Tracking by Animation: Unsupervised Learning of Multi-Object Attentive Trackers

Zhen He[1,2,3*] Jian Li[2] Daxue Liu[2] Hangen He[2] David Barber[3,4]

[1]Academy of Military Medical Sciences
[2]National University of Defense Technology
[3]University College London
[4]The Alan Turing Institute

Abstract

Online Multi-Object Tracking (MOT) from videos is a challenging computer vision task which has been extensively studied for decades. Most of the existing MOT algorithms are based on the Tracking-by-Detection (TBD) paradigm combined with popular machine learning approaches which largely reduce the human effort to tune algorithm parameters. However, the commonly used supervised learning approaches require the labeled data (e.g., bounding boxes), which is expensive for videos. Also, the TBD framework is usually suboptimal since it is not end-to-end, i.e., it considers the task as detection and tracking, but not jointly. To achieve both label-free and end-to-end learning of MOT, we propose a Tracking-by-Animation *framework, where a differentiable neural model first tracks objects from input frames and then animates these objects into reconstructed frames. Learning is then driven by the reconstruction error through backpropagation. We further propose a* Reprioritized Attentive Tracking *to improve the robustness of data association. Experiments conducted on both synthetic and real video datasets show the potential of the proposed model. Our project page is publicly available at:* https://github.com/zhen-he/tracking-by-animation

1. Introduction

We consider the problem of online 2D multi-object tracking from videos. Given the historical input frames, the goal is to extract a set of 2D object bounding boxes from the current input frame. Each bounding box should have an one-to-one correspondence to an object and thus should not change its identity across different frames.

MOT is a challenging task since one must deal with: (i) unknown number of objects, which requires the tracker to be correctly reinitialized/terminated when the object appears/disappears; (ii) frequent object occlusions, which require the tracker to reason about the depth relationship among objects; (iii) abrupt pose (e.g., rotation, scale, and position), shape, and appearance changes for the same object, or similar properties across different objects, both of which make data association hard; (iv) background noises (e.g., illumination changes and shadows), which can mislead tracking.

To overcome the above issues, one can seek to use expressive features, or improve the robustness of data association. E.g., in the predominant Tracking-by-Detection (TBD) paradigm [1, 21, 7, 8], well-performed object detectors are first applied to extract object features (e.g., potential bounding boxes) from each input frame, then appropriate matching algorithms are employed to associate these candidates of different frames, forming object trajectories. To reduce the human effort to manually tune parameters for object detectors or matching algorithms, many machine learning approaches are integrated into the TBD framework and have largely improved the performance [68, 54, 53, 39]. However, most of these approaches are based on supervised learning, while manually labeling the video data is very time-consuming. Also, the TBD framework does not consider the feature extraction and data association jointly, i.e., it is not end-to-end, thereby usually leading to suboptimal solutions.

In this paper, we propose a novel framework to achieve both label-free and end-to-end learning for MOT tasks. In summary, we make the following contributions:

- We propose a *Tracking-by-Animation (TBA)* framework, where a differentiable neural model first tracks objects from input frames and then animates these objects into reconstructed frames. Learning is then driven by the reconstruction error through backpropagation.
- We propose a *Reprioritized Attentive Tracking (RAT)* to mitigate overfitting and disrupted tracking, improving the robustness of data association.
- We evaluate our model on two synthetic datasets (MNIST-MOT and Sprites-MOT) and one real dataset (DukeMTMC [49]), showing its potential.

*Correspondence to Zhen He (email: hezhen.cs@gmail.com).

2. Tracking by Animation

Our TBA framework consists of four components: (i) a *feature extractor* that extracts input features from each input frame; (ii) a *tracker array* where each tracker receives input features, updates its state, and emits outputs representing the tracked object; (iii) a *renderer* (parameterless) that renders tracker outputs into a reconstructed frame; (iv) a *loss* that uses the reconstruction error to drive the learning of Components (i) and (ii), both label-free and end-to-end.

2.1. Feature Extractor

To reduce the computation complexity when associating trackers to the current observation, we first use a neural network NN^{feat}, parameterized by $\boldsymbol{\theta}^{feat}$, as a feature extractor to *compress* the input frame at each timestep $t \in \{1, 2, \ldots, T\}$:

$$\boldsymbol{C}_t = \mathrm{NN}^{feat}\left(\boldsymbol{X}_t; \boldsymbol{\theta}^{feat}\right) \tag{1}$$

where $\boldsymbol{X}_t \in [0,1]^{H \times W \times D}$ is the input frame of height H, width W, and channel size D, and $\boldsymbol{C}_t \in \mathbb{R}^{M \times N \times S}$ is the extracted input feature of height M, width N, and channel size S, containing much fewer elements than $\boldsymbol{X}_t$.

2.2. Tracker Array

The tracker array comprises I neural trackers indexed by $i \in \{1, 2, \ldots, I\}$ (thus I is the maximum number of tracked objects). Let $\boldsymbol{h}_{t,i} \in \mathbb{R}^R$ be the state vector (vectors are assumed to be in row form throughout this paper) of Tracker i at time t, and $\mathcal{H}_t = \{\boldsymbol{h}_{t,1}, \boldsymbol{h}_{t,2}, \ldots, \boldsymbol{h}_{t,I}\}$ be the set of all tracker states. Tracking is performed by iterating over two stages:

(i) **State Update**. The trackers first associate input features from $\boldsymbol{C}_t$ to update their states $\mathcal{H}_t$, through a neural network NN^{upd} parameterized by $\boldsymbol{\theta}^{upd}$:

$$\mathcal{H}_t = \mathrm{NN}^{upd}\left(\mathcal{H}_{t-1}, \boldsymbol{C}_t; \boldsymbol{\theta}^{upd}\right) \tag{2}$$

Whilst it is straightforward to set NN^{upd} as a Recurrent Neural Network (RNN) [52, 16, 11] (with all variables vectorized), we introduce a novel RAT to model NN^{upd} in order to increase the robustness of data association, which will be discussed in Sec. 3.

(ii) **Output Generation**. Then, each tracker generates its output from $\boldsymbol{h}_{t,i}$ via a neural network NN^{out} parameterized by $\boldsymbol{\theta}^{out}$:

$$\mathcal{Y}_{t,i} = \mathrm{NN}^{out}\left(\boldsymbol{h}_{t,i}; \boldsymbol{\theta}^{out}\right) \tag{3}$$

where NN^{out} is shared by all trackers, and the output $\mathcal{Y}_{t,i} = \left\{y^c_{t,i}, \boldsymbol{y}^l_{t,i}, \boldsymbol{y}^p_{t,i}, \boldsymbol{Y}^s_{t,i}, \boldsymbol{Y}^a_{t,i}\right\}$ is a *mid-level* representation of objects on 2D image planes, including:

Confidence $y^c_{t,i} \in [0,1]$ Probability of having captured an object, which can be thought as a *soft* sign of the trajectory validity (1/0 denotes valid/invalid). When time proceeds, an increase/decrease of $y^c_{t,i}$ can be thought as a *soft* initialization/termination of the trajectory.

Layer $\boldsymbol{y}^l_{t,i} \in \{0,1\}^K$ One-hot encoding of the image layer possessed by the object. We consider each image to be composed of K object layers and a background layer, where higher layer objects occlude lower layer objects and the background is the 0-th (lowest) layer. E.g., when $K = 4$, $\boldsymbol{y}^l_{t,i} = [0,0,1,0]$ denotes the 3-rd layer.

Pose $\boldsymbol{y}^p_{t,i} = [\widehat{s}^x_{t,i}, \widehat{s}^y_{t,i}, \widehat{t}^x_{t,i}, \widehat{t}^y_{t,i}] \in [-1,1]^4$ Normalized object pose for calculating the scale $[s^x_{t,i}, s^y_{t,i}] = [1 + \eta^x \widehat{s}^x_{t,i}, 1 + \eta^y \widehat{s}^y_{t,i}]$ and the translation $[t^x_{t,i}, t^y_{t,i}] = [\frac{W}{2}\widehat{t}^x_{t,i}, \frac{H}{2}\widehat{t}^y_{t,i}]$, where $\eta^x, \eta^y > 0$ are constants.

Shape $\boldsymbol{Y}^s_{t,i} \in \{0,1\}^{U \times V \times 1}$ Binary object shape mask with height U, width V, and channel size 1.

Appearance $\boldsymbol{Y}^a_{t,i} \in [0,1]^{U \times V \times D}$ Object appearance with height U, width V, and channel size D.

In the output layer of NN^{out}, $y^c_{t,i}$ and $\boldsymbol{Y}^a_{t,i}$ are generated by the sigmoid function, $\boldsymbol{y}^p_{t,i}$ is generated by the tanh function, and $\boldsymbol{y}^l_{t,i}$ and $\boldsymbol{Y}^s_{t,i}$ are sampled from the Categorical and Bernoulli distributions, respectively. As sampling is not differentiable, we use the Straight-Through Gumbel-Softmax estimator [26] to reparameterize both distributions so that backpropagation can still be applied.

The above-defined mid-level representation is not only flexible, but also can be directly used for input frame reconstruction, enforcing the output variables to be disentangled (as would be shown later). Note that through our experiments, we have found that the discreteness of $\boldsymbol{y}^l_{t,i}$ and $\boldsymbol{Y}^s_{t,i}$ is also very important for this disentanglement.

2.3. Renderer

To define a training objective with only the tracker outputs but no training labels, we first use a differentiable renderer to convert all tracker outputs into reconstructed frames, and then minimize the reconstruction error through backpropagation. Note that we make the renderer both parameterless and deterministic so that correct tracker outputs can be encouraged in order to get correct reconstructions, enforcing the feature extractor and tracker array to learn to generate desired outputs. The rendering process contains three stages:

(i) **Spatial Transformation**. We first scale and shift $\boldsymbol{Y}^s_{t,i}$ and $\boldsymbol{Y}^a_{t,i}$ according to $\boldsymbol{y}^p_{t,i}$ via a Spatial Transformer Network (STN) [25]:

$$\boldsymbol{T}^s_{t,i} = \mathrm{STN}\left(\boldsymbol{Y}^s_{t,i}, \boldsymbol{y}^p_{t,i}\right) \tag{4}$$

$$\boldsymbol{T}^a_{t,i} = \mathrm{STN}\left(\boldsymbol{Y}^a_{t,i}, \boldsymbol{y}^p_{t,i}\right) \tag{5}$$

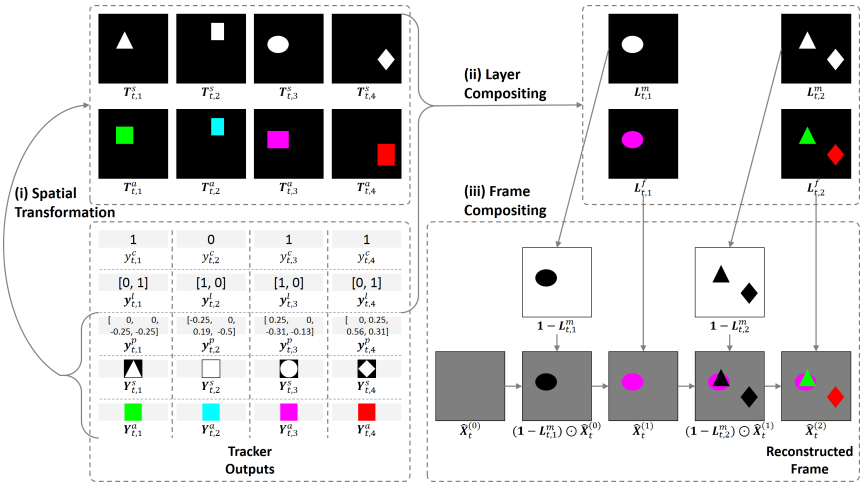

Figure 1: Illustration of the rendering process converting the tracker outputs into a reconstructed frame at time t, where the tracker number $I=4$ and the layer number $K=2$.

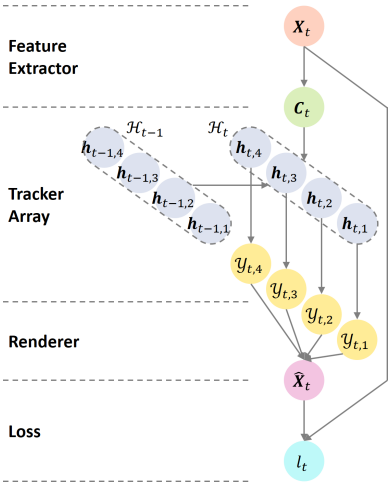

Figure 2: Overview of the TBA framework, where the tracker number $I=4$.

where $\boldsymbol{T}^s_{t,i} \in \{0,1\}^{H\times W\times 1}$ and $\boldsymbol{T}^a_{t,i} \in [0,1]^{H\times W\times D}$ are the spatially transformed shape and appearance, respectively.

(ii) **Layer Compositing**. Then, we synthesize K image layers, where each layer can contain several objects. The k-th layer is composited by:

$$\boldsymbol{L}^m_{t,k} = \min\left(1, \sum_i y^c_{t,i} y^l_{t,i,k} \boldsymbol{T}^s_{t,i}\right) \tag{6}$$

$$\boldsymbol{L}^f_{t,k} = \sum_i y^c_{t,i} y^l_{t,i,k} \boldsymbol{T}^s_{t,i} \odot \boldsymbol{T}^a_{t,i} \tag{7}$$

where $\boldsymbol{L}^m_{t,k} \in [0,1]^{H\times W\times 1}$ is the layer foreground mask, $\boldsymbol{L}^f_{t,k} \in [0,I]^{H\times W\times D}$ is the layer foreground, and $\odot$ is the element-wise multiplication which broadcasts its operands when they are in different sizes.

(iii) **Frame Compositing**. Finally, we iteratively reconstruct the input frame layer-by-layer, i.e., for $k = 1, 2, \ldots, K$:

$$\widehat{\boldsymbol{X}}^{(k)}_t = \left(\mathbf{1} - \boldsymbol{L}^m_{t,k}\right) \odot \widehat{\boldsymbol{X}}^{(k-1)}_t + \boldsymbol{L}^f_{t,k} \tag{8}$$

where $\widehat{\boldsymbol{X}}^{(0)}_t$ is the extracted background, and $\widehat{\boldsymbol{X}}^{(K)}_t$ is the final reconstruction. The whole rendering process is illustrated in Fig. 1, where $\eta^x = \eta^y = 1$.

Whilst the layer compositing can be parallelized by matrix operations, it cannot model occlusion since pixel values in overlapped object regions are simply added; conversely, the frame compositing well-models occlusion, but the iteration process cannot be parallelized, consuming more time and memory. Thus, we combine the two to both reduce the computation complexity and maintain the ability of occlusion modeling. Our key insight is that though the number of occluded objects can be large, the occlusion *depth* is usually small. Thus, occlusion can be modeled efficiently by using a small layer number K (e.g., $K=3$), in which case each layer will be shared by several non-occluded objects.

2.4. Loss

To drive the learning of the feature extractor as well as the tracker array, we define a loss l_t for each timestep:

$$l_t = \text{MSE}\left(\widehat{\boldsymbol{X}}_t, \boldsymbol{X}_t\right) + \lambda \cdot \frac{1}{I} \sum_i s^x_{t,i}\, s^y_{t,i} \tag{9}$$

where, on the RHS, the first term is the reconstruction Mean Squared Error, and the second term, weighted by a constant $\lambda > 0$, is the *tightness* constraint penalizing large scales $[s^x_{t,i}, s^y_{t,i}]$ in order to make object bounding boxes more compact. An overview of our TBA framework is shown in Fig. 2.

3. Reprioritized Attentive Tracking

In this section, we focus on designing the tracker state update network NN^{upd} defined in (2). Although NN^{upd} can be naturally set as a single RNN as mentioned in Sec. 2.2, there can be two issues: (i) *overfitting*, since there is no mechanism to capture the data regularity that similar patterns are usually shared by different objects; (ii) *disrupted tracking*, since there is no incentive to drive each tracker to associate its relevant input features. Therefore, we propose the RAT, which tackles Issue (i) by modeling each tracker independently and sharing parameters for different trackers (this also reduces the parameter number and makes learning more scalable with the tracker number), and tackles Issue (ii) by utilizing attention to achieve explicit data association (Sec. 3.1). RAT also avoids *conflicted tracking* by employing memories to allow tracker interaction (Sec. 3.2) and reprioritizing trackers to make data association more robust

(Sec. 3.3), and improves efficiency by adapting the computation time according to the number of objects presented in the scene (Sec. 3.4).

3.1. Using Attention

To make Tracker i explicitly associate its relevant input features from $\boldsymbol{C}_t$ to avoid disrupted tracking, we adopt a content-based addressing. Firstly, the previous tracker state $\boldsymbol{h}_{t-1,i}$ is used to generate key variables $\boldsymbol{k}_{t,i}$ and $\beta_{t,i}$:

$$\left\{\boldsymbol{k}_{t,i}, \widehat{\beta}_{t,i}\right\} = \text{Linear}\left(\boldsymbol{h}_{t-1,i}; \boldsymbol{\theta}^{key}\right) \tag{10}$$

$$\beta_{t,i} = 1 + \ln\left(1 + \exp\left(\widehat{\beta}_{t,i}\right)\right) \tag{11}$$

where Linear is the linear transformation parameterized by $\boldsymbol{\theta}^{key}$, $\boldsymbol{k}_{t,i} \in \mathbb{R}^S$ is the addressing key, and $\widehat{\beta}_{t,i} \in \mathbb{R}$ is the activation for the key strength $\beta_{t,i} \in (1, +\infty)$. Then, $\boldsymbol{k}_{t,i}$ is used to match each feature vector in $\boldsymbol{C}_t$, denoted by $\boldsymbol{c}_{t,m,n} \in \mathbb{R}^S$ where $m \in \{1, 2, \ldots, M\}$ and $n \in \{1, 2, \ldots, N\}$, to get attention weights:

$$W_{t,i,m,n} = \frac{\exp\left(\beta_{t,i}\, \mathrm{K}\left(\boldsymbol{k}_{t,i}, \boldsymbol{c}_{t,m,n}\right)\right)}{\sum_{m',n'} \exp\left(\beta_{t,i}\, \mathrm{K}\left(\boldsymbol{k}_{t,i}, \boldsymbol{c}_{t,m',n'}\right)\right)} \tag{12}$$

where K is the cosine similarity defined as $\mathrm{K}(\boldsymbol{p}, \boldsymbol{q}) = \boldsymbol{p}\boldsymbol{q}^{\mathsf{T}} / (\|\boldsymbol{p}\|\|\boldsymbol{q}\|)$, and $W_{t,i,m,n}$ is an element of the attention weight $\boldsymbol{W}_{t,i} \in [0, 1]^{M \times N}$, satisfying $\sum_{m,n} W_{t,i,m,n} = 1$. Next, a read operation is defined as a weighted combination of all feature vectors of $\boldsymbol{C}_t$:

$$\boldsymbol{r}_{t,i} = \sum_{m,n} W_{t,i,m,n}\, \boldsymbol{c}_{t,m,n} \tag{13}$$

where $\boldsymbol{r}_{t,i} \in \mathbb{R}^S$ is the read vector, representing the associated input feature for Tracker i. Finally, the tracker state is updated with an RNN parameterized by $\boldsymbol{\theta}^{rnn}$, taking $\boldsymbol{r}_{t,i}$ instead of $\boldsymbol{C}_t$ as its input feature:

$$\boldsymbol{h}_{t,i} = \text{RNN}\left(\boldsymbol{h}_{t-1,i}, \boldsymbol{r}_{t,i}; \boldsymbol{\theta}^{rnn}\right) \tag{14}$$

Whilst each tracker can now attentively access $\boldsymbol{C}_t$, it still cannot attentively access $\boldsymbol{X}_t$ if the receptive field of each feature vector $\boldsymbol{c}_{t,m,n}$ is too large. In this case, it remains hard for the tracker to correctly associate an object from $\boldsymbol{X}_t$. Therefore, we set the feature extractor NN^{feat} as a Fully Convolutional Network (FCN) [37, 70, 61] purely consisting of convolution layers. By designing the kernel size of each convolution/pooling layer, we can control the receptive field of $\boldsymbol{c}_{t,m,n}$ to be a local region on the image so that the tracker can also attentively access $\boldsymbol{X}_t$. Moreover, parameter sharing in FCN captures the spatial regularity that similar patterns are shared by objects on different image locations. As a local image region contains little information about the object translation $[t^x_{t,i}, t^y_{t,i}]$, we add this information by appending the 2D image coordinates as two additional channels to $\boldsymbol{X}_t$.

3.2. Input as Memory

To allow trackers to interact with each other to avoid conflicted tracking, at each timestep, we take the input feature $\boldsymbol{C}_t$ as an external memory through which trackers can pass messages. Concretely, Let $\boldsymbol{C}^{(0)}_t = \boldsymbol{C}_t$ be the initial memory, we arrange trackers to sequentially read from and write to it, so that $\boldsymbol{C}^{(i)}_t$ records all messages written by the past i trackers. In the i-th iteration ($i = 1, 2, \ldots, I$), Tracker i first reads from $\boldsymbol{C}^{(i-1)}_t$ to update its state $\boldsymbol{h}_{t,i}$ by using (10)–(14) (where $\boldsymbol{C}_t$ is replaced by $\boldsymbol{C}^{(i-1)}_t$). Then, an erase vector $\boldsymbol{e}_{t,i} \in [0, 1]^S$ and a write vector $\boldsymbol{v}_{t,i} \in \mathbb{R}^S$ are emitted by:

$$\{\widehat{\boldsymbol{e}}_{t,i}, \boldsymbol{v}_{t,i}\} = \text{Linear}\left(\boldsymbol{h}_{t,i}; \boldsymbol{\theta}^{wrt}\right) \tag{15}$$

$$\boldsymbol{e}_{t,i} = \text{sigmoid}\left(\widehat{\boldsymbol{e}}_{t,i}\right) \tag{16}$$

With the attention weight $\boldsymbol{W}_{t,i}$ produced by (12), we then define a write operation, where each feature vector in the memory is modified as:

$$\boldsymbol{c}^{(i)}_{t,m,n} = \left(\boldsymbol{1} - W_{t,i,m,n}\boldsymbol{e}_{t,i}\right) \odot \boldsymbol{c}^{(i-1)}_{t,m,n} + W_{t,i,m,n}\boldsymbol{v}_{t,i} \tag{17}$$

Our tracker state update network defined in (10)–(17) is inspired by the Neural Turing Machine [18, 19]. Since trackers (controllers) interact through the external memory by using interface variables, they do not need to encode messages of other trackers into their own working memories (i.e., states), making tracking more efficient.

3.3. Reprioritizing Trackers

Whilst memories are used for tracker interaction, it is hard for high-priority (small i) but low-confidence trackers to associate data correctly. E.g., when the first tracker ($i = 1$) is free ($y^c_{t-1,1} = 0$), it is very likely for it to associate or, say, 'steal' a tracked object from a succeeding tracker, since from the unmodified initial memory $\boldsymbol{C}^{(0)}_t$, all objects are equally chanced to be associated by a free tracker.

To avoid this situation, we first update high-confidence trackers so that features corresponding to the tracked objects can be firstly associated and modified. Therefore, we define the priority $p_{t,i} \in \{1, 2, \ldots, I\}$ of Tracker i as its previous (at time $t-1$) confidence ranking (in descending order) instead of its index i, and then we can update Tracker i in the $p_{t,i}$-th iteration to make data association more robust.

3.4. Using Adaptive Computation Time

Since the object number varies with time and is usually less than the tracker number I (assuming I is set large enough), iterating over all trackers at every timestep is inefficient. To overcome this, we adapt the idea of Adaptive Computation Time (ACT) [17] to RAT. At each timestep t, we terminate the iteration at Tracker i (also disable the write operation) once $y^c_{t-1,i} < 0.5$ and $y^c_{t,i} < 0.5$, in which case there are unlikely to be more tracked/new objects. While for the remaining trackers, we do no use them to generate outputs. An illustration of the RAT is shown in Fig. 3. The algorithm of the full TBA framework is presented in Fig. 4.

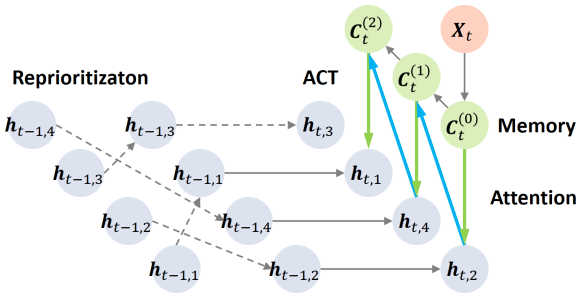

Figure 3: Illustration of the RAT with the tracker number $I = 4$. Green/Blue bold lines denote attentive read/write operations on memory. Dashed arrows denote copy operations. At time t, the iteration is performed by 3 times and terminated at Tracker 1.

# Initialization
for $i \leftarrow 1$ **to** I **do**
  $\boldsymbol{h}_{0,i} \leftarrow \mathbf{0}$
  $y^c_{0,i} \leftarrow 0$
end for
# Forward pass
for $t \leftarrow 1$ **to** T **do**
  # (i) Feature extractor
  extract $\boldsymbol{C}_t$ from $\boldsymbol{X}_t$, see (1)
  # (ii) Tracker array
  $\boldsymbol{C}^{(0)}_t \leftarrow \boldsymbol{C}_t$
  use $y^c_{t-1,1}, y^c_{t-1,2}, \ldots, y^c_{t-1,I}$ to calculate $p_{t,1}, p_{t,2}, \ldots, p_{t,I}$
  for $j \leftarrow 1$ **to** I **do**
    select the i-th tracker whose priority $p_{t,i} = j$
    use $\boldsymbol{h}_{t-1,i}$ and $\boldsymbol{C}^{(j-1)}_t$ to generate $\boldsymbol{W}_{t,i}$, see (10)–(12)
    read from $\boldsymbol{C}^{(j-1)}_t$ according to $\boldsymbol{W}_{t,i}$, and update $\boldsymbol{h}_{t-1,i}$ to $\boldsymbol{h}_{t,i}$, see (13) and (14)
    use $\boldsymbol{h}_{t,i}$ to generate $\mathcal{Y}_{t,i}$, see (3)
    if $y_{t-1,i} < 0.5$ **and** $y_{t,i} < 0.5$ **then**
      break
    end if
    write to $\boldsymbol{C}^{(j-1)}_t$ using $\boldsymbol{h}_{t,i}$ and $\boldsymbol{W}_{t,i}$, obtaining $\boldsymbol{C}^{(j)}_t$, see (15)–(17)
  end for
  # (iii) Renderer
  use $\mathcal{Y}_{t,1}, \mathcal{Y}_{t,2}, \ldots, \mathcal{Y}_{t,I}$ to render $\widehat{\boldsymbol{X}}_t$, see (4)–(8)
  # (iv) Loss
  calculate l_t, see (9)
end for

Figure 4: Algorithm of the TBA framework.

4. Experiments

The main purposes of our experiments are: (i) investigating the importance of each component in our model, and (ii) testing whether our model is applicable to real videos. For Purpose (i), we create two synthetic datasets (MNIST-MOT and Sprites-MOT), and consider the following configurations:

TBA The full TBA model as described in Sec. 2 and Sec. 3.

TBAc TBA with constant computation time, by not using the ACT described in Sec. 3.4.

TBAc-noOcc TBAc without occlusion modeling, by setting the layer number $K = 1$.

TBAc-noAtt TBAc without attention, by reshaping the memory $\boldsymbol{C}_t$ into size $[1, 1, MNS]$, in which case the attention weight degrades to a scalar ($\boldsymbol{W}_{t,i} = W_{t,i,1,1} = 1$).

TBAc-noMem TBAc without memories, by disabling the write operation defined in (15)–(17).

TBAc-noRep TBAc without the tracker reprioritization described in Sec. 3.3.

AIR Our implementation of the 'Attend, Infer, Repeat' (AIR) [13] for qualitative evaluation, which is a probabilistic generative model that can be used to detect objects from individual images through inference.

Note that it is hard to set a supervised counterpart of our model for online MOT, since calculating the supervised loss with ground truth data is *per se* an optimization problem which requires to access complete trajectories and thus is usually done offline [54]. For Purpose (ii), we evaluate TBA on the challenging DukeMTMC dataset [49], and compare it to the state-of-the-art methods. In this paper, we only consider videos with static backgrounds $\widehat{\boldsymbol{X}}^{(0)}_t$, and use the IMBS algorithm [6] to extract them for input reconstruction.

Implementation details of our experiments are given in Appendix A.1. The MNIST-MOT experiment is reported in Appendix A.2. The appendix can be downloaded from our project page.

4.1. Sprites-MOT

In this toy task, we aim to test whether our model can robustly handle occlusion and track the pose, shape, and appearance of the object that can appear/disappear from the scene, providing accurate and consistent bounding boxes. Thus, we create a new Sprites-MOT dataset containing 2M frames, where each frame is of size 128×128×3, consisting of a black background and at most three moving

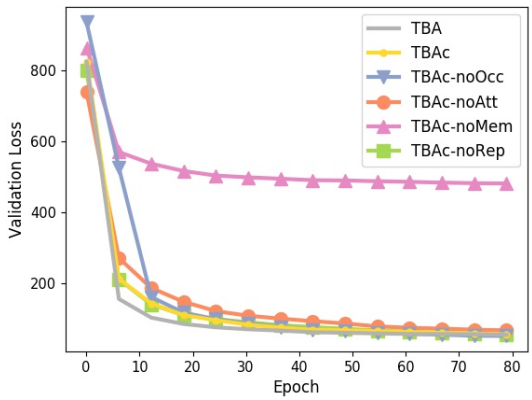

Figure 5: Training curves of different configurations on Sprites-MOT.

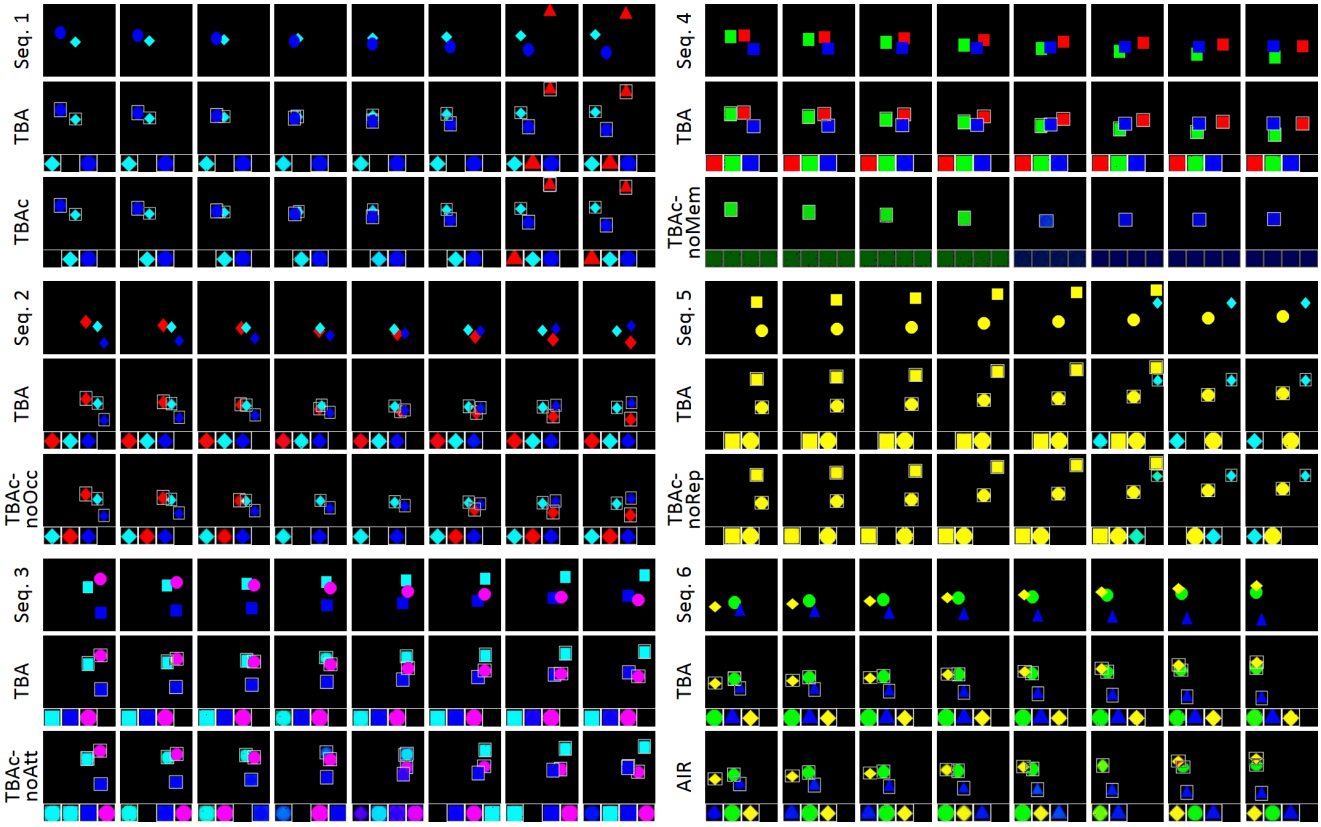

Figure 6: Qualitative results of different configurations on Sprites-MOT. For each configuration, we show the reconstructed frames (top) and the tracker outputs (bottom). For each frame, tracker outputs from left to right correspond to tracker 1 to I (here $I=4$), respectively. Each tracker output $\mathcal{Y}_{t,i}$ is visualized as $\left(y^c_{t,i}\ \boldsymbol{Y}^s_{t,i} \odot \boldsymbol{Y}^a_{t,i}\right) \in [0,1]^{U\times V\times D}$.

Table 1: Tracking performances of different configurations on Sprites-MOT.

Configuration	IDF1↑	IDP↑	IDR↑	MOTA↑	MOTP↑	FAF↓	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag↓
TBA	99.2	99.3	99.2	99.2	79.1	0.01	985	1	60	80	30	22
TBAc	99.0	99.2	98.9	99.1	78.8	0.01	981	0	72	83	36	29
TBAc-noOcc	93.3	93.9	92.7	98.5	77.9	0	969	0	48	227	64	105
TBAc-noAtt	43.2	41.4	45.1	52.6	78.6	0.19	982	0	1,862	198	8,425	89
TBAc-noMem	0	–	0	0	–	0	0	987	0	22,096	0	0
TBAc-noRep	93.0	92.5	93.6	96.9	78.8	0.02	978	0	232	185	267	94

sprites that can occlude each other. Each sprite is randomly scaled from a $21\times21\times3$ image patch with a random shape (circle/triangle/rectangle/diamond) and a random color (red/green/blue/yellow/magenta/cyan), moves towards a random direction, and appears/disappears only once. To solve this task, for TBA configurations we set the tracker number $I=4$ and layer number $K=3$.

Training curves are shown in Fig. 5. TBAc-noMem has the highest validation loss, indicating that it cannot well reconstruct the input frames, while other configurations perform similarly and have significantly lower validation losses. However, TBA converges the fastest, which we conjecture benefits from the regularization effect introduced by ACT.

To check the tracking performance, we compare TBA against other configurations on several sampled sequences, as shown in Fig. 6. We can see that TBA consistently performs well on all situations, where in Seq. 1 TBAc perform as well as TBA. However, TBAc-noOcc fails to track objects from occluded patterns (in Frames 4 and 5 of Seq. 2, the red diamond is lost by Tracker 2). We conjecture the reason is that adding values of occluded pixels into a single layer can result in high reconstruction errors, and thereby the model just learns to suppress tracker outputs when occlusion occurs. Disrupted tracking frequently occurs on TBAc-noAtt which does not use attention explicitly (in Seq. 3, trackers frequently change their targets). For TBAc-noMem, all trackers know nothing about each other and compete for a same object, resulting in identical tracking with low confidences. For TBAc-noRep, free trackers incorrectly associate the objects tracked by the follow-up trackers. Since AIR does not consider the temporal dependency of sequence data, it fails to track objects across different timesteps.

We further quantitatively evaluate different configurations using the standard CLEAR MOT metrics (Multi-Object Tracking Accuracy (MOTA), Multi-Object Tracking Precision (MOTP), etc.) [4] that count how often the tracker makes incorrect decisions, and the recently proposed ID metrics (Identification F-measure (IDF1), Identification Precision (IDP), and Identification Recall (IDR)) [49] that measure how long the tracker correctly tracks targets. Note that we only consider tracker outputs $\mathcal{Y}_{t,i}$ with confidences $y^c_{t,i} > 0.5$ and convert the corresponding poses $\boldsymbol{y}^p_{t,i}$ into object bounding boxes for evaluation. Table 1 reports the tracking performance. Both TBA and TBAc gain good performances and TBA performs slightly better than TBAc. For TBAc-noOcc, it has a significantly higher False Negative (FN) (227), ID Switch (IDS) (64), and Fragmentation (Frag) (105), which is consistent with our conjecture from the qualitative results that using a single layer can sometimes suppress tracker outputs. TBAc-noAtt performs poorly on most of the metrics, especially with a very high IDS of 8425 potentially caused by disrupted tracking. Note that TBAc-noMem has no valid outputs as all tracker confidences are below 0.5. Without tracker reprioritization, TBAc-noRep is less robust than TBA and TBAc, with a higher False Positive (FP) (232), FN (185), and IDS (267) that we conjecture are mainly caused by conflicted tracking.

4.2. DukeMTMC

To test whether our model can be applied to the real applications involving highly complex and time-varying data patterns, we evaluate the full TBA on the challenging DukeMTMC dataset [49]. It consists of 8 videos of resolution 1080×1920, with each split into 50/10/25 minutes long for training/test(hard)/test(easy). The videos are taken from 8 fixed cameras recording movements of people on various places of Duke university campus at 60fps. For TBA configurations, we set the tracker number $I = 10$ and layer number $K = 3$. Input frames are down-sampled to 10fps and resized to 108×192 to ease processing. Since the hard test set contains very different people statistics from the training set, we only evaluate our model on the easy test set.

Fig. 7 shows sampled qualitative results. TBA performs well under various situations: (i) frequent object appearing/disappearing; (ii) highly-varying object numbers, e.g., a single person (Seq. 4) or ten persons (Frame 1 in Seq. 1); (iii) frequent object occlusions, e.g., when people walk towards each other (Seq. 1); (iv) perspective scale changes, e.g., when people walk close to the camera (Seq. 3); (v) frequent shape/appearance changes; (vi) similar shapes/appearances for different objects (Seq. 6).

Quantitative performances are presented in Table 2. We can see that TBA gains an IDF1 of 82.4%, a MOTA of 79.6%, and a MOTP of 80.4% which is the highest, being very competitive to the state-of-the-art methods in performance. However, unlike these methods, our model is the first one free of any training labels or extracted features.

4.3. Visualizing the RAT

To get more insights into how the model works, we visualize the process of RAT on Sprites-MOT (see Fig. 8). At time t, Tracker i is updated in the $p_{t,i}$-th iteration, using its attention weight $\boldsymbol{W}_{t,i}$ to read from and write to the memory $\boldsymbol{C}_t^{(p_{t,i}-1)}$, obtaining $\boldsymbol{C}_t^{(p_{t,i})}$. We can see that the memory content (bright region) related to the associated object is attentively *erased* (becomes dark) by the write operation, thereby preventing the next tracker from reading it again. Note that at time $(t+1)$, Tracker 1 is reprioritized with a priority $p_{t+1,1} = 3$ and thus is updated at the 3-rd iteration, and the memory value has not been modified in the 3-rd iteration by Tracker 1 at which the iteration is terminated (since $y^c_{t,1} < 0.5$ and $y^c_{t+1,1} < 0.5$).

5. Related Work

Unsupervised Learning for Visual Data Understanding There are many works focusing on extracting interpretable representations from visual data using unsupervised learning: some attempt to find low-level disentangled factors ([33, 10, 51] for images and [43, 29, 20, 12, 15] for videos), some aim to extract mid-level semantics ([35, 41, 24] for images and [28, 63, 67, 22] for videos), while the remaining seek to discover high-level semantics ([13, 71, 48, 57, 66, 14] for images and [62, 65] for videos). However, none of these works deal with MOT tasks. To the best of our knowledge, the proposed method first achieves unsupervised end-to-end learning of MOT.

Data Association for online MOT In MOT tasks, data association can be either offline [73, 42, 34, 3, 45, 9, 40] or online [59, 2, 64], deterministic [44, 23, 69] or probabilistic [55, 5, 30, 60], greedy [7, 8, 56] or global [47, 31, 46]. Since the proposed RAT deals with online MOT and uses soft attention to greedily associate data based on tracker confidence ranking, it belongs to the probabilistic and greedy online methods. However, unlike these traditional methods, RAT is learnable, i.e., the tracker array can learn to generate matching features, evolve tracker states, and modify input features. Moreover, as RAT is not based on TBD and is end-to-end, the feature extractor can also learn to provide discriminative features to ease data association.

6. Conclusion

We introduced the TBA framework which achieves unsupervised end-to-end learning of MOT tasks. We also introduced the RAT to improve the robustness of data association. We validated our model on different tasks, showing its potential for real applications such as video surveillance. Our future work is to extend the model to handle videos with dynamic backgrounds. We hope our method could pave the way towards more general unsupervised MOT.

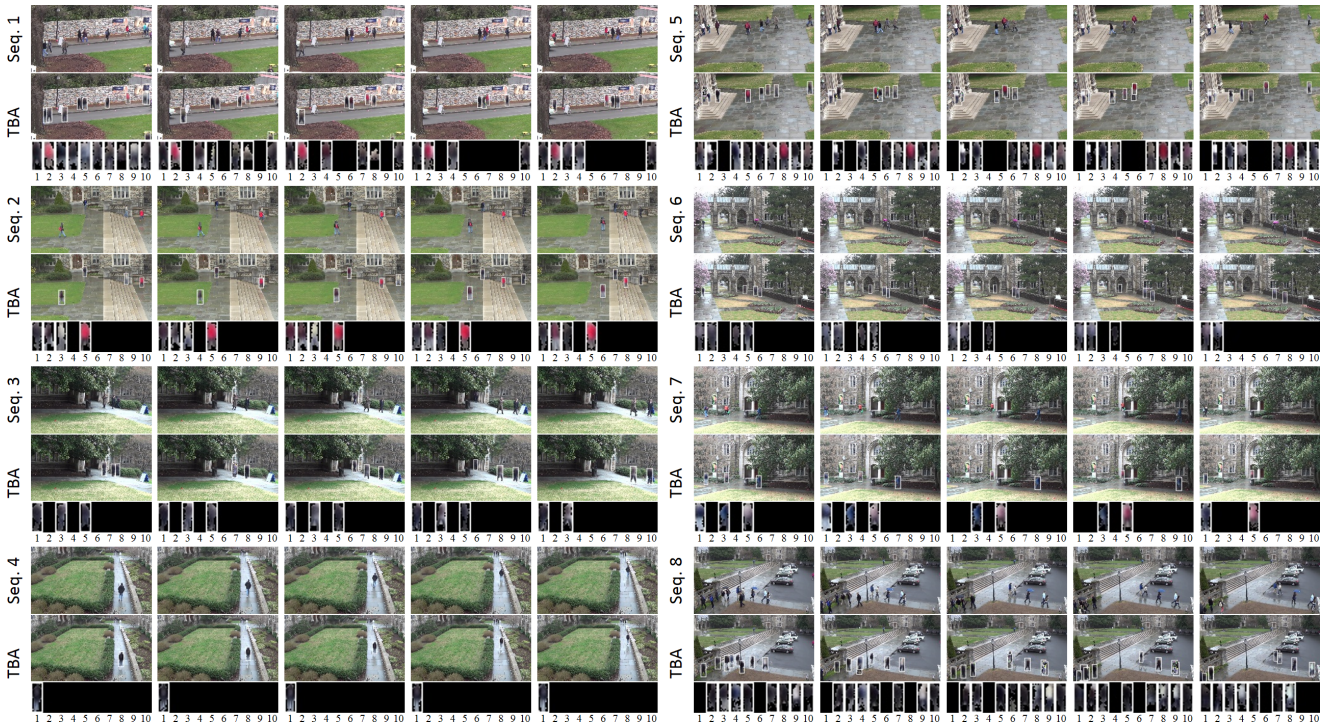

Figure 7: Qualitative results of TBA on DukeMTMC. For each sequence, we show the input frames (top), reconstructed frames (middle), and the tracker outputs (bottom). For each frame, tracker outputs from left to right correspond to tracker 1 to I (here $I=10$), respectively. Each tracker output $\mathcal{Y}_{t,i}$ is visualized as $\left(y^c_{t,i}\, \boldsymbol{Y}^s_{t,i} \odot \boldsymbol{Y}^a_{t,i}\right) \in [0,1]^{U \times V \times D}$.

Table 2: Tracking performances of different methods on DukeMTMC.

Method	IDF1↑	IDP↑	IDR↑	MOTA↑	MOTP↑	FAF↓	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag↓
DeepCC [50]	89.2	91.7	86.7	87.5	77.1	0.05	1,103	29	37,280	94,399	202	753
TAREIDMTMC [27]	83.8	87.6	80.4	83.3	75.5	0.06	1,051	17	44,691	131,220	383	2,428
TBA (ours)*	82.4	86.1	79.0	79.6	80.4	0.09	1,026	46	64,002	151,483	875	1,481
MYTRACKER [72]	80.3	87.3	74.4	78.3	78.4	0.05	914	72	35,580	193,253	406	1,116
MTMC_CDSC [58]	77.0	87.6	68.6	70.9	75.8	0.05	740	110	38,655	268,398	693	4,717
PT_BIPCC [38]	71.2	84.8	61.4	59.3	78.7	0.09	666	234	68,634	361,589	290	783
BIPCC [49]	70.1	83.6	60.4	59.4	78.7	0.09	665	234	68,147	361,672	300	801

* The results are hosted at https://motchallenge.net/results/DukeMTMCT, where our TBA tracker is named as 'MOT_TBA'.

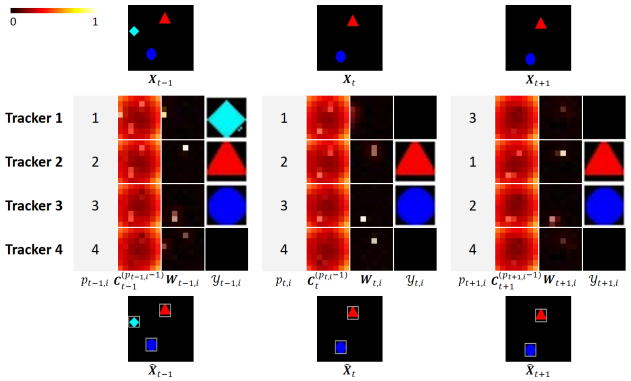

Figure 8: Visualization of the RAT on Sprites-MOT. Both the memory $\boldsymbol{C}_t$ and the attention weight $\boldsymbol{W}_{t,i}$ are visualized as $M \times N$ (8×8) matrices, where for $\boldsymbol{C}_t$ the matrix denotes its channel mean $\frac{1}{S}\sum_{s=1}^{S} \boldsymbol{C}_{t,1:M,1:N,s}$ normalized in $[0,1]$.

References

[1] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. People-tracking-by-detection and people-detection-by-tracking. In *CVPR*, 2008. 1

[2] Seung-Hwan Bae and Kuk-Jin Yoon. Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning. In *CVPR*, 2014. 7

[3] Jerome Berclaz, Francois Fleuret, Engin Turetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE TPAMI*, 33(9):1806–1819, 2011. 7

[4] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *Journal on Image and Video Processing*, 2008:1, 2008. 7

[5] Samuel S Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, 2004. 7

[6] Domenico Bloisi and Luca Iocchi. Independent multimodal background subtraction. In *CompIMAGE*, 2012. 5

[7] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Robust tracking-by-detection using a detector confidence particle filter. In *ICCV*, 2009. 1, 7

[8] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *IEEE TPAMI*, 33(9):1820–1833, 2011. 1, 7

[9] Asad A Butt and Robert T Collins. Multi-target tracking by lagrangian relaxation to min-cost network flow. In *CVPR*, 2013. 7

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016. 7

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 2, 11

[12] Emily L Denton et al. Unsupervised learning of disentangled representations from video. In *NIPS*, 2017. 7

[13] SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *NIPS*, 2016. 5, 7

[14] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 7

[15] Marco Fraccaro, Simon Kamronn, Ulrich Paquet, and Ole Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *NIPS*, 2017. 7

[16] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000. 2

[17] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016. 4

[18] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 4

[19] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. 4

[20] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *NIPS*, 2017. 7

[21] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *ECCV*, 2012. 1

[22] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *NeurIPS*, 2018. 7

[23] Chang Huang, Bo Wu, and Ramakant Nevatia. Robust object tracking by hierarchical association of detection responses. In *ECCV*, 2008. 7

[24] Jonathan Huang and Kevin Murphy. Efficient inference in occlusion-aware generative models of images. In *ICLR Workshop*, 2016. 7

[25] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015. 2

[26] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017. 2

[27] Na Jiang, SiChen Bai, Yue Xu, Chang Xing, Zhong Zhou, and Wei Wu. Online inter-camera trajectory association exploiting person re-identification and camera topology. In *ACM International Conference on Multimedia*, 2018. 8

[28] Nebojsa Jojic and Brendan J Frey. Learning flexible sprites in video layers. In *CVPR*, 2001. 7

[29] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *ICLR*, 2017. 7

[30] Zia Khan, Tucker Balch, and Frank Dellaert. Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE TPAMI*, 27(11):1805–1819, 2005. 7

[31] Suna Kim, Suha Kwak, Jan Feyereisl, and Bohyung Han. Online multi-target tracking by large margin structured learning. In *ACCV*, 2012. 7

[32] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 11

[33] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015. 7

[34] Cheng-Hao Kuo, Chang Huang, and Ramakant Nevatia. Multi-target tracking by on-line learned discriminative appearance models. In *CVPR*, 2010. 7

[35] Nicolas Le Roux, Nicolas Heess, Jamie Shotton, and John Winn. Learning a generative model of images by factoring appearance and shape. *Neural Computation*, 23(3):593–650, 2011. 7

[36] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 11

[37] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 4

[38] Andrii Maksai, Xinchao Wang, Francois Fleuret, and Pascal Fua. Non-markovian globally consistent multi-object tracking. In *ICCV*, 2017. 8

[39] Anton Milan, Seyed Hamid Rezatofighi, Anthony R Dick, Ian D Reid, and Konrad Schindler. Online multi-target tracking using recurrent neural networks. In *AAAI*, 2017. 1

[40] Anton Milan, Stefan Roth, and Konrad Schindler. Continuous energy minimization for multitarget tracking. *IEEE TPAMI*, 36(1):58–72, 2014. 7

[41] Pol Moreno, Christopher KI Williams, Charlie Nash, and Pushmeet Kohli. Overcoming occlusion with inverse graphics. In *ECCV*, 2016. 7

[42] Juan Carlos Niebles, Bohyung Han, and Li Fei-Fei. Efficient extraction of human motion volumes by tracking. In *CVPR*, 2010. 7

[43] Peter Ondrúška and Ingmar Posner. Deep tracking: seeing beyond seeing using recurrent neural networks. In *AAAI*, 2016. 7

[44] AG Amitha Perera, Chukka Srinivas, Anthony Hoogs, Glen Brooksby, and Wensheng Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *CVPR*, 2006. 7

[45] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR*, 2011. 7

[46] Zhen Qin and Christian R Shelton. Improving multi-target tracking via social grouping. In *CVPR*, 2012. 7

[47] Vladimir Reilly, Haroon Idrees, and Mubarak Shah. Detection and tracking of large number of targets in wide area surveillance. In *ECCV*, 2010. 7

[48] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *NIPS*, 2016. 7

[49] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. In *ECCV*, 2016. 1, 5, 7, 8

[50] Ergys Ristani and Carlo Tomasi. Features for multi-target multi-camera tracking and re-identification. In *CVPR*, 2018. 8

[51] Jason Tyler Rolfe. Discrete variational autoencoders. In *ICLR*, 2017. 7

[52] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 2

[53] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. In *ICCV*, 2017. 1

[54] Samuel Schulter, Paul Vernaza, Wongun Choi, and Manmohan Chandraker. Deep network flow for multi-object tracking. In *CVPR*, 2017. 1, 5

[55] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters. *The International Journal of Robotics Research*, 22(2):99–116, 2003. 7

[56] Guang Shu, Afshin Dehghan, Omar Oreifej, Emily Hand, and Mubarak Shah. Part-based multiple-person tracking with partial occlusion handling. In *CVPR*, 2012. 7

[57] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *AAAI*, 2017. 7

[58] Yonatan Tariku Tesfaye, Eyasu Zemene, Andrea Prati, Marcello Pelillo, and Mubarak Shah. Multi-target tracking in multiple non-overlapping cameras using constrained dominant sets. *arXiv preprint arXiv:1706.06196*, 2017. 8

[59] Ryan D Turner, Steven Bottone, and Bhargav Avasarala. A complete variational tracker. In *NIPS*, 2014. 7

[60] B-N Vo and W-K Ma. The gaussian mixture probability hypothesis density filter. *IEEE Transactions on Signal Processing*, 54(11):4091–4104, 2006. 7

[61] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *ICCV*, 2015. 4

[62] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *NIPS*, 2017. 7

[63] John Winn and Andrew Blake. Generative affine localisation and tracking. In *NIPS*, 2005. 7

[64] Bo Wu and Ram Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *IJCV*, 75(2):247–266, 2007. 7

[65] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *NIPS*, 2017. 7

[66] Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *CVPR*, 2017. 7

[67] Jonas Wulff and Michael Julian Black. Modeling blurred video with layers. In *ECCV*, 2014. 7

[68] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *ICCV*, 2015. 1

[69] Junliang Xing, Haizhou Ai, and Shihong Lao. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *CVPR*, 2009. 7

[70] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015. 4

[71] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *NIPS*, 2016. 7

[72] Kwangjin Yoon, Young-min Song, and Moongu Jeon. Multiple hypothesis tracking algorithm for multi-target multi-camera tracking with disjoint views. *IET Image Processing*, 2018. 8

[73] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In *CVPR*, 2008. 7

A. Supplementary Materials for Experiments

A.1. Implementation Details

Model Configuration There are some common model configurations for all tasks. For the NN^{feat} defined in (1), we set it as a FCN, where each convolution layer is composed via convolution, adaptive max-pooling, and ReLU and the convolution stride is set to 1 for all layers. For the RNN defined in (14), we set it as a Gated Recurrent Unit (GRU) [11] to capture long-range temporal dependencies. For the NN^{out} defined in (3), we set it as a Fully-Connected network (FC), where the ReLU is chosen as the activation function for each hidden layer. For the loss defined in (9), we set $\lambda = 1$. For the model configurations specified to each task, please see in Table 3. Note that to use attention, the receptive field of $\boldsymbol{c}_{t,m,n}$ is crafted as a local region on $\boldsymbol{X}_t$, i.e., 40×40 for MNIST-MOT and Sprites-MOT, and 44×24 for DukeMTMC (this can be calculated using the FCN hyper-parameters in Table 3).

Training Configuration For MNIST-MOT and Sprites-MOT, we split the data into a proportion of 90/5/5 for training/validation/test; for DukeMTMC, we split the provided training data into a proportion of 95/5 for training/validation. For all tasks, in each iteration we feed the model with a mini-batch of 64 subsequences of length 20. During the forward pass, the tracker states and confidences at the last time step are preserved to initialize the next iteration. To train the model, we minimize the averaged loss on the training set w.r.t. all model parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}^{feat}, \boldsymbol{\theta}^{upd}, \boldsymbol{\theta}^{out}\}$ using Adam [32] with a learning rate of 5×10^{-4}. Early stopping is used to terminate training.

A.2. MNIST-MOT

As a pilot experiment, we focus on testing whether our model can robustly track the position and appearance of each object that can appear/disappear from the scene. Thus, we create a new MNIST-MOT dataset containing 2M frames, where each frame is of size 128×128×1, consisting of a black background and at most three moving digits. Each digit is a 28×28×1 image patch randomly drawn from the MNIST dataset [36], moves towards a random direction, and appears/disappears only once. When digits overlap, pixel values are added and clamped in $[0, 1]$. To solve this task, for TBA configurations we set the tracker number $I = 4$ and layer number $K = 1$, and fix the scale $s^x_{t,i} = s^y_{t,i} = 1$ and shape $\boldsymbol{Y}^s_{t,i} = \mathbf{1}$, thereby only compositing a single layer by adding up all transformed appearances. We also clamp the pixel values of the reconstructed frames in $[0, 1]$ for all configurations.

Training curves are shown in Fig. 9. The TBA, TBAc, and TBAc-noRep have similar validation losses which are slightly better than that of TBAc-noAtt. Similar to the results on Sprites-MOT, TBA converges the fastest, and TBAc-noMem has a significantly higher validation loss as all trackers are likely to focus on a same object, which affects the reconstruction.

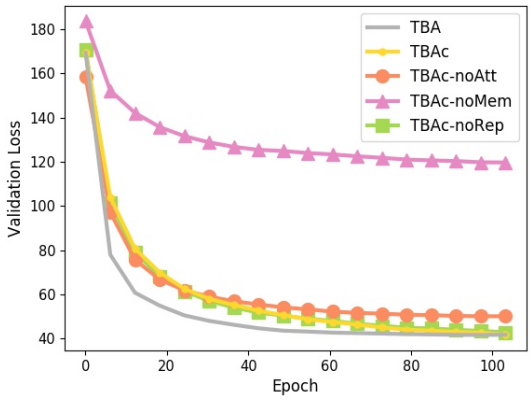

Figure 9: Training curves of different configurations on MNIST-MOT.

Qualitative results are shown in Fig. 10. Similar phenomena are observed as in Sprites-MOT, revealing the importance of the disabled mechanisms. Specifically, as temporal dependency is not considered in AIR, overlapped objects are failed to be disambiguated (Seq. 5).

We further quantitatively evaluate different configurations. Results are reported in Table 4, which are similar to those of the Sprites-MOT.

Table 3: Model configurations specified to each task, where 'conv h×w' denotes a convolution layer with kernel size h×w, 'fc' denotes a fully-connected layer, and 'out' denotes an output layer. Note that for NN^{feat}, the first layer has two additional channels than $\boldsymbol{X}_t$, which are the 2D image coordinates (as mentioned in Sec. 3.1).

Hyper-parameter	**MNIST-MOT**	**Sprites-MOT**	**DukeMTMC**
Size of $\boldsymbol{X}_t$: $[H, W, D]$	[128, 128, 1]	[128, 128, 3]	[108, 192, 3]
Size of $\boldsymbol{C}_t$: $[M, N, S]$	[8, 8, 50]	[8, 8, 20]	[9, 16, 200]
Size of $\boldsymbol{Y}^a_{t,i}$: $[U, V, D]$	[28, 28, 1]	[21, 21, 3]	[9, 23, 3]
Size of $\boldsymbol{h}_{t,i}$: R	200	80	800
Tracker number: I	4	4	10
Layer number: K	1	3	3
Coef. of $[\widehat{s}^x_{t,i}, \widehat{s}^y_{t,i}]$: $[\eta^x, \eta^y]$	[0, 0]	[0.2, 0.2]	[0.4, 0.4]
Layer sizes of NN^{feat} (FCN)	[128, 128, 3] (conv 5×5) [64, 64, 32] (conv 3×3) [32, 32, 64] (conv 1×1) [16, 16, 128] (conv 3×3) [8, 8, 256] (conv 1×1) [8, 8, 50] (out)	[128, 128, 5] (conv 5×5) [64, 64, 32] (conv 3×3) [32, 32, 64] (conv 1×1) [16, 16, 128] (conv 3×3) [8, 8, 256] (conv 1×1) [8, 8, 20] (out)	[108, 192, 5] (conv 5×5) [108, 192, 32] (conv 5×3) [36, 64, 128] (conv 5×3) [18, 32, 256] (conv 3×1) [9, 16, 512] (conv 1×1) [9, 16, 200] (out)
Layer sizes of NN^{out} (FC)	200 (fc) 397 (fc) 787 (out)	80 (fc) 377 (fc) 1772 (out)	800 (fc) 818 (fc) 836 (out)
Number of parameters	1.21 M	1.02 M	5.65 M

Table 4: Tracking performances of different configurations on MNIST-MOT.

Configuration	**IDF1↑**	**IDP↑**	**IDR↑**	**MOTA↑**	**MOTP↑**	**FAF↓**	**MT↑**	**ML↓**	**FP↓**	**FN↓**	**IDS↓**	**Frag↓**
TBA	99.6	99.6	99.6	99.5	78.4	0	978	0	49	49	22	7
TBAc	99.2	99.3	99.2	99.4	78.1	0.01	977	0	54	52	26	11
TBAc-noAtt	45.2	43.9	46.6	59.8	81.8	0.20	976	0	1,951	219	6,762	86
TBAc-noMem	0	–	0	0	–	0	0	983	0	22,219	0	0
TBAc-noRep	94.3	92.9	95.7	98.7	77.8	0.01	980	0	126	55	103	10

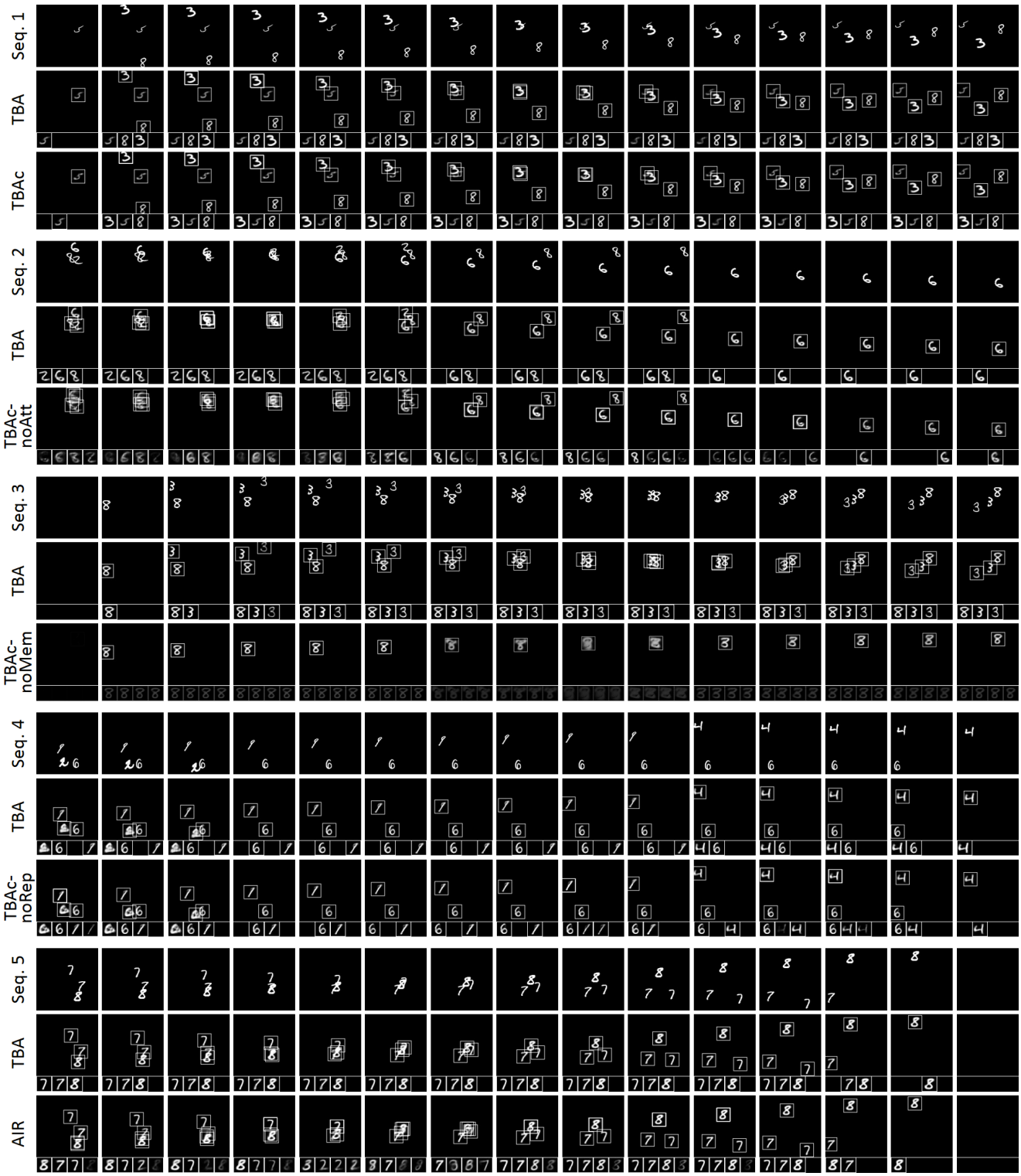

Figure 10: Qualitative results of different configurations on MNIST-MOT. For each configuration, we show the reconstructed frames (top) and the tracker outputs (bottom). For each frame, tracker outputs from left to right correspond to tracker 1 to I (here $I=4$), respectively. Each tracker output $\mathcal{Y}_{t,i}$ is visualized as $\left(y^c_{t,i}\ \boldsymbol{Y}^s_{t,i} \odot \boldsymbol{Y}^a_{t,i}\right) \in [0,1]^{U \times V \times D}$.