmentation network, and then warp all the pixels that belongs to an interested object to the new frame, which gives more accurate estimation of locations.

An optical flow estimation network can also be trained using tracking data instead of using an off-the-shelf one, and it might performs better because the network could capture task-specific patterns. Here we also want to include those CNN-based and other feed-forward neural network models that estimate motion fields, since these models and optical flow motion models are similar in terms of both concepts and structures.

One of the examples is CenterTrack [51]. CenterTrack follows the structure of CenterNet [52] (discussed in Section 2.1.3), but it is trained such that the location offset maps now represent a motion field between two adjacent frames. Thanks to CenterNet's strong detection ability, CenterTrack performs quite well in MOT and has become one of the state-of-the-art trackers.

2.4.5 Historical Features and Recurrent Motion Models

Optical flow motion models exploit appearance features and are therefore powerful in estimating motions. However, these models only take consider of the last (and/or current) frame and neglect rich historical information in the previous frames which could be potentially useful.

Exploiting historical features Several methods have been proposed to make good use of these information. For example, the appearance of an object in one frame may not be reliable all the time since there could be motion blur or occlusion, but by involving appearance in a few previous frames we could have a more robust basis for estimation. A simple approach can be directly averaging historical appearance features that have been stored in a buffer. An improved version is to take weighted average of historical features, where the weights can be obtained by comparing features at each step with a learned vector that captures temporal relations [48]. This is inspired by the sequence-to-sequence attention approaches in

the domain of natural language processing. The weights can also be constructed using manually selected properties, *e.g.,* visibility of objects [18].

Recurrent motion models The methods of utilizing historical features introduced above are effective and easy to implement, but they require tuning hyperparameters such as the maximum number of historical frames, and considerable extra memory space for storing these historical features (usually one buffer per trajectory). Recurrent Neural Networks (RNNs) can avoid these problems, because RNNs can handle variable length of inputs, and the historical information are captured and represented by the hidden state of RNN. We briefly review two recurrent motion models as follows.

The most representative recurrent motion model is the one proposed by Milan et al. [36]. Their network is a customized RNN (rather than a vanilla one, LSTM or GRU) which consists of three parts: prediction, update and birth/death control. Firstly the network give a predicted object state based on the last known state of the object and its hidden state. Inspired by Kalman Filter, it then refines the prediction using the measurements (detection results) in the current frame, an association matrix (given by another data association network which we will not discuss further in detail), and previous confidences of targets being trajectories. The output of the network is the predicted and refined object state as well as confidences in current time step. Experimental results of this network is limited since only the geometry and location information are included in the object state. The authors claim that by integrating rich features such as appearance, the performance of the network could be significantly improved.

Babaee et al. [1] utilize RNN as a postprocessing method that works on results of any tracker. The aim of their auxiliary network is to correct id-switches caused by occlusion, or in other words, to connect tracklets that belong to the same object. The network is just a vanilla RNN, and their approach is extremely simple: input motions of an existing tracklet at each step, and after it ends, predict the motion of the object for a fixed number of steps. Although this postprocessing method can improve performance in terms of id-switches, it is questionable whether this results from the ability of RNN, because intuitively long-term motion history may not have significant effect on current motion prediction.

Chapter 3

Methods

In this chapter we first describe a novel motion model proposed for tackling occlusions in multiple object tracking in detail (Section 3.1). We name our model *Neural Motion Model*. Then we introduce other important parts of our tracking algorithm (Section 3.2).

3.1 Neural Motion Model

3.1.1 Overview

The overall structure of Neural Motion Model is shown as Figure 3.1. Neural Motion Model takes three kinds of data as input: the whole image of one frame, historical and current image patches of interested objects (RoIs), and object motion at previous time step. The output of the model is the predicted motion for each track at current time step.

Neural Motion Model mainly consists of four sub-modules: appearance feature extractor, identity branch (including ReID network and identity encoding network), visibility estimator, and motion residual regressor. We will introduce these four sub-modules in detail in the following sections, but here we would like to give a quick overview of the whole model.

The appearance feature extractor and ReID feature extractor are both designed to encode an image (or image patch) in latent space. The ReID features are trained to represent identity information of objects, whereas the appearance features focus more on classification and localization. Then, historical and current ReID features of a track are fed into and "compared" by the identity encoding network to check whether they represent the same object. The resulting feature and appearance feature are used as input of the visibility estimator which predicts a visibility value (between 0 and 1) of the corresponding object.

So far the visibility estimator seems to be useless. To understand its significance, we have to introduce the main branch of Neural Motion Model at the same time. On the main branch, motion prediction is performed in a residual learning fashion, similar to ResNet [17]. However, our motivation for doing this is quite different. Specifically, if the estimated object visibility is low, then the RoI appearance features are very likely to be unreliable, because part or most of the interested object is occluded and the object's appearance may not provide sufficient information for motion prediction. It is also likely that another interested object is the occluder (*i.e.,* in the front) of this object, and in this case the RoI

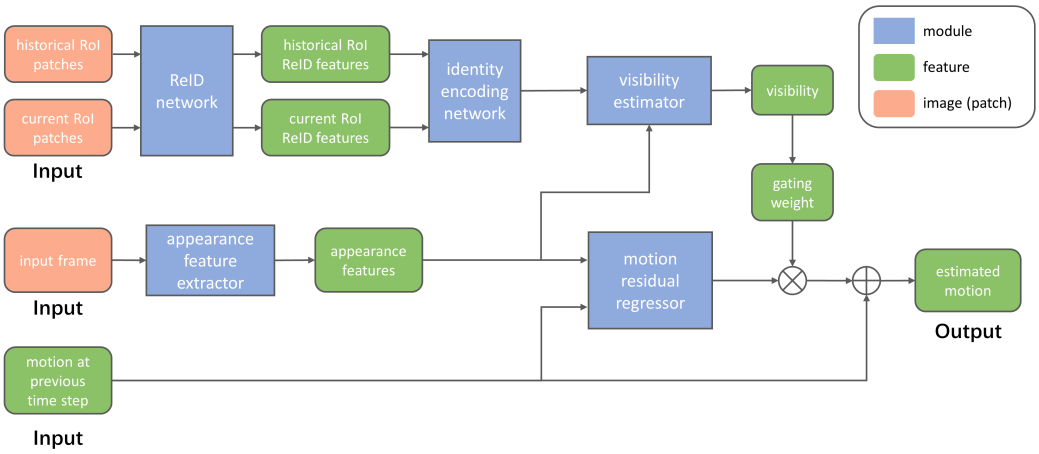


Figure 3.1: The structure of Neural Motion Model.

appearance features is mostly representing the occluder, thus misleads motion prediction.

Therefore, we use the estimated visibility as a *weight of gating* that controls the proportion of motion residual to be reserved. The motion residual is estimated by motion residual regressor from appearance features and previous motion. We do not want to rely too much on appearance features if object visibility is low, and in this case the gating weight is almost zero, so the output motion (previous motion plus motion residual) is basically the same as the previous one. Here we assume that the previous motion is a good estimated value for current motion, inspired by constant velocity motion model. On the other hand, if the object visibility is high, then the reliability of appearance features is also high. In this case the gating weight should be near one, and the estimated motion residual is mostly preserved, since effective appearance features may provide richer information for predicting motion.

3.1.2 Appearance Feature Extractor

The appearance feature extractor takes the whole image of current frame as input, and for each RoI a fixed length latent vector is encoded as the corresponding appearance representation. It consists of three parts: backbone, RoI pooling, and two appearance heads (see Figure 3.2 for details).

Backbone We use ResNet-50 [17] with Feature Pyramid Network (FPN) [30] as our appearance feature extraction backbone, which is identical to the backbone used in the Faster R-CNN detector of Tracktor++ [2]. This is to ensure consistency when we compare our model with Tracktor++ in the experiment sections.

FPN is designed to extract multiple feature maps with different scales from different depth of a convolutional network. Feature maps from shallow layers have larger scales and retain more spatial information, while those from deep layers retain more high-level semantics but their spatial information is largely lost. The purpose of FPN is to combine the advantages of shallow and deep feature maps. In our setting, FPN takes conv3, conv4, conv5 outputs of ResNet-50 as inputs. It then iteratively enriches the low-level, large scale feature maps by upscaling and adding high-level, small scale feature maps to the former ones. We refer readers to [30] for more details of FPN.

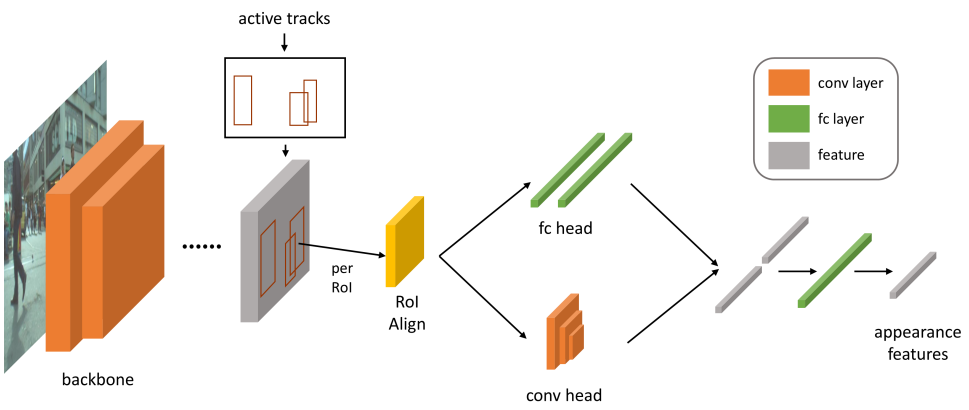


Figure 3.2: The structure of appearance feature extractor. Multi-scale outputs of FPN and feature map assignments for RoI pooling are omitted in the figure for simplicity.

RoI pooling Although in Faster R-CNN the vanilla RoIPool [13] has been used, here we use RoIAlign [16] for RoI pooling, following the implementation of Tracktor++. The purpose of RoI pooling is to perform adaptive pooling operations on patches of feature maps that are cropped at arbitrary locations with arbitrary sizes (given by bounding box coordinates), such that the pooled patches have fixed spatial resolutions (7×7 in our setting).

The difficulty is that bounding box coordinates are real numbers. The strategy of RoIPool to handle this is by dividing the bounding box area into cells such that the number of cells equals to the output resolution. Then for each cell, its coordinates are rounded up to nearest integers, and global pooling is performed in the cell to generate one spatial output. In contrast, RoIAlign is a more accurate approach. Firstly, for each cell four points at fixed locations are sampled. Then the value at each sampled point is computed by interpolation from the nearest four locations on the original feature maps. Finally global pooling is performed using the four sampled points.

There still remains a problem of how to adapt RoI pooling to multi-scale feature maps generated by FPN. We follows the methods proposed in the FPN paper [30]. According to the area s of a bounding box, the feature maps at k-th level are used for RoI pooling where k is determined by the heuristic of

$$k = \lfloor k_0 + \log_2(\sqrt{s}/224) \rfloor. \tag{3.1}$$

Here we do not go into further details, but one point need to be mentioned is that feature maps with larger scales (low k) are assigned to bounding boxes with smaller areas.

Appearance heads RoI pooled features are further fed into a two-headed network, namely the conv head and the fc head (Figure 3.2). The conv head consists of three convolution layers with 3×3 kernels, so the output resolution becomes 1×1. The fc head consists of two fully-connected layers with ReLU activations. The output dimension of both heads is 1024. Then the features from two heads are concatenated and fused by another fc-layer to get the final 1024-dimensional appearance features.

Although the two-headed structure appears to be redundant (the conv head in particular), empirically we find that removing either one of the heads would result in performance decrease. The reason

might be that the RoI pooled features still possess spatial information, so convolution operation is still favored for raising performance.

3.1.3 Identity Branch

The identity branch is designed to provide identity information to the visibility estimator. It consists of two parts: a ReID network and an identity encoding network (see Figure 3.3). Our motivation here is that if the visibility estimator only relies on RoI appearance features at current frame, then the estimated visibility might be incorrect where there are occlusions between interested objects.

Imagine two pedestrians overlap each other in the 2D image and both of them are correctly tracked before the occlusion. At the time step when occlusion happens, we still need to estimate the visibility for each pedestrian, which is used in downstream network for predicting their motions. For the pedestrian in the front, we would like to obtain a high visibility, while for the pedestrian in the back a low visibility is preferred. However, the problem here is that there is no significant difference between the coordinates of their bounding boxes at this time, so if the visibility is predicted only from RoI appearance features, then the estimated values for the two pedestrians should be almost same. More precisely, a high visibility would be given for both pedestrians. This might be acceptable for the pedestrian in the front, but totally wrong for the other one.

Therefore, we add identity branch to the visibility estimator so that it not only depends on appearance features but also on identity information of each track. The way we encode identity information to a track is by comparing its historical ReID features against its current ReID features. We use historical ReID features to represent the true identity information of an object. If there is large discrepancy between historical and current features, then this is a signal for the visibility estimator that it should not trust too much on the appearance features.

Unlike the common approach of taking the features at most recent k time steps as a gallery, here we use the features at first k time steps of a track's life, following [8]. In this period of time the object is less likely to be occluded. For aggregation of historical features, learnable weighted average is theoretically a good choice (discussed in Section 2.4.5) but we find that simple arithmetic average works better in our case.

Empirically we also confirm that identity information has positive effects on the prediction accuracy of the visibility estimator and the whole Neural Motion Model (see results of ablation experiments in Section 4.6 and Table A.2).

ReID network We directly use the off-the-shelf ReID network of Tracktor++. It is a ResNet-50 model that takes a cropped RoI image patch as input, and yields a latent ReID embedding of 128 dimensions. The network is trained using triplet loss (Equation 2.3) on MOT17 [35] training dataset. For more details we refer readers to [2].

Identity encoding network As discussed above, after obtaining historical and current ReID features of a track through the ReID network, the historical features are averaged and fed into the identity encoding network along with current ReID features, where they are "compared" to represent identity information.

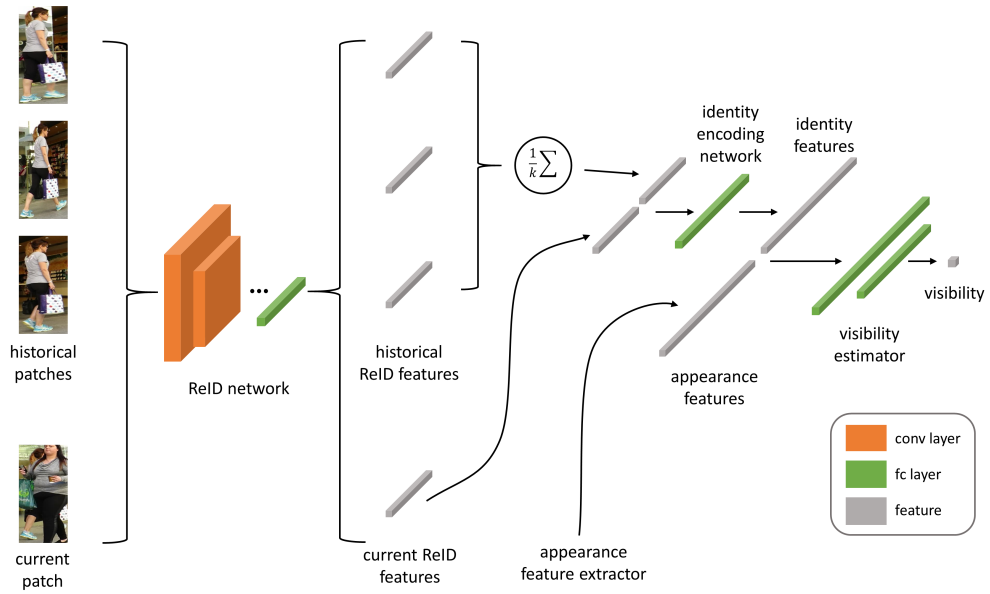


Figure 3.3: The structure of identity branch and visibility estimator.

The structure of identity encoding network is very simple: it is just a fully-connect layer with ReLU activation. Two kinds of features are concatenated (256-dimensional) and fused by the network. The output dimension is also 256. Although L2 distance between ReID features represents similarity of corresponding objects (Section 2.2.1), we do not explicitly introduce such operations but instead use concatenation, since a neural network with the latter one is able to represent a class of basic operations. Empirically we also find that concatenation yields better results.

3.1.4 Visibility Estimator

As discussed in Section 3.1.3, the identity branch provides features that contains information indicating whether the RoI appearance features truly correspond to *this* tracked object, since the object in the front dominates the appearance features.

As shown in Figure 3.3, the visibility estimator is composed of two fully-connected layers: first a layer with ReLU activation that accepts the concatenation of identity features and appearance features, and reduce the dimension to 512. Then another layer directly predicts a scalar output, followed by a sigmoid funtion squeezing the output into the range from 0 to 1. This is the estimated visibility value for the object.

3.1.5 Motion Residual Regressor

Motion embeddings Before we introduce motion residual regressor, first we need to clarify the embeddings of motions, *i.e.,* how the motions are represented at the input and output of our Neural Motion Model. To begin with, a certain style of bounding box coordinates need to be chosen. There are two kinds of possible representations: 1) (x_1, y_1, x_2, y_2), where (x_1, y_1) signifies top-left corner, and (x_2, y_2) is the bottom-right corner of the bounding box (which we refer to as "two-point" representation); 2)(x, y, w, h), where (x, y) denotes the top-left corner or the center point, and w, h denote the

width and height of the bounding box respectively (which we refer to as "point-scale" representation). We use the latter one, because scale is an intrinsic attribute of a bounding box (and the corresponding object). By contrast, the two-point representation only contains location (which is an extrinsic attribute) information. Although scales can be computed from the locations of two points, we want the network to be trained based on essential cues.

Using the point-scale representation of bounding boxes, we cannot encode motion by simple subtractions. This is because the distribution and the rate of change of locations and scales are quite different. Locations could be approximated as following a uniform distribution in the range of image height or width, whereas scales have a more aggregated distribution. Also, the rates of change of scales are relatively small on average, while the displacements of objects could be either large or small. Therefore, we adopt the form of coordinate refinements in bounding box regression proposed in [13] as our motion representation:

$$
\begin{aligned}
\mathbf{s}_1 &= (x_1, y_1, w_1, h_1), \\
\mathbf{s}_2 &= (x_2, y_2, w_2, h_2), \\
v_x &= \frac{x_2 - x_1}{x_1}, \\
v_y &= \frac{y_2 - y_1}{y_1}, \\
v_w &= \log \frac{w_2}{w_1}, \\
v_h &= \log \frac{h_2}{h_1}, \\
\mathbf{v}_{1,2} &= (v_x, v_y, v_w, v_h),
\end{aligned}
\tag{3.2}
$$

where $\mathbf{v}_{1,2}$ denotes the motion from the state of bounding box 1 ($\mathbf{s}_1$) to the state of bounding box 2 ($\mathbf{s}_2$). It has been proved effective in practice by previous works [13, 38, 11].

Motion residual regressor This is the core sub-module of Neural Motion Model. As discussed in Section 3.1.1, motion residual regressor predicts a residual value that is to be added to the input motion (or the motion at previous time step). Compared with the complicated structure of identity branch, the structure of this module is quite simple (shown in Figure 3.4): firstly the input motion is further embedded to a 512-dimensional latent vector through a fully-connected layer with ReLU activation, then it is concatenated to the appearance features and passed through a three-layer fully-connected network. The output dimensions of these three layers are 1536, 512 and 4. The first and second fc-layer are followed by a ReLU activation respectively, while the third layer's output is not activated. The reason is that the ranges of all the coordinates in our motion representation (see the "Motion embeddings" paragraph) v_x, v_y, v_w and v_h are expanded to the entire real number field.

Gating weight There still remains a small issue to be discussed, although technically it is not a part of the motion residual regressor. We mentioned that the estimated visibility is acting as a weight of gating that controls the amount of motion residual to be added to the input motion (Section 3.1.1). However, small differences still exists among the 4 coordinates in our motion representation. For example, the width of a bounding box may have larger changes than the height when the tracked objects are horizon-

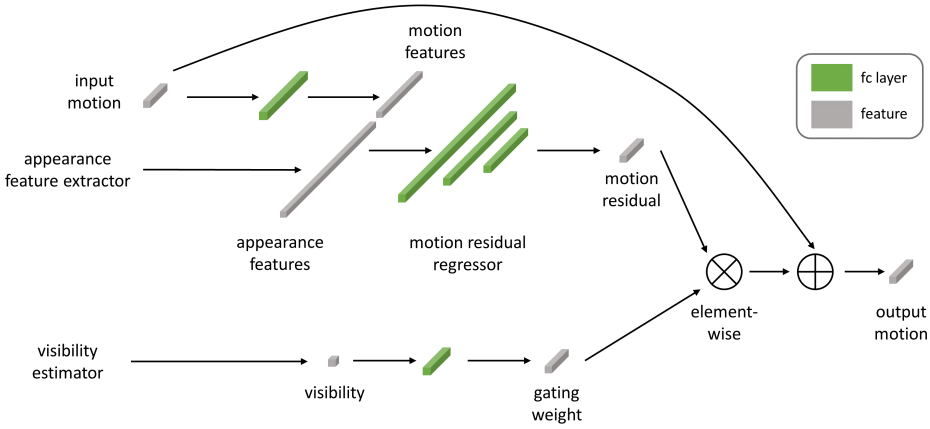


Figure 3.4: The structure of the main branch of Neural Motion Model, including motion residual regressor, gating operation and the calculation for output motion.

tally moving pedestrians, since their leg can move back and forth beyond the horizontal range of their body.

Therefore, we apply another small fc-layer that maps the visibility value to a 4-dimensional vector. It is activated by sigmoid function again, so now every coordinate in the motion representation has its own learnable gating weight. Empirically the performance slightly increases after applying this minor modification.

3.1.6 Overall Forward Algorithm of Neural Motion Model

The overall forward algorithm of Neural Motion Model can be summarized as Algorithm 3.1. Note that the historical ReID features are extracted in the outer tracking algorithm (see Algorithm 3.2) at previous time steps, so here we directly take them as input. When testing, for newly initialized tracks, the required previous states are set to be identical with current states, and current ReID features are used for historical ones.

3.1.7 Loss

Neural Motion Model requires two supervision signals for training, namely object visibility and state (*i.e.,* bounding box coordinates). The training loss should also be considered from this two aspects separately.

For visibility loss, we use weighted mean squared error, given by

$$L_{vis} = \frac{1}{N} \sum_{i=1}^{N} \gamma^{1-v_i^*} \left(v_i - v_i^*\right)^2 , \tag{3.3}$$

where v denotes predicted visibility and v^* denotes ground truth visibility. γ is a hyperparameter ($\gamma \geq 1$) that controls the relative magnitude of weights for different visibility. The intuition here is that we want the loss for low visibility samples to be higher, since these are the "hard examples". However, in practice we experiment a set of values for γ and check the *vanilla* mean squared error for low-vis examples on

Algorithm 3.1: Forward algorithm of Neural Motion Model

Input : Image at current time step I_{t-1}

States for all n active tracks at previous and current time step $\mathcal{S}_{t-2}, \mathcal{S}_{t-1}$ where $\mathcal{S}_{t-1} = \{\mathbf{s}_{t-1}^{(1)}, \mathbf{s}_{t-1}^{(2)}, \cdots, \mathbf{s}_{t-1}^{(n)}\}$

Historical ReID features for all active tracks $\mathcal{F} = \{\mathcal{F}^{(1)}, \mathcal{F}^{(2)}, \cdots, \mathcal{F}^{(n)}\}$ with $\mathcal{F}^{(i)} = \{\mathbf{f}_{t_{0i}}^{(i)}, \mathbf{f}_{t_{0i}+1}^{(i)}, \cdots, \mathbf{f}_{\min\{t_{0i}+k,t-2\}}^{(i)}\}$ where t_{0i} is the initialization time of track i, k is the maximum length of history

Output: Predicted states for all active tracks at next time step $\mathcal{S}_t = \{\mathbf{s}_t^{(1)}, \mathbf{s}_t^{(2)}, \cdots, \mathbf{s}_t^{(n)}\}$

Current ReID features $\mathcal{F}_{t-1} = \{\mathbf{f}_{t-1}^{(1)}, \mathbf{f}_{t-1}^{(2)}, \cdots, \mathbf{f}_{t-1}^{(n)}\}$ (to be saved in history buffer)

$\mathcal{V}_{t-1} = \text{encode_motions}(\mathcal{S}_{t-2}, \mathcal{S}_{t-1})$; `// using Equation 3.2`

`// appearance feature extractor`

$\mathcal{A}_{t-1} = \text{appearance_extractor}(I_{t-1}, \mathcal{S}_{t-1})$;

`// identity branch`

$\mathcal{I}_{reid} = \text{crop_and_resize}(I_{t-1}, \mathcal{S}_{t-1})$;

$\mathcal{F}_{t-1} = \text{ReID_network}(\mathcal{I}_{reid})$;

$\mathcal{F}_{agg_his} = \emptyset$;

foreach $\mathcal{F}^{(i)} \in \mathcal{F}$ **do**

  $\mathbf{f}_{agg_his}^{(i)} = \frac{1}{\min\{k,t-2-t_{0i}\}} \sum_{j=t_{0i}}^{\min\{t_{0i}+k,t-2\}} \mathbf{f}_j^{(i)}$; `// average for each dim`

  $\mathcal{F}_{agg_his} = \mathcal{F}_{agg_his} + \{\mathbf{f}_{agg_his}^{(i)}\}$;

end

$\mathcal{C}_{t-1} = \text{identity_network}(\mathcal{F}_{agg_his}, \mathcal{F}_{t-1})$; `// identity features`

`// visibility estimator`

$\mathcal{R}_{t-1} = \text{visibility_estimator}(\mathcal{A}_{t-1}, \mathcal{C}_{t-1})$; `// visibility`

`// motion residual regressor`

$\Delta\mathcal{V}_{t-1} = \text{motion_residual_regressor}(\mathcal{V}_{t-1}, \mathcal{A}_{t-1})$;

$\mathcal{W}_{t-1} = \sigma(\text{fc}(\mathcal{R}_{t-1}))$; `// gating weight`

$\mathcal{V}_t = \emptyset$;

foreach $\mathbf{v}_{t-1}^{(i)}, \Delta\mathbf{v}_{t-1}^{(i)}, \mathbf{w}_{t-1}^{(i)} \in \text{zip}(\mathcal{V}_{t-1}, \Delta\mathcal{V}_{t-1}, \mathcal{W}_{t-1})$ **do**

  $\mathbf{v}_t^{(i)} = \mathbf{v}_{t-1}^{(i)} + \mathbf{w}_{t-1}^{(i)} \odot \Delta\mathbf{v}_{t-1}^{(i)}$; `// ⊙ denotes elem-wise multiplication`

  $\mathcal{V}_t = \mathcal{V}_t + \{\mathbf{v}_t^{(i)}\}$;

end

$\mathcal{S}_t = \text{decode_motions}(\mathcal{S}_{t-1}, \mathcal{V}_t)$; `// using Equation 3.2`

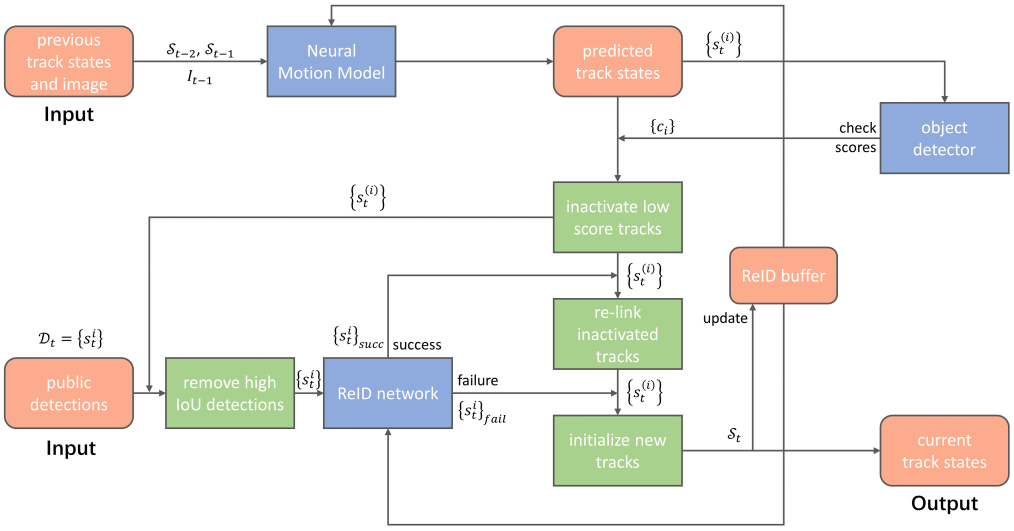


Figure 3.5: The overall framework of our tracking method.

validation set, and it proves that $\gamma = 1$ performs best, *i.e.,* the loss without re-weighting. Therefore, we use the vanilla version for all experiments in Chapter 4.

For bounding box state, we use Huber loss [19], also known as smooth-L1 loss. As the name suggests, it is essentially a L1 loss but "smoothed" near the non-differentiable point, so it becomes fully differentiable on its domain. By using L1 loss we can stabilize the training process since it is less sensitive to outlier samples. It has been proved effective in the object detection field [13]. The bounding box loss in our setting is defined as

$$L_{loc} = \frac{1}{4N} \sum_{i=1}^{N} \sum_{s \in \{x,y,w,h\}} \text{Huber}(s - s^*), \tag{3.4}$$

where

$$\text{Huber}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}. \tag{3.5}$$

The network can be trained end-to-end by combining the two losses:

$$L = L_{vis} + \lambda L_{loc}, \tag{3.6}$$

where λ is a hyperparameter controlling the ratio of the two losses. However, we find that training in such way will result in unconvergence of visibility estimation. To deal with this problem, we choose to pretrain the visibility estimator and identity branch firstly, then freeze their parameters and train the main branch. More details can be found in Section 4.3.

3.2 Tracking Algorithm

In Section 3.1 we comprehensively introduced Neural Motion Model, which is the main part of our proposed multiple object tracker. In this section we mainly focus on the parts except Neural Motion

Model but are still important components of the tracker. Besides, the overall tracking framework is illustrated in Figure 3.5, and a formal description of our tracking algorithm is given by Algorithm 3.2.

3.2.1 Image Alignment

Since the camera itself is not always static in all video sequences, we perform image alignment between adjacent frames as a rigid motion estimator (see Section 2.4.1). Specifically we adopt Enhanced Correlation Coefficient (ECC) maximization [10] (implemented in OpenCV [6]) to estimate a warping matrix between two frames and transform the states of the bounding boxes by the matrix before applying Neural Motion Model.

3.2.2 Association

As mentioned in Section 2.2.2, for association a frequently adopted approach is Hungarian algorithm, which optimally assigns detections in the new frame to the predicted states of active tracks such that the assigned detections become new states of the tracks. However, here we follow the rare method of directly taking the predicted states *as* the new states, ignoring the new detections (they are useful in track management, see the following section).

There are two reasons for doing so. Firstly, yet again we want to keep our algorithm consistent with the algorithm of Tracktor++. Secondly, even if we do not consider the first reason, it would be impossible for the model to tackle occlusions if prediction-detection association is used. This is because object detectors could almost *never* give a detection that is occluded by another interested object of the same class, since most of the object detectors use non-maximum suppression in postprocessing to eliminate highly overlapping bounding boxes. Therefore, there will be no detection to be assigned to the predicted states of objects that are being occluded.

3.2.3 Track Management

Our Neural Motion Model is only responsible for extending active (*i.e.,* existing) tracks, but the track management methods are also necessary in a multiple object tracking algorithm. These include managing the births and deaths of tracks, and postprocessing operations such as re-linking two tracks that belong to the same object.

The method of initializing new tracks of our algorithm is the same as that of Tracktor++. Firstly, the detections in the new frame are filtered by a threshold. Only detections with confidence greater than the threshold are kept. Then, for each remaining detection the IoU between it and all active tracks are examined and only if the maximum IoU is less than a threshold, this detection can be used to initialize a new track.

For terminating existing tracks, we only consider situations of a track having been occluded by a non-interested object for a long time, or moving out of frame. We do not terminate a track when it is occluded by another interested object as Tracktor++, since we want the motion model to continue tracking in the situations of having such occlusions. This can be simply implemented by checking the detection confidences right after updating tracks. Here we take the bounding boxes of updated tracks as *proposals* and fed them into the Faster R-CNN detector to obtain the confidences.

After being terminated, the track is not immediately deleted but instead recorded in a buffer that will be kept for a few time steps. Specifically, its most recent k ReID features (before elimination) and last motion are saved. The virtual bounding box state of the track is updated using its recorded last motion in the following time steps. The ReID features are used to be compared with the ReID features of newly initialized tracks, and for those pairs whose similarity and IoU are greater than some thresholds, the new tracks are re-linked to their counterparts in the buffer. This strategy is identical with that of Tracktor++.

Algorithm 3.2: Tracking algorithm

Input : Video sequence $\mathcal{I} = \{I_1, I_2, \cdots, I_T\}$

Public detections for each frame $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_T\}$

Output: Track of objects $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_N\}$ with $\mathcal{T}_i = \{\mathbf{s}^{(i)}_{t_{0i}}, \mathbf{s}^{(i)}_{t_{0i}+1}, \cdots, \mathbf{s}^{(i)}_{t_{1i}}\}$, where t_{0i} and t_{1i} are the initialization time and termination time of track i respectively

$\mathcal{T}, \mathcal{T}_{act}, \mathcal{T}_{inact} = \emptyset$;
foreach $t \in \{1, 2, \cdots, T\}$ **do**
  $\mathcal{S}_{t-2}, \mathcal{S}_{t-1} = \text{get_previous_current_state}(\mathcal{T}_{act})$;
  $\mathcal{S}_{t-2} = \text{ecc_align}(I_{t-1}, I_t, \mathcal{S}_{t-2}),\ \mathcal{S}_{t-1} = \text{ecc_align}(I_{t-1}, I_t, \mathcal{S}_{t-1})$;
  $\mathcal{S}_t, \mathcal{F}_{t-1} = \text{Neural_MM}(\mathcal{S}_{t-2}, \mathcal{S}_{t-1}, \text{get_historical_ReID}(\mathcal{T}_{act}))$; `// Algo. 3.1`
  $\mathcal{T}_{act} = \text{update_state}(\mathcal{T}_{act}, \mathcal{S}_t)$;
  $\mathcal{T}_{act} = \text{update_historical_and_recent_ReID}(\mathcal{T}_{act}, \mathcal{F}_{t-1})$;
  $\mathcal{T}_{inact} = \text{apply_last_motion}(\mathcal{T}_{inact})$;
  $\mathcal{C} = \text{detector.confidence}(\mathcal{S}_t)$;
  foreach $T_i, c_i \in \text{zip}(\mathcal{T}_{act}, \mathcal{C})$ **do**
    if $c_i < \sigma_{act}$ **then**
      $\mathcal{T}_{act} = \mathcal{T}_{act} - \{T_i\},\ \mathcal{T}_{inact} = \mathcal{T}_{inact} + \{T_i\}$; `// termination`
    end
  end
  $\mathcal{D}_t = \text{filter}(\mathcal{D}_t, c \geq \sigma_{act})$;
  foreach $(\mathbf{s}^i_t, c^i_t) \in \mathcal{D}_t$ **do**
    if $\max(\text{IoU}(\mathbf{s}^i_t, \text{get_state}(\mathcal{T}_{act}))) > \lambda_{new})$ **then** $\mathcal{D}_t = \mathcal{D}_t - \{(\mathbf{s}^i_t, c^i_t)\}$;
  end
  $\mathbf{M} = \text{build_similarity_matrix}($
   $\text{ReID_network}(\text{crop_and_resize}(I_t, \text{get_bboxes}(\mathcal{D}_t))), \text{get_recent_ReID}(\mathcal{T}_{inact}))$;
  foreach $(\mathbf{s}^i_t, T_j) \in \text{get_bboxes}(\mathcal{D}_t) \times \mathcal{T}_{inact}$ **do**
    if $\text{IoU}(\mathbf{s}^i_t, \text{get_state}(T_j)) < \lambda_{reid}$ **then** $\mathbf{M}_{ij} = -\infty$;
  end
  $\mathcal{A} = \text{Hungarian_assignment}(\mathbf{M})$; `// re-linking results in k-v pairs`
  foreach $\mathbf{s}^i_t \in \text{get_bboxes}(\mathcal{D}_t)$ **do**
    if $\text{has_key}(\mathcal{A}, \mathbf{s}^i_t)$ **then** `// re-linking`
      $\mathcal{T}_{inact} = \mathcal{T}_{inact} - \{\text{get_value}(\mathcal{A}, \mathbf{s}^i_t)\}$;
      $\mathcal{T}_{act} = \mathcal{T}_{act} + \text{update_state}(\text{get_value}(\mathcal{A}, \mathbf{s}^i_t), \mathbf{s}^i_t))$;
    else $\mathcal{T}_{act} = \mathcal{T}_{act} + \{\text{new_track}(\mathbf{s}^i_t)\}$; `// initialization`
  end
  $\mathcal{T}_{act} = \text{save_previous_state}(\mathcal{T}_{act})$;
  $\mathcal{T}_{inact} = \text{remove_inactive_tracks_too_old}(\mathcal{T}_{inact})$;
  $\mathcal{T} = \text{update_results}(\mathcal{T}, \mathcal{T}_{act})$;
end

Chapter 4

Experiments and Results

In this chapter, we begin by briefly introducing the dataset on which we train and evaluate our proposed model. Next we go over the metrics measuring the accuracy of motion prediction and multiple tracking performance, and particularly we explain the latter one in detail since it is complicated and difficult to understand. Then we present our quantitative and qualitative experimental results accompanied with analyses in both motion prediction and MOT experiments. In the end, we perform ablation studies on the motion model.

4.1 Dataset

We use MOT17 tracking dataset provided by MOTChallenge benchmark [35] for training and testing the performance of our proposed tracker. There are 7 video sequences in the training set, with approximately 800 frames per sequence in average. Most of the videos have relatively high frame rate of 30 fps. The only class of object to be tracked are pedestrians, whose detections in each frame are generated by detectors of Faster R-CNN [38], DPM [12] and SDP [50]. Despite limited number of sequences, a variety of scenarios has been covered, *e.g.,* daytime/nighttime, static/moving camera, high/low density of objects, low/medium/high camera angles. Therefore, we believe that MOT17 is a proper dataset for comprehensive evaluation of our tracker's performance and robustness.

Note that MOTChallenge does not provide official splits of validation sets, so we take the last 20% of each sequence and utilize them as our validation data. In the following of this chapter, such validation data is referred to as *MOT17 Validation Dataset*.

4.2 Evaluation Metrics

In this thesis, we mainly use two types of metrics, namely motion prediction losses and multiple object tracking metrics. The former type is used for testing the performance of pure motion models, and the latter one is used for testing the overall performance of tracking algorithms.

4.2.1 Motion Prediction Loss

We directly use the previously mentioned bounding box loss (Equation 3.4) on MOT17 Validation Dataset for motion model evaluation.

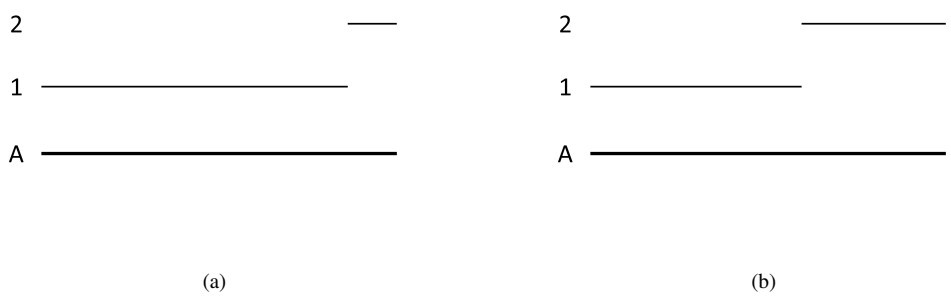


Figure 4.1: An example illustrating the heuristic of CLEAR MOT, adopted from [3] and rearranged. Identity A represents a ground truth trajectory, and identities 1 and 2 are the corresponding predicted trajectories. The idea of CLEAR MOT is that if predicted trajectories are assigned to ground truth trajectories globally and mismatches are accumulated for each frame, then the penalization for (a) and (b) should be different, but essentially the error made by the tracker in this two scenarios are the same (*i.e.,* one id-switch) and it should not be considered relevant to the time.

4.2.2 Multiple Object Tracking Metrics

For evaluation of MOT performance, we use MOTA, MOTP, IDSw and IDF1. In fact, there are two different measurement systems for MOT that follow different heuristics into which the four MOT metrics can be categorized: the first, second and third one are part of the CLEAR MOT metrics [3], and the last one is part of the ID metrics [39]. To explain their difference in one sentence, the CLEAR MOT metrics are local, event-based, distance/similarity-aware measurements, whilst the ID metrics are global and unaware of concrete distance/similarity values.

CLEAR MOT metrics The CLEAR MOT measurement algorithm applies optimal matching between predicted track locations and ground truth object locations at each frame and count the frequencies of three kinds of events: false positive, false negative and identity switch. It *does not* penalize identity mismatches between predicted and ground truth trajectories for each frame, but rather penalize the *event* that a switch happens. The intuition can be illustrated by comparing Figure 4.1(a) and 4.1(b). The matching procedure for each frame is as follows:

1. For each of the ground truth and prediction pairs matched in the last frame, check whether the distance between these two identities in this frame is smaller than a threshold t_d. If so, then this pair is preserved in this frame.

2. For all the remaining (unmatched) ground truths and predictions in this frame, build a distance matrix, and mask out those pairs with distance greater than the threshold t_d. Then optimal matching is performed using Hungarian algorithm. This step is similar to the approach used in our Algorithm 3.2 (line 19-23).

3. Compare all the matches in this frame with matches in the last frame. For each ground truth object, if the two assigned prediction identities are not the same, record one mismatch event.

4. Record one false positive for each unmatched prediction, and one false negative for each unmatched ground truth.

After performing the above matching procedure for each frame, the CLEAR MOT metrics can be computed as follows:

1. Multiple object tracking accuracy (MOTA), defined as

$$\mathrm{MOTA} = 1 - \frac{\sum_t (\mathrm{FP}_t + \mathrm{FN}_t + \mathrm{IDSw}_t)}{\sum_t g_t}, \tag{4.1}$$

   where FP_t is the number of false positives, FN_t is the number of false negatives, IDSw_t is the number of id-switches and g_t is the number of ground truth objects, all at frame t.

2. Multiple object tracking precision (MOTP), defined as

$$\mathrm{MOTP} = \frac{\sum_t \sum_i d_t^i}{\sum_t c_t}, \tag{4.2}$$

   where d_t^i denotes the distance between predicted object i and its corresponding ground truth object, c_t is the number of matched pairs, in frame t. Here we follow the definition in MOTChallenge [28], and this is slightly different from the original one in the CLEAR MOT paper [3].

3. Total identity switches (IDSw), defined as

$$\mathrm{IDSw} = \sum_t \mathrm{IDSw}_t. \tag{4.3}$$

ID metrics Compared with CLEAR MOT measurements, the algorithm of ID metrics applies global matching between predicted and ground truth *trajectories* instead of bounding box locations in each frame. See Figure 4.2 for the intuition mentioned in the original paper [39]. Specifically, a bipartite graph is built with two set of nodes. The first set contains nodes of ground truth trajectories τ and "false positive" nodes of predicted trajectories f_γ^+. The second set contains nodes of predicted trajectories γ and "false negative" nodes of ground truth trajectories f_τ^-.

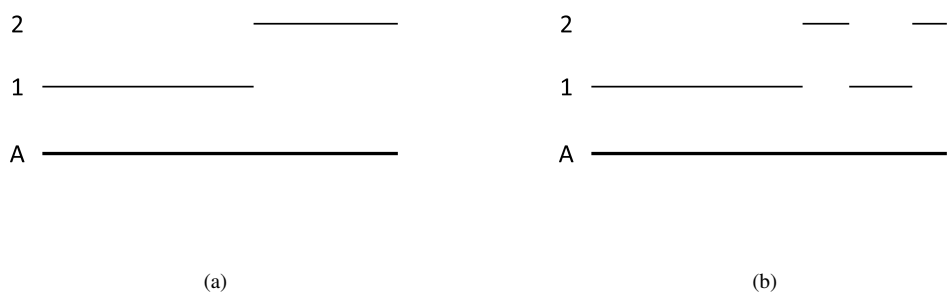
We then define the cost or dissimilarity of each edge in the bipartite graph. Firstly we explain the dissimilarity computation between two "real" trajectories τ and γ. At frame t, if the distance (or IoU) between these two trajectories is lower than a certain threshold Δ, we use a *binary* variable to record a miss:

$$m(\tau, \gamma, t) = 1. \tag{4.4}$$

Then, the cost between two "real" trajectories are computed by

$$c(\tau, \gamma) = \sum_{t \in \mathcal{T}_\tau} m(\tau, \gamma, t) + \sum_{t \in \mathcal{T}_\gamma} m(\tau, \gamma, t), \tag{4.5}$$

which is the sum of the numbers of true negative locations and true positive locations. For an edge between a "real" trajectory and a "false" trajectory, the cost is the length of the "real" trajectory since the "false" trajectory could be regarded as null. The cost between two "false" trajectories is simply zero.



(a) (b)

Figure 4.2: An example illustrating the heuristic of ID metrics, adopted from [39] and rearranged. The tracker in (a) has higher CLEAR MOT scores than tracker in (b) since it has less id-switches. However, tracker in (b) has higher IDF1 score because trajectory 1 is assigned to be the "correct" one by the algorithm of ID metrics, and the correspondence remains longer in (b) than in (a). Please compare this figure with Figure 4.1.

Note that this is very different from CLEAR MOT, where the costs involve real values of distances, but here for ID metrics the costs are just counts.

After defining the costs, Hungarian algorithm is preformed on the bipartite graph to find optimal matching. Then we only consider matches between two "real" trajectories, and compute ID statistics as follows:

$$\text{IDFN} = \sum_{\tau \in AT} \sum_{t \in \mathcal{T}_\tau} m(\tau, \gamma_m(\tau), t), \tag{4.6}$$

$$\text{IDFP} = \sum_{\gamma \in AC} \sum_{t \in \mathcal{T}_\gamma} m(\tau_m(\gamma), \gamma, t), \tag{4.7}$$

$$\text{IDTP} = \sum_{\tau \in AT} \sum_{t \in \mathcal{T}_\tau} (1 - m(\tau, \gamma_m(\tau), t)) = \sum_{\gamma \in AC} \sum_{t \in \mathcal{T}_\gamma} (1 - m(\tau_m(\gamma), \gamma, t)), \tag{4.8}$$

where AT, AC denote sets of matched identities of ground truth and predicted trajectories respectively.

The ID metrics are finally given by

$$\text{IDP} = \frac{\text{IDTP}}{\text{IDTP} + \text{IDFP}}, \tag{4.9}$$

$$\text{IDR} = \frac{\text{IDTP}}{\text{IDTP} + \text{IDFN}}, \tag{4.10}$$

$$\text{IDF1} = \frac{2\text{IDTP}}{2\text{IDTP} + \text{IDFP} + \text{IDFN}}. \tag{4.11}$$

In this thesis we concentrate on the IDF1 metric, since it gives a more comprehensive scoring of tracker's performance.

4.3 Training

As mentioned in Section 3.1.7, we split the training process into two separate parts: pretrain the identity branch and visibility estimator using visibility labels, and then train the main branch using bounding box labels. We initialize the backbone of the appearance feature extractor by the parameters of Track-

Motion Model	Loss (all samples)	Loss (low vis samples)	Loss (high vis samples)
Neural Motion Model	1.30	0.95	1.44
Tracktor++	3.48	10.37	1.11
Constant velocity	0.59	0.37	0.71

Table 4.1: Motion prediction evaluation results of Neural Motion Model compared with Tracktor++ and constant velocity motion model on MOT17 Validation Dataset. "low vis" refers to visibility values less than 0.3, and "high vis" refers to visibility values greater than 0.7. Lower loss is preferred. Neural Motion Model outperforms Tracktor++ on average and for low visibility samples, while Tracktor++ gives better results for high visibility samples. Although constant velocity motion model seems to have great advantages over the other two in this experiment, it can not handle the dynamics in long-term tracking.

tor++. During training we do not update the backbone parameters, for two reasons: firstly, we find that finetuning the backbone leads to slow convergence; secondly, the accuracy of visibility prediction would decrease in stage-two training since the appearance feature extractor is also involved, which affects the model's performance in MOT evaluation.

For visibility pretraining, we optimize the network using Adam [22] optimizer with learning rate 1×10^{-4}, weight decay 5×10^{-5} and batch size 32. We randomly add zero mean Gaussian noise to bounding box coordinates (x_1, y_1, x_2, y_2) as data augmentation. The standard deviation of Gaussian noise added to x_1 and x_2 is $0.05 \times (x_2 - x_1)$, and that added to y_1 and y_2 is $0.05 \times (y_2 - y_1)$. We perform early stopping on MOT17 Validation Dataset.

For main branch training, we split the ground truth trajectories into small clips of tracklets with length $2k_m + 1$. We not only perform bounding box jittering as in stage-one but also randomly select a frame gap k ($k \in \{1, \cdots, k_m\}$) for each sample when computing motions. For example, if $k = 3$ then frame 1 (in a clip) is the previous frame, frame 4 is the current frame and frame 7 is the next frame. We set $k_m = 2$ in our training. This serves as data augmentation that prevents overfitting to frame rates. We use Adam optimizer with learning rate 1×10^{-4}, weight decay 1×10^{-4} and batch size 32. Early stopping is performed on validation set.

4.4 Motion Prediction Evaluation

The bounding box state prediction losses (Equation 3.4) for Tracktor++, Neural Motion Model and constant velocity motion model on MOT17 Validation Dataset are shown as Table 4.1. We also report the prediction losses for low visibility examples to evaluate the performances of these models in occlusion scenarios, and losses for high visibility examples to evaluate their localization abilities.

It could be clearly seen in Table 4.1 that our proposed Neural Motion Model gives much more accurate motion prediction results than Tracktor++ on average. Particularly for low visibility (or occluded) samples, Tracktor++ performs very poorly, yielding a loss that is 991% higher than the loss of Neural Motion Model for low visibility samples, and 198% higher than the loss of itself for all samples. This

is because Tracktor++ is totally unaware of identities, since the tracker is just an object detector trained for classification and localization. Therefore, it is almost certain that when occlusion of two interested objects happens, the predicted bounding box by Tracktor++ would follow the object in the front, and thus an id-switch would happen and the occluded object would lose its track as well. This can be confirmed by visualization results, as shown in Figure 4.3. On the other hand, though, Tracktor++ gives better predictions for high visibility samples than Neural Motion Model. Thanks to its strong localization ability, Tracktor++ (Faster R-CNN, in essence) is able to produce very accurate bounding boxes for clearly visible objects.

The loss of Neural Motion Model for low visibility samples is quite interesting: it is even lower than the overall loss. In other words, it performs worse for high visibility samples than for low visibility ones (Table 4.1). This implies that the Neural Motion Model might still have rooms for improvements on localization ability. Again this can be confirmed by MOT experiments in Section 4.5.2. As for its performance for low visibility samples, recall that Neural Motion Model essentially relies on motion at the previous time step in such scenarios (Section 3.1.1). Therefore, having a low loss for low visibility samples is a evidence indicating that our assumption is reasonable.

It is noteworthy that the prediction losses of constant velocity model, no matter for high, low visibility samples or on average, are much lower than the other two models. This is because in this experiment we are using *ground truth* motions as inputs and according to our above mentioned assumption, there should be hardly any difference between velocities at two consecutive time steps because of relatively high video framerate. The advantage in this experiment, however, does not necessarily mean that constant velocity motion model can perform well in the *tracking* experiments. One can easily imagine that a tracker that only relies on constant velocity motion model is not able to deal with any long-term dynamics (*i.e.,* change of speeds or directions, deformations of the object) although in short term these seems not to be obvious.

Also note that the loss of constant velocity motion model for low visibility samples is lower than average as well, same as Neural Motion Model. This provides us another explanation: the annotations for occluded object are probably determined by linear interpolations from accurate annotations of visible objects.

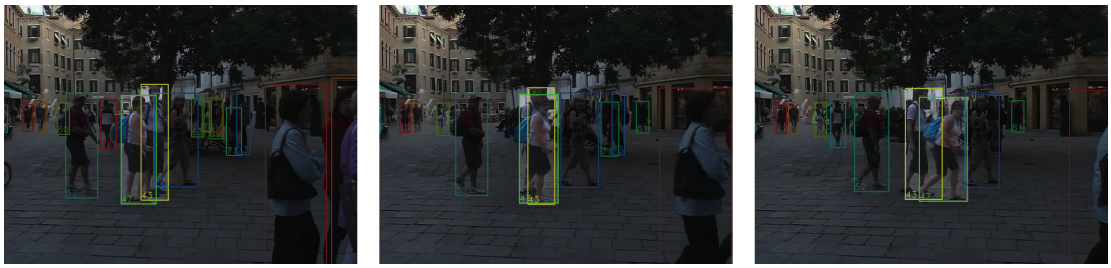
4.5 Multiple Object Tracking Evaluation

4.5.1 Qualitative Results

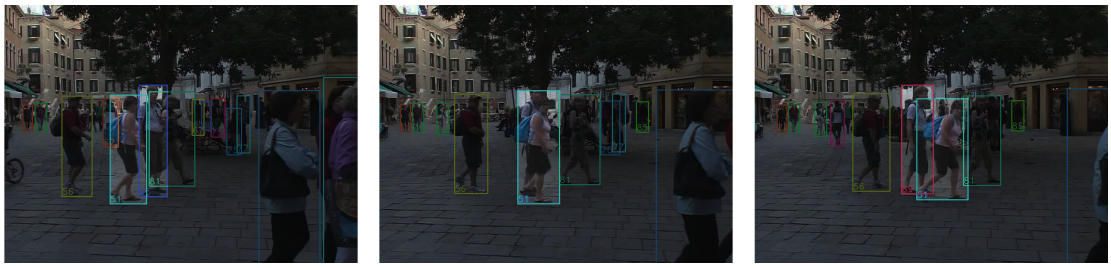
Figure 4.3 shows two examples where Neural Motion Model successfully tackles occlusions between two pedestrians by preventing an id-switch. In Figure 4.3(a) and 4.3(c), Neural Motion Model preserves identities of two objects after the occlusion. Note that this is not done by temporarily terminating the occluded track and then re-linking a newly initialized track to the former one by the ReID network. The track generated by Neural Motion Model keeps active in the whole process, and during occlusion the bounding boxes are able to show approximate states of the occluded object, as can be observed in the second column of Figure 4.3.

(a) Neural Motion Model

(b) Tracktor++



(c) Neural Motion Model



(d) Tracktor++

Figure 4.3: Two examples showing that Neural Motion Model is able to preserve identities when occlusion between two interested objects happens. In each sub-figure three frames selected before, during and after an occlusion are shown. The two objects of interest involved in the occlusion are highlighted for clarity. When generating this visualization result, we removed non-maximum suppression for terminating overlapping tracks in the track management algorithm of Tracktor++, otherwise the potential id-switches can not be observed.

Method	IDF1(%) ↑	MOTA(%) ↑	MOTP(%) ↑	IDSw ↓
Tracktor++	72.9	60.3	89.8	39
Ours (Neural Motion Model)	72.1	59.4	83.6	38

Table 4.2: Multiple object tracking evaluation results of our method (Neural Motion Model) compared with Tracktor++ on MOT17 Validation Dataset. Direction of an arrow indicates either a high or low value is preferred for the corresponding metric. Our method have less id-switches but failed to outperform Tracktor++ on the other three metrics.

In contrast, Tracktor++ is totally unable to deal with occlusions. In both cases of Figure 4.3(b) and Figure 4.3(d), the track of the occluded object drifts to the object in the front during the occlusion, and keeps following that one, not being aware of the occurred id-switch. The object that loses its track is then initialized with a new identity. As mentioned in Section 4.4, this is because Tracktor++ is essentially an object detector. Object detectors considers class of objects rather than identities of objects. For a class of objects with different identities, their appearance features extracted by object detectors should be very similar so that they can be correctly classified by the downstream network. Therefore, although having provided a relatively concise and effective algorithm for multiple object tracking, Tracktor++ has defects and is certainly not the destination of multiple object trackers.

4.5.2 Quantitative Results

The evaluation results in MOT metrics are shown as Table 4.2. It can be observed that our method with Neural Motion Model failed to outperform Tracktor++ on IDF1, MOTA and MOTP, but performs better on IDSw. However, we have to admit that the advantage of our method on IDSw over Tracktor++ is quite limited, especially taking into account that Neural Motion Model is designed on purpose to prevent identity switches.

ID-switches To explain the situation here, we have to focus on Tracktor++ rather than Neural Motion Model. In other words, the main reason is not related to our method (although there are failures in occlusion handling, to be discussed in Section 5.3), but rather it is because Tracktor++ performs so well on IDSw that it seems to be contradictory to the fact that Tracktor++ is unable to prevent id-switches, as previously shown in Table 4.1 and Figure 4.3. Essentially, it is the track management method of Tracktor++ that prevents most of the potential id-switches from happening.

Tracktor++ terminates a track in two kinds of situations. The first one is as soon as the track is occluded by a non-object or is out of frame. This strategy is the same as ours, as discussed in Section 3.2.3. The second one, however, is to terminate a track when it overlaps another object of interest. Specifically, non-maximum suppression is performed after the track states being updated and if the IoU between two tracks is greater than a threshold, the track with lower detection confidence would be terminated. By applying this method, there will be hardly any id-switches since large overlaps will not happen and thus bounding box states will not be regressed to the same object. Even if an id-switch happens and two tracks now correspond to the same object, in the next time step one of the tracks will be deactivated by non-maximum suppression and there will be no false positives in the future.

In order to validate our analysis, we perform an experiment by disabling non-maximum suppression for preventing object-object overlaps in Tracktor++, and the results are shown as Table A.1. IDSw is the most affected metric among the four, which rises by 15.4% compared with the original method. Also, Tracktor++ without non-maximum suppression performs worse than our method (except on MOTP, which is determined by localization precision). To sum up, the track management method of Tracktor++ is delicately designed and it covers up the shortcoming of Tracktor++ that it is unable to capture identity information.

MOTA, MOTP and IDF1 Our method obtains lower scores on these three metrics than Tracktor++. This is due to the following factors: localization quality of Neural Motion Model, calculation of metrics, and properties of data. Here we focus on these issues themselves and for potential solutions or direction of improvements, please see Chapter 5.

Firstly, as discussed in Section 4.4, Neural Motion Model can not be competitive with Tracktor++ in terms of localization ability, since its prediction accuracy is lower for high visibility examples. This has been verified again in the MOT evaluation experiment. It can be observed from Table 4.2 that there is a huge gap between the MOTP score of our method and Tracktor++. Recall the definition introduced in Section 4.2.2, MOTP can be regarded as the average localization error over matched pairs of predicted bounding boxes and ground truth bounding boxes, which is unrelated to false positives, false negatives and id-switches. Therefore, MOTP is a "pure" indicator for localization quality in MOT. With relatively inadequate localization ability, Neural Motion Model sometimes gives imprecise predictions even in situations without occlusions (see the example in Section 5.1). This further leads to false positives, false negatives, and eventually decrease in MOTA and IDF1.

Secondly, the results for occluded and non-occluded track clips are treated equally in the calculations of MOTA and IDF1. However, in some of the real-world applications (*e.g.,* autonomous driving) we are more interested in or give more importance to estimating the position of an object when it is not directly visible. In these cases, Tracktor++ will not be favored because it totally gives up tracking occluded objects. By contrast, our method is able to infer the possible object locations according to previous cues of motions and appearances, and thus might obtain better scores if occluded track clips have greater weights in metric calculations.

Thirdly, we have to admit that the proportion of the data that our method can handle is quite limited. Multiple object tracking deals with all kinds of complex scenarios where occlusions happen not only frequently but in all kinds of forms as well. There are indeed some kinds of occlusions that our method is unable to deal with. For instance, if the occluded object appears much smaller in the image than the occluder, this will not be regarded as an object-object occlusion, but rather a non-object-object occlusion by our method, and then the occluded track will be directly terminated. This is because the bounding box area of the occluded track is so small that it is not able to cover enough large reception field which enables the object classifier to recognize the front object. Therefore, only objects whose scales are similar to that of their occluders will be continued tracking by our method, and the advantages of our method in these cases are not obvious in terms of MOT metrics.

Motion Model	motion residual regression		visibility estimation		Loss (all)	Loss (low vis)
	current appearance	previous motion	current appearance	historical identity		
Neural Motion Model	✓	✓	✓	✓	1.30	0.95
-identity branch	✓	✓	✓	-	1.36	1.07
-visibility estimator	✓	✓	-	-	1.41	1.07
-input motion features	✓	-	✓	✓	3.67	4.24

Table 4.3: Motion prediction evaluation results of ablation experiments on MOT17 Validation Dataset. Details of which component/input is enabled in the model is also specified in the table. "low vis" refers to samples whose visibility values are less than 0.3, and "all" refers to all samples. Lower loss is preferred. The motion prediction performance worsens in all the three experiments, compared with the original method.

4.6 Ablation Experiments

In this section we perform a few ablation experiments in order to explore the function and necessity of each component and/or kind of input features in Neural Motion Model.

In the first experiment we remove the identity branch that is used in visibility estimation. As discussed in Section 3.1.3, by doing so the visibility estimator would lose the ability of correctly predicting visibility for occluded objects, and this further leads to large errors in motion prediction. We conduct a supplementary experiment on visibility prediction precision of this model. The result (Table A.2) confirms the former point, showing that identity branch has positive effect on visibility prediction. The latter point is verified by the results in Table 4.3 and 4.4. Particularly, the motion prediction loss for low visibility examples rises by 12.6% compared with the original Neural Motion Model. In contrast, the overall loss only increases by 4.6%. This indicates that identity information is indeed crucial to correctly estimating motions for low visibility examples, especially in scenarios where two objects overlaps each other largely. In the example shown in Figure 4.4(b), the model without identity branch also failed in tackling the occlusion, which leads to an id-switch.

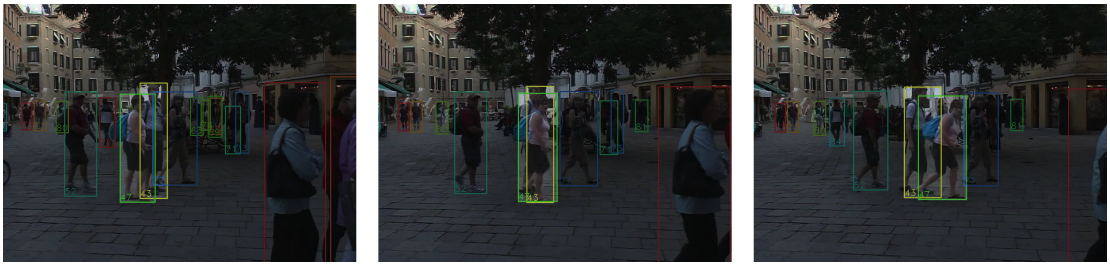
Based on the settings of the first experiment, we further remove the visibility estimator in the second experiment. This leads to increases in the overall motion prediction loss and IDSw, and decreases in IDF1, MOTA and MOTP, shown in Table 4.3 and 4.4. Among all the metrics, IDSw is the one that changes the most, which increases by 44.7% compared with the original method, which indicates that visibility estimation is a core mechanism for tackling occlusions in Neural Motion Model. Also, without visibility estimator the first training step (see Section 4.3) is no longer necessary. However, since the appearance heads (Section 3.1.2) are designed to be trained in both two steps, removing the first training step means that the appearance heads are not able to obtain benefits from pretraining on visibility. This can be observed from the decrease of localization precision (see the MOTP scores in Table 4.4). As can be seen in Figure 4.4(c), the visualization result of this experiment is similar to that in the first experiment, where the track of the occluded object drifts to the occluder and a new track is initialized for

Method	IDF1(%) ↑	MOTA(%) ↑	MOTP(%) ↑	IDSw ↓
Ours (Neural Motion Model)	72.1	59.4	83.6	38
-identity branch	71.4	58.3	83.5	39
-visibility estimator	68.4	55.9	80.4	55
-input motion features	66.3	55.1	78.1	83

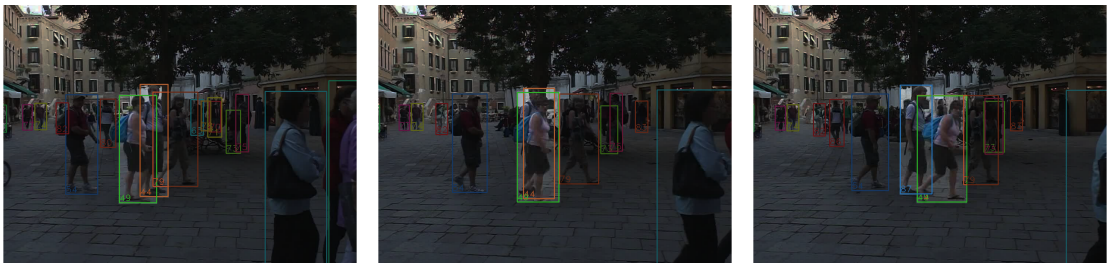
Table 4.4: Multiple object tracking evaluation results of ablation experiments on MOT17 Validation Dataset. Direction of an arrow indicates either a high or low value is preferred for the corresponding metric. All the metrics worsen in all the three experiments. Please also refer to Table 4.3 for details of model components/inputs enabled in each experiment.

the former one after occlusion.

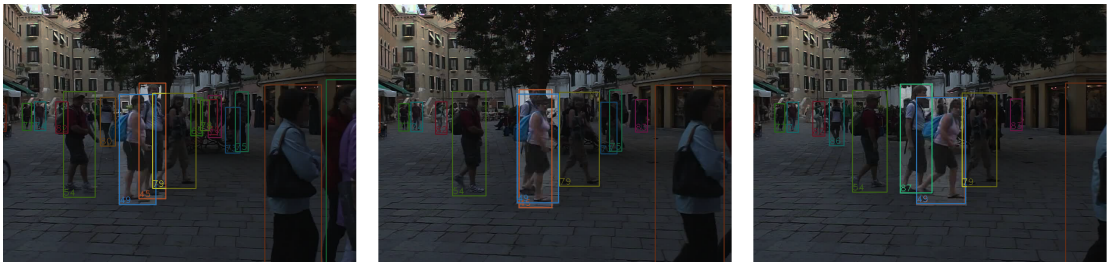
Finally, in the third experiment we remove motion features from the input of motion residual regressor. The performance of the model degrade sharply. This is because the model is designed to estimate motion residual, which is to be added to the motion of the previous time step. If the motion is not provided in the input, then the motion residual regressor would have no clue about approximately how large its prediction should be, or in other words, input motion acts as a reference value. It can be observed in Figure 4.4(d) that the localization precision is quite poor. The track of the occluded object does not even drift to the occluder, because it has already been terminated before the occlusion happens due to bad localization.



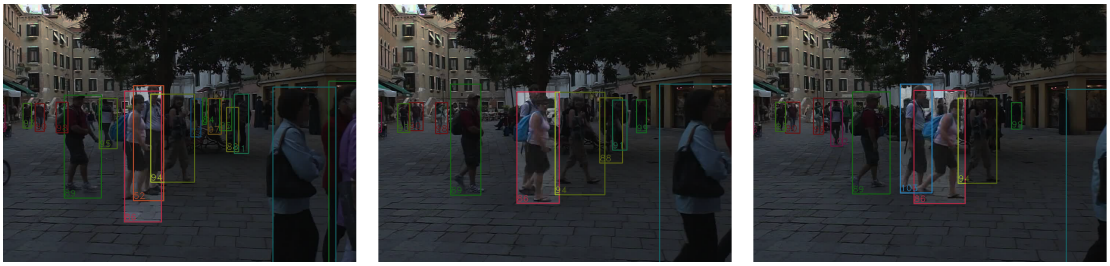
(a) Neural Motion Model



(b) Neural Motion Model without identity branch



(c) Neural Motion Model without visibility estimator



(d) Neural Motion Model without input motion features

Figure 4.4: Visualization results of ablation experiments. In this example all the methods except the original Neural Motion Model failed to preserve the identity of the occluded pedestrian. Please compare this example with Figure 4.3(c) and 4.3(d).

Chapter 5

Discussion

In this chapter we discuss the shortcomings or failure examples of our proposed method in three aspects: localization, track management and occlusion handling, as well as corresponding possible solutions or improvements.

5.1 Localization

In Chapter 4 we have already mentioned and conducted experiments on the localization ability of Neural Motion Model, and it is clear that our model has room for improvements. In order to explore possible explanations for this, we provide an visualized example showing that Neural Motion Model almost failed to bound an object in a very special case, as can be seen in Figure 5.1. The camera is at a high angle overlooking pedestrians, and two pedestrians moving vertically in the image plane with their backs to the camera are not well localized for as long as 150 frames (although they are tracked correctly). Indeed this is a very rare example in the training data where three uncommon factors (*i.e.,* high-angle camera, vertical movement, backing to camera) appear simultaneously, but the targets are not occluded at all and should be clearly recognized as pedestrians. In other words, the network lacks the ability of generalizing from commonly seen data to these rare cases, or we could say it overfits to training data to some extent.

Although the MOT dataset covers a variety of scenes, the amount of training data is still limited considering the complexity of these scenes that MOT deals with. What is more, for Neural Motion Model, only 80% of training data are actually learned, whereas Tracktor++ exploits 100%. Therefore, the most direct way to solve the overfitting problem might be providing the network with more training data. In fact, some of the multiple object trackers [51] relies on external data for pretraining, and it has been shown that this provides a large boost to the performance of the tracker, especially detection or localization performance.

However, considering the fairness in benchmarks evaluations as well as data availability, it is not always feasible to increase the amount of training data. Therefore, other aspects, *e.g.,* data augmentation, loss should be focused on in order to optimize the training process and better exploit existing data. Here we discuss a modification of training loss. Recall that Neural Motion Model gives better predictions for low visibility examples than high visibility ones (Section 4.4). Hence, the high visibility examples actually become “hard” ones, and we could assign more weights to these “hard” examples so that the

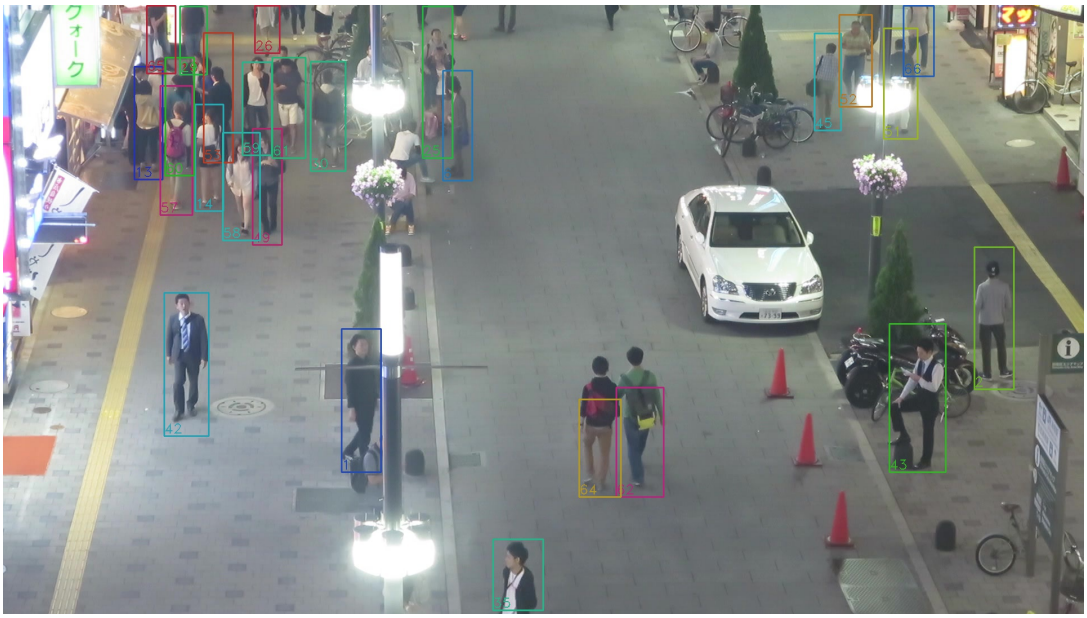


Figure 5.1: An example of localization failure. Neural Motion Model is unable to precisely localize the two person in the middle of the image for about 150 frames.

network can learn faster on them. Specifically, the bounding box loss (Equation 3.4) can be modified by imitating the form of the visibility loss (Equation 3.3).

5.2 Track Management

We have discussed in Section 4.5.2 that some of the object-object occlusions can not be handled by our model where the scales of two objects are very different. The reason is that our track management method mainly follows the framework of Tracktor++ and is not specially designed for such cases. For the moment, we have to admit that the optimal track management method of Neural Motion Model can not be as simple as that of Tracktor++, and it might consists of various rule-based strategies to handle all kinds of situations as well as avoiding shortcomings of Neural Motion Model.

For instance, in order to handle the aforementioned problem in object-object occlusions, an intuitive idea might be adding another condition to the non-object-object occlusion termination criterion: the track to be stopped whose overlapping ratio (occluded by other objects) must be smaller than a certain threshold. However, this, as well as all other proposed rule-based strategies, need to be examined carefully in practice since they might have potential disadvantages. For example, if the motion model only performs well during short-term occlusions, it is even questionable whether explicitly handling the aforementioned problem can improve the model's performance in terms of MOT metrics, since the small object will be occluded for a long time.

5.3 Occlusion Handling

We notice that when handling object-object occlusions, Neural Motion Model can sometimes have failures. Figure 5.2 shows one of such examples. Although the resemblance between appearances of the two

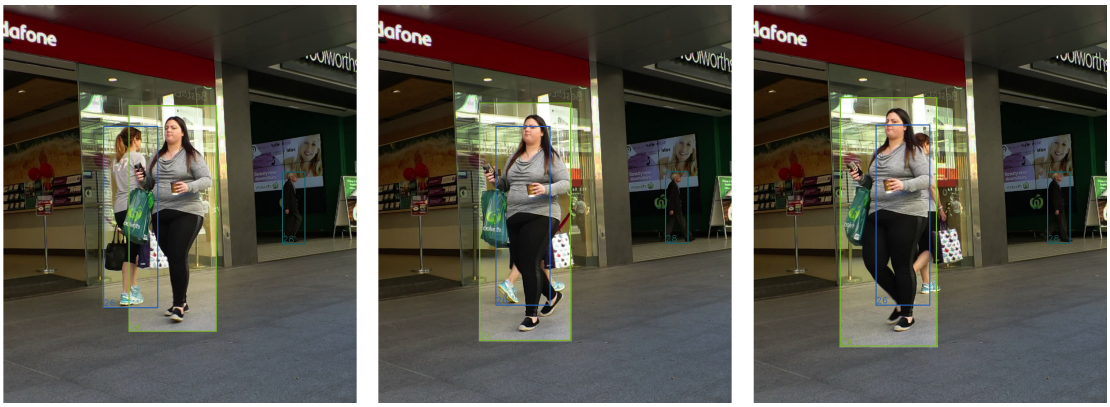


Figure 5.2: An failure example of occlusion handling by Neural Motion Model. In the beginning of the occlusion (between the first and second frame shown above), the model tries to predict a precise bounding box for the partly occluded object, and thus its velocity slows down (second frame). Since the model relies greatly on previous motion during the following period when the object is almost fully occluded, the predicted track lags behind the occluded object afterwards (third frame) and is then terminated.

pedestrians can be a good reason, the phenomenon we observe in Figure 5.2 leads to another possible explanation. Briefly speaking, during the intermediate period the model only bounds the *visible* part of occluded object, which slows down the track. To verify our hypothesis, we lower the frame rate of this video by 50% (*i.e.,* make predictions every two frames) in order to decrease the number of intermediate frames, and the result shows that the occluded object in this example can be correctly tracked.

This problem might be related to the training method. The situation here is similar to that of RNN in time series prediction: when training, ground truth at step $t-1$ is the input, and ground truth at step t is the label; when testing, however, the predicted value (output) at previous step $t-1$ is treated as the input at current step t. Hence, if the prediction at some step goes quite wrong, the following predictions can hardly be precise, because the model is very unlikely to have seen these low probability inputs during training. To solve this problem, one of the ways can be adjusting the training method and let it imitates testing method: we can use ground truths as inputs every k steps instead of using them every step, and at the remaining steps we use previous predictions. This actually acts as a kind of data augmentation that enables the model to deal with input variations more robustly.

Chapter 6

Conclusion

In this thesis, we have proposed a novel motion model for multiple object tracking that filled in the research gap of directly predicting track states using multimodal information for occluded objects. We have presented a new method in the design of the model that leverages visibility information as the criterion of appearance feature reliability. This mechanism automatically balances the dependence on two kinds of information, namely appearance and previous motion, for future motion prediction. We have shown that our model substantially outperformed the high performance tracker Tracktor++ on one-step motion prediction accuracy, especially for low visibility examples. For MOT metrics, our method has advantage over Tracktor++ in terms of id-switches, which is the most representative metric for identity preservation. To conclude, our model possesses the ability of precisely tracking objects during occlusions. Although our method failed to surpass Tracktor++ on the other MOT metrics, we showed that this was mainly related to the model's localization ability and the track management methods, which are not among the main objectives of this thesis. In these aspects, we presented a few possible directions of improvements which are able to push our method towards state-of-the-art. We hope that our method provides inspirations for designing occlusion-aware multiple object trackers.

6.1 Practical Issues and Future Work

Except the improvements mentioned in Chapter 5, here we also brief discuss issues and possible modifications of the motion model itself.

Firstly, one of the prerequisites of applying our model is to pretrain a visibility estimator for the objects to be recognized, but visibility information is seldom provided as labels in public datasets other than MOTChallenge [35]. In industrial settings such information is not difficult to obtain. For instance, one may prepare an off-the-shelf instance segmentation network such as Mask R-CNN [16] and images where most of the interested objects are totally visible. Then, the visibility training data can be prepared by copying and pasting objects such that they randomly overlaps each other. The visibility values are simply the ratio of overlapped parts. However, in academic benchmarks where external data and model are not allowed, application of our method is indeed impeded. In these cases, it might be workable to use detection confidence scores as alternative to visibility.

Secondly, our motion model works totally in a "prediction" mode and does not leverage the newly

observed information. To be specific, when computing object states at time step t, the model only relies on image I_{t-1}, but at this time the current image I_t is also available. Therefore, it might be beneficial to add a "correction" stage after "prediction", which imitates the work mode of Kalman filter.

Appendix A

Supplemental Experiments

Method	IDF1(%) ↑	MOTA(%) ↑	MOTP(%) ↑	IDSw ↓
Ours (Neural Motion Model)	72.1	59.4	83.6	38
Tracktor++	72.9	60.3	89.8	39
Tracktor++ w/o NMS	71.5	57.5	89.6	45

Table A.1: Comparing the performance of Tracktor++ with or without non-maximum suppression (NMS) on multiple object tracking metrics, evaluated on MOT17 Validation Dataset. Direction of an arrow indicates either a high or low value is preferred for the corresponding metric. After NMS being disabled, all the metrics worsen sharply. Note that our method do not apply NMS when object-object overlap happens.

Method	Visibility prediction loss
Ours (Neural Motion Model)	0.0135
-identity branch	0.0142

Table A.2: Ablation experiment on visibility prediction of disabling identity branch of Neural Motion Model. Please refer to Equation 3.3 for the definition of visibility prediction loss. The loss increases after identity branch being disabled.

Bibliography

[1] M. Babaee, Z. Li, and G. Rigoll. Occlusion handling in tracking multiple people using rnn. In *2018 IEEE International Conference on Image Processing (ICIP)*, pages 2715–2719, 2018.

[2] P. Bergmann, T. Meinhardt, and L. Leal-Taixé. Tracking without bells and whistles. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[3] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: the CLEAR MOT metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008.

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

[5] E. Bochinski, V. Eiselein, and T. Sikora. High-speed tracking-by-detection without using image information. In *International Workshop on Traffic and Street Surveillance for Safety and Security at IEEE AVSS 2017*, 2017.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] S. Bullinger, C. Bodensteiner, and M. Arens. Instance flow based online multiple object tracking. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 785–789. IEEE, 2017.

[8] J. Cai, Y. Wang, H. Zhang, H.-M. Hsu, C. Ma, and J.-N. Hwang. IA-MOT: Instance-aware multi-object tracking with motion consistency. *arXiv preprint arXiv:2006.13458*, 2020.

[9] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766. IEEE, 2015.

[10] G. D. Evangelidis and E. Z. Psarakis. Parametric image alignment using enhanced correlation coefficient maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1858–1865, 2008.

[11] C. Feichtenhofer, A. Pinz, and A. Zisserman. Detect to track and track to detect. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[12] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2009.

[13] R. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448. IEEE, 2015.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. IEEE, 2014.

[15] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, pages 1735–1742. IEEE, 2006.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988. IEEE, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[18] P. Huang, S. Han, J. Zhao, D. Liu, H. Wang, E. Yu, and A. C. Kot. Refinements in motion and appearance for online multi-object tracking. *arXiv preprint arXiv:2003.07177*, 2020.

[19] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[20] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82D:35–45, 01 1960.

[21] K. Kang, W. Ouyang, H. Li, and X. Wang. Object detection from video tubelets with convolutional neural networks. In *CVPR*, 2016.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[24] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, 2016.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[26] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics*, 2(1-2):83–97, 1955.

[27] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[28] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942*, 2015.

[29] W. Li, J. Mu, and G. Liu. Multiple object tracking with motion and appearance cues. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[30] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944. IEEE, 2017.

[31] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007. IEEE, 2017.

[32] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common objects in context. *arXiv preprint arXiv:1405.0312*, 2014.

[33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *ECCV*, 2016.

[34] Z. Lu, V. Rathod, R. Votel, and J. Huang. RetinaTrack: Online single stage joint detection and tracking. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[35] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[36] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler. Online multi-target tracking using recurrent neural networks. In *AAAI*, February 2017.

[37] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525. IEEE, 2017.

[38] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[39] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. In *European Conference on Computer Vision Workshop on Benchmarking Multi-Target Tracking*, 2016.

[40] A. Sadeghian, A. Alahi, and S. Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 300–311, 2017.

[41] F. Saleh, S. Aliakbarian, M. Salzmann, and S. Gould. ArTIST: Autoregressive trajectory inpainting and scoring for tracking. *arXiv preprint arXiv:2004.07482*, 2020.

[42] R. Sanchez-Matilla and A. Cavallaro. A predictor of moving objects for first-person vision. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2189–2193. IEEE, 2019.

[43] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823. IEEE, 2015.

[44] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[45] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders. Segmentation as selective search for object recognition. In *2011 International Conference on Computer Vision*, pages 1879–1886, 2011.

[46] P. Voigtlaender, M. Krause, A. Ošep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe. MOTS: Multi-object tracking and segmentation. In *CVPR*, 2019.

[47] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[48] J. Xu, Y. Cao, Z. Zhang, and H. Hu. Spatial-temporal relation networks for multi-object tracking. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3987–3997. IEEE, 2019.

[49] Y. Xu, A. Osep, Y. Ban, R. Horaud, L. Leal-Taixé, and X. Alameda-Pineda. How to train your deep multi-object tracker. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6787–6796, 2020.

[50] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2129–2137. IEEE, 2016.

[51] X. Zhou, V. Koltun, and P. Krähenbühl. Tracking objects as points. *arXiv preprint arXiv:2004.01177*, 2020.

[52] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.