Multiple Object Tracking by Link Prediction using Graph Convolution Networks

Master Thesis

presented by
Marius Bock
Matriculation Number 1632186

submitted to the
Data and Web Science Group
Prof. Dr.-Ing. Margret Keuper
University of Mannheim

November 2020

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Abstract

Multiple object tracking (MOT) is a fundamental problem in computer vision which finds many real-world applications [7]. Most state-of-the-art multiple object trackers follow the tracking-by-detection paradigm. The paradigm divides MOT into two subproblems, namely detecting objects within a scene and linking them together to form trajectories. Recent methods have employed clustering methods to solve the latter part of the paradigm. This thesis demonstrates the applicability of GCNs as proposed by Wang et al. [62] to cluster detections and from trajectories. The overall approach differs from previously proposed methods as it learns to cluster detections using only local information surrounding the node. This locality of the decision-making process is expected to make the method less affected by global constraints and being able to correctly classify difficult detections. This thesis extends the implementation by Wang et al. [62] by employing different methods of how localized information is presented to the GCN, applying primal feasible search heuristics as seen in [34] and utilizing preprocessed detections as seen in [4]. The evaluated best setting of the approach ranks 64th within the multiple object tracking benchmark dataset from 2017 (MOT17) while being trained using only one sequence and having no hyperparameter-tuning performed.

1.1 Problem Statement

Tracking multiple objects within a scene is a fundamental problem in computer vision and an essential part of many applications such as autonomous driving, biology and surveillance [7]. More specifically, multiple object tracking (MOT) describes the identification of trajectories of objects within a scene. By nature, a moving scene can cause objects to be only partially visible, occluded or cropped

throughout time [48]. Additionally, there can be reflections in mirrors or windows as well as objects within the scene that look very similar to each other. Such difficulties and ambiguities make MOT a difficult problem to solve. Most state-of-the-art multiple object trackers follow the tracking-by-detection paradigm which divides MOT into a two-step problem [7]. Firstly, detecting objects on a frame-by-frame basis and secondly, linking detections across time to form trajectories. The former part of the paradigm can be achieved using learning-based detectors [18, 52, 66], while the latter is usually translated to a graph partitioning problem [7]. Within said problem, the nodes within the graph represent the obtained detections, i.e., bounding boxes, proposed by the learning-based detectors. An edge between two nodes constitutes to both detections belonging to the same object and thus are part of the same trajectory. To determine the set of edges, which partition the graph into trajectories, each edge is assigned a weight, representing the link likelihood of the two involved nodes. Said weights can then be used to apply a graph optimization framework to find an optimal partitioning.

1.2 Contribution

This thesis illustrates the applicability of Graph Convolutional Networks (GCNs) as proposed by Wang et al. [62] on MOT. In their work, Wang et al. apply their proposed network to cluster images of faces. To do so their approach structures the image data as a graph with nodes representing images. Afterward, a GCN is employed as a link prediction method to predict linkage likelihoods between nodes and pseudo label propagation as seen in [67] is applied to obtain a final clustering of the graph, grouping instances of the same face. Applying this to MOT, this thesis uses the implementation of Wang et al. [62] to cluster object detections within a sequence with each resulting cluster representing a trajectory throughout the sequence. To evaluate the applicability and effectiveness as well as tracking performance compared to other state-of-the-art trackers, the MOT17 [40, 48], a publicly available MOT dataset, was used. The challenge provides users with a set of train and test sequences, corresponding pedestrian detections provided by three different detectors [18, 52, 66] as well as a set of evaluation metrics, which can be used by users to evaluate tracking performance on the training sequences. As of handing in this thesis the proposed method ranks 64th within the public leaderboard of the MOT17 challenge. The tracker's performance ranks close to other state-of-the-art trackers seen in [33, 4] while being trained using only one train sequence and having no hyperparameter-tuning performed. Contributions of this thesis include a dataset creation script, a modified version of the implementation of Wang et al. [62], a postprocessing script, which converts predicted clusters into trajectories and

an evaluation script. The dataset script provides users the ability to create five types of datasets, each encoding different information using the MOT17 dataset, a label creation process and two additional types of KNN graph calculations. Further, the implementation of Wang et al. [62] was extended by adding different variants how local information is encoded within the GCN as well as provide additional information for users to analyze a network's behavior. Additionally, it was shown that results are able to be improved using primal feasible search heuristics as seen in [34] instead of pseudo label propagation seen in [67] as well as training the GCN using a preprocessed version of the detection datasets as seen in [4].

The thesis is structured into five chapters. In Chapter 2, the theoretical background of this thesis is explained. Chapter 3 gives details on the MOTChallenge, explaining the composition of the MOT17 benchmark dataset as well as employed evaluation metrics of the challenge. Chapter 4 gives details on the implementation used during this thesis, highlighting differences to the implementation proposed by Wang et al. [62] as well as postprocessing and evaluation scripts needed to obtain and assess object trajectories from the final clustering. Afterward, the overall implementation is evaluated in Chapter 5. Experiments conducted during this thesis feature the evaluation of the base tracking performance of the GCN proposed by Wang et al. [62] using each feature dataset as input (see Chapter 5.1) as well as their consistency across runs (see Chapter 5.2). Further, modifications to the implementation of Wang et al. [62] are assessed. Said modifications include two new types of KNN graph calculations (see Chapter 5.3), different calculation of node features within the data feeder of the implementation (see Chapter 5.4) and using primal feasible search heuristics as seen in [34] as opposed to pseudo label propagation [67] to obtain a final clustering. Next, the best setting is identified (see Chapter 5.6) and evaluated using a preprocessed version of the detection dataset as proposed by Bergmann et al. [4]. Finally, the overall best-performing setting is submitted to the MOTChallenge website to report results on the testing sequences and rank the proposed method's tracking performance compared to other trackers. Lastly, Chapter 6 concludes the findings of this thesis, gives a final assessment of the applicability of GCNs to MOT and touches on aspects that could be further investigated and improved during future research.

1.3 Related Work

Related work within the field of graph-based MOT defines data associations among nodes either as maximum flow [3] or, equivalently, minimum cost problem [7]. The latter is achieved by either estimating fixed costs based on distances between nodes [31, 51, 43], using motion models [39] or learned costs [37]. Within this

work, edge weights are being predicted using GCNs, so it can be categorized within the group of learned cost approaches. For implementations to learn costs, i.e., edge weights, discriminative node features are needed [63]. Early work within the field of MOT included features being manually designed [51, 6, 43]. Recent work [57, 2, 68, 64] focuses on employing Convolutional Neural Networks (CNNs) to extract appearance-based features. Other work suggests extending the graph formulation, incorporating external vision input like joint detection [58], activity recognition [11], multi-camera sequences [38, 65], keypoint trajectories [10] or segmentation [49]. Within this work, node features are created, inter alia, using three different CNNs [72, 25, 61]. Each network's feature representation is used to extract features surrounding the detections. Additionally, the proposed network is learned using only spatial features, i.e., the coordinates of the bounding boxes as well as the frame they occur in.

Besides clustering detections, disjoint path methods have proven to be another way to obtain trajectories [43, 3, 27]. Unlike the clustering approach, disjoint path methods assume that there can be no more than one detection of an object per frame. If e.g., trackers predict multiple detections for a given detection in a frame, the disjoint path method would declare each of them belonging to different trajectories. In contrast to this, clustering approaches do not pose this constraint and group detections disregarding whether they appear in the same frame. Additionally, disjoint path methods pose the constraint that a detection needs to have a predecessor and successor in the neighboring frames as they suppose that objects cannot disappear within sequences. If, for example, an object is partially occluded or changes its appearance within the next frame, said constraint of the disjoint path methods would force trajectories to include low-confidence detections as successors and consequently to not break. Both constraints hold advantages as well as disadvantages as they make trajectories produced by the disjoint path methods less flexible compared to using clustering methods.

While working on this thesis two works were published which also investigate the applicability of GCNs to MOT [7, 63]. Braso et al. [7] propose a tracker that uses a message-passing network (MPN), performs feature learning as well as predicts trajectories. Their proposed MPN is able to learn to identify global interactions among detections within the graph by combining *"deep features into higher-order information across the graph"* [7]. Weng et al. [63] propose to use GCNs to learn feature interactions among nodes. With each step of the GCN, nodes are able to update their associated features by aggregating feature information of other nodes within the graph. They further provide a feature extractor paired with an ensemble training architecture that learns *"discriminative motion and appearance features from both 2D and 3D"* [63]. The approaches proposed in [7] and [63] differ from the one proposed within this thesis as updates of feature vectors are per-

formed by considering the graph as a whole, i.e., posing global constraints on the graph. Braso et al. [7] claim that said design is able to learn global dependencies among nodes within the graph. As of now, the work proposed in this thesis is the only application of GCNs on MOT which estimates the link likelihood between nodes by only their local neighborhood. It is expected that said non-restriction helps the network predicting especially difficult object tracking tasks correctly as it is not penalized through global constraints and is able to make predictions only dependent on local information. Within Chapter 5.5 it is shown how using primal feasible search heuristics as seen in [34] impact results instead of using pseudo label propagation [67]. Keuper et al. [34] demonstrate how primal feasible search heurisitcs can be applied to MOT by using it to find a feasible solution for their defined co-correlation clustering problem illustrated in [33]. Lastly, the implementation within this thesis makes use of the preprocessing of the MOTChallenge data suggested in [4], in which Bergmann et al. demonstrate that their preprocessing improves overall tracking quality.

Chapter 2

Theoretical Framework

A common theroretical background is needed to properly understand the implementation (see Chapter 4) as well as the described experiments and their evaluation metrics (see Chapter 5). In order to avoid readers having different understandings, the key terms *link prediction* (see Section 2.1) and *Graph Convolution Network* (see Section 2.2) are going to be defined in more detail.

2.1 Link Prediction

Link prediction describes the task of predicting the likelihood that two nodes are connected within a graph, i.e., are connected via an edge [44]. Sample use cases of link prediction include social network analyzes [44] and recommender systems [36]. The former tries to predict interactions, collaboration or influence between entities within a social context [44], while the latter tries to produce a set of products, i.e., edges with the highest link likelihood, to recommend to a user assuming that the products are most likely to be consumed by the customer [36].

To compute the link likelihood between two nodes link prediction algorithms do so by either analyzing the complete graph or only the neighbors of the two nodes [62]. Sample algorithms for the former type of algorithms include PageRank [8] and SimRank [30], while for the latter preferential attachment [1] and resource allocation [73] are sample methods. The link prediction problem proposed by Wang et al. [62] can be categorized within the latter group of algorithms. It draws inspiration from implementations proposed by Zhang and Chen [70, 69] and proposes to predict the link likelihood of a node and its one-hop neighbors by analyzing the local structure surrounding the node pair. They define the local structure of a pivot instance, i.e., the node of interest, using Instance Pivot Subgraphs (IPS). Unlike e.g., PageRank [8], which works using basic matrix operations, their approach ap-

plies a Graph Convolutional Networks (GCN) using a graph defined through the IPS of each detection as input to the network. For each node the GCN predicts the link likelihood between the node and its one-hop neighbors.

As demonstrated by Zhang and Chen in [69], one can reduce clustering to a link prediction problem. Within this setting instances that are to be clustered are represented as nodes. An edge between two nodes, i.e., two instances, represents the likelihood that these two instances are describing the same instance. Wang et al. [62] apply this reduction in order to cluster images of faces with all images within a cluster describing the same instance. This thesis aims at evaluating whether this approach of clustering images can be applied to MOT. Within said scenario nodes would represent detections of objects, e.g., persons, throughout a sequence while edges with their associated predicted link likelihood would represent the likelihood that a node pair, i.e., two detections, describe the same instance within a sequence. Consequently, clusters of nodes would amount to all detections of an object, which could thus be translated to said object's track within a sequence.

2.2 Graph Convolution Networks

Graph convolution networks (GCNs) are neural networks that use data organized as graphs as input [60]. According to Velickovic et al. [60] GCNs *"consist of an iterative process, which propagates the node states [of a graph] until equilibrium; followed by a neural network, which produces an output for each node based on its state."* GCNs can be categorized by the type of convolution which is applied on the graph and how train and test data are structured [62]. If the applied convolution within a GCN is based on Graph Fourier Transform, the network is said to be a spectral GCN [9, 23, 60]. By contrast, if a GCN applies user-defined convolution directly on its graph nodes and their neighbors, it is said to be a spatial GCN [35, 13]. A GCN is said to be applied in a transductive setting if train and test data are merged into one single graph GCNs [23, 60]. Differently, it is said to be applied in an inductive setting if train and test data are kept as separate graphs [35, 13]. The GCN proposed by Wang et al. [62] is to be classified as a spatial GCN which is applied in an inductive setting.

Early work in the field of GCNs included implementations using recursive neural networks working on directed acyclic graphs [19, 56]. Gori et al. [22] and Scarselli et al. [54] improved upon said first implementations by proposing a generalization of a recurrent neural network, which was able to use a much broader class of graphs (i.e., cyclic, acyclic, directed and undirected graphs) as input [60]. Said approach was later extended by Li et al. [42] replacing contraction maps as propagation functions with modern recurrent neural network techniques [35].

The GCN employed by Wang et al. is a modified version of the GCN proposed by Kipf et al. [35]. Kipf et al. claim that the contribution of their work is two-fold. First, they propose a *"well-behaved layer-wise propagation rule for neural network models"* [35] which can be directly applied on top of the graph. Secondly, they claim that their approach is able to provide *"fast and scalable semi-supervised classification of nodes in a graph"* [35]. Kipf et al. [35] claim that, unlike the approach seen in [17], their proposed method scales to large graphs with wide node degree distribution. As mentioned above, the input graph to the GCN employed in this thesis is defined by the IPS of each detection. The goal of the GCN is to use said local structure to learn to predict link likelihoods between each detection and its one-hop neighbors within its IPS. More specifically, the network outputs per node pair the probability of a link and no link between the two nodes.

Chapter 3

MOTChallenge

To make results of this thesis comparable to other research within the field of MOT, the MOT17 [40, 48] was chosen as input data for experiments. The goal of the challenge is to track pedestrians within videos using three sets of detections provided by the competition. In the following, the dataset (see Section 3.1) as well as the employed evaluation metrics by the MOTChallenge (see Section 3.2) will be described in detail.

3.1 Dataset

The MOT17 dataset consists of seven train and seven test sequences. Each of the fourteen sequences can be categorized by the type of camera used (moving or static), the viewpoint of the camera (high, medium or low) as well as the weather conditions (sunny, cloudy, night or indoor). Table 3.1 gives an overview of the settings employed in the sequences included in the MOT17 dataset. Each train sequence has a corresponding test sequence, which was filmed using (approximately) the same setting.

The MOT17 dataset provides pedestrian detections of three public detectors: DPM [18], FRCNN [52] and SDP [66]. Additionally, for train sequences ground truth detections are supplied. Table 3.2 gives an overview of how many detections there are per detector per sequence.[1] To avoid overfitting, ground truth annotations of the test sequences are not released by the challenge [48].

[1] Ground truth information, as well as a download link of the MOT17 dataset, can be found on the official MOTChallenge website `https://motchallenge.net/data/MOT17/` The number of detections per detector were calculated by counting detections included in the respective *det.txt* files of the data download.

Training sequences						
Name	FPS	Resolution	Length	Camera	Viewpoint	Conditions
MOT17-02	30	1920x1080	600 (00:20)	static	medium	cloudy
MOT17-04	30	1920x1080	1,050 (00:35)	static	high	night
MOT17-05	14	640x480	837 (01:00)	moving	medium	sunny
MOT17-09	30	1920x1080	525 (00:18)	static	low	indoor
MOT17-10	30	1920x1080	654 (00:22)	moving	medium	night
MOT17-11	30	1920x1080	900 (00:30)	moving	medium	indoor
MOT17-13	25	1920x1080	750 (00:30)	moving	high	sunny
Total Training			**5,316 (3:35)**			

Testing sequences						
Name	FPS	Resolution	Length	Camera	Viewpoint	Conditions
MOT17-01	30	1920x1080	450 (00:15)	static	medium	cloudy
MOT17-03	30	1920x1080	1,500 (00:50)	static	high	night
MOT17-06	14	640x480	1,194 (01:25)	moving	medium	sunny
MOT17-07	30	1920x1080	500 (00:17)	moving	medium	shadow
MOT17-08	30	1920x1080	625 (00:21)	static	medium	sunny
MOT17-12	30	1920x1080	900 (00:30)	moving	medium	indoor
MOT17-14	25	1920x1080	750 (00:30)	moving	high	sunny
Total Testing			**5,919 (4:08)**			

Table 3.1: Employed settings for each sequence included in the MOT17 benchmark dataset [48]

.

3.2 Evaluation

In addition to the data, the MOTChallenge also provides standardized evaluation metrics, which can be used to evaluate results as well as determine the final ranking within the challenge's leaderboard. The metrics consist of the CLEAR MOT metrics proposed by Bernadin and Stiefelhagen [5] as well as metrics introduced by Ristani et al. [53] and Li et al. [41]. In the following, each evaluation metric and how the final ranking within the MOTChallenge is determined will be explained.

Tracker-to-target assignments. Two central metrics surrounding MOT are *identity switches (IDS)* and *fragmentations (Frag)*. An *IDS* occurs if the hypothesis assigned to a ground truth trajectory changes along the track. Contrarily, a *fragmentation* occurs if a ground truth trajectory, which was previously assigned a hypothesis, is not assigned to any hypothesis anymore in the next frame. In order to come up with a clear definition of what an *identity switch*, as well as a *fragmentation* of a trajectory is, one must first define a one-to-one mapping between the tracker's detections and the ground truth detections [48].

Training sequences							
		Detections			Ground Truth		
Name	FPS	DPM	FRCNN	SDP	Detections	Tracks	Density
MOT17-02	30	7,267	8,186	11,639	18,581	62	31.0
MOT17-04	30	39,437	28,406	37,150	47,557	83	45.3
MOT17-05	14	4,333	3,848	4,767	6,917	133	8.3
MOT17-09	30	5,976	3,049	3,607	5,325	26	10.1
MOT17-10	30	8,832	9,701	10,371	12,839	57	19.6
MOT17-11	30	8,590	6,007	7,509	9,436	75	10.5
MOT17-13	25	5,355	8,442	7,744	11,642	110	15.5
Total Training		79,790	67,639	82,787	336,891	1638	21.1

Testing sequences				
		Detections		
Name	FPS	DPM	FRCNN	SDP
MOT17-01	30	3,775	5,514	5,837
MOT17-03	30	85,854	65,739	80,241
MOT17-06	14	7,851	7,809	8,283
MOT17-07	30	11,309	9,377	10,273
MOT17-08	30	10,042	6,921	8,118
MOT17-12	30	7,764	4,276	5,440
MOT17-14	25	8,781	10,055	10,461
Total Testing		135,376	109,691	128,653

Table 3.2: Number of DPM, FRCNN and SDP as well as ground truth detections for each sequence included in the MOT17 benchmark dataset.

First, for each hypothesized output of a tracker one has to decide whether the output is to be considered a *true positive (TP)*, meaning there is a corresponding ground truth detection or *false positive (FP)*, meaning it is representing something other than the target class. This is done via a distance (or dissimilarity) measure d between the detection and all target detections, within the frame. If d does not exceed a pre-defined threshold t_d for at least one target detection, then the tracker's detection is to be considered a *true positive (TP)*. As distance metric between two bounding boxes, Milan et al. [48] suggest using the intersection over union (IoU), which is the ratio of overlap to union of two bounding boxes and employ 50% (or 0.5) as threshold t_d. All other target (ground truth) detections that were missed by the tracker, i.e., were not within the threshold distance of a tracker's detection, are to be considered *false negatives (FN)*. Generally speaking, a tracker should try to minimize the amount of FP and FN across all frames. Besides the absolute number, one can also look at the average number of *false alarms per frame (FAF)*, i.e., the false positive ratio across frames.

Using *FP*, *FN* and *TP* one can also compute *recall* and *precision* of the tracker. The computation of the metrics seen in 3.1 are taken from the Chapter *Precision and Recall* in [16]. Applied to the scenario at hand, *recall* is the fraction of ground truth bounding boxes that were assigned an ID compared to all ground truth detections [48]. In contrast, *precision* is the ratio of detections that do belong to a ground truth detection divided by the total amount of the tracker's detections (see Equation 3.1).

$$Recall = \frac{TP}{TP + FN} \tag{3.1a}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.1b}$$

Using the above-mentioned method to come up with *FP*, *FN* and *TP* can result in multiple detections being assigned to the same ground truth detection, which distorts a tracker's *TP* values. Milan et al. [48] explain how they decide on a final one-to-one correspondence. First, they assume that each ground truth trajectory, i.e., bounding boxes of a person across frames, has one unique start and end point. If a person leaves the field-of-view of the camera and reappears at a later point in time, the person is assigned a new identity. Thus, person reidentification is not explicitly handled within the evaluation of the MOT17 dataset. Milan et al. [48] then exemplify a stricter definition of identity switches than proposed by Bernadin and Stiefelhagen [5], which draws inspiration from Li et al. [41]. Within said approach matching is not performed independently for each frame [48]. More specifically, if a ground truth trajectory i is matched to a tracker trajectory (or hypothesis) j in frame $t - 1$, they are also matched in frame t if the distance (dissimilarity) between their respective bounding boxes in frame t is below t_d, even if there are other hypotheses whose distance (dissimilarity) in frame t is smaller.

Target-like annotations. Milan et al. define the target class as *"all upright people, standing or walking, that are reachable along the viewing ray without a physical obstacle, i.e. reflections, people behind a transparent wall or window are excluded."* [48]. Said definition also excludes people on bicycles or other vehicles from the target class. Ground truth detections which overlap more than 50% with excluded classes (i.e., distractor, static person, reflection, person on vehicle) are excluded from the final evaluation. Milan et al. [48] follow a strategy similar to the one employed in [47] as they do not reward nor penalize trackers for tracking or not tracking the excluded ground truth detections [48].

Track quality measures. Once correspondences have been identified, one is able to calculate track quality measures. First, one can identify *identity switches (IDS)* and *fragmentations (Frag)* [48]. The former occurs when a ground truth trajectory is assigned a hypothesis different from the one it was last assigned to. The latter occurs when a ground truth trajectory was matched to a hypothesis in frame $t-1$ but is not matched to any hypothesis in frame t or when a ground truth trajectory is interrupted. Besides counting the absolute number of *IDS* and *fragmentations*, one can also set them into relation to the calculated *recall* measure. As trackers which produce more trajectories will have likely more *IDS* and *fragmentations*, looking at the *identity switch ratio (IDSR)* and *fragmentation ratio (FRR)* values will deliver a better understanding of a tracker's performance.

Depending on the percentage of tracked bounding boxes, i.e., bounding boxes being assigned to any hypothesis, ground truth trajectories can be classified as *mostly tracked (MT)* or *mostly lost (ML)*. The former are tracks of which at least 80% of the bounding boxes are assigned to any hypothesis, while the latter describes all tracks of which less than 20% of bounding boxes are assigned. Here it is to be noted that the measures disregard *identity switches* as the number of hypotheses being matched does not matter for a ground truth trajectory to be *mostly tracked (MT)*.

Multi-Object Tracking Accuracy (MOTA). The *MOTA* score combines three sources of errors, namely *FP*, *FN* and *IDS* [48]. The exact computation can be seen in Equation 3.2, where t is the frame index. If the tracker makes more mistakes than there are ground truth detections, the *MOTA* can be negative.

$$MOTA = 1 - \frac{\sum_t FN_t + FP_t + IDS_t}{\sum_t GT_t} \tag{3.2}$$

As the main intention of the MOTChallenge is to test trackers for their robustness and ability to perform on unconstrained environments as well as unseen data, Milan et al. [48] also calculate the standard deviation of the *MOTA* score across sequences. A large standard deviation might hint towards the tracker overfitting on certain scenarios and data.

Identity F1 Score (IDF1). To calculate the *IDF1* score of a tracker a bipartite graph $G = (V_T, V_C, E)$ is constructed [53]. The vertex set V_T contains a 'regular' node τ per ground truth trajectory and a 'false positive' node f_γ^+ per hypothesis produced by the tracker, while the vertex set V_C contains one 'regular' node γ per hypothesis and one 'false negative' node f_τ^- per ground truth trajectory. 'Regular' τ nodes are connected via an edge $e \in E$ with their corresponding 'false negative'

f_τ^- node, as well as 'regular' γ nodes are connected to their corresponding 'false positive' f_γ^+ node. An edge $(\tau, \gamma) \in E$ exists between two 'regular' nodes if the corresponding trajectories overlap in time and is associated with a cost that is the number of FP and FN incurred from choosing said matching of trajectories. More specifically, one can define the cost of an edge $(\tau, \gamma) \in E$ as seen in Equation 3.3, where $m(\tau, \gamma, t, \Delta)$ is 1 if two trajectories are not within threshold distance Δ in frame t and $\mathcal{T}_\tau$ ($\mathcal{T}_\gamma$) is the set of frames that the trajectory τ (γ) covers.

$$c(\tau, \gamma, \Delta) = \underbrace{\sum_{t \in \mathcal{T}_\tau} m(\tau, \gamma, t, \Delta)}_{\text{False Negatives}} + \underbrace{\sum_{t \in \mathcal{T}_\gamma} m(\tau, \gamma, t, \Delta)}_{\text{False Positives}} \tag{3.3}$$

The resulting graph can be reduced to a one-to-one matching using minimum-cost solution resulting in a set of (τ, γ) pairs. In the following *True Positive ID (IDTP)*, *False Positive ID (IDFP)* and, *False Negative ID (IDFN)* are calculated as seen in Equation 3.4. Within the equation, AT is the set of all ground truth trajectories τ that are matched to a hypothesis γ and $\tau_m(\gamma)$ is a function that returns hypotheses γ that τ is matched with via an edge. Respectively AC is the set of all hypotheses γ that are matched to a ground truth trajectory τ and $\gamma_m(\tau)$ is a function that returns ground truth trajectories τ that γ is matched with via an edge. Lastly, len(τ) and len(γ) are the number of detections that the trajectories consist of.

$$IDFN = \sum_{\tau \in AT} \sum_{t \in \mathcal{T}_\tau} m(\tau, \gamma_m(\tau), t, \Delta) \tag{3.4a}$$

$$IDFP = \sum_{\gamma \in AC} \sum_{t \in \mathcal{T}_\gamma} m(\gamma, \tau_m(\gamma), t, \Delta) \tag{3.4b}$$

$$IDTP = \sum_{\tau \in AT} \text{len}(\tau) - IDFN = \sum_{\gamma \in AC} \text{len}(\gamma) - IDFP \tag{3.4c}$$

Using the values for $IDFN$, $IDFP$ and, $IDTP$ one can now calculate the *identity precision (IDP)* and *identity recall (IDR)* and subsequently the *identity F1 (IDF1)* score (see Equation 3.5). Ristani et al. define the *IDF1* score as the *"ratio of correctly identified detections over the average number of ground-truth and computed detections."* [53].

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{3.5a}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{3.5b}$$

$$IDF1 = \frac{2IDTP}{2IDTP + IDFP + IDFN} \tag{3.5c}$$

Multi-Object Tracking Precision (MOTP). The *MOTP* evaluation metric focuses on analyzing how well a tracker is able to identify *TP* as precisely as possible [48]. It does so by looking at the average distance d between each matched ground truth and tracker detection across all frames. The exact computation can be seen in Equation 3.6, where c_t denotes the number of matches in frame t and $d_{t,i}$ is the bounding box overlap of the tracker's detection i and the matched ground truth detection, i.e. fulfilling the distance threshold t_d. Since detections are provided by the MOTChallenge and the *MOTP* score focuses on evaluating the precision of the detections, it is to be considered less significant in the evaluation of a tracker's performance.

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \tag{3.6}$$

For the same reason as with the *MOTA* score, Milan et al. [48] advice calculating the standard deviation of the *MOTP* score across all sequences.

Tracker Ranking. Due to the different perspective that each evaluation measure can give, it is difficult to come up with a final ranking of detectors. On the official MOTChallenge website, two types of leaderboards[2] can be accessed. Both leaderboards are by default sorted according to the *MOTA* score of each tracker but can be changed to be sorted by any other of the evaluation metrics. Besides a public leaderboard, which uses the detection datasets as is, there also exists a private leaderboard. Submission on the private leaderboard differ from the public leaderboard as in that data from outside the competition was used to train the trackers. Table B.2 within the appendix gives an overview of all MOT17 evaluation metrics.

[2]See `https://motchallenge.net/results/MOT17/`.

Chapter 4

Implementation

The following chapter is going to describe the core implementation that was used to perform experiments. The implementation can be accessed on GitHub[3], where a quick start guide is provided if one wants to perform their own experiments. The implementation can be split into four parts. Section 4.1 outlines the dataset creation which uses the MOT17 dataset as input to create all necessary input files for subsequent steps. Afterward, Section 4.2 describes the GCN clustering which is based on the implementation provided by Wang et al. [62], but was extended during the course of this thesis. Next, Section 4.3 highlights the workflow of the postprocessing script which was used to convert outputs of the GCNs into trajectories. Lastly, Section 4.4 is going to sketch the evaluation script which consists of computing the official MOT17 metrics described in Chapter 3.2 using the evaluation script of Christoph Heindl [4] as well as additional plots used for analyzing the GCN.

4.1 Dataset Creation

The GCN clustering implementation by Wang et al. [62] requires three main input files. First, a file containing the labels of each node, i.e., detection, within the graph. Second, a features file which contains the features of each node within the graph, i.e., features associated with each detected bounding box, within the graph. Third, a KNN graph file which per row contains the ID of a detection as well as the IDs of its k-nearest neighboring detections. In the following, the creation process of each of the three input files, as well as necessary assumptions that needed to be made to do so, will be described.

[3] See `https://github.com/mariusbock/master-thesis`.
[4] See `https://github.com/cheind/py-motmetrics`.

Labels. Detection files in the MOT17 dataset do not provide any labels. Nevertheless, labels are needed during training as the proposed clustering approach is a supervised learning algorithm. To overcome this, IoU between a detection and each ground truth detection was used. A large overlap between a detection and a ground truth bounding box is assumed to indicate that both bounding boxes are describing the same person and are both to be assigned the same label. Contrarily, a small (or no) overlap of a detection and a ground truth bounding box indicates that they are not describing the same person and are not sharing the same label.

Using these assumptions, the IoU between a detection and each ground truth bounding box is calculated. Given the largest IoU d_{IoU} calculated between the detection and a ground truth bounding box and a predefined upper o_h and lower threshold o_l, three cases are to be differentiated that determine the final label of the detection:

- If $d_{IoU} > o_h$, it is assumed that both detections are describing the same person. The detection is assigned the label of the ground truth detection.
- If $d_{IoU} < o_l$, it is assumed the detection is not describing a person but is detecting background. The detection is assigned a background label b.
- If $o_h > d_{IoU} > o_l$, it is assumed that the detection is neither describing a person nor is to be considered a background detection. The detection is assigned an unclassified label u.

The label creation process is summarized in Algorithm 1. Within the algorithm, D represents the detector dataset and GT represents the ground truth data file. Further, O is the employed IoU threshold pair (o_h, o_l) to be used during labeling and bh (uh) is the method that determines what label is assigned to background (unclassified) detections. There are currently two background (unclassified) labeling strategies implemented, namely cluster or singleton label assignment. The former assigns each background (unclassified) detection the same label, e.g., 0 (-1), while the latter assigns each background (unclassified) detection a different label by incrementing the largest assigned label within the ground truth detection file iteratively by one.

Note that Algorithm 1 is only a high-level overview of the actual implemented function.[5] For reasons of simplicity in Algorithm 1, O is not a list of threshold pairs, but just a pair of thresholds. If one would want to implement the former case, one needs to keep track of multiple lists, each representative of one threshold pair. Instead of storing the results in arrays, the actual implementation stores

[5] See function *create_label_dataset* within the *dataset_creation.py* script.

Algorithm 1

CREATELABELDATASET(D, GT, O, bh, uh)

```
▷ initialize arrays to store information in
iou_array ← array of size len(D); initialized with all −1
label_array ← array of size len(D); initialized with all −1
curr_max_label ← largest label assigned in GT
for i, det_D in D do
    ▷ filter ground truth detection file to only be detections from the same frame as det_D to increase speed of algorithm
    GT_fil := all ground truth detections with the same frame as det_D
    for det_GT in GT_fil do
        ▷ calculate IoU between det_D and det_GT
        curr_iou ← iou(det_D, det_GT)
        if curr_iou > iou_array[i] and curr_iou > 0 then
            iou_array[i] ← curr_iou
            if curr_iou > o_h then
                label_array[i] ← label of det_GT
            else if o_h ≥ curr_iou ≥ o_l then
                if uh = 'clusters' then
                    label_array[i] ← −1
                else if uh = 'singletons' then
                    curr_max_label = curr_max_label + 1
                    label_array[i] ← curr_max_label
                end if
            else if curr_iou < o_l then
                if bh = 'clusters' then
                    label_array[i] ← 0
                else if bh = 'singletons' then
                    curr_max_label = curr_max_label + 1
                    label_array[i] ← curr_max_label
                end if
            end if
        end if
    end for
end for
```

all relevant information of the label creation process in a metadata file, in order to reconstruct it at a later point in time as well as make the process easier to debug and more understandable. Besides label and IoU information for each threshold pair $(o_h, o_h) \in O$, the implementation stores which ground truth and detection bounding box were matched, frame information, employed background (unclassified) handling method as well as boolean arrays which allow for easy filtering to obtain all person, background or unclassified detections.

Features. During the course of this thesis, five types of feature datasets were created. Each feature dataset tries to encode different types of information and viewpoints to solve MOT. For the creation of the reidentification, classification and person feature dataset pre-trained networks were used to extract feature maps [72, 25, 61]. The choice of the three networks was made because of what kind of feature maps they created. The networks can be interchanged with any other implementation that tries to solve the same problem. The result of each dataset creation is an array F that contains one feature vector $det_F \in F$ per detection. The size of the feature vector depends on the method that is used. The idea behind each dataset and what a network would need to provide in order to be interchanged with the current selection of networks, will be described in more detail below.

(Person) Reidentification Features. The reidentification dataset is calculated using a modified version of the implementation provided by Zheng et al. [72].[6] Given a set of images and a sample image of a person of interest, (person) reidentification aims at identifying all other occurrences of that person within the set of images. This is especially challenging due to intra-class variations of a person across images due to e.g., different camera angles. To solve this issue, solutions focused on exposing the network to (intra-class) variations of a person during training by generating new images. A popular choice to do so are Generative Adversarial Networks (GANs) [21]. Performance of GANs is rated by diversity and 'realism' of the produced images [72]. According to Zheng et al., other approaches, which use GANs to improve reidentification learning, struggle to align optimization of the GAN with the reidentification learning task. The network proposed by Zheng et al., called DG-Net, combines both discriminative and generative learning in a unified network. The generative part of their solution encodes a pedestrian image into two latent spaces, namely an appearance and structure space. The former encodes *"appearance and other identity related semantics"* [72], while the latter

[6] See `https://github.com/layumi/Person_reID_baseline_pytorch` for an inofficial implementation of [72]. Note that the repository provides multiple pretrained models with ft_net resembling the model proposed in [72].

encodes mostly spatial information. The interactions between the generative and discriminative part of the solution are three-fold. Firstly, images generated by the generative part of the solution are used to refine the appearance encoder. Secondly, the appearance encoder improves the generative part with better encoding. Thirdly, as the appearance encoder serves as the backbone for the reidentification learning, both modules are jointly optimized based on the shared encoder. The image generation is performed by switching the appearance or structure spaces of two images, resulting in variations of images with the same or different identity. For example, given two images of different pedestrians one is able to create new images by applying the pose, clothing, viewpoint or background of the one person on the other person.

Algorithm 2 describes the main approach that is employed to perform feature extraction using a pretrained network. The algorithm is similarly used to create the person and classification dataset. The main premise of the algorithm is to alter existing pretrained networks so that they return their feature representations. In case of the DG-NET, the network is altered to convert an input image into a 2048-channel feature map. The feature map can then be cropped according to each detection bounding box coordinates and thus provide a feature representation per detection which encodes e.g., reidentification features of a pedestrian. Input to Algorithm 2 is the modified network M, the detections D for which a feature vector is to be obtained, the images I of the corresponding sequence, the number of channels out of the feature map M returns and a boolean indicating whether max pooling should be applied or not. Iterating over all images $i \in I$ each image is passed through the network resulting in a feature map F_i. For each detection within that image, bounding box coordinates are extracted and adjusted according to the height and width reduction network M caused to i. Reduction factors are calculated by dividing the original height (width) by the height (width) of the feature map F_i. As bounding box sizes differ from detection to detection, one is not able to just crop images according to the raw (adjusted) bounding box coordinates.

To solve this, RoIAlign as proposed by He et al. [24] is used, which is a variation of RoIPool [20]. Both methods are able to extract a feature map of any size, e.g., four by four, from a region of interest (RoI). Given a RoI R of size $x \times y$ and a target feature map F of size $n \times n$, RoI pool divides both x and y into n bins and max pools across each resulting sub patch of R. The bin size is calculated by dividing each axis, i.e., x and y, by n and rounding the results. This quantization may cause the last bin to not be the same size as the other bins, depending on the size of the RoI, resulting in misalignments and inaccurate feature maps. Instead of using rounded strides, He et al. [24] propose to define bins using floating strides, apply bilinear interpolation [29] within each bin at four regularly sampled locations and aggregate the result using max or average pooling. This

Algorithm 2

EXTRACTFEATURES(D, I, M, out, max_pool)

▷ *Initialize empty output feature array; if max pooling is applied array has 16-times more columns*
if max_pool **then**
    output_feat ← empty array of size [len(D), ($out * 16$)]
else
    output_feat ← empty array of size [len(D), out]
end if
▷ *load model weights*
$M = load\_weights(M)$
for $i \in I$ **do**
    ▷ *apply preprocessing to each image*
    $i = preprocess(i)$
    D_{fil} ← all detections that are in image i
    ▷ *feed image i through network M to obtain feature representation of i*
    F_i ← returned feature map after feeding image i into M
    δ_w ← width reduction factor of network; obtained by dividing the original image width with the width of F_i
    δ_h ← height reduction factor of network; obtained by dividing the original image height with the height of F_i
    B ← empty of size [len(D_{fil}), (4)]
    $B[:, 0]$ ← x-coordinate of all detections in D_{fil} divided by δ_w
    $B[:, 1]$ ← y-coordinate of all detections in D_{fil} divided by δ_h
    $B[:, 2]$ ← x-coordinate + (40%*height of all detections in D_{fil}) / δ_w
    $B[:, 3]$ ← y-coordinate + (height of all detections in D_{fil}) / δ_h
    ▷ *extract (4,4) image patches corresponding to the detections in B using roi_align*
    $F_D = roi\_align(B, F_i, (4, 4))$
    ▷ *if max pooling is applied image patches are shrunk down to (1,1)*
    if max_pool **then**
        $F_D = max\_pool(F_D)$
    end if
    output_feat ← save detection feature arrays in F_D at the same index as the original detection
end for

results in more accurate feature maps and avoids misalignments [24]. Applied to the scenario at hand, a RoI is a detected bounding box within an image. Output

of the RoIAlign algorithm within Algorithm 2 is a $(4 \times 4 \times x)$ feature map with x being the amount of feature channels, e.g., 2048 in case of the reidentification dataset. If wanted, the feature map can further be reduced to be of size $(1 \times 1 \times x)$ by applying max pooling on each (4×4) feature map. Whether max pooling is applied or not, the feature map is flattened to a one-dimensional feature vector. The resulting feature vector is saved within the final feature output file according to the index of its corresponding detection. Note that Algorithm 2 represents a simplified version of the actual implemented algorithm. The actual implementation is able to process multiple images simultaneously and extracts detections within a frame at once instead of handling them individually. This results in a significant speed up compared to the workflow seen in Algorithm 2.

Classification Features. The classification dataset is calculated using a modified version of a residual network (ResNet) as proposed by He et al. [25]. He et al. witnessed a degradation problem with increased depth of a network. Typically, one would expect that with increased depth a network tends to overfit more and hence training loss would converge to zero. Instead, He et al. observed a plateauing followed by an increase in training error with increased depth. They address this problem by introducing a deep residual learning framework which features residual building blocks. Unlike prior networks like VGGNet [55], the residual building blocks introduce skip connections allowing networks to skip one or more weight layers, i.e., learning an identity mapping for said layers. These identity mappings do not increase computation time, as no additional parameters need to be learned and the whole network can still be learned via stochastic gradient descent using backpropagation. Their proposed network ResNet in [25] is inspired by VGGNet [55], but adds an additional 15 layers to the network, while having fewer filters and lower complexity.

The ResNet used to create the feature dataset for this thesis consist of 50 (ResNet-50) instead of 34 as proposed in [25] and was pretrained on the ImageNet dataset [14]. The ImageNet dataset features 1000 target classes. Unlike the underlying network reidentification dataset, the pretrained ResNet delivers a more generalized feature embedding of each detected pedestrian. The classification dataset is thus to be considered to deliver a less specialized view. As with the reidentification dataset creation, the ResNet was modified to return its 2048-channel feature map instead of passing it to the classifier and coming up with a final prediction. Using Algorithm 2, the modified network was used to come up with a feature embedding per detection. The final feature dataset is of size (len(D), 2048) if max pooling is applied or of size (len(D), 32768) without max pooling applied with D being the original detection dataset.

Person Features. The person dataset is calculated using a modified version of the implementation provided by Wang [61]. Goal of Wang's implementation is to detect persons within images by coming up with a set of bounding boxes. The proposed approach, called Adapted Center and Scale Prediction (ACSP) network, focuses on leveraging capabilities of the approach of Liu et al. [45], called Center and Scale Prediction (CSP), by solving its sensitivity to batch size and different input scales as well as improve its occlusion and crowd handling process [61]. Originally, the CSP detector was the first anchor-free pedestrian detection method. The detector is made of a feature extraction and a detection head part. Within the feature extraction part, a backbone architecture, e.g., a ResNet-50 [25], is used to extract features from an input image. Unlike Liu et al. [45], Wang feeds images with their original dimensions into the network [61]. The result of the feature extraction layer is a feature map with dimensions $(256, \frac{h}{4}, \frac{w}{4})$, where h and w are the original input image's height and width. This feature map is fed into the detection head part which consists of three (parallel) convolutions that try to predict center (1×1 convolution), scale (1×1 convolution) and offset (2×2 convolution) of the bounding boxes respectively. Unlike the CSP network, the ACSP network proposed by Wang uses switchable normalization (SN) layers as proposed by Luo et al. [46] instead of batch normalization (BN) layers as they are sensitive in their performance for different batch sizes as well as image input dimensions [61]. SN layers combine batch, layer and instance normalization by taking the weighted average of mean and variance values of each normalization type [46]. Wang also reports better results compared to CSP when switching to a ResNet-101 [25] backbone instead of ResNet-50 [25] backbone as higher-level semantic features are only learned using deeper feature maps. To calculate the final loss of the network the three-fold loss of the detection head (i.e., classfication, scale and offset loss) is combined using L1 loss with weights being set to 0.01 for the classification loss, 0.05 for the scale loss and 0.1 for the offset loss [61].

To create the feature dataset used as input for the GCN clustering, the ACSP network was slightly modified. As with the classification and reidentification feature dataset, goal was to use the feature representation of the ACSP network as input to the GCN. To achieve this, the detection head was omitted from the calculation so that the network returned the feature map of the feature extraction layer. To initialize the network, Wang [61] provided trained weights for the ResNet-101 and ACSP optimized using L1 loss.[7] To create a feature vector for each bounding box, images of the corresponding sequence as well as the ACSP network initialized using the pretrained weights provided by Wang are passed into Algorithm 2. Un-

[7] See the download links provided in the GitHub repository of the implementation `https://github.com/WangWenhao0716/Adapted-Center-and-Scale-Prediction`.

like the networks used during the reidentification and classification dataset creation the ACSP network requires input images to be of aspect ratio 1:2. This is caused by the original network as proposed by Wang [61] being trained on the CityPersons dataset [71], which only contains images of aspect ratio 1:2. As images contained in the MOT17 dataset are mostly of aspect ratio 16:9, preprocessing applied during the person feature dataset creation includes zero-padding images until they suffice the required aspect ratio. The images are then resized to be of size $(640, 1280)$ to reduce memory-usage. The zero-padding of the images prevents that the images are getting distorted due to the resizing. Lastly, normalization as seen in during the construction of the reidentification and classification dataset is applied to each image. Applying the feature extraction algorithm (see Algorithm 2) using the modified version of the ACSP network results in a feature array of size (len(D), 256) (or (len(D), 1024) without max pooling being applied), where each row encodes the corresponding detection's likelihood of representing a person.

Spatial Features. A feature vector in the spatial feature dataset consists of the frame index, the x- and y-coordinate, the height and the width of a detection bounding box. The idea of the spatial feature dataset is to capture spatial (and temporal) information of a detection, i.e., in which frame and where in the frame the detection is located. This assumes that a large spatial (and/ or temporal) distance between two detections is an indicator that they are not describing the same instance.

Extra Features. The extra feature dataset holds some additional features that were calculated to be used as an additional information and hence (potentially) complement the above-mentioned features. These features include the absolute size of the detection bounding box and the detector confidence of each detection. The former is obtained by multiplying the width and height of a bounding box, while the latter is provided with the MOT17 dataset and expresses the detectors certainty that the detection it has produced is a true detection, i.e., a person.

KNN Graphs. A KNN graph consists of a pivot node and its k nearest neighbors. Applying this to the scenario at hand, a node represents a detection within a frame. To decide which nodes within a dataset are to be considered 'nearest' to the pivot node, a distance metric $\delta(v_1, v_2)$ is applied where v_1 and v_2 are the feature vectors of the two nodes of which the distance is to be measured. In total three variants were implemented to come up with the set of k-nearest neighbors.[8] Regardless of which method is used, the result is an array which contains per row the index of the

[8] see function *create_knn_graph_dataset* in *dataset_creation.py*

pivot node followed by the indices of its k closest neighbors sorted from smallest to largest distance to the pivot node according to the employed distance metric δ.

The (brute-force) approach to compute the set of nearest neighbors for a pivot node is to calculate the distance between the pivot node and all other nodes within the feature dataset, sort the nodes according to distance and select the k nodes with the smallest distance. This is known to be the standard way of constructing a KNN graph and implementations to apply this technique on a dataset are readily available (e.g. *NearestNeighbors* implementation of *scikit-learn*.[9]) While performing experiments using the spatial feature dataset (see Chapter 5.2), it was noted that results showed a high degree of variance. Chapter 5.2 will further try to analyze this flaw in greater detail by analyzing the label composition and structure of IPS during used training as well as provide results using optimized versions of the KNN graph calculation.

To ensure temporal proximity of detections within the KNN graphs another type of KNN graph was implemented (see Algorithm 3). Unlike the 'standard' KNN described above this approach does not calculate the distance between all other detections within the feature dataset F, when coming up with the set of k nearest neighbors for a given detection's feature vector $det_F \in F$, but instead pre-filters F to only include detections within a certain frame distance d_f to det_F. Given the frame index f_{det} of det_F, the filtered dataset that is to be used for calculation of the KNN graph only contains detections whose frame distance is greater or equal than $f_{det} - d_f$ and less or equal than $f_{det} + d_f$. This decreases the likelihood of neighborhoods consisting of detections that have a large temporal distance to detection det_F. As it is possible that there are not enough detections within frame distance d_f of det_F, d_f is iteratively increased until at least $k + 1$ detections satisfy the condition.

Additionally, the approach to compute the KNN on a filtered version of F was further extended with a slight variation (see Algorithm 4). Instead of filtering the original feature dataset F according to the frame distance f_d, this variation pre-filters F according to some filter dataset F_{fil} before it is filtered according to f_d. This is done by computing the 'standard' KNN graph using F_{fil} as input obtaining a set of k_{fil} neighbors per detection feature vector $det_F \in F$. Note that k_{fil} should preferably be set to a large value, e.g. 500, and needs to be at least greater or equal than k. Input to the above mentioned frame-distance variant of the KNN graph construction is then only those k_{fil} pre-filtered neighbor detections, ensuring that the calculated closest neighbors are not only temporally proximate, but also proximate according to the feature-dimensions of the filter dataset.

[9] See the scikit-learn documentation `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html`.

Algorithm 3

FRAMEDISTANCEKNN(k, F, d_f, γ)

$output\_knn \leftarrow$ empty array of size [len(F), $k+1$]
for $det_F \in F$ **do**
  $id_{det} \leftarrow$ index of current detection
  $f_{det} \leftarrow$ frame index of current detection
  $F_{knn} \leftarrow$ all detections $\mathrm{det}_{fil} \in F$ with frame index f_{fil} where $f_{det} - d_f \leq f_{fil} \leq f_{det} + d_f$
  while len(F_{knn}) $\leq k+1$ **do**
    $d_f = d_f + 1$
    $F_{knn} \leftarrow$ all detections $\mathrm{det}_{fil} \in F$ with frame index f_{fil} where $f_{det} - d_f \leq f_{fil} \leq f_{det} + d_f$
  end while
  ▷ *obtain indices of k nearest neighbors of det_F using F_{knn} as input feature and γ as distance metric*
  $neighbors_{det} \leftarrow get\_neighbors(k, det_F, F_{knn}, \gamma)$
  $output\_knn[idx, 0] = id_{det}$
  $output\_knn[idx, 1:] = neighbors_{det}$
end for

4.2 GCN Clustering

In the following, the workflow of the GCN clustering part of the implementation will be illustrated. The implementation can be split into four parts namely the data feeder, the GCN model, the link merging and the training/ testing methods. The clustering is based on the implementation of Wang et al. [62], but was modified to alter computation of the GCN (e.g., by introducing multiplicative parts) and deliver additional functionalities like extraction of intermediate losses and feature maps for plotting or stacking of different networks.

The main idea of the implementation as proposed by Wang et al. is learning to predict the likelihood that two nodes are linked within a graph by looking at their respective neighborhoods. Unlike methods like PageRank [8] their method as seen in [62] does not need look at the whole graph in order to solve the link prediction problem. Furthermore, the algorithm only needs to predict the linkage likelihood between an instance and its k_1 nearest (one-hop) neighbors, which in turn lowers computation costs significantly [62]. As mentioned in Chapter 2.1 one can reduce clustering to a link prediction problem [69]. Within this setting instances that are to be clustered are represented as nodes. An edge between two nodes, i.e., two instances, describes the likelihood that both instances describe the same entity, i.e.,

Algorithm 4

PREFILTEREDKNN($k, F, d_f, k_f, F_{fil}, \gamma$)

$output\_knn \leftarrow$ empty array of size [len(F), $k+1$]
▷ *compute 'standard' KNN graph for dataset* F_{fil}
$knn_{fil} \leftarrow compute\_knn(k_f, F_{fil}, \gamma)$
for $det_F \in F$ **do**
  $id_{det} \leftarrow$ index of current detection
  $f_{det} \leftarrow$ frame index of current detection
  $F_{knn} \leftarrow$ all detections $\mathrm{det}_{fil} \in F_{fil}$ with frame index f_{fil} where $f_{det} - d_f \leq f_{fil} \leq f_{det} + d_f$]
  while len(F_{fil}) $\leq k+1$ **do**
    $d_f = d_f + 1$
    $F_{knn} \leftarrow$ all detections $\mathrm{det}_{fil} \in F_{fil}$ with frame index f_{fil} where $f_{det} - d_f \leq f_{fil} \leq f_{det} + d_f$]
  end while
  ▷ *obtain indices of* k *nearest neighbors of* det_F *using* F_{knn} *as input feature and* γ *as distance metric*
  $neighbors_{det} \leftarrow get\_neighbors(k, F_{knn}, \gamma)$
  $output\_knn[id, 0] = id_{det}$
  $output\_knn[id, 1:] = neighbors_{det}$
end for

share the same label. Using a heuristic, e.g., pseudo label propagation [67], one is able to obtain clusters from the predicted graph [62], with each cluster representing instances of one class. Wang et al. apply their framework to cluster images of faces. As mentioned in Chapter 1, the main contribution of this thesis is to investigate the applicability of the above-mentioned framework to MOT. Translating the problem statement mentioned in [62] to MOT gives us a set of detections $D = [d_1, ...d_N]^T \in \mathbb{R}^{N \times F}$ where N is the number of detections and F is the amount of feature dimensions describing each detection (i.e. each person). The goal of the illustrated GCN clustering implementation is to assign a pseudo label y_i to each $i \in 1, 2, ..., N$ so that detections assigned the same label form a cluster.

Pipeline Overview. The pipeline employed in [62] can be broken down into three steps. First, for each detection d its instance pivot subgraph (IPS) is constructed, which tries to encode the information of the neighborhood of d. It consists of the k nearest neighbors of i as well as higher-order neighbors. Secondly, the IPS is used an input data to the GCN which outputs the link likelihood of the detection d and each of its k nearest neighbors. Thirdly, a heuristic tries to find the best possible

clustering using the graph with computed edge weights, i.e., link likelihoods, as input. As heuristic Wang et al. [62] use pseudo label propagation [67]. In addition to said heuristic, primal feasible search heuristic as seen in [34] were also evaluated to see if clustering could be improved.

Data feeder. The main objective of the data feeder is upon query of an detection d to construct and return its IPS [62]. The IPS construction can be broken down into three steps. First, given a pivot instance p (i.e., the detection that is queried) its h-hop neighbors are collected [62]. How many nearest neighbors are chosen to construct the IPS at each hop is denoted by k_i with $i = 1, 2, ..., h$. For example a sample IPS $\mathbf{G}_p(\mathbf{V}_p, \mathbf{E}_p)$ with $h = 2$ and $k_1 = 20$, $k_2 = 5$ will consist of 20 nearest neighbors of the pivot instance p (i.e. one-hop neighbors of p) and 5 nearest neighbors of each of those nearest neighbors (i.e. two-hop neighbors of p). Note that p is not contained in $\mathbf{V}_p$. The second step is to normalize the features of the nodes in $\mathbf{V}_p$. To do so the node features of the pivot instance x_p are subtracted from the features of each node in $\mathbf{V}_p$ namely $\{x_q | q \in \mathbf{V}_p\}$. Result are the normalized node features,

$$\mathcal{F}_p = [..., \mathbf{x}_q - \mathbf{x}_p, ...]^T, \text{ for all } q \in \mathbf{V}_p \tag{4.1}$$

where $\mathcal{F}_p \in \mathbb{R}^{|\mathbf{V}_p| \times F}$. Lastly, edges are added among the nodes in $\mathbf{V}_p$. To do so another parameter u is introduced. Using u the u-nearest neighbors within the entire dataset F of each node $q \in \mathbf{V}_p$ are determined. If a node r is among the u-nearest neighbors of q and is in $\mathbf{V}_p$, then an edge (q, r) is added to $\mathbf{E}_p$. According to Wang et al. [62] this ensures that the degree of nodes within $\mathbf{G}$ does not vary too much. After completing the above mentioned three steps, the normalized node features $\mathbf{F}_p$ as well an adjacency matrix $\mathbf{A}_p \in \mathbb{R}^{|\mathcal{V}_p| \times |\mathbf{V}_p|}$ according to the previously computed edges and nodes are returned by the IPS construction algorithm.

The code implementation of the data feeder during training and validation takes the label, feature and KNN graph dataset as input.[10] The implementation of Wang et al. was slightly modified by introducing new attributes to enable different behaviors of the feeder during the IPS construction. In the following each of these modifications will be described.

As mentioned above node features of the pivot instance are subtracted from each of its neighbors. Wang et al. [62] used signed differences within their implementation (see Equation 4.3). Within this thesis' implementation users are given the choice to use absolute distances, i.e., taking the absolute values of each value within the normalized feature vectors $\mathcal{F}_p$. Said modification is applicable no matter the input feature dataset.

[10] See *Feeder* class in *gcn_clustering/feeder/feeder.py*.

Other modifications that were made focus on experiments involving the spatial feature dataset. Using the original way of calculating the normalized node feature vector (see Equation 4.3), both absolute and signed differences represent the distance between the spatial coordinates as well the timely occurence of the detection (i.e., frame distance). To better compare distances of different detections to the pivot instance, the bounding box coordinates were normalized according to their size. Given the bounding box coordinates $x_{1,q}$, $y_{1,q}$, w_q and h_q of a detection q and the bounding box coordinates $x_{1,p}$, $y_{1,p}$, w_p and h_p of the pivot instance p the size-normalized distances are calculated as follows,

$$x_{1,norm} = \frac{x_{1,q} - x_{1,p}}{w_q + w_p} \tag{4.2a}$$

$$y_{1,norm} = \frac{y_{1,q} - y_{1,p}}{h_q + h_p} \tag{4.2b}$$

$$w_{norm} = \frac{x_{1,q} - x_{1,p}}{w_q + w_p} \tag{4.2c}$$

$$h_{norm} = \frac{y_{1,q} - y_{1,p}}{h_q + h_p} \tag{4.2d}$$

with $x_{1,norm}$, $y_{1,norm}$, w_{norm} and h_{norm} being the size-normalized coordinate distances between the bounding boxes of detections p and q. Note that the difference in frame information as well as all other features included in the dataset are still calculated using the original way proposed by Wang et al. [62] (i.e. simple subtraction of vector values).

Lastly, users are given the option to introduce a multiplicative into the GCN. This multiplicative part is achieved by calculating element-wise products between either all feature columns contained in the spatial dataset or between the frame distance column and all other features. The former is achieved by multiplying each column pairwise with all other feature columns using the Hadamard product. More specifically, for a given IPS with its normalized node features $\mathcal{F}_p$ for each unique feature column pair $(f_i, f_j) \in \mathcal{F}_p$ calculate a new feature column $f_{i,j}$, using

$$f_{i,j} = f_i \circ f_j \tag{4.3}$$

This results in $\binom{n}{2}$ new columns, where n is the number of feature columns. The latter is achieved by multiplying the frame distance feature column with all other feature columns using the Hadamard product, resulting in $n-1$ new columns, where n is again the number of feature columns. Note that during the combined training setup the appended sequence identifier is omitted from said pairwise calculations.

GCN model. The computed normalized node features $\mathcal{F}_p$ and adjacency matrix A_p of each pivot instance p are next fed into the GCN. The GCN as proposed by Wang et al. [62] applies at each graph convolution layer,

$$\mathbf{Y} = \sigma([\mathbf{X} \parallel \mathbf{GX}]\mathbf{W}), \tag{4.4}$$

with $\mathbf{X} \in \mathbb{R}^{N \times d_{in}}$ being the normalized node features, $\mathbf{Y} \in \mathbb{R}^{N \times d_{out}}$ being the output feature matrix of the GCN layer with N being the number of nodes and d_{in} and d_{out} being the input and output dimensions of the GCN layer. $\mathbf{G} = g(\mathbf{X}, \mathbf{A})$ is an aggregation matrix of size $N \times N$ with $\mathbf{A}$ being the adjacency matrix and $g(\cdot)$ being a function of $\mathbf{X}$ and $\mathbf{A}$. $\mathbf{W}$ is a learnable weight matrix of size $2d_{in} \times d_{out}$ of the GCN layer, operator $\parallel$ symbolizes matrix concatenation along the feature dimension, i.e. the y-axis and $\sigma(\cdot)$ is a non-linear activation function. The computation can be summarized into three steps. First, the node features $\mathbf{X}$ are left multiplied with the aggregation matrix $\mathbf{G}$ and afterward concatenated again with the node features $\mathbf{X}$ along the feature axis. Next, the concatenated feature matrix is multiplied with the weight matrix. Lastly, the activation function $\sigma(\cdot)$ is applied on the results of the previous steps.

Regarding the aggregation function, Wang et al. [62] provide only the mean aggregation option within their code. As only said aggregation function was evaluated during the thesis, the other aggregation function will not be explained. Nevertheless, they could potentially improve the clustering and could be evaluated in the future (see Chapter 6.2). As proposed by Wang et al. [62] the mean aggregation matrix is defined as $\mathbf{G} = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{A}\mathbf{\Lambda}^{-\frac{1}{2}}$, where $\mathbf{A}$ is the adjacency matrix and $\mathbf{\Lambda}$ a diagonal matrix with $\mathbf{\Lambda}_{ii} = \sum_j \mathbf{A_{ij}}$. Once multiplied with the feature matrix $\mathbf{X}$ it encodes the information of the neighborhood of the pivot instance by performing average pooling among neighbors. The overall architecture of the GCN as a whole is depicted in Figure 4.1.

As during this thesis multiple feature datasets with different amount of input dimensions were evaluated, the proposed architecture by Wang et al. [62] was extended by an input adjustment layer. This input adjustment layer converts the input feature dataset from a given feature dimension (depicted as X in Figure 4.1) to a 512-feature dimension using 1×1 convolutions. Following the input adjustment layer, the architecture is unchanged from the one provided by Wang et al. [62].

First, the input, i.e., the normalized node features of the IPS of a given instance, are passed through a batch normalization layer, which is used to speed up and stabilize training [28]. Afterward, four graph convolution layers, as described above and activated by the rectified linear unit (ReLU) [50] function, are applied. As mentioned previously Wang et al. [62] only provide the option to use mean aggregation within their code, thus said aggregation is used in all four graph con-

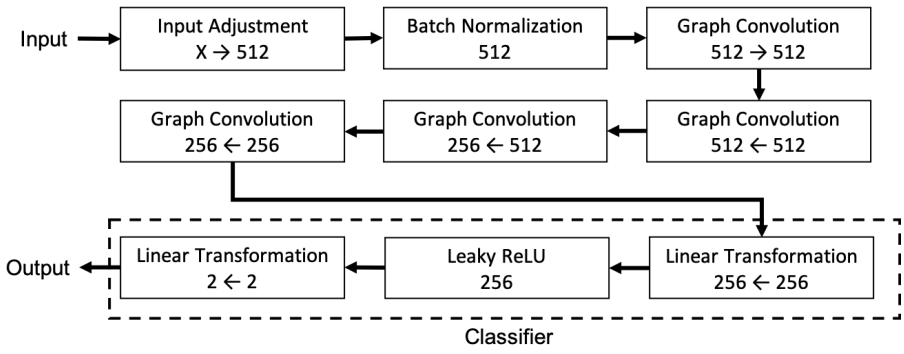


Figure 4.1: Overview of the GCN architecture.

volution layers. Applying the four layers, the feature dimension is shrunk down to 256 features. Lastly, the features are passed on to a classifier which predicts the link likelihoods, i.e., probability that the two nodes link or not link. Wang et al. state in their paper that they only predict the link likelihood between pivot instances and their one-hop neighbors within each IPS, hence the classifier is only passed the feature vectors of the one-hop neighbors [62]. The classifier consists of three layers. First, a fully-connected layer with 256 as input and output dimension, then a layer applying parametric ReLU (PReLU) [26] on top of the 256-output vector of the last layer and lastly a fully connected layer shrinking feature dimensions from 256 to 2 [62]. The parametric ReLU differs from the non-parametric ReLU such that it introduces a learnable parameter a to the calculation of the activation function [25] (see Equation 4.5a (ReLU) and 4.5b (PReLU) for comparison of the two activation functions).

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ 0, & \text{if } y_i \leq 0 \end{cases} \tag{4.5a}$$

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases} \tag{4.5b}$$

Finally, the softmax activation function is applied on the classifier's output and cross-entropy loss is calculated as the objective function [62]. As only the one-hop neighbors are predicted, only their loss is backpropagated through the network and used to adjust weights. According to Wang et al. [62] excluding higher-order neighbors causes a considerable decrease in computation time of the network as well as accuracy gain. They account this behavior to higher-order neighbors being

mostly negative examples, i.e., not linked via an edge to the pivot instance, causing the distribution of positive and negative samples to be less skewed when only considering one-hop neighbors.

Link Merging. Once the link likelihoods for each instance to its one-hop neighbors have been produced, a final clustering needs to be obtained. To achieve this Wang et al. [62] use pseudo label propagation (see Algorithm 5) as seen in [67]. The heuristic takes as input a graph $\mathbf{G} = (\mathbf{E}, \mathbf{V})$ with positive edge weights $\mathbf{W_E}$. Note that G does not need to be a connected graph [67]. First, the algorithm identifies all connected components c, i.e., clusters, and puts them in a queue Q. Then, for each connected component c within Q. If the size of component c exceeds a predefined maximum cluster size t_c, all edges $e \in \mathbf{E}$ whose weights $w_e \in \mathbf{W_E}$ are below a certain threshold t_w are cut. If a connected component does not exceed the predefined limit, then all nodes within said component are assigned the same pseudo label. Once every component has been processed, the remaining connected components, which are not yet assigned a label, are added back to q and the cut threshold t_w is increased. This process is repeated until all nodes are assigned a pseudo label, i.e., there being no connected components added to q after an iteration. Applied to the scenario at hand, $\mathbf{G}$ is defined using the predicted link likelihoods of the GCN with nodes being the detections and edges (as well as their weights) being defined by the one-hop neighborhoods of each pivot instance (as well as their link likelihood).

Algorithm 5

PSEUDOLABELPROPAGATION(G, t_c, t_w)

$Q \leftarrow$ all connected components c in G
$p = 0$
while Q not empty **do**
  for $c \in Q$ **do**
    if $|c| \geq t_c$ **then**
      $c \leftarrow$ cut all edges in c that are below t_w
    else if $|c| < t_c$ **then**
      $c \leftarrow$ assign all nodes in c pseudo label p
      $p = p1$
    end if
  end for
  $Q \leftarrow$ all connected components c in G without a pseudo label assigned
end while

During this thesis it was also evaluated if a performance gain can be achieved by using a heuristic other than pseudo label propagation. For that primal feasible search heuristics as seen in [34] were applied on the edge weights. Within their work, Keuper et al. [34] propose a generalization of the Minimum Cost Multicut Problem (MP) [12, 15] namely the Minimum Cost Lifted Multicut Problem (LMP) [34]. Given a graph $G = (V, E)$ with edge weights $c_e, \forall e \in E$, goal of the MP is to find a partitioning of the graph into r non-empty subsets such that the sum of edge weights of edges that are cut is minimized [12]. The problem at hand, i.e. obtaining a clustering from the predicted one-hop neighbors can be seen as a MP with nodes being the detections, edges being defined by the one-hop neighborhoods of each IPS and edge weights being the predicted link likelihoods of said edges. With LMP, Keuper et al. [34] propose an optimization problem, whose feasible solutions relate one-to-one to decompositions of a graph with an objective function that can assign for any given pair of nodes a cost if said nodes are to be placed in different components. An instance of the LMP is defined via an undirected graph $G = (V, E)$, a set of additional edges $F \subseteq \binom{V}{2} \setminus E$ and for every edge $vw \in E \cup F$ associated costs $c_{vw} \in \mathbb{R}$. F is defined as the set of edges which are not in E, but whose nodes are within a maximum distance d^* to each other. Note that a MP problem can be seen as a LMP-problem where V is an empty set. As the problem at hand, namely the predicted link likelihood graph of the GCN, is to be classified as a MP, properties of the LMP for cases where V not being empty will not be discussed in this thesis. The two primal feasible search heuristics defined by Keuper et. al [34] are applicable to LMP problems as well as MP problems (i.e., LMP with V being empty). The first algorithm proposed is called the Greedy Additive Edge Contraction (GAEC) algorithm. It starts by decomposing the graph into single nodes and then, iteratively, joining nodes, which minimize the objective function. If no join minimizes the objective function, the algorithm terminates. The second algorithm proposed is an extension of the Kerninghan-Lin (KL) algorithm [32]. It starts with the initial decomposition of the graph and at each step attempts to improve the decomposition, as far as the objective function is concerned, by either moving a node between two neighboring components, moving a node into a separate, new component or joining two components. This is repeated until no improvement can be made. Overall, the KL algorithm can be considered less efficient than the GAEC algorithm, which by design performs more greedy moves resulting in earlier termination.

The heurisitic proposed by Keuper et al. [34] needs to be applied separate from the actual GCN implementation. Therefore, the implementation of Wang et al. was modified, so that each GCN produces a text file per validation sequence, which contains all relevant information for the heuristic as seen in [34] to be employed. Additionally, users are given the option to add 'dummy' edges to the text file. A

'dummy' edge is defined as an edge between two detections which are within the same frame and do not overlap, i.e. have an IoU of zero, therefore assuming they are definite non-match. Each 'dummy' edge is assigned the edge weight zero. Initially, it was noted that the implementation[11] of the heurisitic provided by Keuper et al. [34] was unable to be applied to the problem at hand. This was caused by weights of reoccurring edges being added up for said edge. Applied to the problem at hand it is highly likely for detections that are certain matches to end up in each other's one-hop neighborhood. Consequently, this would cause the final weight to be set too high for said two nodes. To avoid this behavior, the graph loading process within the heuristic was altered. The new workflow loads the graph edge by edge using an input text file containing predicted link likelihoods of a GCN. Once an edge reoccurs, the heuristic either:

1. Only keeps the larger of the two contradicting edge weights.
2. Only keeps the smaller of the two contradicting edge weights.
3. If both edge weights are above 0.5 keeps the larger of the two contradicting edge weights or else if both edge weights are below 0.5 keeps the smaller of the two contradicting edge weights or else sets the edge weight to be zero.

Each of the methods on how to handle multiple defined edge weights were evaluated during experiments (see Chapter 5.5.) Besides an input and an output text file, the implementation of Keuper et al. [34] takes two additional arguments, namely a *theta* and a *beta* value. The former is the coefficients of a polynomial (starting with the constant term), while the latter is the logistic prior probability for edges to be cut.

Training and testing methods. There are two different training types implemented within this thesis. As there are multiple sequences within the MOT17 dataset, one can either choose to train the GCN using multiple sequences in a combined or sequential setup. Within the combined approach, feature datasets from different sequences are concatenated along the x-axis with an identifier column added to the dataset indicating to which sequence the detection belongs to. To avoid having the KNN graphs consider detections from other sequences, they are only calculated using one sequence as input. Afterward, the KNN graph datasets are appended in the same way as the feature dataset and labeling is adjusted so that it is consistent with the combined approach, as (row) indices of the KNN graph will not be correct after being appended to the combined dataset. Subsequently, the combined label dataset is created using the same approach with label datasets

[11] See `https://github.com/bjoern-andres/graph`.

from the different sequences being concatenated along the x-axis and labeling being consistent with the new row indices. The sequential training approach, on the other hand, trains a network iteratively and separately using a list of training sequences. Weights of the network resulting from one training iteration using one of the training sequences from the list are then used to initialize the network for the next training sequence until there are no sequences left to train with.

In addition to the straightforward approach of running only one GCN, an ensemble approach was implemented using outputs of multiple, intermediate GCNs as input for a final GCN (see Figure 4.2). To do so, a different GCN architecture was implemented that instead of returning softmax-probabilities, returns the feature map produced after the second layer of the classifier. The produced feature maps with 256 input channels are concatenated along the y-axis resulting in a feature dataset with $256x$ input channels with x being the amount of GbCNs that were trained. Input to the final GCN is hence the combined feature maps of the previous GCNs, a KNN graph computed using either the combined feature maps as input or any other feature dataset combination and the label dataset that was also used to train the GCNs. Note that the position where feature maps of the intermediate GCNs are extracted is only a proposal and can be altered within the implementation of the intermediate GCNs.

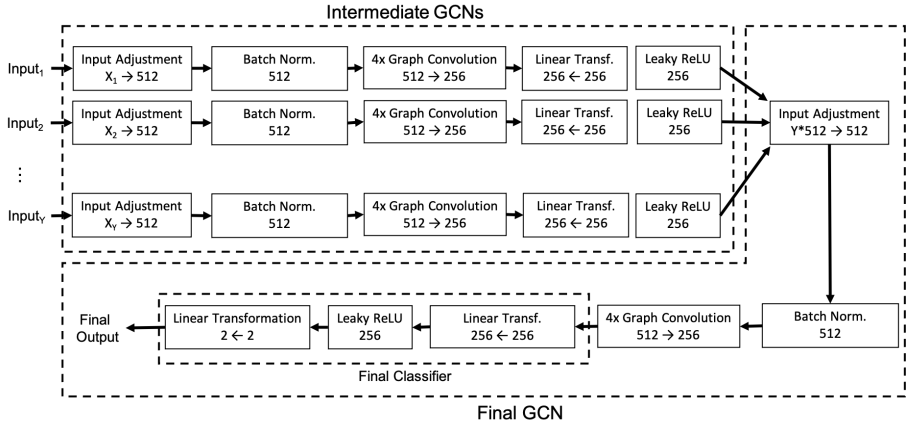


Figure 4.2: Overview of the ensemble GCN architecture.

Miscellaneous. As the KNN graph determines which nodes can potentially link and thus influences the final clustering, inputs for the GCN and KNN graph were separated allowing KNN graphs to be calculated on features different from the ones seen during training.

To further analyze created IPS, the user is given the option to have IPS plots created during runtime. There are two types of IPS plots created. The first type of plots focuses on analyzing the label composition of IPS graphs (especially of their one-hop neighborhoods), while the second type focuses on analyzing the embedding structure of IPS graphs. To avoid confusion, the former group of plots will be referred to as KNN graph plots, while the latter group will be referred to as feature embedding plots. KNN graph and feature embedding plots are created using the original features dataset of training and validation sequences as input. Moreover, plots are also created using the feature map returned after the fourth graph convolution layer within each training epoch. Note that the plotted IPS are equivalent to the ones seen during training and validation. For each input dataset plots are created for the tenth index and the index that is approximately half the number of total detections within said dataset. To allow the features to be plotted in a two-dimensional space, dimensionality of the data was reduced using t-distributed stochastic neighbor embedding (t-SNE) as proposed by van der Maaten and Hinton [59].[12] Using t-SNE the feature dataset is reduced to its two most prominent dimensions resulting in a x and y coordinate for each detection. Note that the KNN graph is still calculated using the original feature dataset and not the reduced feature dataset using t-SNE. Both type of plots resembles the IPS of the detection of interest (i.e., the pivot instance), which is colored green. A neighbor is colored grey if it does not share the same label as the pivot instance and is colored red if it does share the same label.

A KNN graph plot is divided into two subplots (see Figure 4.3). The left subplot focuses on analyzing the one-hop neighborhood of the IPS, while the right subplot analyzes the frame distance between the pivot instance and its neighbors. Within the left KNN graph subplot, a dashed line is drawn between the pivot instance and a neighbor if the neighbor is part of the one-hop neighborhood of the pivot instance. The edge is colored red if said neighbor shares the same label and is colored gray if their labels differ. Within the right subplot, nodes are color graded according to their frame distance to the pivot instance with a color grading scale provided next to the plot.

Contrarily, within a feature embedding plot all edges of all nodes are plotted, i.e. also those contained in the IPS of nodes other than the pivot instance. Additionally, same as for the KNN graphs, the frame distance between the pivot instance

[12] Used the sklearn implementation of t-SNE.

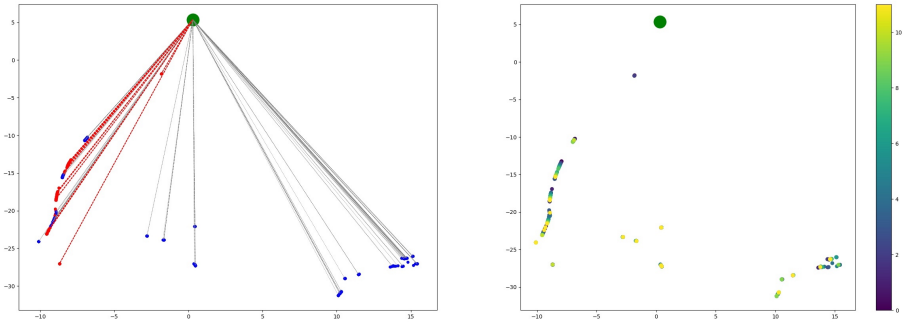


Figure 4.3: Sample IPS used during training computed using pre-filtered frame-distance KNN graph with the spatial DPM dataset used as input.

and all other instances is also plotted in a separate plot.

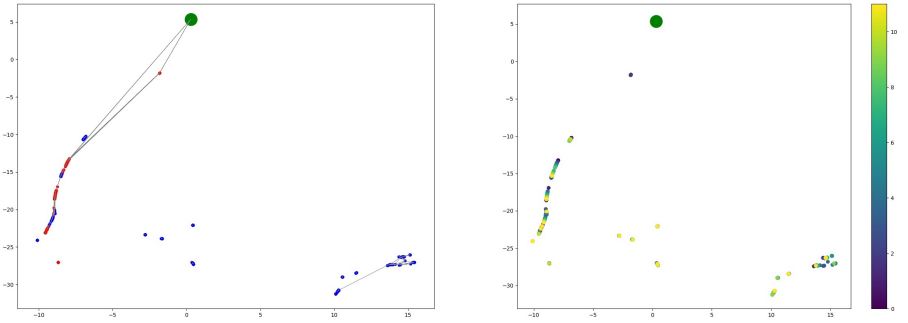


Figure 4.4: Same IPS graph as shown in Figure 4.3, but now illustrating feature embedding information.

Note that due to changes in the implementation not all plots (KNN graph and feature embedding plots) have the pivot instance marked green within the second subplot. Within said subplots the color grading resembles the frame information of the node and not its frame distance to the pivot instance.

To further analyze networks across multiple runs (see Chapter 5.2), training losses for each epoch (batch-wise and average values) and validation losses (batch-wise and averages) of each validation sequence are saved during training and validation time. Loss information are used during the evaluation to analyze whether a network is overfitting or not. The created plots which are used to perform said

analysis are described in Section 4.4. Due to an error within the implementation the creation of some of the plots was not performed correctly. For those plots to be meaningful and convey the right information, training data needs to be divided into a training and validation set. Unfortunately, this was not done during experiments performed in this thesis. Nevertheless, Section 4.4 will describe the initial idea behind each plot.

4.3 Postprocessing

Last step in the implementation is to convert the final clustering, produced by one of the heuristics, into tracker hypothesis. To do so a MatLab script was used.[13] Input to the script is a text file which contains per detection its frame information, predicted label and bounding box coordinates (i.e. x and y coordinate, width and height). The script first filters out detections which are assigned the label -1, which consequently causes these detections to not be considered during evaluation. Next, the confidence per detection is obtained via the original detection file. This value expresses how certain the original detector was that the detection at hand is the class of interest, i.e., a pedestrian. Additionally, bounding boxes are converted into a four-coordinate format, i.e., into x_1, x_2, y_1 and y_2 coordinates. The remaining detections are then filtered to only include detections which exceed a certain detector confidence threshold t_d. Similarly, the resulting remaining clusters are filtered to only consider those clusters for evaluation, whose average detector confidence among all detections contained in said cluster, is above a confidence threshold t_c and consists of at least t_s detections.

Afterward, clusters are converted to tracks, with each cluster handled individually. First, it is checked whether detections appearing within the cluster are at least from t_s different frames. If not, the cluster is not considered during evaluation. Next, if multiple detections appear in the same frame, the script either merges them by using interpolation or only keeps the detection that appears first in the detection file. Interpolation is done by taking the mean value of the bounding box coordinates of all detections. At this point tracks might still have detections missing in intermediate frames causing the tracks to skip frames. If the gap is less or equal to t_g frames long, said gaps are filled with interpolated detections. If the gap is larger than t_g, the track is split into two separate tracks with the last detection before the gap being the end point within the initial track and the detection at the end of the gap being the starting point of a new track. This handling of gaps is based on the assumption that if a gap is more than t_g frames, the instance that is

[13] The script is an altered version of a script provided by Professor Margret Keuper (see *evaluation/postprocessing_script.m*).

to be tracked is likely to be not visible in said timeframe, e.g., by being covered by another object or temporally leaving the camera frame. This definition is in line with the *IDS* defintion provided by the MOTChallenge described in Chapter 3.2. The interpolated detections to fill gaps are produced by one-dimensional interpolating each of the bounding box coordinates (x_1, y_1, x_2 and y_2). To interpolate, the coordinate that is to be interpolated is used as y-axis and the frame information is used as x-axis. Finally, the postprocessed tracks are converted into the evaluation format specified by the MOTChallenge and saved into a text file.

4.4 Evaluation

Using the text file produced by the postprocessing script, one is now able to evaluate hypothesis as described in Chapter 3.2 and rate the GCNs tracking performance. To apply the evaluation metrics a Python-based implementation called *py-motmetrics*, which was implemented by Christoph Heindl, was used.[14] The implementation differs from the official MOT evaluation script since it does not report the average number of *false alarms per frame (FAF)* and *identity switch ratio (IDSR)*. Nevertheless, if wanted, the missing metrics can be calculated using the provided other metrics. For this thesis, said values were not calculated as it sufficed for analyzing the experiments to have the absolute values of said metrics, i.e., *false positives (FP)* and *identity switches (IDS)*. Log files produced by *py-motmetrics* include provide evaluation metrics which are not part of the official MOT evaluation metrics. Said metrics are *identity precision (IDP)*, *identity recall (IDR)*, *number of unique objects (GT)*, *partially tracked (PT)*, *IDt*, *IDa* and *IDm*. *IDP* and *IDR* are used to calculate the *identity F1 (IDF1) score* (see Chapter 3.2). *PT* represents the number of objects (i.e. persons) within the ground truth file which are *partially tracked*. An object is *partially tracked* if it is neither *mostly lost (ML)* nor *mostly tracked (MT)*, i.e. ground truth tracks of which less than 80% and at least 20% of all bounding boxes are assigned to any hypotheses. *GT* represents the number of ground truth tracks within a sequence. Heindl's implementation provides no explanation what the evaluation metrics *IDt*, *IDa* and *IDm* represent. These evaluation metrics can thus be ignored within the log files. The MOTChallenge and *py-motmetrics* use different abbreviations for the evaluation metrics. To avoid confusion when reading log files, Table B.1 within the appendix of this thesis provides an overview of said used abbreviations.

If only parts of sequences were used during validation, users are given the op-

[14] See `https://github.com/cheind/py-motmetrics`. Note that the PyPi package (`https://pypi.org/project/motmetrics/`) of *py-motmetrics* was used during this thesis and results might differ using the GitHub version of *py-motmetrics*.

tion to alter the validation process to use a modified version of the corresponding ground truth file. Within said modified ground truth file all detections with a label not appearing in the subset of the detection file used during validation were excluded during evaluation. This way the evaluation process focuses more on assessing how well the instances contained in the split were tracked and ignores evaluation of excluded ones. Besides evaluating experiments using the *py-motmetrics*, postprocessed outputs of the best performing setting were also submitted to the MOT website to obtain results on the test sequences. Ground truth files of the test sequences are not publicly available to avoid trackers overfitting on them. The final ranking within the MOT17 leaderboard thus can be used to compare the performance of the method proposed within this thesis to other state-of-the-art trackers.

As mentioned previously, losses saved during training and validation are used to create additional plots that can be used to analyze a network's tendency to overfit (see Chapter 5.2). The plots that are created during each experiment include:

1. Average and batch-wise training and validation losses across epochs.
2. Average validation loss of each sequence (x-axis) compared to the *MOTA* score of said sequence (y-axis).[15]
3. Average loss on the validation set of the training data (x-axis) compared against the *MOTA* score of each validation sequence (y-axis).
4. Average training loss (x-axis) compared against the *MOTA* score of each validation sequence (y-axis).
5. Average loss on the validation set of the training data sequence divided by the average training loss (x-axis) against the *MOTA* score of each validation sequence (y-axis).

[15]This includes the validation set of the training data.

Plot groups two to five were additionally created using *IDS* information instead of *MOTA* scores. Note that above mentioned average losses within plot groups two to five are calculated by averaging the loss of the last two batches. For the first group of plots, witnessing a low training loss (close to zero) combined with high validation losses could indicate that the model does not learn to generalize well across sequences and only learns to predict the training sequence. Contrarily, a late increase in training loss could indicate that the learning rate is set too high, indicating that the model does not converge to a local minimum or that the number of epochs should be decreased. As mentioned previously (see Chapter 3.1) every sequence has certain characteristics (e.g., camera angle, time of day, etc.). A strong correlation within the second group of plots paired with a low correlation within the third group of plots could indicate that a model overfits on the characteristics of the training data/ sequence, but not on the data itself. If, on the other hand, there is as well a strong correlation within the third group of plots, then one can say that the model generalizes well across sequences and does not overfit on said characteristics of the training sequence. Similarly, a strong correlation within the fourth group of plots would also indicate that the model does not overfit on the training data. Lastly, within the fifth group of plots the x-axis value captures the degree of overfitting as an increased gap between the average loss on the validation set of the training data and the average training loss could indicate that the model tends to only learn to predict the training data but does not generalize and is unable to predict unseen data. This measurement is set into relation with the computed *MOTA* (and *IDS*) score.

Due to errors within the implementation that were noticed too late to rerun all experiments, some plots are not correctly computed and thus convey not the message that they intended to. First error was to not calculate the validation loss of the training sequence on a separate, unseen part of the training data, but have it calculated on the whole sequence. This caused validation losses for training sequences to not be representative, as detections were seen during training. As of handing in this thesis this error still remains within the current experimental setup, but can be fixed by adjusting its workflow using already implemented functions. The second error was a confusion between x- and y-axis values within certain plots, which was fixed within the workflow, but still exists within some of the plots provided with this thesis.

Chapter 5

Experimental Evaluation

In the following chapter, the experimental evaluation performed during this thesis will be illustrated. This chapter is divided into six sections, each describing a different type of experiment. Section 5.1 will describe the baseline experiments which tested the performance of each dataset by itself as well as in combination with the spatial dataset using a base setting. Next, Section 5.2, will illustrate consistency checks performed on the baseline experiments to check their stability with varying as well as same seeds. During these experiments it was observed that GCNs trained using the spatial dataset showed high variance. Section 5.2 will discuss what caused the variance and how the problem was (partially) solved by switching to the frame-distance and pre-filtered frame-distance KNN graph calculation. Afterward, Section 5.3, will try to determine the best input setting, i.e., node features and KNN graph input, for experiments using the frame-distance and pre-filtered frame-distance KNN graph. Section 5.4 will analyze how GCNs performed using different kind of feeder modifications. Analyzed modifications include using absolute differences instead of signed differences, introducing a multiplicative factor by appending element-wise products between columns and normalizing distances between bounding boxes. Further, the chapter will also analyze the combination of different feeder modifications, which showed the most promise performance-wise. Section 5.5 will describe how results changed once primal feasible search heuristics as seen in [34] were used to obtain the final clustering instead of pseudo label propagation [67]. Lastly, Section 5.6 will determine the best setting and evaluate whether performance can be increased using a preprocessed version of the detection dataset as seen in [4]. Furthermore, results reported by submitting predictions to the MOTChallenge website will be illustrated as well as a final ranking compared to other state-of-the-art trackers will be given.

As the MOTChallenge limits participants in their number of submissions to avoid overfitting on the test data, only the best performing setting was submitted. To determine the best setting each experiment was trained using the MOT17-04 sequence and validated on all other training sequences, i.e., MOT17-02, MOT17-05, MOT17-09, MOT17-10, MOT17-11 and MOT17-13. For each experiment, results are reported with and without the train sequence included during validation. The former is written in brackets within each table, except for the consistency checks, where both evaluation settings are divided into two tables due to variance being reported in brackets. Evaluation metrics reported within this thesis are the amount of *mostly tracked (MT)* and *mostly lost (ML)* ground truth trajectories, the *multiple object tracking accuracy (MOTA)* score, *identity F1 (IDF1)* score and the amount of *identity switches (IDS)*, *false positives (FP)* and *false negatives (FN)* (see Chapter 3.2 for a detailed description how these metrics are defined). Within a table, the best scoring values are printed in bold font. Other evaluation metrics as well as a breakdown of the performance per sequence was omitted within this thesis unless needed and can be found in the log files provided with this thesis. It is to be noted that the proposed removal of singleton nodes as seen in [62] did not yield better results in any of the conducted experiments. Nevertheless, evaluation results using said method were (mostly) computed and can be found under *evaluation results (removed)* within the log files.

Dataset creation settings. As described in Chapter 4.1 the dataset creation process offers users options in how each dataset is created. Due to time limitations of this thesis only one setting was employed resulting in the same set of input datasets for each experiment. For the label dataset an upper IoU range of 0.7 for training and 0.5 for validation and testing was used. This means that detections which have at least an IoU of 0.7 (or 0.5 in case of validation/ testing) with a ground truth detection, which shares the same frame information, are considered to be pedestrian detections and assigned the label of the matched ground truth detection. For the lower IoU range 0.3 was employed during training, validation and testing. This means that detections which have no more than an IoU of 0.3 with any ground truth detection within the frame that they occur, are considered to be background detections. Within all experiments background detections were assigned a unified label, grouping them into one cluster. Detections which do not fall in any of the two categories are like background detections assigned a unified label, i.e. -1. This is done to make the network robust to bad input detections and learn to exclude those during prediction time. For the creation of the reidentification, person and classification dataset max pooling was used. Lastly, as mentioned in Chapter 4.4, validation of the training sequence was not done correctly. For evaluation results

reported on the training sequence to be of value, the training sequence would have needed to be split into a training and validation part. Not doing so causes validation results including the train sequence to score too high as the data that is to be predicted was already seen during training and data leakage took place. Not all overfitting plots created during evaluation (see Chapter 4.4) are therefore able to be analyzed properly as some values used to create the plot assume the previously mentioned split. Note that the implementation provided allows for such a setup and splits can be defined during the label creation process. Users would only need to adjust the experimental pipeline, which was skipped over during this thesis as there was not enough time left when the error was noticed.

Hyperparameter settings. Hyperparameter tuning was not performed during this thesis due to being limited in computational resources. To make the results of performed experiments comparable only one setting of hyperparameters was used. This setting is identical to the hyperparameter setting employed by Wang et al. [62]. Table 5.1 summarizes both hyperparameter settings. For training a two-hop IPS was used with the one-hop neighborhood (k_1) being of size 200 and the two-hop neighborhood (k_2) being of size ten. Parameter u was set to ten as well. Networks were trained for four epochs with weight decay (wd) being set to 0.0004, momentum (m) set to 0.9 and learning rate (lr) set to 0.01. Note that the learning rate is decreased by factor ten after each epoch as seen in [62]. For validation and testing a smaller two-hop IPS was used with the one-hop neighborhood being of size 20 and the two-hop neighborhood being of size five. Parameter u was set to five as well. It is to be noted that parameters are named differently within the implementation namely *k_at_hop* being a list of k_i (i being the index within the list) defining the number of nodes within each h-hop neighborhood and *active_connection* being parameter u. As only one sequence was used for training (namely MOT17-04), there is no difference between a combined or sequential training setup (see Chapter 4.2). Nevertheless, for consistency reasons, all experiments were trained using a combined setup.

Hyperparameter settings							
	k_1	k_2	u	wd	m	lr	seed
Training	200	10	10	0.0004	0.9	0.01	1
Validation/ Testing	20	5	5	-	-	-	1

Table 5.1: Overview of the employed hyperparameter settings during training, validation and testing.

Postprocessing settings. The input parameters for the postprocessing script were - like the hyperparameters - not tuned and the initial setting provided with the script was used throughout all experiments (see Table 5.18).[16] The four threshold parameters, namely minimum detection confidence t_d, minimum average detection confidence within a cluster t_c, minimum cluster size (i.e., track length) t_s and maximum gap distance between detections in a track t_g need to be set individually for each sequence. In addition to the thresholds the script also needs to be given a boolean flag for each sequence indicating whether tracks for said sequence are to be smoothed. For smoothing the implementation provided by Jonas Lundgren was used.[17] Withing this thesis t_g was being set to be 30 for all sequences, i.e., gaps within tracks of up to 30 frames are interpolated. For t_c and t_d values were set differently for detections provided by the DPM detector as its detections on average are less confident. For the DPM detector t_d was set to be -0.1 and t_c to be 0.2. For the FRCNN and SDP detector t_d was set to 0 and t_c was set to 0.5. For all detectors, the minimum track length t_s was set to 5.

Postprocessing settings					
Detector	t_d	t_c	t_s	t_g	smoothing
DPM	-0.1	0.2	5	30	no
FRCNN/ SDP	0	0.5	5	30	no

Table 5.2: Overview of the employed postprocessing settings for each detector.

Caveats. In the following a list of caveats will be provided which need to be considered when reading results of conducted experiments. This thesis was conducted under a limited timeframe using a shared pool of resources provided by the University of Mannheim. Even though all experiments used the same amount of resources, runtime performance of the experiments was sometimes skewed, as the server was occupied by other students conducting their experiments. Due to the limited amount of experiments the shared resource pool allowed to conduct, only experiments mentioned in the consistency check section were conducted over multiple runs. This means that only for these experiments' variance was able to be estimated and reported. Consequently, all other experiments (especially the ones using spatial data) would show some sort of variance from the results reported within this thesis when being conducted again. This makes results only to give a tendency if

[16] The setting was proposed by Professor Margret Keuper and was optimized based on results seen in [33].

[17] See *splinefit* function within the *postprocessing_script.m* file. The function was provided by Professor Margret Keuper.

the employed methods work, but not their final result over multiple runs. Furthermore, as mentioned above, no hyperparameter tuning was performed. Settings for hyperparameters were used as proposed by Wang et al. [62], meaning that the possibility exists that tuning them could lead to better results. Furthermore, some plots created to analyze a network's tendency to overfit were not correctly calculated. As the correct calculation would have required to split the training sequence into a training and validation part and x- and y-axis values were confused within certain plots, the plot groups comparing the *MOTA* (*IDS*) score to average losses do not convey the correct message and should not be considered for analysis of the experiments performed during this thesis. Nevertheless, plotting was corrected within the evaluation script and would produce correct results if one employs the above mentioned split. As of handing in this thesis, the calculation of the normalized bounding box distances and frame-pairwise element-wise products only works in the combined training setup and when the spatial feature dataset is occupying the first indices within a concatenated feature array. This means that if multiple feature datasets are to be combined, the spatial feature dataset needs to be the first within the order of concatenation. It is further not recommended to calculate the pairwise element-wise products using any other feature dataset than the spatial dataset since it will blow up the feature space significantly and most likely exceed memory of most working machines. Even though it was explained in Chapter 4.2, an ensemble GCN setup as seen in Figure 4.2 did not show significant results and ended up predicting each detection as a singleton cluster. The setup, which was initially tried consisted of three separate GCNs, each trained using either the reidentification, person or spatial dataset. For each network the feature representation after the second classifier layer of each detection was extracted, concatenated to be one dataset and used as input to a final GCN. Even when changing the KNN graph calculation of the final GCN to be based upon the spatial dataset, the GCN did not provide any useful results. As of handing in this thesis, the implementation does not contain a working ensemble setup script as the experimental workflow changed and the code which was used to obtain the above-mentioned results was not functional anymore. Nevertheless, this setup is expected to hold some value and will be discussed in more detail within the future research section (see Chapter 6.2) of this thesis.

5.1 Baseline experiments

The baseline experiments focused on testing each of the created feature datasets performance. Table 5.3 summarizes the baseline experiments results. Each experiment was performed using the above-mentioned dataset creation, hyperparameter and postprocessing settings. The KNN graph within each experiment was calcu-

lated using the same features that were used to train the GCN. The person, reidentification (ReID) and classification (Clf) dataset are all appearance-based features, i.e., features which are calculated solely based on how a bounding box looks like. It was therefore also evaluated how each dataset performed when being combined with the spatial (Spa) feature dataset (see experiments 5, 6 and 7 in Table 5.3). As the spatial dataset encodes where bounding boxes are located within a frame as well as when they occur (frame information), it was expected to deliver a more complete view of each sequence and hence improve a tracker's ability to track objects.

One can see that among all datasets the spatial feature dataset performed best. Additionally, the spatial and the reidentification feature dataset performed significantly better than the classification and person feature dataset. As the underlying networks of the latter datasets were trained solely with the purpose to classify detections as persons (or other classes in the case of the classification dataset) and not as different instances of persons, they are less able to depict intra-class variations between different pedestrian detections. Thus, feature embeddings provided by both networks are likely to be similar for different detections, which makes it difficult for a GCN to learn to differentiate different detections and account them to the correct entity. Therefore, both the classification and person dataset were not further investigated during experiments. Nevertheless, one could investigate whether using a different objective for both datasets (e.g., predicting whether detection is person or background) could hold value when later being combined with the predictions of MOT. This proposition will be further illustrated in Chapter 6.2. Experiments 5, 6 and 7 show that combining different datasets and thus combining different views of the data holds value as results improve compared to using the datasets individually.

5.2 Consistency checks

During the consistency check experiments both the spatial and reidentification dataset were checked for same seed as well as varying seed performance across multiple runs. Ideally, if the random seed is set at the same value, the network will perform identically throughout runs. On the contrary, if the seed is changed throughout runs, the network should not vary significantly in its performance. Table 5.4 and 5.5 are summarizing results over ten runs for the spatial and reidentification dataset with varying (see Table 5.4) and same random seed (see Table 5.5). The tables are split into two sub tables with the former illustrating results validating using validation sequences and the latter using training and validation sequences. Each evaluation metric is provided with variance witnessed across runs. Experi-

Baseline Experiments								
ID	Dataset	MT	ML	FN	FP	IDS	IDF1	MOTA
1	Clf	10 (52)	1036 (1121)	161,524 (233,030)	**8,908** (**10,473**)	**1,202** (**1,654**)	16.0% (32.3%)	11.6% (27.2%)
2	Spa	74 (120)	742 (829)	128,813 (198,570)	13,617 (19,274)	2,000 (2,504)	25.9% (34.9%)	25.6% (34.6%)
3	ReID	37 (98)	832 (908)	142,303 (204,599)	9,309 (12,522)	1,671 (2,006)	22.1% (37.8%)	21.1% (35.0%)
4	Person	12 (24)	955 (1105)	157,807 (264,594)	30,092 (56,146)	4,622 (6,100)	11.4% (12.8%)	0.9% (3.0%)
5	Clf, Spa	78 (**134**)	**735** (**814**)	128,334 (**191,385**)	12,283 (16,399)	1,598 (1,901)	**26.6%** (**38.3%**)	26.8% (**37.8%**)
6	ReID, Spa	**80** (**134**)	764 (845)	**128,316** (192,671)	11,777 (15,336)	1,671 (1,964)	26.2% (37.5%)	**27.0%** (37.7%)
7	Person, Spa	78 (125)	745 (831)	129,596 (199,826)	13,025 (18,785)	1,859 (2,272)	25.7% (35.1%)	25.6% (34.4%)

Table 5.3: Evaluation results of baseline experiments validated using validation sequences (validation and training sequences) as input.

ment 8 and 9 show that the spatial dataset using the normal (*norm*) KNN graph calculation does not produce consistent results even with the random seed being fixed at 1. However, no variance was observed with fixed seeds across runs for experiments using the reidentification dataset.

Consistency check experiments (validation data)								
ID	Input (KNN)	MT	ML	FN	FP	IDS	IDF1	MOTA
8	Spa (norm)	72.7 (±3.38)	747.3 (±8.51)	129,314.5 (±1,154.96)	13,544.4 (±324.5)	1,906.3 (±48.45)	26.03% (±0.44%)	25.48% (±0.6%)
9	ReID (norm)	30.9 (±5.05)	845.8 (±15.47)	142,815.3 (±836.58)	9,580.2 (±479.18)	1,709.1 (±46.8)	21.41 (±0.54%)	20.66% (±0.52%)
10	Spa (f-dist)	**137** (±10.23)	**655.6** (±12.31)	**124,415** (±1,841.57)	4,521 (±143.00)	4,391.6 (±153.67)	26.5 (±0.81%)	31.34 (±0.91%)
11	Spa (pre-fil)	92.9 (±3.24)	726.9 (±6.98)	125,308.8 (±403.52)	**4,422.5** (±151.38)	**1,611.4** (±36.52)	**29.86%** (±0.29%)	**32.37%** (±0.17%)

Consistency check experiments (train & validation data results)								
ID	Input (KNN)	MT	ML	FN	FP	IDS	IDF1	MOTA
8	Spa (norm)	118.2 (±3.34)	834 (±7.94)	199,650.4 (±1,348.73)	19,046.9 (±533.73)	2,346.6 (±58.33)	35.01 (±0.41%)	34.4% (±0.38%)
9	ReID (norm)	92 (±5.71)	920.2 (±15)	205,116.8 (±1,176.78)	12,621.3 (±584.46)	2,062.3 (±52.63)	37.25 (±0.43%)	34.75% (±0.41%)
10	Spa (f-dist)	**196.5** (±10.3)	**735** (±14.04)	**189,708.9** (±2,345.98)	**6,794** (±166.05)	7,845.7 (±334.15)	29.49% (±0.74%)	39.34% (±0.71%)
11	Spa (pre-fil)	149.9 (±2.88)	808.7 (±7.21)	193,827 (±677.78)	7,175.8 (±332.37)	**2,003.6** (±48.82)	**37.84%** (±0.34%)	**39.75%** (±0.24%)

Table 5.4: Evaluation results of consistency check experiments (and observed variance across runs) using varying seed across runs.

Consistency check experiments (validation data results)								
ID	Input (KNN)	MT	ML	FN	FP	IDS	IDF1	MOTA
12	Spa (normal)	75 (±3.29)	740.8 (±7.81)	128,443.9 (±464.86)	13,459.2 (±494.51)	1,938.1 (±49.8)	26.34% (±0.21%)	25.94% (±0.28%)
13	ReID (normal)	37 (±0)	832 (±0)	142,303 (±0)	9,309 (±0)	1,671 (±0)	22.1% (±0%)	21.1% (±0%)
14	Spa (f-dist)	**122.3** (±3.49)	**666.7** (±6.03)	126,472 (±410.37)	4,599.8 (±76.3)	4,720.5 (±94.09)	25.13% (±0.43%)	30.08% (±0.26%)
15	Spa (pre-fil)	93.2 (±3.12)	725.3 (±2.87)	**124,978** (± 175.35)	**4,480.5** (± 135.46)	**1564.1** (± 22.36)	**30.26%** (±0.11%)	**32.51%** (± 0.09%)

Consistency check experiments (train & validation data)								
ID	Input (KNN)	MT	ML	FN	FP	IDS	IDF1	MOTA
12	Spa (norm)	119.3 (±3.9)	827.5 (±7.68)	198,128.8 (±691.2)	19,460.5 (±633.21)	2,421.2 (±52.89)	35.08% (±0.4%)	34.67% (±0.3%)
13	ReID (norm)	98 (±0)	908 (±0)	204,599 (±0)	12,522 (±0)	2,006 (±0)	37.8% (±0%)	35% (±0%)
14	Spa (f-dist)	**181.6** (±3.8)	**748.3** (±5.81)	**192,934** (±410.94)	**6,842.9** (±81.65)	8,290.7 (±90.54)	28.25% (± 0.23%)	38.24% (±0.15%)
15	Spa (pre-fil)	149.2 (±3.49)	806.1 (±2.21)	193,717.4 (±563.08)	7,522.8 (±107.8%)	**1,970** (±25.25)	**37.82** (±0.17%)	**39.68** (±0.17%)

Table 5.5: Evaluation results of consistency check experiments (and observed variance across runs) using same seed across runs.

When visualizing results of experiments 8 and 9 it was noted that the network being trained using spatial data produced errors, which were not expected to be made given that the network had information about the location of the detections. One can find videos visualizing results within the files provided with this thesis. Said errors included labels constantly changing for a given instance even though it was moving at a steady pace in the same direction without being covered by other instances and labels switching between instances that are walking close by each other without overlapping. By contrast, when looking at videos produced by the GCN trained using reidentification features, said errors were less prominent and/ or mostly absent. Unlike for the network being trained using spatial features, identity switches mostly occurred when persons were partly covered and then uncovered by objects or reappeared throughout the video. To solve the inconsistent behavior of the spatial GCN, two additional types of KNN graphs were implemented namely the frame-distance (*f-dist*) and pre-filtered (*pre-fil*) frame-distance KNN graph. As these graphs focus on constructing neighborhood graphs by favoring detections which are proximate to the pivot instance timewise, it was hoped to improve the quality of IPS and consequently help the GCN learn to avoid switching labels in the above-mentioned cases. Experiments 10 and 11 in Table 5.5 show that tracking performance increased using both types of KNNs, but in the case of the frame-distance KNN graph *IDS* significantly increased. Furthermore, variance was mostly minimized, but not eliminated. Comparing videos produced by the GCN using the normal KNN graph with videos produced by the GCNs using the modi-

fied KNN graphs, one can see that trackers still produce the same kind of mistakes as mentioned above.

To better understand the training process and identify reasons why the network being trained on spatial features made said mistakes, the training process was modified to produce plots of sample IPS. The created plots focused on analyzing the IPS embeddings (feature embedding plots) and label composition (KNN graph plots). Each plot consists of the detection used as pivot instance (marked green) and its neighbors within the IPS. A neighbor which is colored red indicates that it shares the same label with the pivot instance, while a neighbor which is colored gray indicates that it shares a different label. Within the feature embedding plots a line between instances indicates that there exists an edge between the two instances in at least one IPS. Contrarily, for the KNN graph plots, a dashed line between the pivot and another node indicates that said node is part of the one-hop neighbors of the pivot instance. Like the color of the nodes, the line is colored red if the instance shares the same label with the pivot instance. Table 5.6 and 5.7 analyze the label distribution of nodes (same or different label) within each IPS for the first run of experiment 8, 9, 10 and 11 using the two sample indices as pivot instances. The two sample indices are chosen to be a detection which occurs early within the training sequence (index 10) and one detection which occurs roughly halfway through the sequence (obtained by dividing the dataset length by two). This choice of indices only allows to compare results per detector, but not across detectors, as detections and thus indices differ across detectors. Within the appendix one can find KNN graph and feature embedding plots created during training of experiments 12 to 15 (see Figures B.1 to B.16).[18]

In the following, the label composition of the IPS (see Table 5.6 and 5.7 will be analyzed. First, one can see that IPS constructed using reidentification features are mostly producing neighborhoods which consist of instances sharing the same label as the pivot instance. This can likely to be accredited to reidentification features being highly discriminant and the network used to produce the feature vectors [72] sharing an objective similar to MOT. By contrast, IPS constructed using the frame-distance KNN graph are more skewed towards containing instances which do not share the same label as the pivot instance. Similarly, as seen in Table 5.7, the two new types of KNN graphs produce IPS for the FRCNN which contain on average detections in future frames. These characteristics of the IPS might hint towards the effectiveness of the proposed methods. Both methods limit the search space of potential neighbors by the point in time they occur, disregarding their appearance, causing constructed KNN graphs to be more likely to group together detections which are proximate timewise. IPS produced using reidentification features and

[18] A high resultion version of the plots can be found within the files provided with this thesis.

spatial features (using a pre-filtered frame-distance KNN graph) are similar when using SDP and FRCNN detections. This may be attributed to the detection quality of the DPM detector being worse than detection quality of the other detectors, making it more difficult the reidentification network to identify instances solely based on their appearance. Furthermore, the average frame-distance of the FRCNN detector is the highest of the detectors as it provides the least amount of detections.

When analyzing the feature embedding plots of the above described IPS plots one can see that IPS graphs which were constructed using the frame-distance and pre-filtered frame-distance graph tend to contain more disjoint components than the IPS produced using the normal KNN graph calculation. As performance improved significantly in the former case, one might be able to attribute this to IPS graphs containing more learnable traits with said disjoint components. GCNs being trained on spatial features might need additional components to learn to classify detections correctly.

DPM Detector									
Dataset	Nodes	Same label	Different label	Avg. f-dist	Median f-dist	One-hop same label	One-hop diff label	One-hop avg. f-dist	One-hop median f-dist
Spa	247	90	157	2.45	2	82	118	2.485	2.5
ReID	425	121	304	4.59	5	70	130	4.44	4
Spa (f-dist)	284	6	278	2.9	3	4	196	1.93	2
Spa (pre-fil)	485	144	341	5.33	5	63	137	4.97	5
FRCNN Detector									
Dataset	Nodes	Same label	Different label	Avg. f-dist	Median f-dist	One-hop same label	One-hop diff label	One-hop avg. f-dist	One-hop median f-dist
Spa	218	185	33	159.56	143	173	27	157.395	143
ReID	487	487	0	141.26	100	200	0	133.25	121.5
Spa (f-dist)	384	12	372	186.17	143	6	194	163.205	48
Spa (pre-fil)	302	302	0	196.84	167	200	0	199.78	167
SDP Detector									
Dataset	Nodes	Same label	Different label	Avg. f-dist	Median f-dist	One-hop same label	One-hop diff label	One-hop avg. f-dist	One-hop median f-dist
Spa	244	170	74	2.9	3	163	37	2.935	3
ReID	224	224	0	2.65	3	200	0	2.66	3
Spa (f-dist)	303	8	295	3.79	4	5	195	2.36	2
Spa (pre-fil)	218	207	11	2.56	2.5	199	1	2.57	2.5

Table 5.6: Analysis of the constructed IPS during training for the detection with index 10 using different datasets and KNN graphs.

All in all, it remains an open question how the IPS graphs based on spatial features can be further improved and how to reduce variance across runs using the same seed. The increased performance witnessed in experiments 9 and 10 show that the two new created KNN graph types hold value and that the network is highly dependent on how IPS graphs are built. Analysis of the composition of the KNN graphs show that there are significant differences between the IPS graphs constructed using different types of input data and KNN graphs. This also shows that the different datasets might need different types of IPS being constructed as they are conveying different information and learning to separate objects using different approaches. Therefore, one can conclude that there is still room for improvement

DPM Detector									
Dataset	Nodes	Same label	Different label	Avg. f-dist	Median f-dist	One-hop same label	One-hop diff label	One-hop avg. f-dist	One-hop median f-dist
Spa	319	124	195	−0.69	−1	90	110	−0.86	−1
ReID	533	222	311	−3.08	−3	91	109	−2.96	−3
Spa (f-dist)	350	117	232	−0.33	0	64	136	−0.39	0
Spa (pre-fil)	547	248	299	−1.9	−2	100	100	−2.15	−2
FRCNN Detector									
Dataset	Nodes	Same label	Different label	Avg. f-dist	Median f-dist	One-hop same label	One-hop diff label	One-hop avg. f-dist	One-hop median f-dist
Spa	227	103	124	−449.81	−459	93	107	−454.9	−459
ReID	230	228	2	−452.06	−459	200	0	−456.39	−459
Spa (f-dist)	385	16	368	485.95	442	8	192	434.11	442
Spa (pre-fil)	283	137	146	131.32	95	132	68	133.15	95
SDP Detector									
Dataset	Nodes	Same label	Different label	Avg. f-dist	Median f-dist	One-hop same label	One-hop diff label	One-hop avg. f-dist	One-hop median f-dist
Spa	226	118	108	1.67	2	106	94	1.71	2
ReID	377	220	157	4.85	5	170	30	4.71	5
Spa (f-dist)	409	11	398	2.31	4	5	195	−0.38	2
Spa (pre-fil)	372	118	254	4.78	5	103	97	4.725	5

Table 5.7: Analysis of the constructed IPS during training for the detection with index roughly half the number of total detections using different datasets and KNN graphs.

finding the best composition of neighboring nodes which are to put in the IPS.

5.3 KNN graph experiments

The next type of experiments that were performed centered around experimenting with both inputs to the KNN graph calculation as well as the GCN. As mentioned above, inputs to the KNN graph calculation were separated from the features that were used to train the GCN. This allowed for multiple input combinations. The following experiments focus on determining the best input combination for the frame-distance and pre-filtered frame distance graph.

For the frame-distance KNN graph the input variation of computing the KNN using both spatial and reidentification features and using spatial features as node features produced the best results. Nevertheless, results end up not differing too much from only using spatial features to compute the KNN (experiment 16) nor from using both feature dataset as input for KNN graph calculation and as node features. As far as runtime is concerned, experiment 16 takes the least amount of time. Note that the current way of computing the frame-distance KNN graph takes significantly more time the more feature dimensions it has to process.

On the one hand, tracking performance of the reidentification dataset improves when using the frame-distance KNN graph to compute nearest neighbors for each instance (see experiment 17 and 21). On the other hand, one can also see a significant decrease in performance when using only reidentification features to compute the KNN graph (see experiments 18 and 23). The difference between the exper-

iments is that in experiment 18 and 23 spatial features are included in the node features of IPS. This hints towards the spatial feature dataset having a large impact on the learning process of the GCN and, when it is present in the node features, leads the GCN to try to learn to differentiate nodes mostly by their spatial features, disregarding reidentification features and thus appearance. If this behavior is to be true, it is likely more difficult for a GCN to learn to differentiate nodes as the KNN graph grouped them by their appearance and not by their spatial relationship. Similarly, one can see an increase in *IDS* when only reidentification features are used as node features, which can be caused by feature vectors differentiating too much once an instance changes its appearance (e.g., due to occlusion or movement in other directions). This in turn could lead to GCNs predicting said nodes as singleton clusters causing tracks to break and *identity switches* to occur.

Frame-Distance Experiments									
ID	Input	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
16	Spa	Spa	94 (140)	772 (865)	135,215 (209,085)	2,868 (4,244)	1,903 (3,367)	25.1% (28.3%)	27.9% (35.7%)
17	ReID	ReID	84 (129)	863 (959)	139,285 (213,979)	1,718 (**2,083**)	2,486 (3,614)	20.5% (27.4%)	26.1% (34.8%)
18	Spa	ReID	60 (100)	925 (1,029)	151,774 (230,064)	1,915 (2,728)	**1,833** (**2,863**)	18.5% (26.8%)	19.9% (30.1%)
19	ReID	Spa	73 (119)	825 (916)	137,039 (210,249)	1,706 (2,387)	2,526 (3,656)	21.5% (28.6%)	27.3% (35.8%)
20	Spa	Spa, ReID	**96** (141)	**760** (**851**)	133,966 (**207,592**)	2,866 (4,309)	1,910 (3,358)	**25.6%** (28.9%)	28.6% (**36.1%**)
21	ReID	Spa, ReID	73 (118)	832 (925)	137,237 (210,996)	**1,685** (2,370)	2,491 (3,586)	21.3% (28.5%)	27.2% (35.6%)
22	Spa, ReID	Spa	91 (142)	809 (904)	135,655 (208,712)	2,904 (3,984)	2,062 (3,337)	24.0% (28.6%)	27.6% (35.9%)
23	Spa, ReID	ReID	53 (102)	932 (1,028)	151,832 (226,425)	2,056 (2,942)	1,980 (2,993)	17.6% (27.2%)	19.7% (31.0%)
24	Spa, ReID	Spa, ReID	92 (**143**)	785 (884)	**134,629** (209,405)	2,834 (3,852)	2,096 (3,240)	24.6% (**29.7%**)	**28.1%** (35.7%)

Table 5.8: Evaluation results of experiments using the frame-distance KNN graph.

For experiments centered around the pre-filtered frame-distance KNN graph the reidentification dataset was used as the filter dataset throughout all experiments. As mentioned in Chapter 4.1 the pre-filtered frame-distance KNN graph can be seen as a variant of the frame-distance KNN graph. The filter dataset is used to calculate a large 'normal' KNN graph (e.g. with 500 nearest-neighbors per instance). The pre-filtered selection for each detection is then used to determine the original intended number of nearest neighbors using the frame-distance KNN graph way of calculating neighbors. In total there were nine different input scenarios tested (see Table 5.9). Most notable within this set of experiments is

the drop-off in tracking performance when only reidentification features are used as node features. This hints towards the fact that instances which are grouped together by the pre-filtered frame-distance KNN graph need spatial information in order to be discriminative. Other than these experiments, all other input settings performed better than compared to using the frame-distance KNN graph.

Pre-Filtered Frame-Distance Experiments									
ID	Input	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
25	Spa	Spa	96 (151)	**727** (**807**)	124,823 (**193,984**)	4,675 (7,471)	1,594 (2,007)	29.8% (37.3%)	**32.5%** (39.6%)
26	ReID	ReID	5 (63)	1,030 (1,106)	162,468 (227,233)	2,954 (**4,248**)	2,453 (2,975)	10.5% (28.7%)	13.6% (30.4%)
27	Spa	ReID	93 (145)	753 (836)	125,826 (194,736)	4,525 (7,526)	1,658 (2,197)	29.0% (36.2%)	32.0% (39.3%)
28	ReID	Spa	5 (68)	1,027 (1,100)	162,075 (226,036)	**2,788** (4,283)	2,333 (2,896)	11.1% (28.4%)	13.9% (30.8%)
29	Spa	Spa, ReID	101 (155)	733 (816)	**124,753** (194,208)	4,330 (7,272)	1,565 (1,963)	**30.5%** (**37.9%**)	32.7% (39.6%)
30	ReID	Spa, ReID	4 (68)	1,015 (1,089)	162,007 (226,087)	2,979 (4,412)	2,341 (2,908)	11.2% (28.9%)	13.8% (30.7%)
31	Spa, ReID	Spa	101 (158)	752 (832)	127,361 (194,660)	4,060 (6,351)	**1,394** (**1,906**)	29.9% (37.6%)	31.6% (39.8%)
32	Spa, ReID	ReID	**118** (**173**)	757 (833)	125,989 (193,606)	4,344 (6,894)	1,592 (2,202)	30.0% (36.0%)	32.1% (39.8%)
33	Spa, ReID	Spa, ReID	98 (150)	752 (829)	126,854 (193,112)	4,224 (6,724)	1,504 (1,994)	29.8% (37.3%)	31.7% (**40.1%**)

Table 5.9: Evaluation results of experiments using the pre-filtered frame-distance KNN graph.

All in all, one can conclude that there is no significant benefit in adding additional features to both the KNN graph construction and node features. As one has to consider there to be variance when using spatial data, the difference in performance between experiments 16 and 20 and 25 and 29 cannot be considered to be significant. This can for example be seen when comparing results of experiment 16 to results witnessed in experiment 14, which employs the same setting as experiment over multiple runs.

Even though tracking performance for both spatial and reidentification dataset improved when using the frame-distance KNN graph, the spatial dataset still outperforms the reidentification dataset. Nevertheless, keeping in mind the different results when visualizing tracker's results for both datasets, each dataset holds different kinds of information which is to be combined in some way. One can also see that GCNs using the two new types of KNN graphs are sensitive in performance depending on the data input they are given. This can be seen in the significant drop-offs in performance in experiments 18, 23, 26, 28 and 30. Keeping in mind

the difference between IPS graphs constructed by the spatial and reidentification dataset (see Section 5.2), it becomes apparent, that the two feature datasets are learning to differentiate between objects using different approaches and are learning different parts of MOT. Combining both perspective of the problem could thus end up in a more complete view of the problem and improve results.

5.4 Feeder experiments

As mentioned above, trying out different input variations did not provide a clear best setup. In the following, datasets will be therefore used separately, evaluating their sole performance. This in turn makes results still comparable to baseline experiments (see Table 5.3). In the following, changes to the data feeder will be evaluated. Each modification is explained in greater detail in Chapter 4 but will be shortly summarized again. These extensions are based on the original data feeder found in the implementation of Wang et al. [62].

The first group of experiments investigated the effect of using absolute instead of signed differences when calculating the normalized node features. Table 5.10 shows results of applying said feeder variation within the settings employed in experiment 2, 3, 16, and 25. One can see that only the setting using spatial features and the frame-distance KNN graph significantly improves in tracking performance. Further, the setting using reidentification features and normal KNN graph calculation slighly improves.

The second group of experiments investigates whether introducing a multiplicative factor to the GCN could improve its performance. This was achieved by calculating element-wise products. In total two variants of element-wise products were calculated:

1. The frame-pairwise approach creates element-wise products between the frame distance column and all other columns.

2. The pairwise approach creates element-wise products of all column pairs.

Table 5.11 shows results of the frame-pairwise approach for each type of KNN graph compared to their baseline performance. Note that the additional features do not influence the KNN graph calculation as they are appended within the feeder after the graph has already been calculated. When comparing the baseline performance compared with the performance having the additional columns appended, one can see a clear improvement for the experiments using the normal and frame-distance KNN graph. Both *MOTA* and *IDF1* score increased in both experiments, with more instances being *mostly tracked (MT)*. Only the experiment using the

Baseline Comparison									
ID	Input	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
2	Spa	normal	74 (120)	742 (829)	128,813 (198,570)	13,617 (19,274)	2,000 (2,504)	25.9% (34.9%)	25.6% (34.6%)
16	Spa	f-dist	94 (140)	772 (865)	135,215 (209,085)	**2,868** (**4,244**)	1,903 (3,367)	25.1% (28.3%)	27.9% (35.7%)
25	Spa	pre-fil	96 (151)	727 (807)	124,823 (193,984)	4,675 (7,471)	1,594 (2,007)	29.8% (37.3%)	32.5% (39.6%)
3	ReID	normal	37 (98)	832 (908)	142,303 (204,599)	9,309 (12,522)	1,671 (2,006)	22.1% (37.8%)	21.1% (35.0%)

Absolute Distances Experiments									
ID	Input	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
34	Spa	normal	76 (133)	749 (829)	127,771 (191,226)	17,875 (27,430)	2,213 (2,897)	24.2% (32.5%)	23.9% (34.2%)
35	Spa	f-dist	**132** (**196**)	**662** (**738**)	**122,386** (**185,348**)	3,198 (4,607)	2,317 (3,199)	**29.2%** (35.1%)	**34.1%** (**42.7%**)
36	Spa	pre-fil	79 (129)	762 (850)	128,814 (200,035)	16,160 (27,176)	1,865 (2,359)	26.0% (33.7%)	24.4% (31.9%)
37	ReID	normal	44 (97)	862 (940)	143,290 (208,906)	6,591 (8,963)	**1,405** (**1,848**)	22.9% (**36.1%**)	22.1% (34.8%)

Table 5.10: Evaluation results of experiments using absolute differences within the data feeder.

pre-filtered frame-distance KNN graph did not improve with results dropping significantly for *MOTA* and *IDF1* score and the tracker only being able to *mostly track (MT)* roughly a third of the instances that it was able to track originally.

Table 5.12 shows results when using pairwise element-wise products. As the pairwise calculation causes $\binom{n}{2}$ additional columns to be appended, with n being the number of feature columns, using the reidentification dataset would result in over two million additional columns being appended. Thus, experiments within Table 5.12 are evaluated using only the spatial dataset as input. As with the frame-pairwise element-wise products, the additional pairwise element-wise products do not influence the KNN graph calculation as they are appended within the feeder after the graph has already been calculated. In total three setups are analyzed each using a different type of KNN graph. When comparing the results to the baseline experiments, one can see that both the setup using the normal KNN as well as the frame-distance KNN profit from using pairwise element-wise products. Both *MOTA* and *IDF1* score increase within both experiments with more instances being *mostly tracked (MT)*. The experiment using the pre-filtered KNN graph did not end up improving performance with pairwise element-wise products being appended. One can see that the *MOTA* score on the validation sequences was lower than without the appended columns as well as less instances were *mostly tracked (MT)*.

Baseline Comparison								
ID	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
2	normal	74 (120)	742 (829)	128,813 (198,570)	13,617 (19,274)	2,000 (2,504)	25.9% (34.9%)	25.6% (34.6%)
16	f-dist	94 (140)	772 (865)	135,215 (209,085)	2,868 (4,244)	1,903 (3,367)	25.1% (28.3%)	27.9% (35.7%)
25	pre-fil	96 (151)	727 (**807**)	**124,823** (193,984)	4,675 (7,471)	1,594 (2,007)	**29.8%** (37.3%)	**32.5%** (**39.6%**)

Frame-Pairwise Element-Wise Products Experiments								
ID	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
38	normal	81 (138)	782 (862)	127,258 (**192,868**)	8,917 (13,036)	1,696 (2,314)	28.3% (**35.5%**)	29.0% (38.2%)
39	f-dist	**123** (**173**)	**722** (835)	127,726 (205,443)	2,956 (3,961)	1,899 (2,815)	28.8% (32.1%)	31.7% (37.0%)
40	pre-fil	33 (38)	1,024 (1,253)	164,068 (299,958)	**1,619** (**2,357**)	**649** (**685**)	15.4% (11.9%)	14.4% (10.1%)

Table 5.11: Evaluation results of experiments using the spatial dataset with appended frame-pairwise element-wise products.

Baseline Comparison								
ID	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
2	normal	74 (120)	742 (829)	128,813 (198,570)	13,617 (19,274)	2,000 (2,504)	25.9% (34.9%)	25.6% (34.6%)
16	f-dist	94 (140)	772 (865)	135,215 (209,085)	2,868 (4,244)	1,903 (3,367)	25.1% (28.3%)	27.9% (35.7%)
25	pre-fil	**96** (**151**)	727 (807)	**124,823** (193,984)	4,675 (7,471)	1,594 (2,007)	29.8% (37.3%)	**32.5%** (39.6%)

Pairwise Element-Wise Products Experiments								
ID	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
41	normal	80 (132)	749 (830)	125,583 (**191,275**)	14,568 (19,914)	1,930 (2,427)	26.4% (35.2%)	26.8% (36.6%)
42	f-dist	**96** (142)	773 (867)	134,776 (208,064)	**2,716** (**4,134**)	1,830 (3,294)	25.8% (29.1%)	28.3% (36.0%)
43	pre-fil	93 (**151**)	**724** (**803**)	125,281 (192,449)	4,648 (7,702)	**1,588** (**2,040**)	**30.0%** (**37.6%**)	32.3% (**40.0%**)

Table 5.12: Evaluation results of experiments using the spatial dataset with appended pairwise element-wise products.

The last group experiments analyzed the influence of using distances between bounding boxes, which are normalized according to the size of the bounding box (see Table 5.13). As bounding box coordinates are only part of the spatial dataset, experiments were only applicable to said dataset. Note that the spatial dataset was used as input for the KNN graph creation and as node features within the GCN.

When comparing results, one can see that only the experiment using the frame-distance KNN graph resulted in significantly better results. Both the normal KNN graph and the pre-filtered frame-distance KNN graph saw a sharp decrease in performance. For the latter results were not able to be obtained as the network ended up predicting all singletons some sequences. This resulted in the postprocessing script being unable to handle the output and failing to convert the prediction into trajectories.

Baseline Comparison								
ID	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
2	normal	74 (120)	742 (829)	128,813 (198,570)	13,617 (19,274)	2,000 (2,504)	25.9% (34.9%)	25.6% (34.6%)
16	f-dist	94 (140)	772 (865)	135,215 (209,085)	**2,868** (**4,244**)	1,903 (3,367)	25.1% (28.3%)	27.9% (35.7%)
25	pre-fil	96 (151)	727 (807)	124,823 (193,984)	4,675 (7,471)	1,594 (2,007)	29.8% (37.3%)	32.5% (39.6%)

Normalized Distances Experiments								
ID	KNN	MT	ML	FN	FP	IDS	IDF1	MOTA
44	normal	29 (57)	1,036 (1,179)	157,979 (256,809)	7,741 (9,915)	**1,370** (**1,692**)	17.0% (24.5%)	14.0% (20.3%)
45	f-dist	**176** (**232**)	**656** (**747**)	**115,710** (**185,295**)	3,757 (5,225)	2,378 (3,621)	**31.7%** (**34.5%**)	**37.3%** (**42.4%**)
46	pre-fil	-	-	-	-	-	-	-

Table 5.13: Evaluation results of experiments using the spatial dataset with normalized distances.

After all feeder modifications were evaluated, experiments 35, 39 and 45 were decided to show the most promise. Since previously they were evaluated in an isolated way, last step in the feeder modification evaluation was to combine the different settings and see how they perform in a joined way. Table 5.14 shows results of pairwise combinations of said feeder modifications. One can see that results did not improve in any case and that each feeder modification performs better individually. Note that experiment 48 ended up predicting only singletons and was thus, like experiment 46, unable to be evaluated properly by the subsequent steps of the implementation.

5.5 Link Merging Experiments

The following experiments illustrate differences in performance employing pseudo label propagation as suggested by Wang et al. [62] compared to primal feasible search heuristics as suggested by Keuper et al. [34]. During this thesis the publicly available GitHub distribution of the heuristic provided by Keuper et al. [34] was

Individual Settings Experiments							
ID	MT	ML	FN	FP	IDS	IDF1	MOTA
35	132 (196)	662 (738)	122,386 (185,348)	3,198 (4,607)	2,317 (3,199)	29.2% (35.1%)	34.1% (42.7%)
39	123 (173)	722 (835)	127,726 (205,443)	**2,956** (**3,961**)	1,899 (2,815)	28.8% (32.1%)	31.7% (37.0%)
45	176 (232)	656 (747)	**115,710** (**185,295**)	3,757 (5,225)	2,378 (3,621)	31.7% (34.5%)	**37.3%** (**42.4%**)

Combined Settings Experiments								
ID	Combines	MT	ML	FN	FP	IDS	IDF1	MOTA
47	35 & 45	**181** (**237**)	**650** (**731**)	119,917 (186,743)	3,522 (5,362)	**1,311** (**2,056**)	**36.0%** (**38.6%**)	35.8% (42.4%)
48	39 & 45	-	-	-	-	-	-	-
49	35 & 39	74 (130)	751 (830)	126,673 (189,999)	13,800 (18,880)	2,054 (2,593)	26.4% (35.7%)	26.6% (37.2%)

Table 5.14: Evaluation results of experiments combining different feeder modifications.

used.[19] The *bias* used within the heuristic was adjusted to be 0.8, while *theta* was left unchanged from the standard configuration. To be able to apply the heuristic proposed by Keuper et al. [34] the implementation of Wang et al. [62] needed to be altered to save information on the predicted link likelihoods in a separate file. Initially, it was noted that the heuristic did not perform well, with *MOTA* scores being negative or close to zero using the predicted link likelihoods of experiment 45 as input.[20] This was due to fact that the original implementation added up edge weights of reoccurring edges when loading up the graph and its weights. Applied to the output of a GCN, loaded graphs thus featured edge weights being set too high, resulting in the heurisitc being unable to process the graph properly. The heurisitic was thus extended to either:

1. Only save the higher edge weight (*max*)
2. Only save the lower edge weight (*min*)
3. Apply an if-statement determining to keep either the higher, lower or set the edge to zero (*if*). See Chapter 4.2 for further details.

Table 5.15 summarizes evaluation results of performed experiments using primal feasible search heuristics as proposed by Keuper et al. [34]. Each experiment used

[19] See `https://github.com/bjoern-andres/graph`.

[20] Note that experiment 45 was needed to be re-run causing the predictions used as input for the following experiments to be slightly different from the ones reported in experiment 45 due to variance.

the predicted link likelihoods of experiment 45 as input. All in all, results of experiment 45 were able to be improved. In addition to using either the KL or GAEC algorithm, performance was also evaluated using both algorithms sequentially, with the output of the GAEC algorithm being used to initialize the graph for the KL algorithm. One can see that there is not a significant difference as far as overall tracking performance. Nevertheless, when applying the above mentioned if-statement to determine edge weights for reoccurring edges, each algorithm setup performed better on FRCNN and SDP detections and worsened for DPM detections. Said difference in performance is especially notable for the combined algorithmic setup. Moreover, one can see that using the maximum edge weight for contradicting edge weights causes tracking performance to worsen. Lastly, appending low-confidence 'dummy' edges to each heuristic input file caused no change in results. It was thus decided to not apply this to other experimental setups as it was also expected to interfere with the intention and effectiveness of the if statement determining edge weights.

Normalized Distances Experiment							
ID	MT	ML	FN	FP	IDS	IDF1	MOTA
45	176 (232)	656 (747)	115,710 (185,295)	**3,757** (**5,225**)	2,378 (3,621)	31.7% (34.5%)	37.3% (42.4%)

Link Merging Experiments									
ID	PFS	Weights	MT	ML	FN	FP	IDS	IDF1	MOTA
47	KL	min	231 (290)	579 (656)	107,513 (172,949)	5,727 (8,061)	2,540 (3,402)	36.5% (40.6%)	**40.4%** (**45.3%**)
48	GAEC	min	231 (291)	581 (658)	107,847 (173,365)	5,582 (7,516)	2,515 (3,416)	36.6% (40.8%)	40.3% (**45.3%**)
49	KL + GAEC	min	232 (291)	578 (**655**)	107,705 (173,087)	5,611 (7,841)	2,507 (3,399)	36.7% (40.9%)	**40.4%** (**45.3%**)
50	KL	max	234 (292)	594 (676)	110,283 (178,111)	4,783 (6,139)	**2,283** (**3,012**)	37.6% (41.6%)	39.6% (44.4%)
51	GAEC	max	230 (286)	596 (678)	111,077 (179,507)	4,662 (5,663)	2,293 (3,073)	37.6% (41.5%)	39.2% (44.1%)
52	KL + GAEC	max	231 (287)	601 (681)	110,996 (178,988)	4,714 (5,974)	2,278 (3,040)	**37.7%** (**41.6%**)	39.3% (44.2%)
53	KL	if	**237** (**295**)	586 (663)	107,469 (173,295)	5,711 (8,389)	2,513 (3,423)	37.0% (40.8%)	**40.4%** (45.1%)
54	GAEC	if	235 (294)	**577** (655)	107,577 (173,386)	5,723 (8,001)	2,496 (3,349)	37.6% (40.8%)	**40.4%** (45.2%)
55	KL + GAEC	if	236 (**295**)	**577** (655)	**107,438** (**173,243**)	5,828 (8,090)	2,482 (3,324)	37.6% (41.4%)	**40.4%** (45.2%)
56	KL	min + dummy	231 (290)	579 (656)	107,513 (172,949)	5,727 (8,061)	2,540 (3,402)	36.5% (40.6%)	**40.4%** (**45.3%**)

Table 5.15: Evaluation results of experiments using different variations of primal feasible search heuristics as seen in [34]

As mentioned in Section 5.2, GCNs trained using reidentification and spatial features use different types of IPS in order to learn to differentiate instances. As both feature datasets encode very different viewpoints of the problem at hand, it is expected that each network would learn to predict different types of situations correctly. Therefore, the idea was expressed to train each network separately and, unlike the ensemble approach, combine the predictions of the two networks. The altered implementation of primal feasible search heuristics can be used to achieve this. As the above-mentioned approaches to handle reoccurring edges allow there to be multiple defined edges within a prediction file, the heuristic input files of two networks are able to be merged by simply concatenating both files. Table 5.16 shows results of applying the primal feasible search heuristic setup as seen in experiment 55 on predicted link likelihoods produced by networks trained in experiment 17 and 45. Unfortunately, results were equal to the ones as seen in experiment 55.

Best Settings								
ID	Input	MT	ML	FN	FP	IDS	IDF1	MOTA
17	ReID	84 (129)	863 (959)	139,285 (213,979)	**1,718** (**2,083**)	2,486 (3,614)	20.5% (27.4%)	26.1% (34.8%)
45	Spa	176 (232)	656 (747)	115,710 (185,295)	3,757 (5,225)	**2,378** (3,621)	31.7% (34.5%)	37.3% (42.4%)

Combined Link Merging Experiments								
ID	Input	MT	ML	FN	FP	IDS	IDF1	MOTA
57	Spa + ReID	**236** (**295**)	**577** (**655**)	**107,438** (**173,243**)	5,828 (8,090)	2,482 (**3,324**)	**37.6%** (**40.6%**)	**40.4%** (**45.2%**)

Table 5.16: Evaluation results of combining results from experiment 17 and 45 using primal feasible search heuristics as seen in experiment 55.

All in all, even though results did not vary too much across experiments, the setup applied in experiment 55 shows the most promise for FRCNN and SDP detections. For DPM detections the heurisitic of Keuper et al. [34] should be applied as seen in experiment 47 or 49. Moreover, combining predictions results as seen in experiment 57 did not yield better results.

5.6 Best Setting

When comparing experiments conducted in Chapter 5.1 to 5.4, experiment 45 sets out to be the best performing single-GCN setup setting as far as tracking is concerned. In the following, the setting in experiment 45 will be evaluated using the set of preprocessed detections used to train the Tracktor tracker proposed by

Bergmann et al. [4]. Within their implementation Bergmann et al. reject false positive and correct misaligned detections by applying regression and classification. Using the preprocessed detections, tracking performance of the GCN dropped significantly (see experiment 58 in Table 5.17). When looking at individual sequence results, one can notice that performance only dropped for GCN trained using FRCNN and SDP provided detections, while it significantly increased for DPM detections. Thus, it was tested if results improved using a mixed setup consisting of SDP and FRCNN predictions of experiment 45 and DPM predictions of experiment 58 (see experiment 59). One can see that results improved using said setup. Experiment 60 shows how results even further improved when applying primal feasible search heuristics as seen in experiment 55.

Normalized Distances Experiment							
ID	MT	ML	FN	FP	IDS	IDF1	MOTA
45	176 (232)	656 (747)	115,710 (185,295)	3,757 (5,225)	2,378 (3,621)	31.7% (34.5%)	37.3% (42.4%)

Preprocessed Detections Experiments								
ID	Setup	MT	ML	FN	FP	IDS	IDF1	MOTA
58	normal	73 (122)	780 (853)	132,741 (197,147)	**306** (**398**)	**1,935** (2,648)	26.4% (33.8%)	30.5% (40.6%)
59	mixed	179 (239)	661 (733)	115,086 (176,133)	2,292 (2,451)	1,986 (2,736)	34.1% (38.2%)	38.5% (46.2%)
60	mixed (PFS)	**238** (**303**)	**592** (**665**)	**106,257** (166,300)	3,281 (3,435)	1,995 (**2,500**)	**40.2%** (**45.6%**)	**42.6%** (**48.9%**)

Table 5.17: Evaluation results of experiments using primal feasible search heuristics to determine a final clustering instead of pseudo label propagation.

As the last experiment, postprocessing hyperparameters were optimized for experiment 60 (see Table 5.18). As the runtime of the postprocessing script is very small, one can quickly obtain results for different setups. It was noted, that lowering the minimum cluster size parameter t_s as well as eliminating the restriction posed by the minimum detection confidence t_d and minimum average detection confidence within a cluster t_c resulted in a better tracking performance for most sequences (see experiment 61 in Table 5.19).

The setting employed in experiment 61 was decided to be the best setting within this thesis and used to predict the test sequences in order to submit results to the MOT17 leaderboard. Table 5.20 shows submission results broken down by each sequence. The tracker proposed in this thesis ranks 64th within the public leaderboard of MOT17.[21] The tracker ranks worse than Keuper et al. approach illustrated in [33] and the approach proposed by Bergmann et al. [4]. Nevertheless, only one

[21] See tracker called *'GCN_LP'* within the public leaderboard.

Optimized Postprocessing Settings					
Detector	t_d	t_c	t_s	t_g	smoothing
DPM	-1	-1	2	30	no
FRCNN/ SDP	-1	-1	3	30	no

Table 5.18: Overview of the employed optimized postprocessing settings used in experiment 61.

Best Setting							
ID	MT	ML	FN	FP	IDS	IDF1	MOTA
60	238 (303)	592 (665)	106,257 (166,300)	**3,281** (**3,435**)	**1,995** (**2,500**)	**40.2%** (**45.6%**)	42.6% (48.9%)

Postprocessing Script Experiments							
ID	MT	ML	FN	FP	IDS	IDF1	MOTA
61	**275** (**344**)	**489** (**559**)	**100,170** (**158,647**)	4,863 (5,059)	3,314 (5,706)	39.7% (45.1%)	**44.2%** (**50.2%**)

Table 5.19: Evaluation results of experiments using primal feasible search heuristics to determine a final clustering instead of pseudo label propagation.

sequence was used to obtain said results. As the MOT17 dataset is structured in that train sequences have a corresponding test sequence which employs a similar setting, it is expected that using additional training sequences (and thus more settings) to train the GCNs will yield better results. A final assessment of the approach proposed within this thesis as well as points of improvement for future research will be given in Chapter 6.

MOT17 Submission Results							
Sequence	MT	ML	FN	FP	IDS	IDF1	MOTA
MOT17-01-DPM	2	13	4458	8	79	21.28%	29.53%
MOT17-01-FRCNN	6	8	3305	1359	74	29.61%	26.54%
MOT17-01-SDP	6	4	2772	816	134	29.64%	42.29%
MOT17-03-DPM	26	29	48542	143	1076	34.79%	52.46%
MOT17-03-FRCNN	36	27	43025	1624	527	44.90%	56.84%
MOT17-03-SDP	66	14	27032	1162	839	47.20%	72.26%
MOT17-06-DPM	17	108	6690	75	353	30.47%	39.60%
MOT17-06-FRCNN	39	63	5032	357	296	46.77%	51.76%
MOT17-06-SDP	57	74	4757	459	288	46.40%	53.29%
MOT17-07-DPM	2	27	11555	61	285	19.50%	29.55%
MOT17-07-FRCNN	5	16	9755	1317	295	32.21%	32.71%
MOT17-07-SDP	10	17	8161	766	309	32.58%	45.33%
MOT17-08-DPM	6	41	15819	47	146	23.85%	24.20%
MOT17-08-FRCNN	6	41	15533	774	116	27.25%	22.25%
MOT17-08-SDP	10	36	14364	402	208	28.35%	29.11%
MOT17-12-DPM	9	47	4981	28	63	42.92%	41.48%
MOT17-12-FRCNN	12	43	4833	524	54	47.82%	37.57%
MOT17-12-SDP	19	38	4319	582	62	51.50%	42.74%
MOT17-14-DPM	3	104	14065	56	324	20.60%	21.85%
MOT17-14-FRCNN	7	80	12371	2432	656	22.29%	16.36%
MOT17-14-SDP	6	63	11156	1544	603	27.08%	28.03%
Overall	350	893	272,525	14,536	6,787	37.78%	47.92%

Table 5.20: Evaluation results of submitting predictions of experiment 61 to the MOT17 website.

Chapter 6

Conclusion

6.1 Summary

This thesis demonstrated the applicability of GCNs on MOT. The proposed approach differs from other related works as it only uses local information surrounding a detection to train the network. The implementation includes a dataset creation script, a modified version of the GCN proposed by Wang et al. [62], a postprocessing script, which converts a given clustering into trajectories and an evaluation script, which calculates official MOT17 evaluation metrics using *py-motmetrics*. The dataset script is able to produce five different feature datasets which all encode different types of knowledge used to solve MOT. Further, it allows users to create two additional types of KNN graphs which focus on prefiltering the localized information that GCNs are trained with. Lastly, a label creation process is proposed which assigns labels to the MOT training data based on their overlap with ground truth detections. Using said dataset creation process all necessary input was created to be able to apply the GCN proposed by Wang et al. [62]. The approach by Wang et al. was extended with optional modifications to the data feeder. Moreover, the network was modified so that it saves the prediction output into a text file which can be used to apply primal feasible search heuristics as seen in [34] to obtain a final clustering. Furthermore, the implementation produces plots of the created IPS graphs as well as plots which analyze the network for overfitting. This additional information allows users to obtain a deeper understanding of how the GCN is trained and identify points of improvement. During experiments each of the created feature datasets was evaluated individually as well as in combination with spatial features. Baseline experiments demonstrated that the reidentification and spatial dataset hold the most value and were thus further analyzed in subsequent experiments. During consistency experiments it was identified that

results produced using the spatial dataset showed a high degree of variance. This variance was reduced using the two frame-distance and pre-filtered frame-distance KNN graph, but not eliminated. Nevertheless, tracking performance was improved for both datasets using the altered KNN graph calculation. Next, different input settings for the KNN graph calculation and node features were tested and it was determined that there is little to no value in combining the different feature datasets via concatenation. Modifications made to the data feeder improved performance of the spatial dataset significantly with the setting using normalized distances scoring the highest. It was also evaluated that combining different feeder modifications worsened results. During the next type of experiments, it was noted that changing the heuristic to obtain a final clustering from using pseudo label propagation [67] to primal feasible search heuristics as seen in [34] improved the tracker's performance. In order to apply the heuristic, the implementation proposed by Keuper et al. [34] was altered to handle reoccurring edge weights. In total users are able to select between three different methods how weights are assigned to reoccurring edges. Additionally, the two algorithms proposed in [34], namely the GAEC and KL algorithm, were evaluated separately as well as in a combined setting. Lastly, the best performing setting was evaluated using preprocessed detections as proposed in [4], which improved results for detections produced by the DPM detector. Consequently, a mixed setup using predictions obtained from training with raw detections (FRCNN and SDP detector) as well as preprocessed detections (DPM detector) was decided to be the best performing setting. Before submitting predictions to the MOT website, the hyperparameters associated with the postprocessing script were altered and a performance improvement was able to be obtained. As of handing in this thesis the approach ranks 64th within the public leaderboard of the MOT17 challenge. It performs worse than approaches seen in [33] and [4] but is only trained using one sequence. All in all, work in this thesis demonstrates that GCNs are able to be applied to solve MOT.

6.2 Future Work

Due to the limited timeframe of this thesis not every idea on how to improve the GCN's performance was validated during experiments. In the following, a list of potential points of improvements for future research will be given. It is expected that said points will improve tracking and thus make the method comparable in terms of performance to other state-of-the-art trackers.

As previously mentioned, two works [7, 63] have been published since the start of this thesis, which also apply GCNs to MOT. Said works differ from the approach illustrated within this thesis since they are not based using local information sur-

rounding a node, but consider the graph as a whole, posing global constraints on it in order to learn global dependencies among nodes [7]. It is expected that the unconstrained environment of presenting information to the GCN causes this thesis' approach to be freer in its decision-making process and to be able to solve difficult object tracking tasks correctly. For future research one could investigate said claim by comparing prediction results produced by the approach of Braso et al. [7] and Weng et al. [63] to the prediction results reported within this thesis. The comparison would deliver insights what the strengths and weaknesses of each approach are.

The most notable difficulty during this thesis was posed by the shared resource pool which was used to conduct experiments with. As all students at the University of Mannheim are given equal access, the server load varied and was sometimes too high for experiments to run properly. The current configuration of the script is limited to only using ten workers, i.e., CPU cores. Computation time of the implementation of Wang et al. [62] centers around the data feeder and is thus very CPU-intensive. Especially the larger the IPS are constructed, the longer the runtime of the script becomes. Furthermore, the computation of the frame-distance and pre-filtered frame-distance KNN graph is holds room for improvement. As of handing in this thesis, neighbors for each node are computed sequentially using a two for-loops. This makes the calculation of the KNN take up more than 80% of the runtime. Using more workers and an optimized calculation of the KNN graphs, it is expected that the runtime of the script, which is currently roughly eight hours, can be significantly decreased. A decreased runtime would also be beneficial when performing hyperparameter tuning. As for the same reason previously mentioned, hyperparameter tuning was not performed during this thesis and the hyperparameter setting employed by Wang et al. [62] was used during all experiments.

As mentioned in Chapter 5.2 the spatial dataset showed a high degree of variance when used with the standard way of calculating KNN graphs. This variance was accredited to the IPS not containing relevant information which help the GCN learn how to differentiate detections by their spatial and temporal features. Relevant plots to analyze IPS graphs were created during this thesis, but it remains an open question how the IPS need to be composed. Lowering computation time of the GCN would allow to try and increase the h-hop neighborhoods of IPS. Additionally, one could consider to not only predict link likelihoods of one-hop neighborhoods, but also predict likelihoods of higher order neighbors. Wang et al. [62] argue that this does not hold any value as most higher order neighbors are negative, i.e., do not share the same label as the detection. Nevertheless, one needs to consider that Wang et al. [62] applied the GCN in a different context, i.e., clustering of faces, which makes their claim not directly transferable to MOT.

Chapter 4.2 introduced an ensemble architecture by combining multiple GCNs.

Within said setup intermediate feature representations of GCNs were combined and used as input to train a final GCN. Furthermore, in Chapter 5.5 it was investigated whether results can be improved by combining predictions of a network trained using reidentification features and one network trained using spatial features using an altered version of the implementation proposed by Keuper et al. [34]. Experiments centered around both approaches did not end up improving tracking performance. Nevertheless, combining knowledge encoded in appearance- and spatial-based features is expected to hold value. As Chapter 5.3 illustrated, simple concatenation of features did not yield significant performance improvements. Alternatively, it could be investigated whether the GCN can be altered to allow datasets to be introduced as external knowledge at an intermediate point of the training process. Additionally, the previously mentioned approaches should be further evaluated with different hyperparameter settings and combine predictions using, e.g., voting strategies.

Experiments performed in this thesis mostly centered around evaluating the reidentification and spatial dataset. As the person and classification dataset, like the reidentification dataset, both deliver an appearance-based viewpoint and did not perform as good as the reidentification dataset, both datasets were omitted from further experiments. Nevertheless, employing, e.g., a voting scheme combining different GCNs, viewpoints provided by said datasets may still hold value. Furthermore, one could alter the objective of the GCN to suit the objective of the network used to create the person dataset and only cluster detections by person and non-person detections.

As mentioned in the previous section, results presented in this thesis were obtained using only one sequence as training input. As each sequence within the MOT sequence employs a different setting under which conditions the sequence was filmed and each train sequence has a corresponding test sequence, the GCN was applied on data which featured unseen settings. It is expected that results are able to be further improved if one includes more sequences and thus more data within the training process. This could be via training using a combined setup, where all sequences are concatenated along the x-axis, or a sequential setup, where the GCN is trained throughout multiple iterations, each iteration being a new sequence used as input. Alternatively, it could also be sufficient to train the network using the MOT17-04 sequence and then fine-tune the network using the corresponding train sequence of the test sequence which is to be predicted. Furthermore, one could validate the effectiveness of the proposed approach using different MOT datasets, e.g., the benchmark dataset from 2016 (MOT16) or 2020 (MOT20).

Lastly, the code of the implementation has minor flaws which would need to be fixed in the future. First, the script performing experiments would need to be adjusted to allow users to use an ensemble architecture. As the setting did not provide

good results and was thus not further evaluated, the current workflow of the ensemble script deviates significantly from the actual workflow. All necessary functions are implemented but need to be combined properly again. Secondly, validation of the training sequence was not done correctly. This caused some plots, which are created during evaluation and can be used to analyze a network for overfitting, to be incorrect and thus hold no analytical value. This can be fixed by splitting the train sequence into a training and validation part. As for the ensemble code, all necessary functions are implemented and splits are created during data creation, but one would need to adjust the workflow of the script responsible for conducting experiments. Thirdly, as mentioned in Chapter 4, Wang et al. [62] only implemented one of three described aggregation functions mentioned in their work. The two missing aggregation functions could lead to improved results; thus, one could work on implementing them based on their description within [62].

Bibliography

[1] Albert-Lszl Barabsi and Rka Albert. Emergence of Scaling in Random Networks. 286(5439):509–512.

[2] E. Baser, V. Balasubramanian, P. Bhattacharyya, and K. Czarnecki. FANTrack: 3D Multi-Object Tracking with Feature Association Network. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1426–1433.

[3] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua. Multiple Object Tracking Using K-Shortest Paths Optimization. 33(9):1806–1819.

[4] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. Tracking Without Bells and Whistles. pages 941–951.

[5] Keni Bernardin and Rainer Stiefelhagen. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. 2008:1–10.

[6] E. Bochinski, V. Eiselein, and T. Sikora. High-Speed tracking-by-detection without using image information. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6.

[7] Guillem Braso and Laura Leal-Taixe. Learning a Neural Solver for Multiple Object Tracking. pages 6247–6257.

[8] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. 30:107–117.

[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs.

[10] Wongun Choi. Near-Online Multi-Target Tracking With Aggregated Local Flow Descriptor. pages 3029–3037.

[11] Wongun Choi and Silvio Savarese. A Unified Framework for Multi-target Tracking and Collective Activity Recognition. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision ECCV 2012*, Lecture Notes in Computer Science, pages 215–230. Springer.

[12] Sunil Chopra and M. R. Rao. The partition problem. 59(1):87–115.

[13] Michal Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. 29:3844–3852.

[14] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[15] Michel Marie Deza and Monique Laurent. *Geometry of Cuts and Metrics*. Springer.

[16] Miriam Drake. *Encyclopedia of Library and Information Science, Second Edition* -. CRC Press.

[17] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc.

[18] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object Detection with Discriminatively Trained Part-Based Models. 32(9):1627–1645.

[19] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. 9(5):768–786.

[20] Ross Girshick. Fast R-CNN. pages 1440–1448.

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks.

[22] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2.

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. 30:1024–1034.

[24] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. pages 2961–2969.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. pages 1026–1034.

[27] Andrea Hornakova, Roberto Henschel, Bodo Rosenhahn, and Paul Swoboda. Lifted Disjoint Paths with Application in Multiple Object Tracking.

[28] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[29] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial Transformer Networks.

[30] Glen Jeh and Jennifer Widom. SimRank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 538–543. Association for Computing Machinery.

[31] H. Jiang, S. Fels, and J. J. Little. A Linear Programming Approach for Multiple Object Tracking. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[32] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. 49(2):291–307.

[33] M. Keuper, S. Tang, B. Andres, T. Brox, and B. Schiele. Motion Segmentation Multiple Object Tracking by Correlation Co-Clustering. 42(1):140–153.

[34] Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavou, Thomas Brox, and Bjorn Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1751–1759.

[35] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks.

[36] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. 42(8):30–37.

[37] Laura Leal-Taixe, Michele Fenzi, Alina Kuznetsova, Bodo Rosenhahn, and Silvio Savarese. Learning an Image-based Motion Context for Multiple People Tracking. pages 3542–3549.

[38] L. Leal-Taix, G. Pons-Moll, and B. Rosenhahn. Branch-and-price global optimization for multi-view multi-target tracking. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1987–1994.

[39] L. Leal-Taix, G. Pons-Moll, and B. Rosenhahn. Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 120–127.

[40] Laura Leal-Taix, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking.

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

[42] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks.

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

[44] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. 58(7):1019–1031.

[45] Wei Liu, Shengcai Liao, Weiqiang Ren, Weidong Hu, and Yinan Yu. High-Level Semantic Feature Detection: A New Perspective for Pedestrian Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5182–5191.

[46] Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. Differentiable Learning-to-Normalize via Switchable Normalization.

[47] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, and Luc Van Gool. Face Detection without Bells and Whistles. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision ECCV 2014*,

Lecture Notes in Computer Science, pages 720–735. Springer International Publishing.

[48] Anton Milan, Laura Leal-Taixe, Ian Reid, Stefan Roth, and Konrad Schindler. MOT16: A Benchmark for Multi-Object Tracking.

[49] Anton Milan, Laura Leal-Taixe, Konrad Schindler, and Ian Reid. Joint Tracking and Segmentation of Multiple Targets. pages 5397–5406.

[50] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines.

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

[52] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc.

[53] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. Performance Measures and a Data Set for Multi-target, Multi-camera Tracking. In Gang Hua and Herv Jgou, editors, *Computer Vision ECCV 2016 Workshops*, Lecture Notes in Computer Science, pages 17–35. Springer International Publishing.

[54] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. 20(1):61–80.

[55] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition.

[56] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. 8(3):714–735.

[57] S. Sun, N. AKHTAR, H. Song, A. S. Mian, and M. Shah. Deep Affinity Network for Multiple Object Tracking. pages 1–1.

[58] Siyu Tang, Mykhaylo Andriluka, Bjoern Andres, and Bernt Schiele. Multiple People Tracking by Lifted Multicut and Person Re-Identification. pages 3539–3548.

[59] L. J. P. van der Maaten and G. E. Hinton. Visualizing High-Dimensional Data Using t-SNE. 9:2579–2605.

[60] Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph Attention Networks.

[61] Wenhao Wang. Adapted Center and Scale Prediction: More Stable and More Accurate.

[62] Zhongdao Wang, Liang Zheng, Yali Li, and Shengjin Wang. Linkage based face clustering via graph convolution network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1117–1125.

[63] Xinshuo Weng, Yongxin Wang, Yunze Man, and Kris M. Kitani. GNN3DMOT: Graph Neural Network for 3D Multi-Object Tracking With 2D-3D Multi-Feature Learning. pages 6499–6508.

[64] 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.

[65] Z. Wu, T. H. Kunz, and M. Betke. Efficient track linking methods for track graphs using network-flow and set-cover techniques. In *CVPR 2011*, pages 1185–1192.

[66] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit All the Layers: Fast and Accurate CNN Object Detector With Scale Dependent Pooling and Cascaded Rejection Classifiers. pages 2129–2137.

[67] Xiaohang Zhan, Ziwei Liu, Junjie Yan, Dahua Lin, and Chen Change Loy. Consensus-driven propagation in massive unlabeled data for face recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 568–583.

[68] Jimuyang Zhang, Sanping Zhou, Jinjun Wang, and Dong Huang. Frame-wise Motion and Appearance for Real-time Multiple Object Tracking.

[69] Muhan Zhang and Yixin Chen. Link Prediction Based on Graph Neural Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5165–5175. Curran Associates, Inc.

[70] Muhan Zhang and Yixin Chen. Weisfeiler-Lehman Neural Machine for Link Prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 575–583. Association for Computing Machinery.

[71] Shanshan Zhang, Rodrigo Benenson, and Bernt Schiele. CityPersons: A Diverse Dataset for Pedestrian Detection.

[72] Zhedong Zheng, Xiaodong Yang, Zhiding Yu, Liang Zheng, Yi Yang, and Jan Kautz. Joint Discriminative and Generative Learning for Person Re-identification.

[73] Tao Zhou, Linyuan L, and Yi-Cheng Zhang. Predicting missing links via local information. 71(4):623–630.

Appendix A

Program Code / Resources

The source code, a detailed documentation and some usage examples are available at `https://github.com/mariusbock/master-thesis`. The state of the repository as of handing in this thesis, a PDF version of this thesis as well as log files of all experiments mentioned within this thesis are contained on the USB flash drive provided with this thesis.

Appendix B

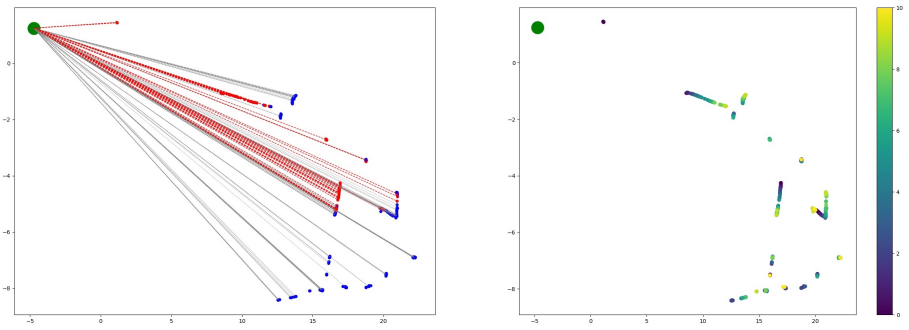
Further Resources

Evaluation metric	MOTChallenge	py-motmetrics
Multiple-object tracking accuracy	MOTA	MOTA
Identity F1	IDF1	IDF1
Multiple-object tracking precision	MOTP	MOTP
Mostly tracked	MT	MT
Mostly lost	ML	ML
False positives	FP	FP
False negatives	FN	FN
Recall	Recall	Rcll
Precision	Precision	Prcn
False alarms per frame	FAF	-
Identity switches	ID Sw.	IDs
Fragmentations	Frag	FM

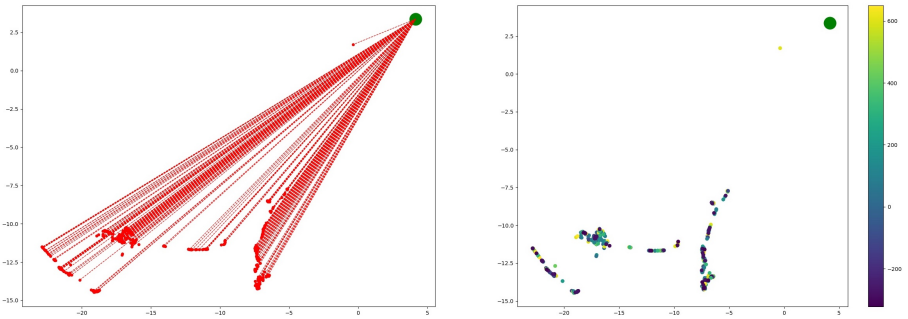
Table B.1: Abbreviations of the MOT evaluation metrics used by the MOTChallenge and *py-motmetrics*.

Evaluation measures			
Measure	Better	Perfect	Description
MOTA	higher	100%	Multi-Object Tracking Accuracy (+/- denotes standard deviation across all sequences). Combines three error sources: false positives (FP), missed targets (FN) and identity switches (IDS).
IDF1	higher	100%	Ratio of correctly identified detections over the average number of ground truth and computed detections
MOTP	higher	100%	Multi-Object Tracking Precision (+/- denotes standard deviation across all sequences). The misalignment between annotated and predicted bounding boxes.
MT	higher	100%	Ratio of ground truth trajectories that are covered by a track hypothesis for at least 80% of their respective life span.
ML	lower	0%	Ratio of ground truth trajectories that are covered by a track hypothesis for at most 20% of their respective life span.
FP	lower	0	Total number of false positives.
FN	lower	0	Total number of false negatives (missed targets).
Recall	higher	100%	Ratio of correct detections to total number of ground truth boxes.
Precision	higher	100%	Ratio of correct detections to total number of detections.
FAF	lower	0	Average number of false alarms (i.e., FP) per frame.
IDS (IDSR)	lower	0 (0%)	Total number of Identity Switches (Identity Switch Ratio = no. Identity Switches / Recall).
Frag (FRR)	lower	0 (0%)	Total number of times a trajectory is fragmented, i.e., interrupted during tracking (Fragmentation Ratio = no. Fragmentations / Recall)).

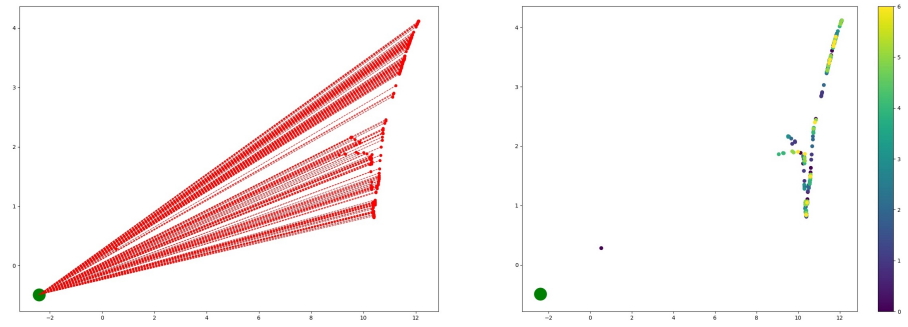
Table B.2: Overview of the official MOT17 evaluation metrics.[22]



(a) DPM detector.



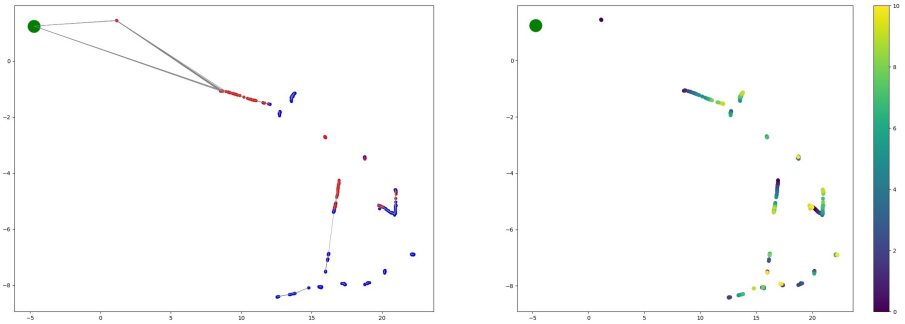
(b) FRCNN detector.



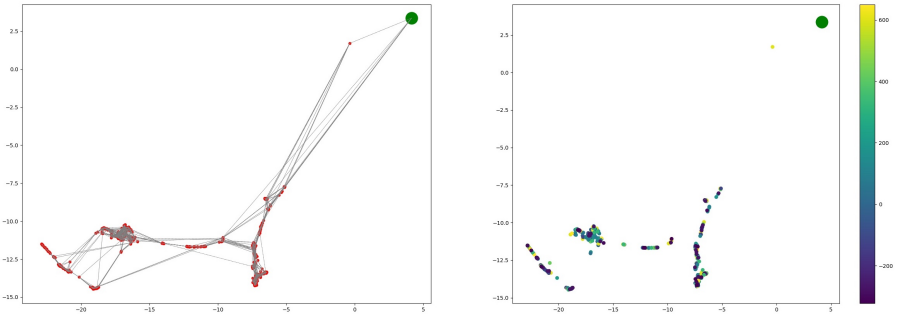
(c) SDP detector.

Figure B.1: KNN graph plots of IPS (index 10) used during training using reidentification dataset as input and normal KNN graph.

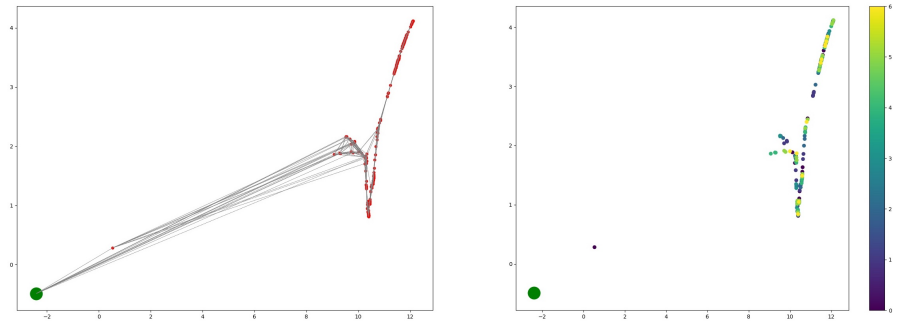
[22]This table was taken (with slight modifications) from the official MOT17 website `https://motchallenge.net/results/MOT17/`.



(a) DPM detector.

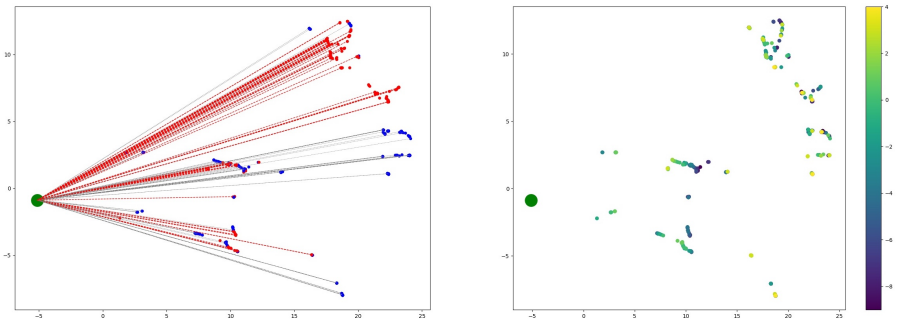


(b) FRCNN detector.

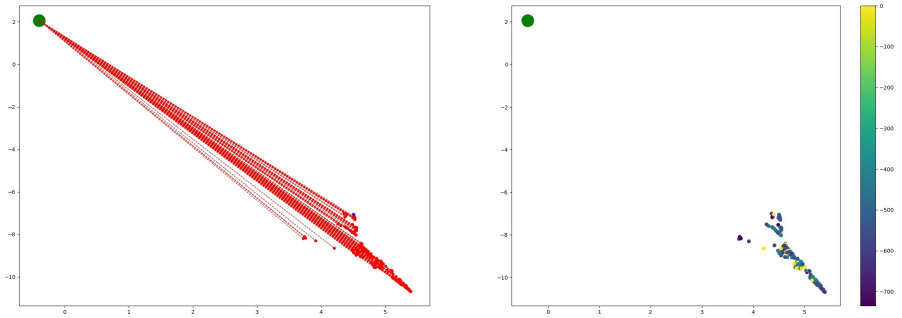


(c) SDP detector.

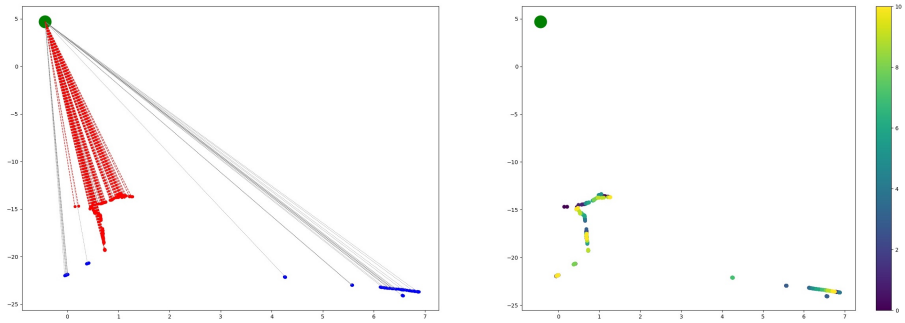
Figure B.2: Feature embedding plots of IPS (index 10) used during training using reidentification dataset as input and normal KNN graph.



(a) DPM detector.

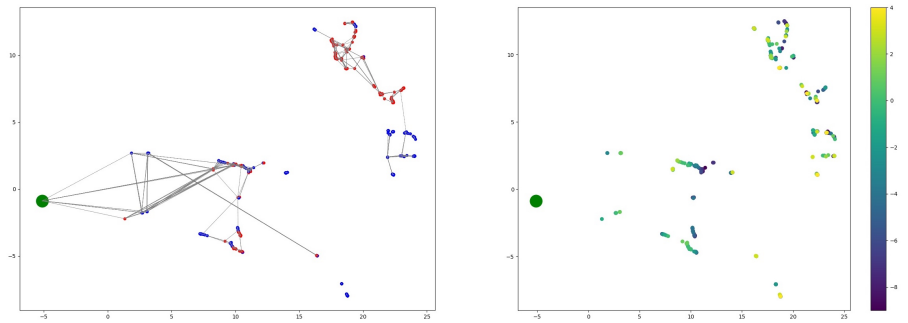


(b) FRCNN detector.

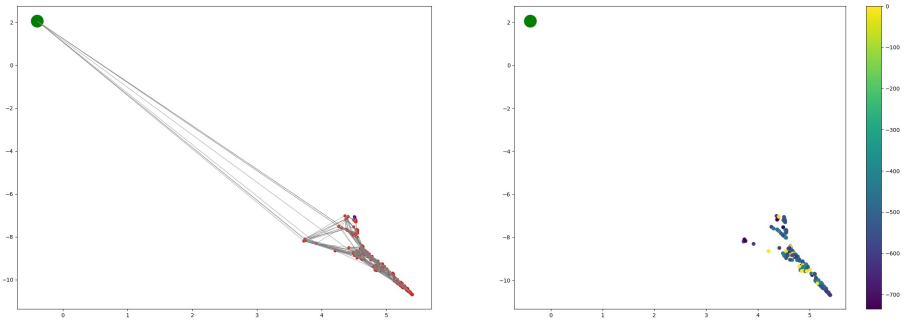


(c) SDP detector.

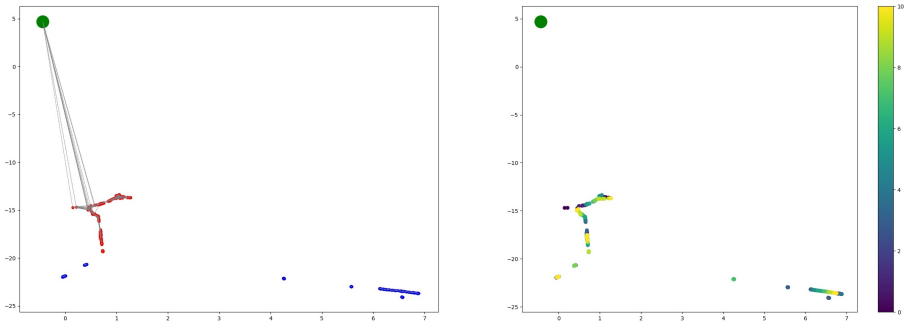
Figure B.3: KNN graph plots of IPS (index half) used during training using reidentification dataset as input and normal KNN graph.



(a) DPM detector.

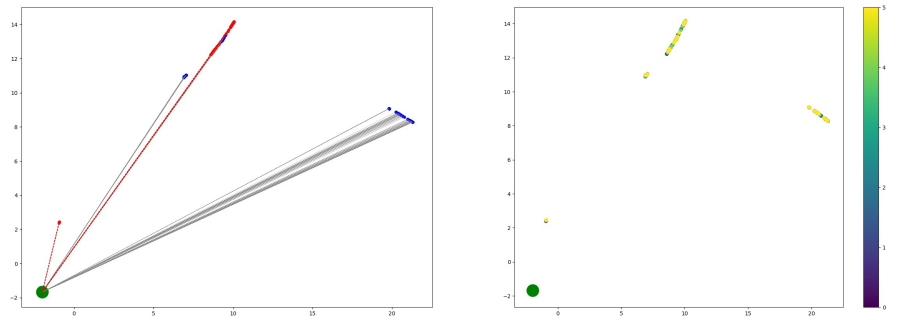


(b) FRCNN detector.

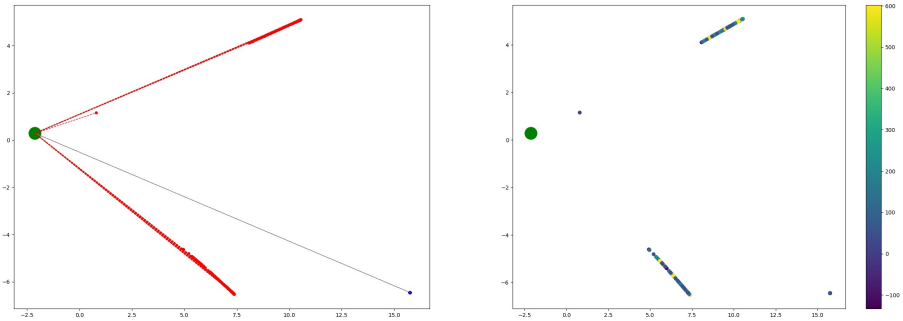


(c) SDP detector.

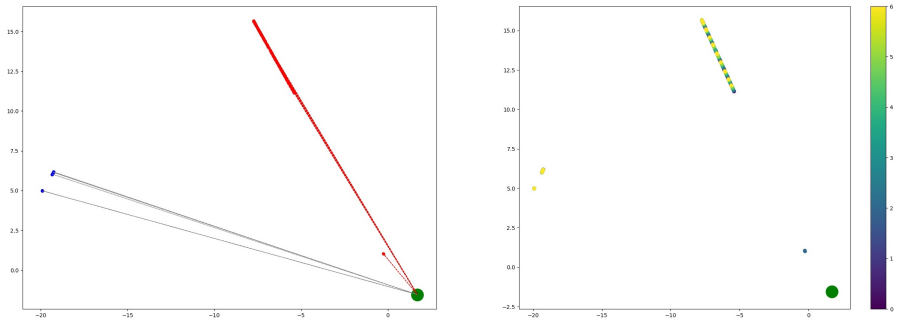
Figure B.4: Feature embedding plots of IPS (index half) used during training using reidentification dataset as input and normal KNN graph.



(a) DPM detector.

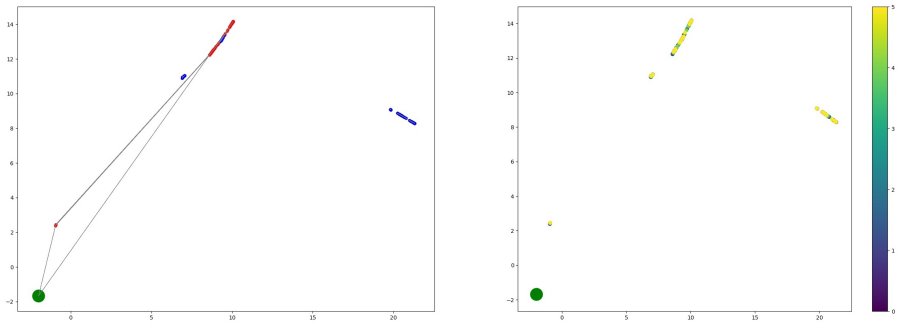


(b) FRCNN detector.

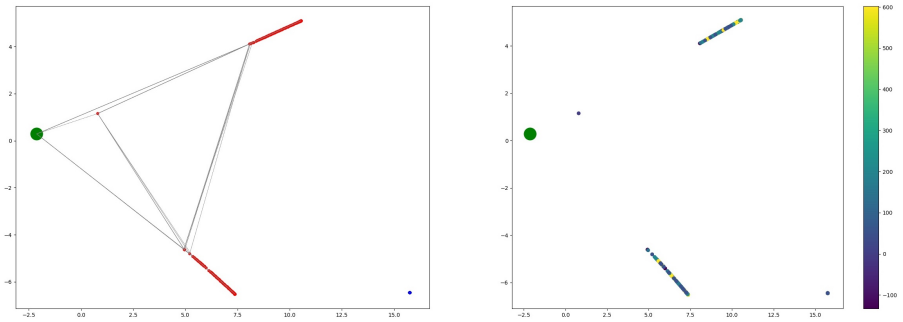


(c) SDP detector.

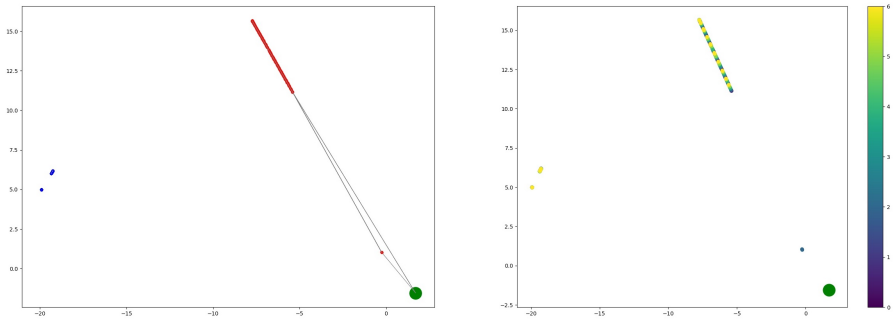
Figure B.5: KNN graph plots of IPS (index 10) used during training using spatial dataset as input and normal KNN graph.



(a) DPM detector.

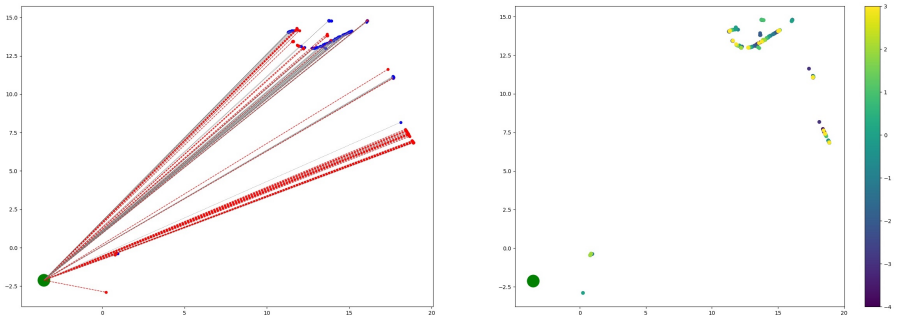


(b) FRCNN detector.

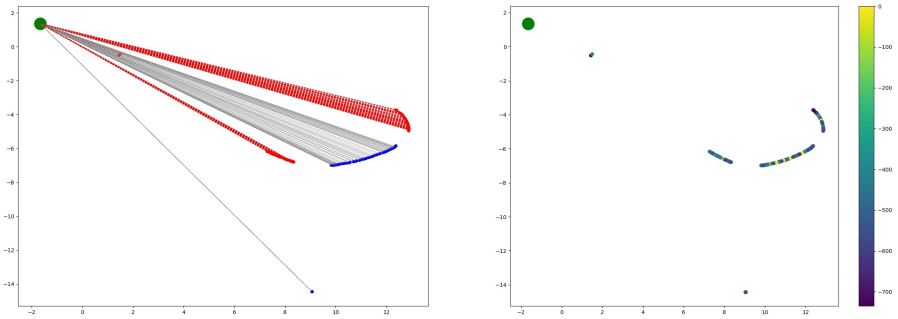


(c) SDP detector.

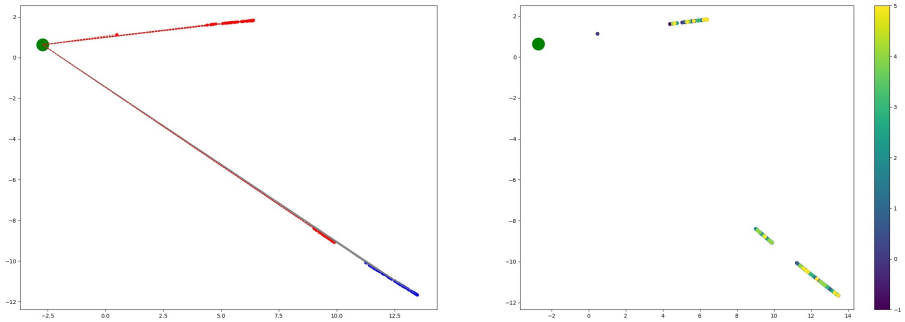
Figure B.6: Feature embedding plots of IPS (index 10) used during training using spatial dataset as input and normal KNN graph.



(a) DPM detector.

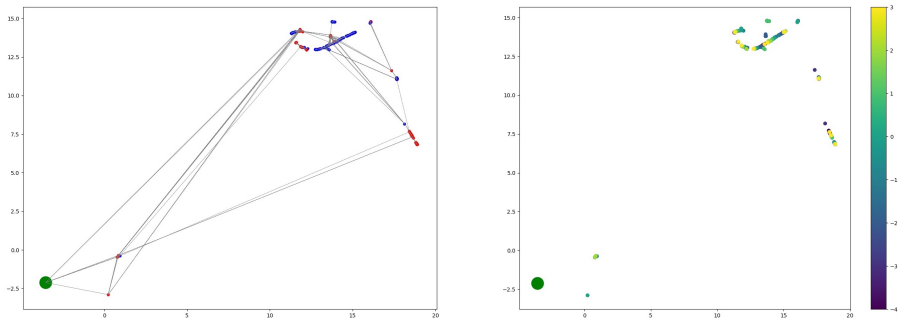


(b) FRCNN detector.

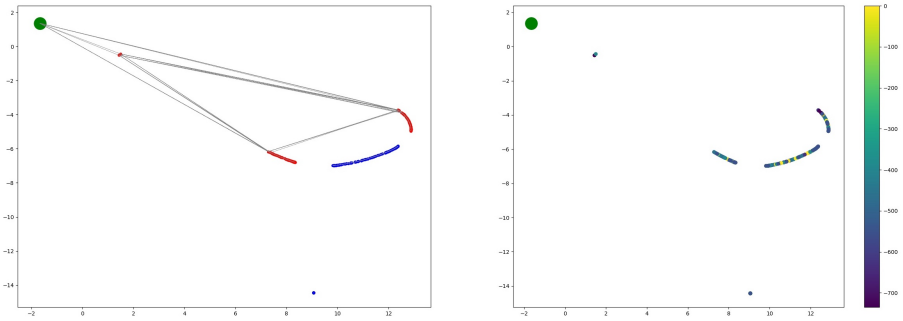


(c) SDP detector.

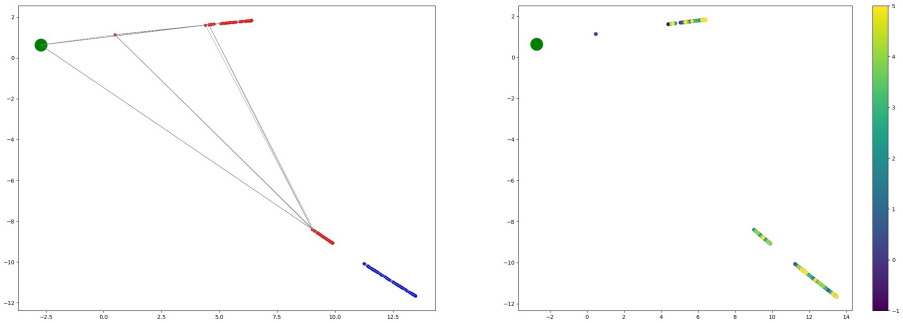
Figure B.7: KNN graph plots of IPS (index half) used during training using spatial dataset as input and normal KNN graph.



(a) DPM detector.

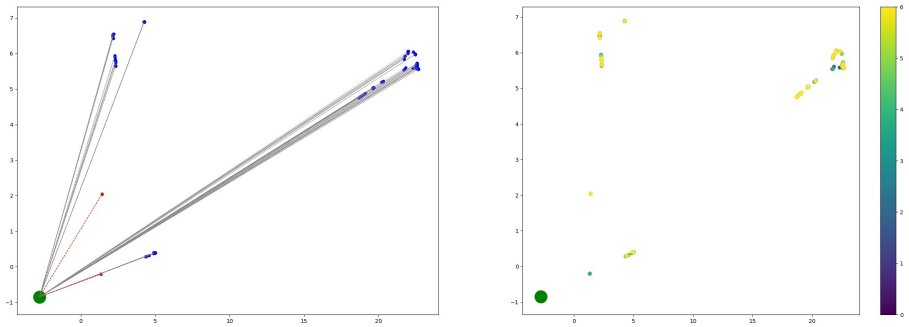


(b) FRCNN detector.

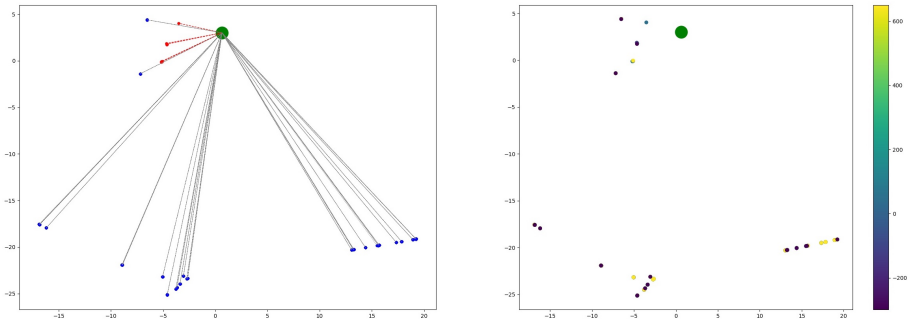


(c) SDP detector.

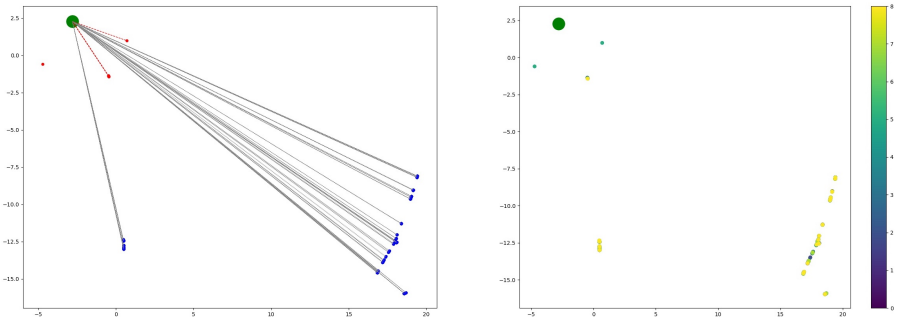
Figure B.8: Feature embedding plots of IPS (index half) used during training using spatial dataset as input and normal KNN graph.



(a) DPM detector.

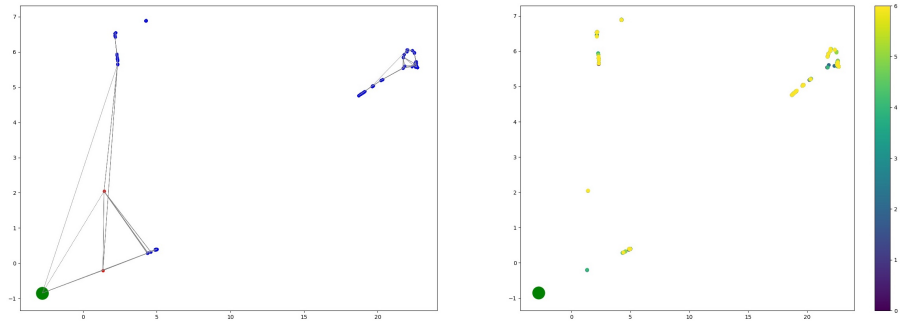


(b) FRCNN detector.

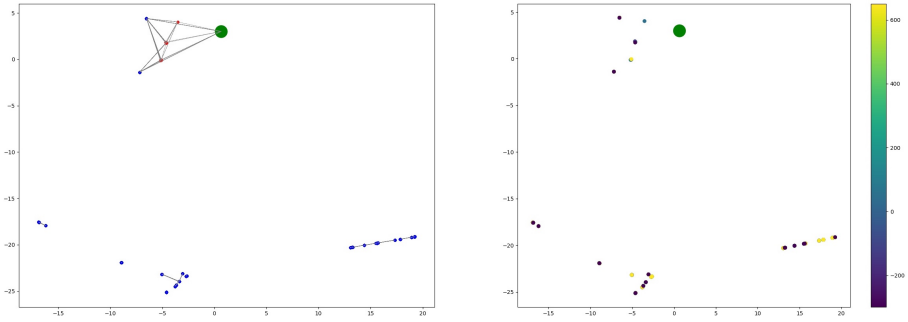


(c) SDP detector.

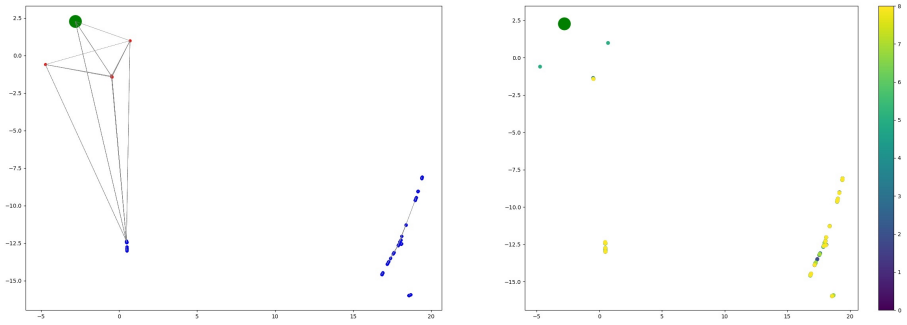
Figure B.9: KNN graph plots of IPS (index 10) used during training using spatial dataset as input and frame-distance KNN graph.



(a) DPM detector.

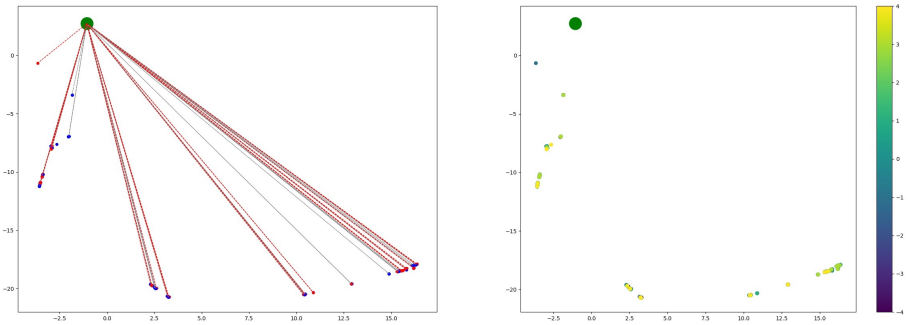


(b) FRCNN detector.

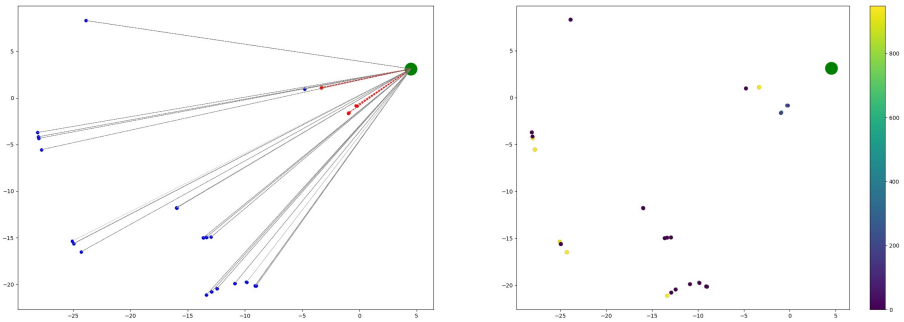


(c) SDP detector.

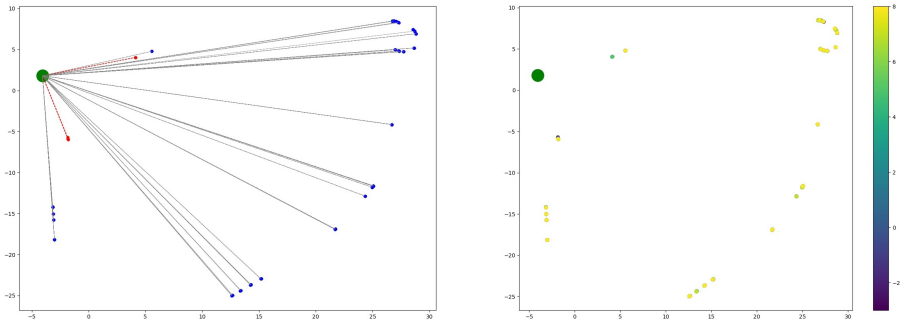
Figure B.10: Feature embedding plots of IPS (index 10) used during training using spatial dataset as input and frame-distance KNN graph.



(a) DPM detector.

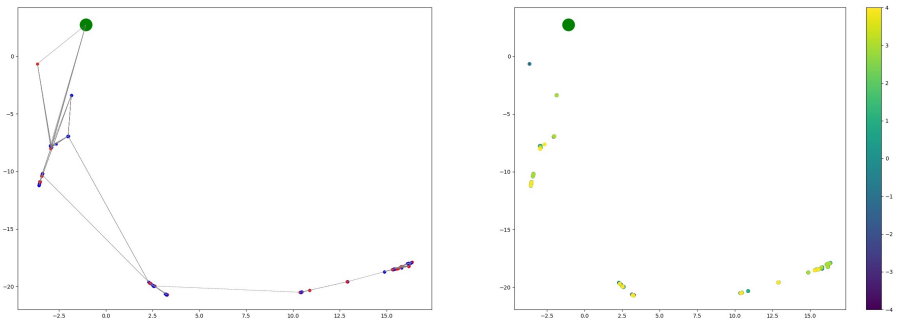


(b) FRCNN detector.

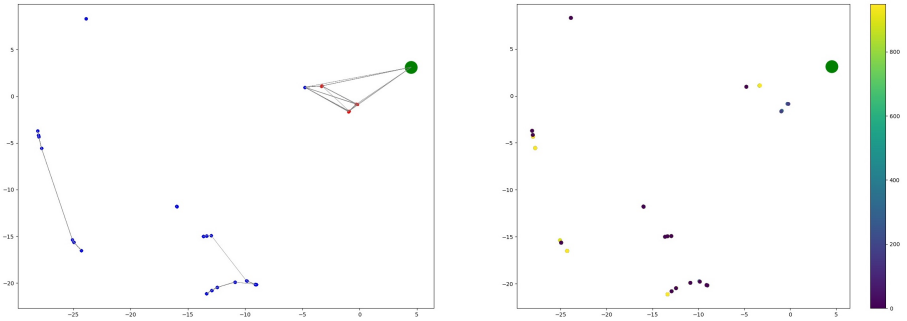


(c) SDP detector.

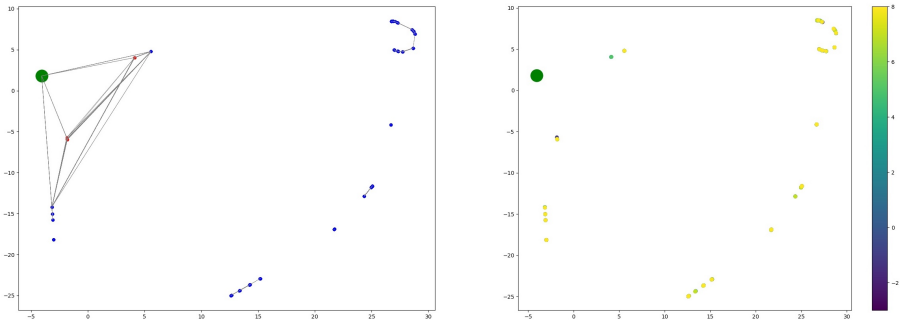
Figure B.11: KNN graph plots of IPS (index half) used during training using spatial dataset as input and frame-distance KNN graph.



(a) DPM detector.

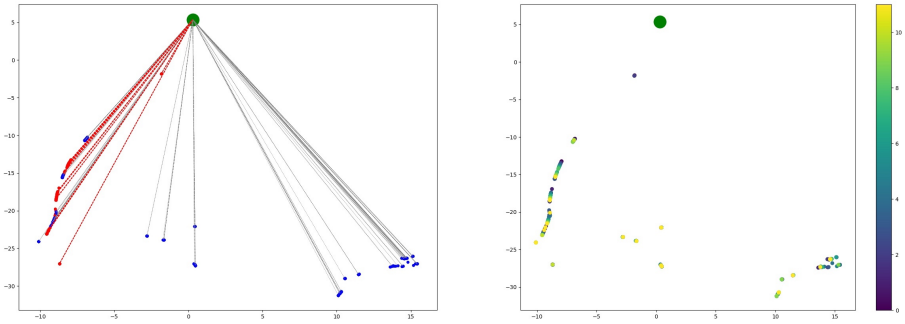


(b) FRCNN detector.

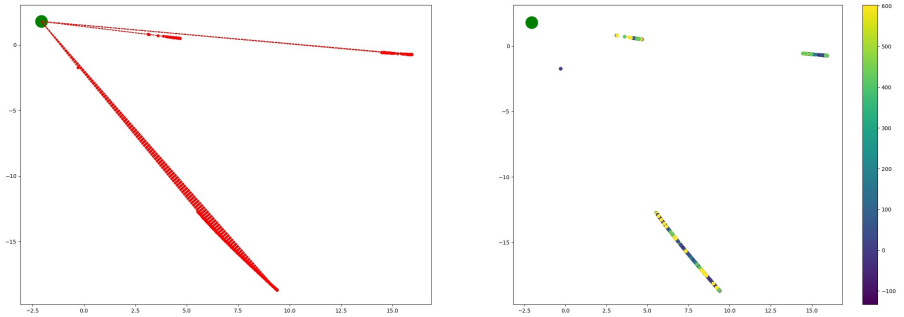


(c) SDP detector.

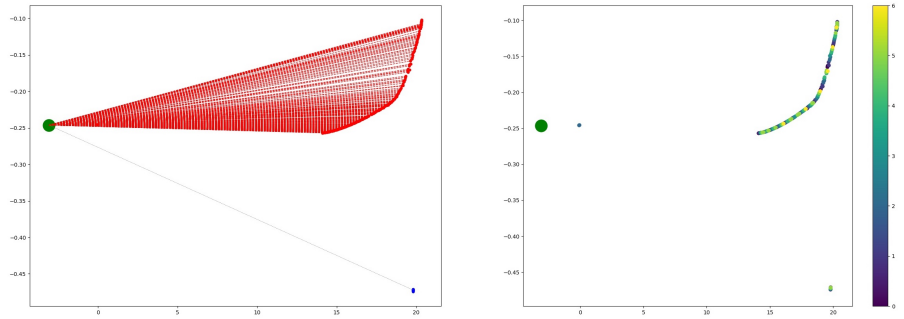
Figure B.12: Feature embedding plots of IPS (index half) used during training using spatial dataset as input and frame-distance KNN graph.



(a) DPM detector.

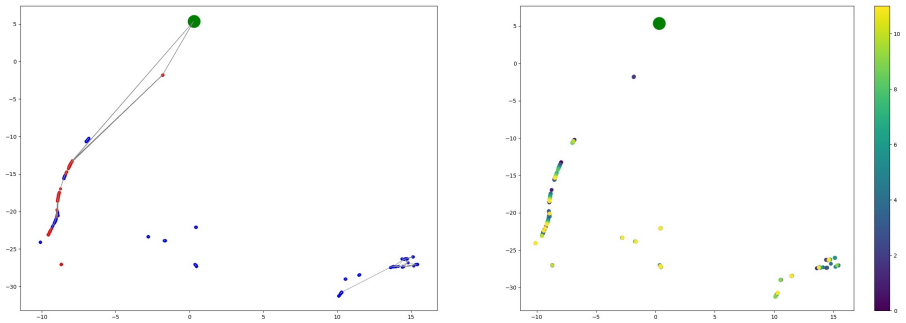


(b) FRCNN detector.

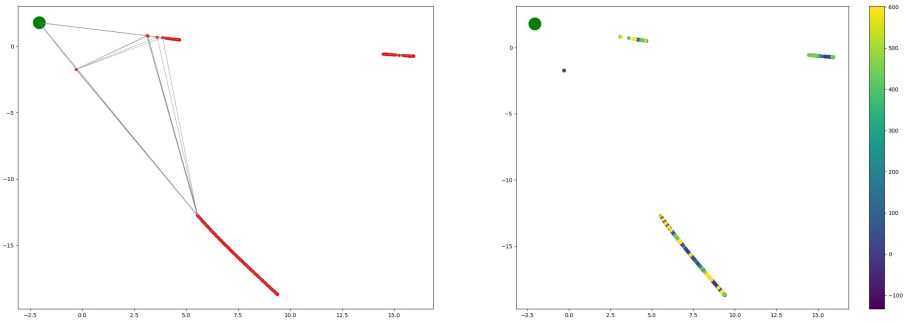


(c) SDP detector.

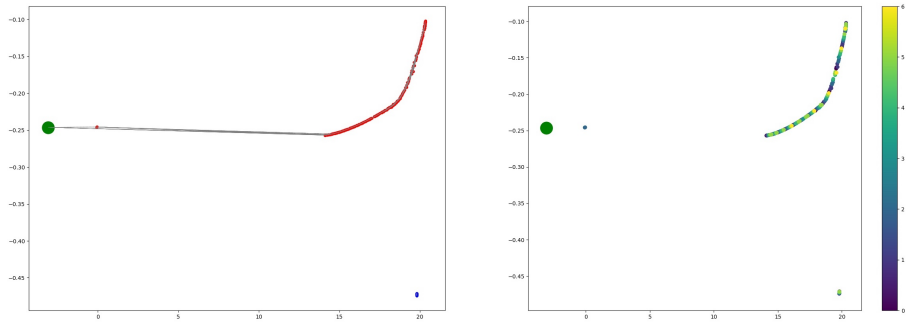
Figure B.13: KNN graph plots of IPS (index 10) used during training using spatial dataset as input and pre-filtered frame-distance KNN graph.



(a) DPM detector.

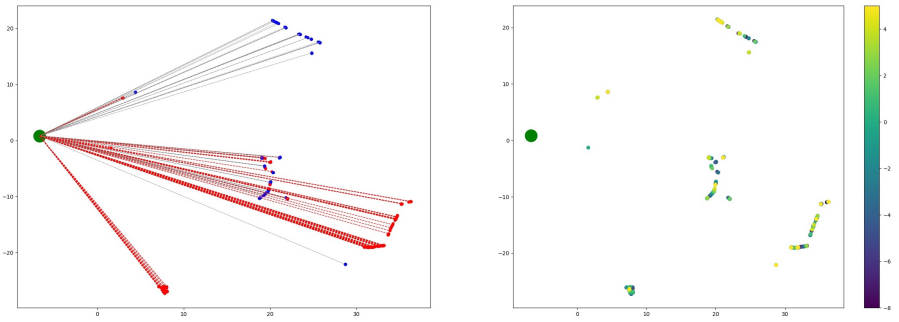


(b) FRCNN detector.

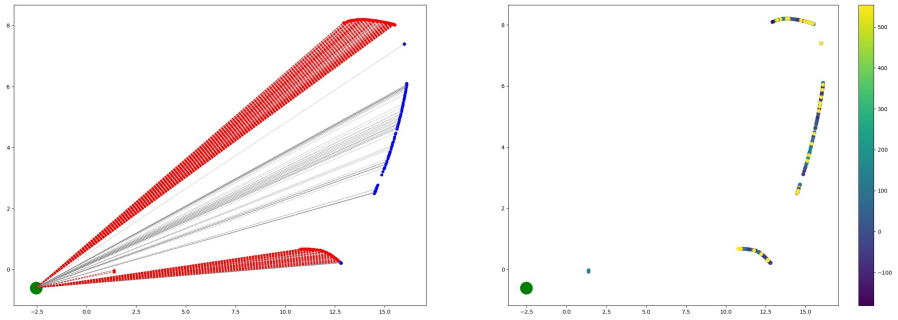


(c) SDP detector.

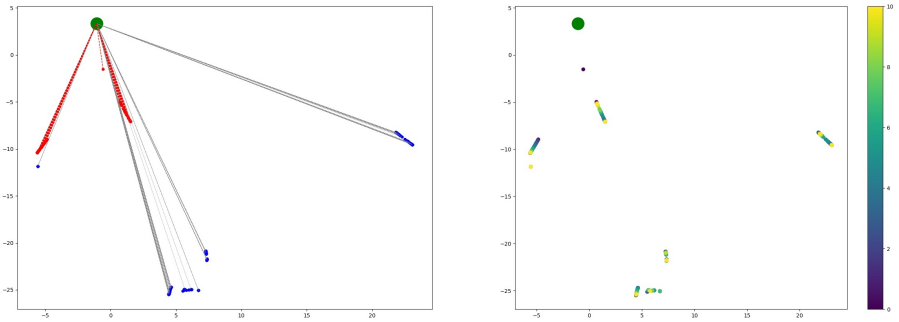
Figure B.14: Feature embedding plots of IPS (index 10) used during training using spatial dataset as input and pre-filtered frame-distance KNN graph.



(a) DPM detector.

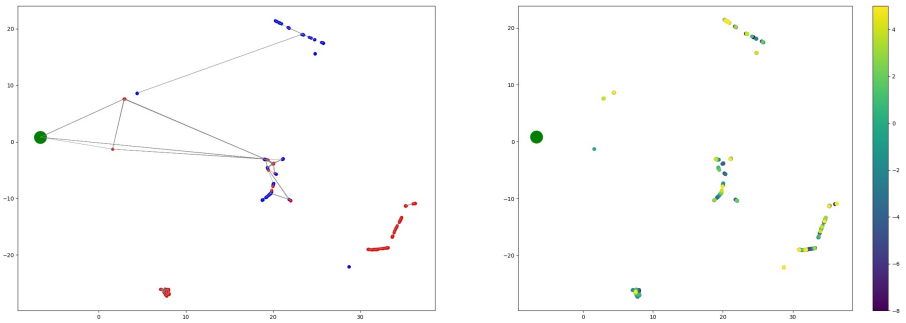


(b) FRCNN detector.

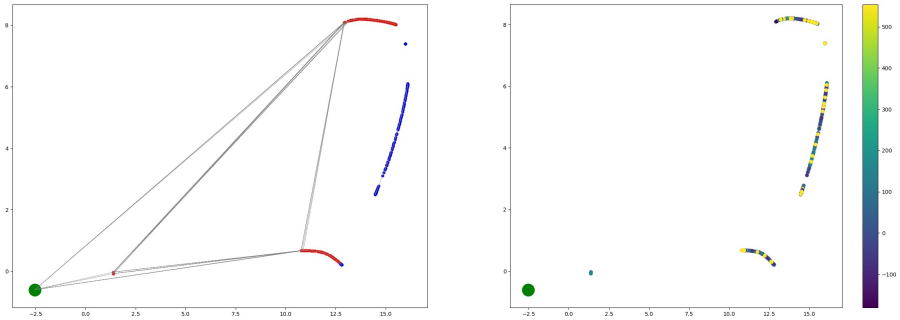


(c) SDP detector.

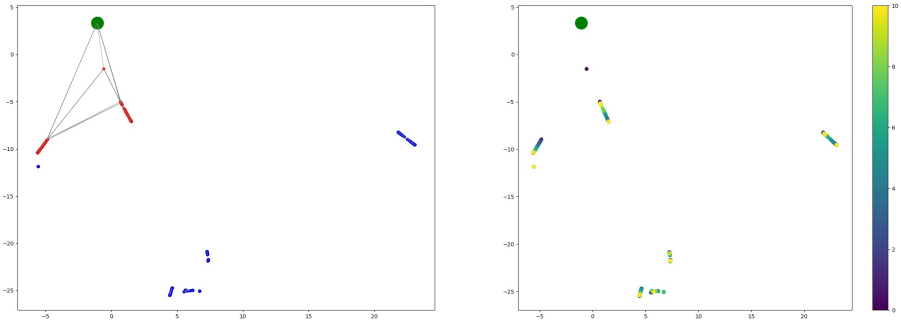
Figure B.15: KNN graph plots of IPS (index half) used during training using spatial dataset as input and pre-filtered frame-distance KNN graph.



(a) DPM detector.



(b) FRCNN detector.



(c) SDP detector.

Figure B.16: Feature embedding plots of IPS (index half) used during training using spatial dataset as input and pre-filtered frame-distance KNN graph.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Masterarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 25.11.2020 Unterschrift