Self-Supervised Multi-Object Tracking with Cross-Input Consistency

Favyen Bastani  
MIT CSAIL  
favyen@csail.mit.edu

Songtao He  
MIT CSAIL  
songtao@csail.mit.edu

Sam Madden  
MIT CSAIL  
madden@csail.mit.edu

Abstract

In this paper, we propose a self-supervised learning procedure for training a robust multi-object tracking (MOT) model given only unlabeled video. While several self-supervisory learning signals have been proposed in prior work on single-object tracking, such as color propagation and cycle-consistency, these signals cannot be directly applied for training RNN models, which are needed to achieve accurate MOT: they yield degenerate models that, for instance, always match new detections to tracks with the closest initial detections. We propose a novel self-supervisory signal that we call cross-input consistency: we construct two distinct inputs for the same sequence of video, by hiding different information about the sequence in each input. We then compute tracks in that sequence by applying an RNN model independently on each input, and train the model to produce consistent tracks across the two inputs. We evaluate our unsupervised method on MOT17 and KITTI — remarkably, we find that, despite training only on unlabeled video, our unsupervised approach outperforms four supervised methods published in the last 1–2 years, including Tracktor++ [1], FAMNet [5], GSM [18], and mmMOT [29].

1 Introduction

Multi-object trackers identify all instances of a particular object type in video, and track each instance through the segment of video in which it is visible in the camera frame. Annotating training data for multi-object tracking is tedious and costly; for example, annotation of pedestrian tracks in just six minutes of video in the training set of the MOT15 Challenge [14] requires an estimated 22 hours [20] of human labeling time using LabelMe [28]. While unsupervised, heuristic detect-to-track methods [2, 4] have been proposed that group detections into tracks by estimating motion using a combination of spatial and visual cues, these methods suffer low-accuracy in scenarios with frequent occlusion where heuristics are insufficient.

Recent work has proposed applying self-supervised learning for training single-object tracking models on unlabeled video [24, 25]. These approaches train a model to propagate instance labels from a reference frame through the rest of a video sequence. In contrast to work on self-supervised representation learning from video, these fully unsupervised approaches do not require fine-tuning to apply the model for single-object tracking.

However, a significant limitation in prior work is that the model independently compares pairs of frames at a time. In multi-object tracking, a key challenge is robustly re-localizing tracks across potentially long occlusions, especially when an object instance is occluded by other instances of the same object type. Pairwise frame comparisons are thus insufficient for high-accuracy multi-object tracking; instead, learning recurrent features that encode the history of a track is crucial for enabling robust re-localization. However, extending prior work to learn RNN parameters is challenging. For example, Wang et al. [25] propose training using forward-backward consistency: from a patch in an initial frame, after tracking forwards through video and then backwards to return to the initial frame,
the final patch should align with the original patch. Training an RNN in this way would be ineffective as the RNN could simply memorize the features of the original patch.

To address this challenge, we propose a novel self-supervised learning method, cross-input consistency. We first compute object detections in each frame of unlabeled video (like unsupervised, heuristic detect-to-track methods, we assume that a robust detector is available). Then, we derive a learning signal from the unlabeled video by sampling a short sequence of contiguous frames from the video, constructing two input variations of that sequence that each hide different information about objects detected in the sequence, and training the tracker to produce consistent tracking outputs when applied independently on each of the two inputs. We propose two alternative input-hiding schemes for computing the input variations: visual-spatial hiding and occlusion-based hiding. Visual-spatial hiding applies the tracker once when only observing spatial inputs (bounding box coordinates in the video frame), and once when only observing visual inputs (pixel values inside detection boxes). Occlusion-based hiding eliminates information about object detections in random intermediate subsequences of frames to simulate occlusion incidents; thus, it constructs two inputs by eliminating different subsequences of detections in each input. After sampling a sequence of video and computing the two input variations under the chosen input-hiding scheme, we apply the tracker model independently on each input, and back-propagate a learning signal that measures the consistency between tracks computed across the two inputs. To attain high consistency, the model must accurately group detections that correspond to the same object: if the model were to instead arbitrarily group detections into tracks, then variations in the inputs would cause the tracker to produce inconsistent outputs.

To implement cross-input consistency, we adapt a now standard RNN model and tracker architecture from prior work [12]: the tracker processes each frame in sequence by matching detections in the current frame with tracks computed up to the previous frame. In prior work, this model is trained under a supervised procedure: they sample a video sequence \( \langle I_0, \ldots, I_n \rangle \) and a track \( t \) in that sequence, and apply the tracker on \( t \) over the sequence. On each frame \( I_j \), the RNN outputs a probability distribution indicating the likelihood that the prefix of a track \( t \) up to \( I_j \) matches with each detection in \( I_j \). Prior work back-propagates the label (i.e., the correct detection of \( t \) in \( I_j \)) under cross entropy loss.

In contrast, under our method, on each training iteration, we propose to sample a sequence \( \langle I_0, \ldots, I_n \rangle \) from a corpus of unlabeled video, and apply the RNN model to compute a transition matrix that specifies the probability that each detection in \( I_0 \) (rows) matches with each detection in \( I_n \) (columns). We select the sequence length \( n \) so that most objects in \( I_0 \) are still visible in \( I_n \). Then, when applying the tracker on two input variations extracted from the sequence, we obtain two transition matrices (one for each input). We compute the dot-product similarity to measure the consistency between these matrices, and back-propagate the negative similarity as a loss function.

We evaluate our approach on the MOT17 and KITTI benchmarks against 9 baselines, including both unsupervised and supervised methods. We train our tracker model using cross-input consistency over a corpus of unlabeled video, which can be cheaply obtained. Like other unsupervised methods, we use an object detector trained on image-level bounding box annotations in COCO [17], but do not use any expensive video-level annotations. We find that, on MOT17, our approach improves both IDF1 and MOTA accuracy over the unsupervised baselines by 14% to 18%. Moreover, remarkably, our fully unsupervised approach outperforms five of the seven supervised methods we compared, even though these methods train on expensive video-level bounding box and track annotations.

Our code is available at https://favyen.com/uns20/

2 Related Work

Self-supervised learning over video has been studied extensively in many contexts. Most work focuses on learning representations of video that can be applied through fine-tuning for tasks such as activity recognition, image classification, and object detection [6, 7, 9, 15, 23, 26]. More closely related to our work, several recent approaches have proposed leveraging widely available unlabeled video to directly train single-object tracking models, without needing fine-tuning [13, 16]. Vondrick et al. [24] train a model to colorize gray-scale video by propagating colors from a colored reference frame. The model is then applied to track objects at inference time by propagating instance IDs instead of colors. Wang et al. [25] train a model to capture correspondence by applying a cycle-consistent loss from
In our novel cross-input consistency method, we derive an learning signal for training an RNN tracker model through a three-step process. We assume that a corpus of unlabeled video is available, along with an object detector for the object category of interest. During pre-processing, we apply the detector on each frame of unlabeled video to compute object detections. Then, during training,
we first repeatedly randomly sample a sequence of contiguous frames from the video, \(\langle I_0, \ldots, I_n\rangle\), where each \(I_k\) is a video frame. Let \(D_k\) be the detections automatically computed in \(I_k\) by the detector, and let \(d^k_i = (im, x, y, w, h)\) be a detection in \(D_k\), where \((x, y, w, h)\) are the 4D spatial coordinates (center point and lengths) of the detection bounding box, and \(im\) is the window of \(I_k\) corresponding to that box. We apply an input-hiding scheme to select two input variations \(A(D), B(D)\) for the video segment, where each variation is a modified sequence of detections in the frames, \(A(D) = \langle D^0, \ldots, D^A_n\rangle, B(D) = \langle D^0, \ldots, D^B_n\rangle\). For example, some detections may be removed entirely, while others may be partially hidden. Second, we apply the tracker model independently on each input variation to derive two probabilistic tracking outputs (one per input), represented as transition matrices. Third, we compare the transition matrices with dot-product similarity to update the RNN parameters.

Figure 1 summarizes our approach.

Below, we first introduce the model architecture that we adapt from prior work in Section 3.1. We then detail our novel training procedure, including the computation of the transition matrices and dot-product loss, in Section 3.2. Finally, we propose two input-hiding schemes for selecting the input variations required by our approach in Section 4.

3.1 Background: Tracker Model

We adopt a tracker model that is similar to prior work [12]. We summarize the architecture in Figure 2. Given a video sequence \(\langle I_0, \ldots, I_n\rangle\), and sets of detections \(D_k = \{d^k_1, \ldots, d^k_m\}\) detected in each frame \(I_k\), to initialize the tracking process, we create a length-1 track \(t_i = (d^k_i)\) for each detection \(d^k_i\) in the first video frame \(I_0\). When processing subsequent frames, we will match the new detections with existing tracks, extending existing tracks if there is a match and initializing new tracks otherwise. Specifically, on each subsequent frame \(I_k\), the model outputs a probability \(p_{i,j}\) that each track \(t_i\) corresponds to each detection \(d^k_j \in D_k\). At inference time, we formulate the problem of matching tracks with detections in \(I_k\) as a bipartite matching problem, where the cost of matching \(t_i\) with \(d^k_j\) is \(1 - p_{i,j}\). We solve this problem and compute a minimum-cost matching using the Hungarian algorithm; for each pair \((t_i, d^k_j)\) in the matching, we append \(d^k_j\) to \(t_i\). For each detection in \(I_k\) that no track matches to, we create a new track for that detection.

The model consists of a CNN, RNN, and matcher network. Together, these components score the likelihood that the \(i\)th track, \(t_i = (d_1^k, \ldots, d_m^k)\), matches with the \(j\)th detection in \(I_k\), \(d_j^k\). We first apply the CNN to derive detection-level features. Given a detection \(d = (im, x, y, w, h)\), the CNN inputs \(im\) resized to \(64 \times 64\), and consists of 6 strided convolutional layers, with ReLU activation in the first 5 layers and linear activation in the last layer. It outputs a 64-vector, which we concatenate with the 4D spatial coordinates to derive a 68-vector detection representation \(f(d)\). Then, we compute track-level features \(f(t_i)\) by applying the RNN (an LSTM with 64 hidden states) over the sequence of detection-level features of detections in the track, \((f(d_1^k), \ldots, f(d_m^k))\). We use the output of the RNN on the last timestep as the track-level features \(f(t_i)\). Finally, we apply a matching network to score the likelihood that \(t_i\) matches \(d_j^k\). The matching network inputs the concatenation of \(f(t_i)\) and \(f(d_j^k)\), applies four fully-connected layers, and outputs a match score.

3.2 Training Procedure

We develop a novel self-supervised learning method for training the model parameters on unlabeled video. During training, we repeatedly sample sequences of video \(\langle I_0, \ldots, I_n\rangle\). We apply one of two input-hiding schemes, which we will detail in the following section, to extract two distinct input variations \(A(D)\) and \(B(D)\) from a sampled video sequence, where each input is a sequence of detections. We then apply the tracker independently on \(A(D)\) and \(B(D)\) to derive two tracking outputs for the same video sequence. In cross-input consistency, we train the model by enforcing similarity between these two outputs.

To represent tracker outputs, we compute an \(|D_0| \times |D_n| + 1\) transition matrix \(M^{(0,n)}\), where \(M^{(0,n)}_{i,j} = \pi_{i,j}\) is the probability that the track \(t_i\) matches \(d^k_j\). We use the last column to represent tracks that are no longer visible in \(I_n\), i.e., \(M^{(0,n)}_{i,|D_n|}\) is the probability that the track \(t_i\) has exited the camera frame. When applying the model over video sequences during training, we update tracks with
new detections based on the scores output by the model on intermediate frames, but do not create additional tracks on frames after \( I_0 \); thus, each track \( t_i \) corresponds directly to a detection \( d^0_i \) in \( I_0 \) (i.e., \( t_i = \{ d^0_i, \ldots \} \)). Thus, we can also think of \( M^{(0,n)} \) as the probability that a detection in the first frame \( d^0_i \) matches a detection in the last frame \( d^n_j \).

Applying the tracker on both input variations yields two transition matrices \( A^{(0,n)} \) and \( B^{(0,n)} \) that match objects detected in \( I_0 \) with those in \( I_n \). We train the model (CNN, RNN, and matching network) to maximize the dot-product similarity between these matrices. In addition, to pushing the model to produce consistent outputs across both inputs, we also design our training method so that the model cannot attain a high similarity score by, for example, saying that all objects visible in \( I_0 \) are no longer visible in \( I_n \).

In our method, it is important that the training sequence length \( n \) be chosen so that, in most sequences, most (but not all) objects in \( I_0 \) are still visible in \( I_n \), but that objects nevertheless move non-trivially during the sequence (so that the tracking task is not too easy). In general, we find that setting \( n \) to one-half of the average time that objects linger in the camera frame works well; this value can be quickly estimated by hand-labeling the duration of a few (e.g., 10-20) objects randomly sampled from the video.

Below, we detail our method to compute transition matrices, and discuss dot-product similarity loss.

**Transition Matrix.** We propose computing a transition matrix \( M^{(0,k)} \) on each frame \( I_k \) to represent the tracker outputs, where \( M_{i,j}^{(0,k)} \) is the probability that the track \( t_i \) matches the detection \( d^k_j \). On intermediate frames, we apply the Hungarian method on \( M^{(0,k)} \) to match detections in \( D_k \) with tracks, updating each track with the matched detection (if any). On the last frame \( I_n \), we use the \( M^{(0,n)} \) matrix produced under different inputs (denoted \( A^{(0,n)} \) and \( B^{(0,n)} \)) to compute and back-propagate a consistency score. Because we do not create new tracks after \( I_0 \) during training, \( M_{i,j}^{(0,n)} \) is the likelihood that \( d^0_i \) and \( d^n_j \) match (since \( t_i \) begins with \( d^0_i \)).

We first construct a score matrix \( S^{(0,k)} \), by computing \( S_{i,j}^{(0,k)} \) as the score (any real number) output by the tracker model given the track \( t_i \) and detection \( d^k_j \). We then transform the score matrix into a probability matrix to derive \( M^{(0,k)} \). We could simply compute \( M^{(0,k)} \) by taking softmax along rows in \( S^{(0,k)} \). However, computing the transition matrix in this way would allow the tracker to cheat and maximize similarity between \( A^{(0,n)} \) and \( B^{(0,n)} \) by simply matching all detections in \( I_0 \) to a single detection \( d^n_j \in D_n \). Indeed, we find that in practice this yields degenerate models.

Thus, instead, we compute \( M^{(0,k),row} \) and \( M^{(0,k),col} \) by applying softmax along rows and columns, respectively, and compute \( M^{(0,k)} = \min (M^{(0,k),row}, M^{(0,k),col}) \):

\[
M^{row}_{i,j} = \frac{\exp(S_{i,j})}{\sum_k \exp(S_{i,k})} \quad M^{col}_{i,j} = \frac{\exp(S_{i,j})}{\sum_k \exp(S_{k,j})} \quad M_{i,j} = \min (M^{row}_{i,j}, M^{col}_{i,j})
\]  
(1)

This produces a transition matrix \( M^{(0,k)} \) that is almost doubly stochastic: rows and columns sum to at most 1, but not necessarily exactly 1. The operation ensures that the model must match each detection in \( I_0 \) to unique detections in \( I_n \) to maximize the consistency score between \( A^{(0,n)} \) and \( B^{(0,n)} \); if two detections in \( I_0 \) are matched to the same detection in \( I_n \), then the columnar softmax would reduce those probabilities in the corresponding matrix to at most 0.5, thereby reducing any dot-products involving those rows.

**Dot-Product Similarity.** We train the RNN tracker by computing two transition matrices \( A^{(0,n)} \) and \( B^{(0,n)} \) over different input variations, and then back-propagating a loss that measures the inconsistency between the matrices. In particular, we use the dot-product to measure the similarity of corresponding rows in the matrices. We define the loss as:

\[
L = -\sum_i \log \sum_j A^{(0,n)}_{i,j} B^{(0,n)}_{i,j}
\]

Here, \( L \) is computed by taking the logarithm of the dot product of corresponding rows in \( A^{(0,n)} \) and \( B^{(0,n)} \), averaged across rows. Note that this is equivalent to the cross-entropy loss between the diagonal matrix and the matrix product of \( A^{(0,n)} \) and the transpose of \( B^{(0,n)} \).
This loss function has several desirable properties. First, the dot-product is maximized when the matrices computed based on different inputs are most similar. This pushes the matching model to learn reasonable visual and spatial tracking constraints, because arbitrarily matching tracks to detections will lead to dissimilarity. Second, the dot-product pushes each matrix to be almost doubly stochastic rather than leaving some rows and columns summing to much less than 1. The model can only produce doubly stochastic $A^{(0,n)}$ and $B^{(0,n)}$ matrices by finding unique detections in $I_n$ for each detection in $I_0$.

**Spatial Mask.** In some cases, training as described above may converge at a local minimum where the model outputs uniform probabilities for all entries in $M$. To mitigate this issue, we add a spatial constraint to the loss that penalizes the tracker when it matches detections that are highly improbable to correspond to the same object based on bounding box positions. We first compute a mask matrix $C^{(0,n)}$ so that $C^{(0,n)}_{i,j} = 0$ if it is “improbable” that $d^{0i}$ matches $d^{ni}$, and $C^{(0,n)}_{i,j} = 1$ otherwise. Then, we compute $L$ as:

$$L = - \sum_i \log \sum_j A_{i,j}^{(0,n)} B_{i,j}^{(0,n)} C_{i,j}^{(0,n)}$$

We determine whether matches in $C$ are improbable by applying a simple floodfill-like algorithm that propagates sets of labels from the first frame $I_0$ to the last frame $I_n$. If the label from a detection $d^{0i}_j$ in $I_0$ does not propagate to a detection $d^{ni}_j$, then it implies there is no sequence of intermediate detections that could form a track between $d^{0i}_j$ and $d^{ni}_j$. In $I_0$, we label each detection $d^{0i}_j$ with a set containing only that detection, i.e., $\{d^{0i}_j\}$. In $I_k$, we label each detection $d^{ki}_j$ with the union of sets of labels of detections $d^{k-l}_j$ in preceding frames $I_{k-l} (1 \leq l \leq 10)$ such that the bounding boxes of $d^{ki}_j$ and $d^{k-l}_j$ intersect. Note that we consider several preceding frames since the detector may occasionally fail to localize an object in an intermediate frame. Then, $C^{(0,n)}_{i,j} = 1$ only if the label set for $d^{ni}_j$ includes $d^{0i}_j$.

**Artificial Detections.** To improve the model’s robustness in learning visual features, we artificially construct additional detections in $I_n$ by pairing the spatial coordinates of detections in $I_n$ with object images selected randomly from frames in the underlying video data that are temporally far from $(I_0, \ldots, I_n)$. Thus, these artificial detections added to $D_n$ have correct spatial coordinates, but include visual cues that do not correspond to any object in $I_0$, and so the tracker model must learn to leverage visual cues so that it does not assign high probabilities in $M^{(0,n)}$ to artificial detections.

We exclude artificial detections in the mask $C$. Then, to perform well under dot-product similarity, the model must learn to leverage visual features to distinguish the correct detections in $I_n$ from artificially constructed ones — a tracker that only considers spatial features would assign half of its probability mass along each row to artificial detections, and thus would be penalized by the loss.

### 4 Input-Hiding Schemes

In this section, we detail two alternative input-hiding schemes for selecting the two input variations, denoted $A(D) = (D^A_0, \ldots, D^A_n)$ and $B(D) = (D^B_0, \ldots, D^B_n)$. Recall that $D$ is the original set of all objects detected in a video sequence $(I_0, \ldots, I_n)$. Although we only introduce two schemes, our cross-input consistency framework is general-purpose, and there may be other input-hiding schemes that offer comparable or better performance.
4.1 Visual-Spatial Hiding

Under visual-spatial hiding, we apply one tracker instance that observes only visual features and one tracker instance that observes only spatial features: in $A(D)$, we set $x = 0$, $y = 0$, $w = 0$, $h = 0$ for all detections (hide all spatial information), and in $B(D)$, we set $im = 0$ (hide the image content).

Training with cross-input consistency forces the model to produce similar outputs between the visual and spatial inputs. We prevent the visual-only instance from focusing on background features when matching with spatial features, the model must learn to match detections based on visual similarity in order to be consistent with matching based on spatial proximity. Similarly, when given spatial features, the model must learn to estimate motion across occlusion, since the visual-only instance would not have difficulty re-localizing a visually distinctive object following frames where it was occluded.

A key issue with visual-spatial hiding is that the visual-only instance can estimate the change in spatial coordinates of an object between two frames by comparing the background of the detection bounding boxes of that object across the frames, similar to optical flow. This reduces performance because the visual-only instance then learns only to process the background rather than learn a robust embedding for contrasting distinct objects. To mitigate the issue, we make adjustments to the training and inference procedures.

Training. We prevent the visual-only instance from focusing on background features when matching detections in $I_n$ with those in $I_0$ by encumbering its ability to aggregate estimates of changes in spatial coordinates across sequences of consecutive frames: although processing the background is sufficient to compute spatial movement between detections that are in close proximity (e.g., detections of the same object between consecutive frames), this strategy fails for detections that share no overlap. Thus, for sufficiently large $n$, where objects move substantially during the sampled video sequence, background processing is only an issue because the tracker can add up changes that it computes between each pair of consecutive frames.

In particular, we eliminate the recurrent unit for the visual-only instance: its matching network inputs visual features for detections in $I_0$ and for detections in $I_n$, and scores each pair of detections in $A^{(0,n)}$ without observing intermediate frames. On the other hand, we make no changes to the spatial-only instance: it computes $B^{(0,n)}$ by processing 4D spatial coordinates for each detection in every frame in the sequence, and employs both the recurrent unit and the matching network. Figure 3 illustrates the training procedure.

Inference. The separation of visual and spatial inputs, and the specialized training procedure that we employ, imposes two challenges during inference. First, because the visual-only and spatial-only instances observe very different inputs, we cannot expect the model to perform well when we provide both inputs—in effect, we have trained two separate models. Second, since we eliminated recurrent features for the visual-only instance during training, the visual-only instance is essentially a re-identification model, and we must decide how to apply it during inference to take advantage of multiple prior observations of a track in previous frames.

To address the first challenge, during inference, for an input video sequence $\langle I_0, \ldots , I_k \rangle$, we independently compute the visual-only tracker output $A^{(0,k)}$ and the spatial-only tracker output $B^{(0,k)}$. We then compute $M^{(0,k)}$ as the minimum of $A^{(0,k)}$ and $B^{(0,k)}$, and use $M^{(0,k)}$ to update tracks before processing the next frame. Taking the minimum for a matching between track $i$ and detection $d_j$
ensures that the final transition probability reflects whichever of the visual features or spatial features that make \( d_k^i \) less likely to match with \( t_i \). This is desirable—for example, two red sedans visible in the same segment of video will have high visual similarity, and we may have to leverage spatial features to distinguish them.

The second challenge is that the visual-only tracker lacks recurrent features, and thus is only able to compare pairs of frames. To address this, when using the visual-only tracker, for each track-detection pair \((t_i, d_k^j)\), we compute 5 scores by applying the model on \( d_k^j \) and 5 randomly selected images associated with the track \( t_i \) in previous frames. We then average these scores to derive \( A_{t_i}^{(0,k)} \). This enables the model to use context from multiple preceding frames when localizing an object in a new frame.

### 4.2 Occlusion-based Hiding

We also experimented with an occlusion-based hiding scheme. Since we found that visual-spatial hiding performs better, we introduce occlusion-based hiding only at a high level here, but include details in the supplementary material.

Occlusion-based hiding produces the variations \( A(D) \) and \( B(D) \) by simulating random occlusion incidents where all detections in occluded frames are eliminated from the input, i.e., if \( I_k \) is occluded for \( A(D) \), then \( D_A^k \) is empty. We only occlude intermediate frames (i.e., a frame \( I_k \) is only considered for occlusion if \( 0 < k < n \)) so that the transition matrices still compare detections in \( I_0 \) with those in \( I_n \). When processing an occluded \( I_k \), the tracker is forced to match all tracks to the absent column in that frame, and re-localize the tracks after the occlusion. In occlusion-based hiding, we also incorporate an RNN hand-off method that cuts off the propagation of RNN features from \( I_0 \) to \( I_n \) by employing two separate RNN executions: for some handoff index \( 1 < h < n \), we apply the model from \( I_0 \) to \( I_h \), and separately apply the model from \( I_h \) to \( I_n \), and combine the transition matrices by taking their product. We summarize the scheme in Figure 4.

### 5 Evaluation

We compare our method and nine baselines on the MOT17 [21] and KITTI [10] benchmarks.

**Baselines.** We compare with two unsupervised methods (SORT [2] and V-IOU [4]) and seven fully supervised methods (Tracktor++ [1], MHT-BLSTM [12], FAMNet [5], LSST [8], GSM [18], mmMOT [29], and CenterTrack [30]). Like our approach, SORT and V-IOU require an object detector, but do not train on any video-level bounding box and track annotations in the MOT17 and KITTI training sets. The fully supervised methods train on video-level annotations; Tracktor++ incorporates a core component that uses only the detector regression network, but requires video-level annotations for training a re-identification network. Results for 8 baselines are available on MOT17, and results for 4 baselines are available on KITTI.

**Dataset.** MOT17 [21] consists of 14 video sequences of pedestrians in a wide range of contexts, including a moving camera inside a shopping mall and a fixed, elevated view of an outdoor plaza. The dataset is split into 7 training sequences and 7 test sequences; each split includes approximately 11 minutes of video. KITTI [10] consists of 48 video sequences captured from vehicle-mounted cameras, split into 20 for training and 28 for testing, and the objective is to track cars.

**Training.** The supervised baselines train on video-level bounding box and track annotations provided by MOT17 and KITTI. In contrast, our method trains only on a corpus of unlabeled video. Because video-level annotations are expensive to label, our method requires substantially less annotation time, and thus greatly reduces the effort needed to apply multi-object tracking on new datasets.

For MOT17, we collect unlabeled video from two sources: we use five hours of video from seven YouTube walking tours, and all train and test sequences from the PathTrack dataset [20] (we do not use the PathTrack ground truth annotations). For KITTI, we use both the 46 minutes of video in the KITTI dataset together with 7 hours of video from Berkeley DeepDrive [27]. We train our tracker model on an NVIDIA Tesla V100 GPU; training time varies between 4 and 24 hours depending on the input-hiding scheme. During training, we randomly select sequence lengths \( n \) between 4 and 16.
<table>
<thead>
<tr>
<th>Method</th>
<th>IDF1</th>
<th>MOTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occlusion (ours)</td>
<td>52.4</td>
<td>56.7</td>
</tr>
<tr>
<td>Visual-Spatial (ours)</td>
<td>57.3</td>
<td>60.2</td>
</tr>
<tr>
<td>Spatial-Only (ours)</td>
<td>56.5</td>
<td>57.8</td>
</tr>
</tbody>
</table>

Table 1: Ablation study on the MOT17 training set.

<table>
<thead>
<tr>
<th>Method</th>
<th>IDF1</th>
<th>MOTA</th>
<th>MT</th>
<th>ML</th>
<th>FP</th>
<th>FN</th>
<th>Idsw</th>
<th>Frag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsupervised Methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual-Spatial (ours)</td>
<td><strong>58.3</strong></td>
<td><strong>56.8</strong></td>
<td>538</td>
<td>880</td>
<td>12K</td>
<td>231K</td>
<td>1K</td>
<td>2K</td>
</tr>
<tr>
<td>SORT [2]</td>
<td>39.8</td>
<td>43.1</td>
<td>295</td>
<td>997</td>
<td>28K</td>
<td>288K</td>
<td>5K</td>
<td>7K</td>
</tr>
<tr>
<td>IOU [3]</td>
<td>39.4</td>
<td>45.5</td>
<td>369</td>
<td>953</td>
<td>20K</td>
<td>282K</td>
<td>6K</td>
<td>7K</td>
</tr>
<tr>
<td>Tracktor++ [1]</td>
<td>52.3</td>
<td>53.5</td>
<td>459</td>
<td>861</td>
<td>12K</td>
<td>248K</td>
<td>2K</td>
<td>5K</td>
</tr>
<tr>
<td>MHT-BLSTM [12]</td>
<td>51.9</td>
<td>47.5</td>
<td>429</td>
<td>981</td>
<td>26K</td>
<td>268K</td>
<td>2K</td>
<td>3K</td>
</tr>
<tr>
<td>FAMNet [5]</td>
<td>48.7</td>
<td>52.0</td>
<td>450</td>
<td>787</td>
<td>14K</td>
<td>254K</td>
<td>3K</td>
<td>5K</td>
</tr>
<tr>
<td>LSST [8]</td>
<td>62.3</td>
<td>54.7</td>
<td>480</td>
<td>944</td>
<td>26K</td>
<td>228K</td>
<td>1K</td>
<td>4K</td>
</tr>
<tr>
<td>GSM [18]</td>
<td>57.8</td>
<td>56.4</td>
<td>523</td>
<td>813</td>
<td>14K</td>
<td>230K</td>
<td>1K</td>
<td>3K</td>
</tr>
<tr>
<td>CenterTrack [30]</td>
<td>59.6</td>
<td>61.5</td>
<td>621</td>
<td>752</td>
<td>14K</td>
<td>201K</td>
<td>3K</td>
<td>5K</td>
</tr>
</tbody>
</table>

Table 2: Performance on the MOT17 test set. We compare methods in terms of IDF1 and MOTA, but include other non-comprehensive metrics from MOT17 as well for completeness.

<table>
<thead>
<tr>
<th>Method</th>
<th>HOTA</th>
<th>DetA</th>
<th>AssA</th>
<th>DetRe</th>
<th>DetPr</th>
<th>AssRe</th>
<th>AssPr</th>
<th>LocA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual-Spatial (ours)</td>
<td><strong>62.5</strong></td>
<td>61.1</td>
<td>65.3</td>
<td>67.7</td>
<td>73.8</td>
<td>69.1</td>
<td>83.1</td>
<td>80.3</td>
</tr>
<tr>
<td>SORT [2]</td>
<td>42.5</td>
<td>44.0</td>
<td>41.3</td>
<td>47.3</td>
<td>73.9</td>
<td>42.8</td>
<td>83.0</td>
<td>80.8</td>
</tr>
<tr>
<td>FAMNet [5]</td>
<td>52.6</td>
<td>61.0</td>
<td>45.5</td>
<td>64.4</td>
<td>78.7</td>
<td>48.7</td>
<td>77.4</td>
<td>81.5</td>
</tr>
<tr>
<td>mmMOT [29]</td>
<td>62.1</td>
<td>72.3</td>
<td>54.0</td>
<td>76.2</td>
<td>84.9</td>
<td>59.0</td>
<td>82.4</td>
<td>86.6</td>
</tr>
<tr>
<td>CenterTrack [30]</td>
<td>73.0</td>
<td>75.6</td>
<td>71.2</td>
<td>80.1</td>
<td>84.6</td>
<td>73.8</td>
<td>89.0</td>
<td>86.5</td>
</tr>
</tbody>
</table>

Table 3: Performance on the KITTI test set (tracking cars). We show unsupervised methods, including our approach, at the top, and methods that require video-level annotations at the bottom. We use HOTA to compare methods, but include other non-comprehensive metrics from KITTI for completeness.

frames, and apply stochastic gradient descent one sequence at a time. We apply the Adam optimizer with learning rate 0.0001, decaying to 0.00001 after plateau.

In contrast to MOT17, KITTI does not provide object detections for use by tracking methods. We extract detections from video using a YOLOv5 model trained on COCO. On MOT17, we pre-process the provided Deformable Parts Model, Faster R-CNN, and Scale-Dependent Pooling detections with classification and regression following the pre-processing method in Tracktor++ [1].

**Metrics.** We use Multi-Object Tracking Accuracy (MOTA) [21] and ID F1 Score (IDF1) [22] on MOT17, and Higher Order Tracking Accuracy (HOTA) [19] for KITTI. Broadly, these comprehensive metrics measure the accuracy of inferred tracks against ground truth tracks, and penalize both when an inferred track contains a detection that doesn’t match to some ground truth detection (or vice versa), and when a ground truth track is split into two or more inferred tracks (or vice versa). MOT17 and KITTI employ several other non-comprehensive metrics, many of which are used to compute MOTA, IDF1, and HOTA; we report these for completeness.

**Ablation Study.** We first compare occlusion-based hiding and visual-spatial hiding on the MOT17 training set in Table 1. Visual-spatial hiding yields higher performance on both MOTA and IDF1 — because objects are often visible in the video for only a short duration, when training under occlusion-based hiding, the model is unable to learn to re-localize objects over simulated occlusions since the simulated occlusion must then also be short. Under Spatial-Only, we show results for visual-spatial hiding when inputting only the spatial coordinates of detections during inference (no image features).
Figure 5: Output of Visual-Spatial on a portion of an MOT17 sequence. Our method tracks objects through several instances of occlusion.

**Quantitative Results.** Table 2 shows results on the MOT17 test set and Table 3 shows results on the KITTI test set. Metrics are automatically computed by the challenge websites. Per the challenge policies, we only submit the best method, and thus show Visual-Spatial performance.

On MOT17, our approach substantially outperforms both of the unsupervised baselines. Moreover, despite training only on unlabeled video, our method outperforms Tracktor++ [1], MHT-BLSTM [12], FAMNet [5], and GSM [18], even though these baselines (all of which except MHT-BLSTM were published in the last 1–2 years) are supervised methods that train on expensive video-level annotations in the MOT17 training set. Our approach is also competitive with LSST [8], yielding higher MOTA but lower IDF1. Nevertheless, CenterTrack [30] yields slightly higher accuracy on both metrics.

Similarly, on KITTI, our approach outperforms SORT [2], FAMNet [5], and mmMOT [29], but yields lower performance than CenterTrack [30].

**Qualitative Results.** We show qualitative results in Figure 5.

**Additional Experiments.** In the supplementary material, we report results for five additional experiments, where we compare MOTA on the MOT17 training set when various experimental parameters are changed, including detector accuracy, unlabeled video corpus size, and the training example sequence length $n$.

6 Conclusion

In this paper, we have shown that a robust, fully unsupervised multi-object tracker can be trained through a novel self-supervisory learning signal, cross-input consistency, that enforces consistency in the tracking outputs across different input variations of one video sequence. Despite training only on unlabeled video, our approach outperforms four supervised trackers published in the last 1–2 years (Tracktor++ [1], FAMNet [5], GSM [18], and mmMOT [29]), which train on expensive video-level bounding box and track annotations.

**Social Impact.** By enabling a robust multi-object tracker to be trained given only unlabeled video, our work promises to greatly reduce the effort for users to apply multi-object tracking on new datasets without sacrificing accuracy. Thus, we believe that our novel self-supervised MOT method can open up new video analytics tasks that were previously too costly. This impact may be positive or negative depending on the nature of these tasks — however, in general, we believe that tasks with greater potential for negative impact such as surveillance and pedestrian tracking would not benefit from the reduction in annotation cost associated with our method.

**Funding Transparency Statement.** This research was supported in part by the Qatar Computing Research Institute (QCRI).

---

1These results are taken from [https://motchallenge.net/results/MOT17/](https://motchallenge.net/results/MOT17/) where our method is denoted UNS20regress. Baselines are denoted SORT17, IOU17, Tracktor++, MHT_BLSTM, FAMNet, LSST17, GSM_Tracktor, and CTTrackPub.

2These results are taken from [http://www.cvlibs.net/datasets/kitti/eval_tracking.php](http://www.cvlibs.net/datasets/kitti/eval_tracking.php)
References


