Unifying Short and Long-Term Tracking with Graph Hierarchies – Supplementary Material –

Orcun Cetintas[1*] Guillem Brasó[12*] Laura Leal-Taixé[1†]
[1]Technical University of Munich [2]Munich Center for Machine Learning

Abstract

In this supplementary material, we provide further details about our method.

1. Additional Details about SUSHI

1.1. Edge association cues

As explained in Section 4.2, we feed an initial vector of concatenated pairwise association features to a light-weight multi-layer perception MLP_{edge} to compute input edge embeddings in each SUSHI block. Specifically, we obtain these features from tracklets and embed information about time distance, reID-based appearance similarity, spatial and motion-based proximity between two nodes.

Time and position information. Similarly to [3], we encode the relative position and time distance among nodes as initial edge features. We naturally extend this notion from single detections to tracks by considering the closest detections in time for each pair of tracks. Formally, given the box coordinate and timestamps of two tracklets T_u and T_v, defined as $u = \{(x_i, y_i, w_i, h_i, t_i)\}_{i=u_1}^{u_{n_u}}$ and $v = \{(x_i, y_i, w_i, h_i, t_i)\}_{i=v_1}^{v_{n_v}}$, assuming $t_{u_{n_u}} < t_{v_1}$ *i.e.* T_u ends before T_v starts, we compute the following position features:

$$\left(\frac{2(x_{n_u} - x_{v_1})}{h_{u_{n_u}} + h_{v_1}}), \frac{2(y_{u_{n_u}} - y_{v_1})}{h_{u_{n_u}} + h_{v_1}}), \log\frac{w_{u_{n_u}}}{w_1}, \log\frac{h_{u_{n_u}}}{h_1^v}\right)$$

and we naturally compute time difference as $t_{u_{n_u}} - t_{v_1}$.

Appearance Representation. For every object detection $o_i \in \mathcal{O}$ we obtain embedding ρ_i representing its appearance by feeding its image patch to a pretrained convolutional network, $\rho_i = \text{CNN}_{app}(a_i)$. Intuitively, while single embeddings can be affected by motion blur or sudden illumination changes, a representation summarizing the entire set can be more robust to such phenomena. Hence, we use the euclidean distance among averaged embeddings of tracks as an appearance similarity term $\|\rho^u_{\text{avg}} - \rho^v_{\text{avg}}\|_2$.

Motion consistency. Trajectories are expected to be continuous in the spatio-temporal domain. We utilize this cue by defining an additional edge feature encoding the motion consistency of each pair of tracklets. Given two tracklets T_u and T_v, we estimate their respective velocities in the pixel domain as v_u and v_v, respectively. Assuming again $t_{n_u} < t_{v_1}$, we use the estimated velocities to forward propagate u's last position and backward propagate v's first position until their middle time point $t^{\text{mid}} := (t_{v_1} - t_{u_{n_u}})/2$, to minimize the prediction horizon from each track. Formally, we compute $pos^{\text{fwrd}}_{u\to v} := b_{u_{n_u}} + t^{\text{mid}} v_u$ and $pos^{\text{bwrd}}_{v\to u} := b_{v_1} - t^{\text{mid}} v_v$, to obtain the edge feature $GIoU(pos^{\text{fwrd}}_{u\to v}, pos^{\text{bwrd}}_{v\to u})$, where $GIoU$ denotes the Generalized Intersection over Union score [4]. We choose the $GIoU$ score over the commonly used Intersection over Union because the former still provides a meaningful signal whenever two boxes do not intersect.

As explained in the main paper SUSHI blocks use the same edge features and their GNNs share weights. The only exception is the first level as motion features are not available at this level becasue each node represents a single detection.

1.2. Additional implementation details

Clip stitching. As explained in the main paper, SUSHI operates over video clips of 512 frames. To obtain trajectories over video sequences of arbitrary length, we process videos in an overlapping sliding window fashion. More specifically, we set the overlap among windows to be 256 frames and therefore process videos into clips with corresponding frame intervals (1, 512), (257, 768), (513, 1024), and so on. To stitch trajectories in overlapping windows, we use a simple bipartite matching-based algorithm. Let $\mathcal{T}_A$ and $\mathcal{T}_B$ represent the sets of tracks in two overlapping windows, respectively, restricted over the frame interval in which they overlap. Since all trajectories in $\mathcal{T}_A$ and $\mathcal{T}_B$ are built over the same initial set of object detections, for every pair of trajectories $T_A \in \mathcal{T}_A$ and $T_B \in \mathcal{T}_B$, we can consider their IoU *i.e.* the ratio of boxes that they share in common:

$$IoU(T_A, T_B) = \frac{\#(T_A \cap T_B)}{\#(T_A \cup T_B)}$$

Note that whenever trajectory predictions among the two

clips are *consistent*, their IoU will be 1, and whenever they don't share any boxes, it will be 0. Once we have computed the IoU between each pair of trajectories $T_A \in \mathcal{T}_A$ and $T_B \in \mathcal{T}_B$, we define their pairwise cost as:

$$c(T_A, T_B) := \begin{cases} 1 - IoU(T_A, T_B) & \text{if } \#(T_A \cap T_B) > 0 \\ \infty & \text{otherwise.} \end{cases}$$

where the second clause prevents non-overlapping tracks from being matched. Using this formulation, we obtain the min-cost bipartite matching between $\mathcal{T}_A$ and $\mathcal{T}_B$, and assign the same identity to matched trajectories.

Edge pruning. As mentioned in Section 5.2 of the main paper, we define the set of edges of each graph in our hierarchy by considering for each node, its top K nearest neighbors (KNNs) according to a position, appearance and motion-based similarity measure. Making graphs sparse with KNN-based edge filtering helps to reduce the number of edges, and therefore computational burden, as well as improving the edge label imbalance. Intuitively, edges between nodes with drastically different appearance or infeasible motion can be discarded early. However, there is a tradeoff: low values of K might also discard edges belonging to ground truth trajectories in case of noisy features. Notably, single monolithic graphs such as MPNTrack's [3] require high values of K to achieve good performance, while in our framework consisting of relatively smaller graphs $K = 10$ suffices, yielding significantly better label distribution.

Choosing the right distance metric to prune edges is crucial for the overall success of this strategy. Intuitively, nodes that are close should be likely to belong to the same trajectory. While MPNTrack relied solely on the distance among appearance embeddings, we take advantage of two features of our hierarchy: i) in lower hierarchy levels, nodes within the same trajectory tend to be very close in space and time ii) in higher levels, we have motion information, which can help us determine physically unreasonable connections. To exploit these facts, for graphs in the first level of our hierarchy, we simply use the coordinate-based distance between each pair of tracks as a similarity measure. In subsequent levels, for each pair of tracklets T_u and T_v we define their distance for edge pruning as: $d(T_u, T_v) := \lambda d_{\text{app}}(T_u, T_v) + (1-\lambda) d_{\text{motion}}(T_u, T_v)$, where d_{app} is the euclidean distance of their appearance embeddings, and d_{motion} is 1 minus their GIoU score. We empirically set $\lambda = 0.05$.

1.3. Message passing network architecture

Time-aware neural message passing updates As explained in Section 4 of the main paper, at the core of our SUSHI blocks there is a message passing GNN that, given a graph at each level of our hierarchy, takes as input its initial set of node and edge embeddings, and produces new embeddings encoding high-order contextual information, that we later use for edge classification. We now explain them in detail. Formally, for each graph $G^l = (V^l, E^l)$ at level l in our hierarchy, we consider embeddings $h_v^{(0)} \in \mathbb{R}^{d_V}$ and $h_{(u,w)}^{(0)} \in \mathbb{R}^{d_E}$ for every node $v \in V^l$ and edge $(u, w) \in E^l$, with d_V and d_E being their respective dimension. For a fixed number of steps, $s = 1, \ldots, S$ and each node $v \in V^l$ and edge $(u, v) \in E^l$ we do:

$$h_{(u,v)}^{(s)} = \text{MLP}_{\text{edge}}^l \left(\left[h_u^{(s-1)}, \bar{h}_{(u,v)}^{(s-1)}, h_v^{(s-1)} \right] \right) \tag{1}$$

$$m_{u \to v}^{(s)} = \begin{cases} \text{MLP}_{\text{past}}^l \left([h_u^{(s-1)}, \bar{h}_{(u,v)}^{(s)}, h_v^{(s-1)}] \right) \text{ if } t_u^{\text{end}} < t_v^{\text{start}} \\ \text{MLP}_{\text{future}}^l \left([h_u^{(s-1)}, \bar{h}_{(u,v)}^{(s)}, h_v^{(s-1)}] \right) \text{ else.} \end{cases} \tag{2}$$

$$h_v^{(s)} = \text{MLP}_{\text{node}}^l \left(\left[\sum_{u | t_u^{\text{end}} < t_v^{\text{start}}} m_{u \to v}^{(s)}, \sum_{u | t_u^{\text{start}} < t_v^{\text{end}}} m_{u \to v}^{(s)} \right] \right) \tag{3}$$

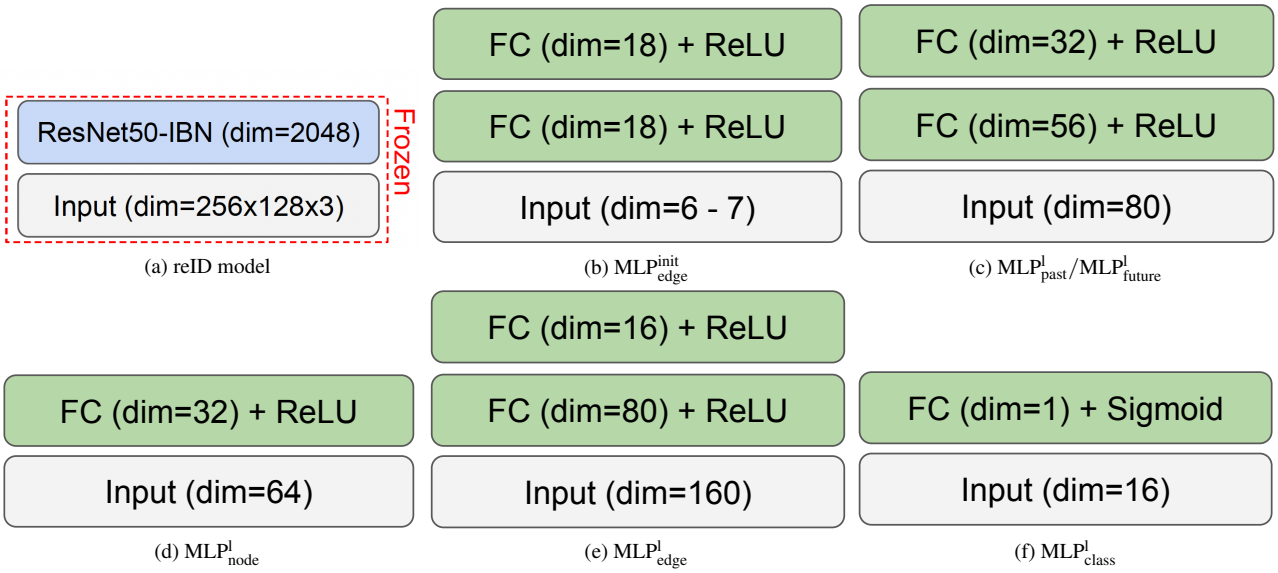
where MLP_*^l denote multi-layer perceptrons that are shared across the entire hierarchy level l, $[*, *]$ denotes concatenation, $\bar{h}_{(u,v)}^{(s)} := [h_{(u,v)}^{(s)}, h_{(u,v)}^{(0)}]$ and t_u^{start} (resp. $t_u^{\text{end}}(u)$) denotes the first (resp. last) timestamp of the tracklet associated to node $u \in V^l$. Intuitively, edges are updated by combining their incident nodes' information. Then nodes are updated by separately aggregating over embeddings from their neighboring incident edges in future and past frames separately, to account for the time directionality. These updates follow the message-passing scheme in [3] and, despite relying on a set of lightweight multi-layer perceptrons, they yield embeddings enabling high-accuracy edge classification.

Detailed MLP architectures. Our SUSHI blocks consist of the MLPs defined above for neural message passing, our edge classifier $\text{MLP}_{\text{class}}$, and an additional MLP used to initialize edge embeddings from their initial features, denoted as $\text{MLP}_{\text{edge}}^{\text{init}}$. All of their exact architectures are detailed in Figure 1. We do not count lernable per-level embeddings due to their negligible cost. Overall, our architecture not including the ResNet50-IBN reID model, has a total of approximately $22K$ parameters, which is notably small for deep learning standards.

1.4. Rounding edge predictions via linear programming

As mentioned in Section 4.2, given a set of edge predictions $y_{(u,v) \in E}^{\text{pred}}$ over a graph $G = (V, E)$, we use a linear programming-based algorithm to obtain a new set of tracklets following [3]. We now explain this algorithm in detail: we first look at the problem constraints and then provide the final algorithm used to enforce them.

Flow conservation-type constraints. Recall that edge predictions aim to approximate the set of edge labels



(a) reID model (b) $\text{MLP}^{\text{init}}_{\text{edge}}$ (c) $\text{MLP}^{\text{l}}_{\text{past}}/\text{MLP}^{\text{l}}_{\text{future}}$

(d) $\text{MLP}^{\text{l}}_{\text{node}}$ (e) $\text{MLP}^{\text{l}}_{\text{edge}}$ (f) $\text{MLP}^{\text{l}}_{\text{class}}$

Figure 1. Detailed architectures of all components of our model.

$\{y^{\text{pred}}_{(u,v)}\}_{(u,v)\in E}$. Further recall that these labels are defined by considering the set of edges coresponding to trajectory-paths in the graph. Specifically, given a time-ordered track $T_k = \{o_{k_i}\}_{i=1}^{n_k}$ with $t_{k_i} < t_{k_{i+1}}$, we consider its corresponding path in G given by its edges $E(T_k) := \{(o_{k_1}, o_{k_2}), \ldots, (o_{k_{n_k-1}}, o_{k_{n_k}})\}$, and hence define, for each $(o_i, o_j) \in E$:

$$y_{(o_i,o_j)} = \begin{cases} 1 & \text{if } \exists T_k \in \mathcal{T}^* \text{ s.t. } (o_i, o_j) \in E(T_k) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Now, notice that since each node (*i.e.* object detection) can belong to at most one trajectory, edge labels need to satisfy the following constraints:

$$\sum_{(o_j,o_i)\in E \text{ s.t. } t_i>t_j} y_{(o_j,o_i)} \leq 1 \quad \forall o_i \in V \tag{5}$$

$$\sum_{(o_i,o_k)\in E \text{ s.t. } t_i<t_k} y_{(o_i,o_k)} \leq 1 \quad \forall o_i \in V \tag{6}$$

Since y's are binary, these constraints state that each node should have, at most, one incident edge labeled as 1 connecting it to a future (resp. past) node, and they are analogous to the conservation constraints used in network flows problems [1].

Projection algorithm. The set of edge predictions, $\{y^{\text{pred}}_{(u,v)}\}_{(u,v)\in E}$ produced by our GNN already satisfies approximately 99% of the aforementioned constraints [3] by simply thresholding them at 0.5. In general, however, having unsatisfied constraints makes it ambiguous to determine the trajectory of an object. In other words, if a node has two positive-labeled edges connecting it to nodes in future locations, it becomes unclear which edge should be selected to form its trajectory. To address these cases, we consider the subgraph of nodes and edges that, after thresholding, violate inequalities 5 or 6, denoted as $\bar{V}$ and $\bar{E}$, respectively, and obtain the closest feasible binary solution y^{round} to our predictions y^{pred} by solving the following integer linear program:

$$\begin{aligned} \min_{y^{\text{round}}} \quad & y^{\text{round}}\left(1 - 2y^{\text{pred}}\right) \\ \text{subject to} \quad & y^{\text{round}} \text{ satisfying ineq. 5 and 6 for all nodes in } \bar{V} \\ & y^{\text{round}}_{(u,v)} \in \{0,1\} \quad \forall (u,v) \in \bar{E} \end{aligned} \tag{7}$$

where we index both y^{round} and y^{pred} indexed for all edges $(u,v) \in \bar{E}$. Note that, since y^{round} is binary, this objective is equivalent to minimizing the euclidean distance $\|y^{\text{round}} - y^{\text{pred}}\|_2$. Moreover, it can be shown that the constraint matrix in 7 is unimodular, and hence solving the linear relaxation of the problem yields a global integer optimum [2]. Overall, Eq. 7 can be very efficiently solved with off-the-shelf linear programming solvers as the graph over which it is defined has very few nodes and edges due to a high percentage of feasible edge predictions produced by our network that can be directly thresholded.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms and applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1993. 3

[2] Jerome Berclaz, Francois Fleuret, Engin Turetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE TPAMI*, 33(9):1806–1819, 2011. 3

[3] Guillem Braso and Laura Leal-Taixe. Learning a neural solver for multiple object tracking. In *CVPR*, 2020. 1, 2, 3

[4] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *CVPR*, pages 658–666, 2019. 1