Supplementary Material: How To Train Your Deep Multi-Object Tracker

Yihong Xu[1] Aljoša Ošep[2] Yutong Ban[1,3] Radu Horaud[1]
Laura Leal-Taixé[2] Xavier Alameda-Pineda[1]
[1]Inria, LJK, Univ. Grenoble Alpes, France [2]Technical University of Munich, Germany
[3]Distributed Robotics Lab, CSAIL, MIT, USA
[1]{firstname.lastname}@inria.fr [2]{aljosa.osep, leal.taixe}@tum.de [3]yban@csail.mit.edu

A. Implementation Details

A.1. DHN

For training the DHN, we use the RMSprop optimizer [17] with a learning rate of 0.0003, gradually decreasing by 5% every $20{,}000$ iterations. We train DHN for 20 epochs (6 hours on a Titan XP GPU). For the focal loss, we weight zero-class by $w_0 = n_1/(n_0 + n_1)$ and one-class by $w_1 = 1 - w_0$. Here n_0 is the number of zeros and n_1 the number of ones in $\mathbf{A}^*$. We also use a modulating factor of 2 in the focal loss. Once the DHN training converges, we freeze the DHN weights and keep them fixed when training trackers with DeepMOT.

Datasets. To train the DHN, we generate training pairs as follows. We first compute distance matrices $\mathbf{D}$ using ground-truth labels (bounding boxes) and object detections provided by the MOTChallenge datasets (MOT 15-17) [11, 13]. We augment the data by setting all entries, higher than the randomly (with an uniform distribution ranging from 0 to 1) selected threshold, to a large value to discourage these assignments. This way, we obtain a rich set of various distance matrices. We then compute assignments using the (Hungarian algorithm) HA (variant used in [2]) to get the corresponding (binary) assignment matrices $\mathbf{A}^*$, used as a supervisory signal. In this way, we obtain a dataset of matrix pairs ($\mathbf{D}$ and $\mathbf{A}^*$), separated into $114{,}483$ training and $17{,}880$ testing instances.

A.2. Trackers

Datasets. For training object trackers, we use the MOT17 train set. For the ablation studies, we divide the MOT17 into train/val sets. We split each sequence into three parts: the first, one containing 50% of frames, the second one 25%, and the third 25%. We use the first 50% for training data and the last 25% for validation to make sure there is no overlap between the two. In total, we use $2{,}664$ frames for the train set, containing $35{,}836$ ground-truth bounding boxes and 306 identities. For the validation split, we have $1{,}328$ frames with 200 identities. The public object detections (obtained by DPM [6], SDP [18] and Faster RCNN [14] detectors) from the MOTChallenge are used only during tracking.

Training. We use the Adam optimizer [10] with a learning rate of 0.0001. We train the SOTs for 15 epochs (72h), and we train Tracktor (regression head and ReID head) for 18 epochs (12h) on a Titan XP GPU.

Loss Hyperparameters. When training trackers using our DeepMOT loss, we set the base value of $\delta = 0.5$, and the loss balancing factors of $\lambda = 5, \gamma = 2$, as determined on the validation set.

Training Details. To train object trackers, we randomly select one training instance from the sequence that corresponds to a pair of consecutive frames. Then, we initialize object trackers using ground-truth detections and predict track continuations in the next frame. At each time step, we use track predictions and ground-truth bounding boxes to compute $\mathbf{D}$, which we pass to our DHN and, finally, compute loss and back-propagate the gradients to the tracker.

Data Augmentation. We initialize trackers using ground-truth bounding boxes. To mimic the effects of imperfect object detectors and prevent over-fitting, we perform the following data augmentations during the training:

- We randomly re-scale the bounding boxes with a scaling factor ranging from 0.8 to 1.2.
- We add random vertical and horizontal offset vectors (bounding box width and/or height scaled by a random factor ranging from 0 to 0.25).

Training with the ReIDhead. While training Tracktor with our **ReIDhead**, we make the following changes. Instead of selecting a pair of video frames, we randomly select ten consecutive frames. This is motivated by the implementation of external ReID mechanism in [1], where tracker averages appearance features over ten most recent frames. At each training step, we compute representative embedding by averaging embeddings of the past video frames and use

it to compute the cosine distance to the ground-truth object embeddings.

Test-time Track Managment. For the MOT-by-SOT baseline, we use detections from three different detectors (DPM, SDP, and FRCNN) to refine the track predictions. When the IoU between a track prediction and detection is higher than 0.6, we output their average. We also reduce FP in the public detections based on detection scores produced by a Faster RCNN detector. For the birth and death processes, we initialize a new track only when detections appear in 3 consecutive frames, and they have a minimal consecutive IoU overlap of 0.3. Tracks that can not be verified by the detector are marked invisible and are terminated after $K = 60$ frames. For Tracktor, we use the same track management and suppression strategy as proposed in [1].

B. Additional DHN Ablation

We perform DHN ablation using our test split of $17{,}880$ DHN training instances, as explained in Sec. A.1. In addition, we evaluate the generalization of DHN by evaluating performing evaluation using distance matrices, generated during the DeepMOT training process.

Accuracy. We compute the weighted accuracy as (using the same weighting factors w_1 and w_0 as for the loss):

$$\text{WA} = \frac{w_1 n_1^* + w_0 n_0^*}{w_1 n_1 + w_0 n_0}. \tag{1}$$

Here, n_1^* and n_0^* are the number of true and false positives, respectively.

Validity. The output of the matching algorithm should be a permutation matrix; *i.e.*, there should be at most one assignment per row/column. In the case of the HA, this is explicitly enforced via constraints on the solution. To study how well the predicted (discretized) assignment matrices preserve this property, we count the number of rows and columns by the following criteria:

- **Several Assignments (SA)** counts the number of rows/columns that have more than one assignment (when performing column-wise maximum and row-wise maximum, respectively).
- **Missing Assignments (MA)** counts the number of rows/columns that are not assigned (when performing column-wise maximum and row-wise maximum, respectively) when ground-truth assignment matrix $\mathbf{A}^*$ has an assignment or inversely, no assignment in $\mathbf{A}^*$ while $\bar{\mathbf{A}}$ (see below) has an assignment in the corresponding rows/columns.

Discretization. To perform the evaluation, we first need to discretize the soft assignment matrix $\tilde{\mathbf{A}}$, predicted by our DHN to obtain a discrete assignment matrix $\bar{\mathbf{A}}$. There are two possibilities.

(i) For each row of $\bar{\mathbf{A}}$, we set the entry of $\bar{\mathbf{A}}$ corresponding to the largest value of the row to 1 (as long as it exceeds 0.5) and the remaining values are set to 0. We refer to this variant as *row-wise maximum*.

(ii) Analogously, we can perform *column-wise maximum* by processing columns instead of rows.

DHN variants. We compare three different DHN architectures:

(i) Sequential DHN (**seq**, see Fig. 1),

(ii) Parallel DHN (**paral**, see Fig. 2),

(iii) 1D Convolutional DHN (**1d_conv**, see Fig. 3).

The recurrent unit of the two recurrent architectures, **seq** and **paral**, is also ablated between long-short term memory units (**lstm**) [8] and gated recurrent units (**gru**) [4].

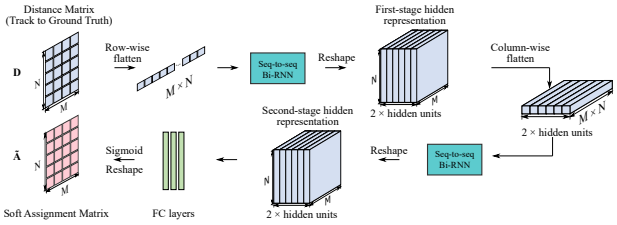

Figure 1. Sequential DHN: Structure of the proposed Deep Hungarian Net. The row-wise and column-wise flattening are inspired by the original Hungarian algorithm, while the Bi-RNN allows for all decisions to be taken globally, thus is accounting for all input entries.

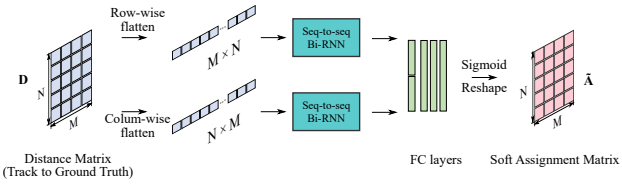

Figure 2. Parallel DHN variant: (i) We perform row-wise and the column-wise flattening of $\mathbf{D}$. (ii) We process the flattened vectors using two different Bi-RNNs. (iii) They then are respectively passed to an FC layer for reducing the number of channels and are concatenated along the channel dimension. (iv) After two FC layers we reshape the vector and apply the sigmoid activation.

From Tab. 1, we see that the proposed sequential DHN (**seq_gru**) obtains the highest WA (92.88% for row-wise maximum and 93.49% for column-wise maximum) compared to others. Compared to the 1D convolutional DHN variant (WA of 56.43% and 56.18% for row-wise and column-wise maximum, respectively), Bi-RNN shows the

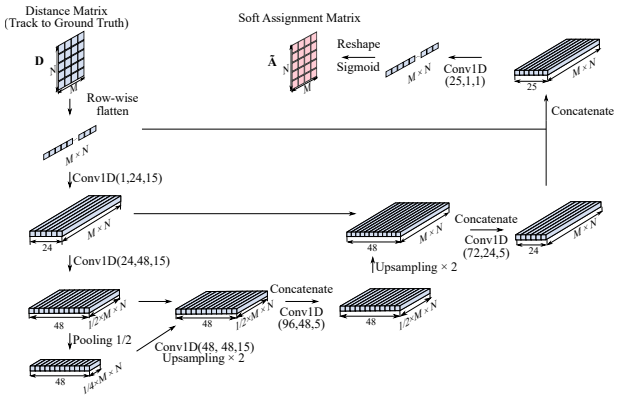

Figure 3. 1D convolutional DHN: Our 1D convolutional DHN variant is inspired by the U-Net [15]. The encoder consists of two 1D-convolution layers of shapes $[1, 24, 15]$ and $[24, 48, 15]$ ([#input channels, #output channels, kernel size]). The decoder consists of two 1D convolutional layers of shapes $[96, 48, 5]$ and $[72, 24, 5]$. Finally, we apply an 1D convolution and a sigmoid activation to produce $\tilde{\mathbf{A}}$.

Discretization	Network	WA % (↑)	MA% (↓)	SA% (↓)
Row-wise maximum	**seq_gru (proposed)**	**92.88**	**4.79**	3.39
	seq_lstm	83.66	13.79	5.98
	paral_gru	89.56	8.21	4.99
	paral_lstm	88.93	8.67	5.38
	1d_conv	56.43	35.06	**2.78**
Column-wise maximum	**seq_gru (proposed)**	**93.49**	**6.41**	26.57
	seq_lstm	87.07	13.54	47.04
	paral_gru	91.01	7.98	46.25
	paral_lstm	90.50	8.60	47.43
	1d_conv	56.18	79.54	**7.73**

Table 1. Evaluation results: comparison of different network structures and settings in terms of WA, MA and SA on the DHN test set.

advantage of its global view due to the receptive field, equal to the entire input. For the sequential DHN setting, we observe in Tab. 1 that **gru** units consistently outperform **lstm** units with WA $+9.22\%$ (row-wise maximum) and $+6.42\%$ (column-wise maximum). Finally, the proposed sequential DHN is more accurate compared to the parallel variant of DHN ($+3.32\%$ for row-wise and $+2.48\%$ for column-wise maximum). As for the validity, the proposed **seq_gru** commits the least missing assignments (MA) (4.79% and 6.41% for row-wise and column-wise maximum, respectively), and commits only a few SA compared to other variants.

DHN is a key component of our proposed DeepMOT training framework. To evaluate how well DHN performs during training as a proxy to deliver gradients from the DeepMOT loss to the tracker, we conduct the following experiment. We evaluate DHN using distance matrices $\mathbf{D}$, collected during the DeepMOT training process. As can be seen in Tab. 2, the proposed sequential DHN (**seq_gru**) outperforms the others variants, with a WA of 92.71% for row-wise and 92.36% for column-wise maximum. For validity, it also attains the lowest MA: 13.17% (row) and 12.21% (column). The SA is kept at a low level with 9.70% and 3.69% for row-wise and column-wise maximum discretizations, respectively. Based on these results, we conclude that (i) our proposed DHN generalizes well to matrices, used to train our trackers, and (ii) it produces outputs that closely resemble valid permutation matrices.

Discretization	Network	WA % (↑)	MA% (↓)	SA% (↓)
Row-wise maximum	**seq_gru (proposed)**	**92.71**	**13.17**	9.70
	seq_lstm	91.64	14.55	10.37
	paral_gru	86.84	23.50	17.15
	paral_lstm	71.58	42.48	22.62
	1d_conv	83.12	32.73	**5.73**
Column-wise maximum	**seq_gru (proposed)**	**92.36**	**12.21**	3.69
	seq_lstm	91.93	13.15	4.71
	paral_gru	87.24	20.56	16.67
	paral_lstm	72.58	39.55	23.16
	1d_conv	82.74	32.94	**1.11**

Table 2. Evaluation results. Comparison of different network structures and settings in terms of WA, MA and SA on distance matrices during training.

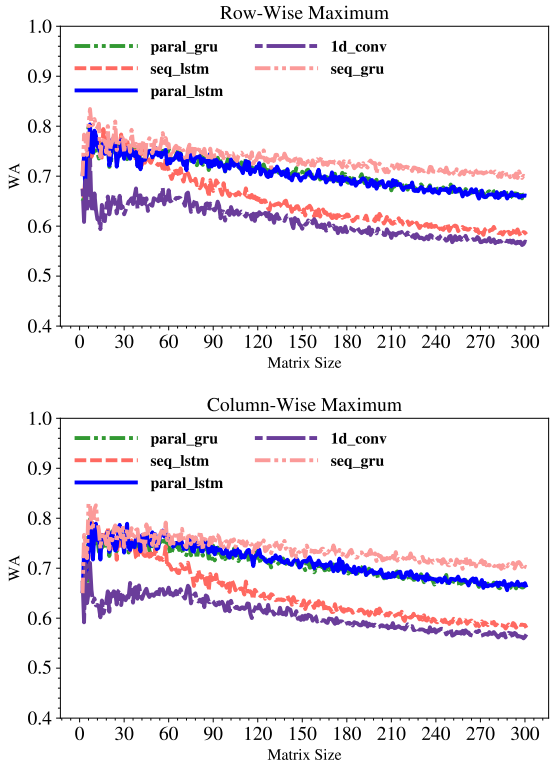

Figure 4. Evaluation of performance of DHN and its variants on $\mathbf{D}$ of different sizes.

Matrix Size. To provide further insights into DHN, we study the impact of the distance matrix size on the assign-

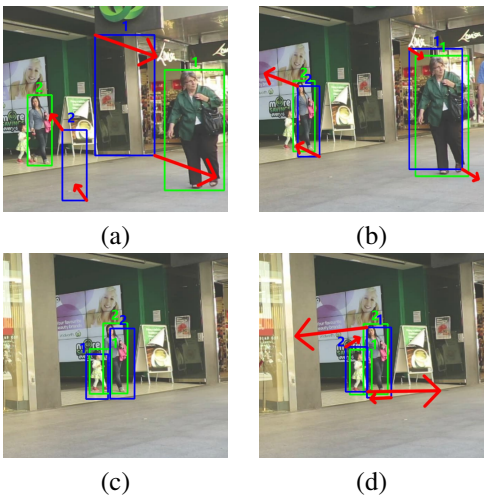

Figure 5. Visualization of negative gradients (direction and magnitude) from different terms in the proposed DeepMOT loss: (a) FP and FN (b) MOTP (c-d) IDS (compare (c) $t-1$ with (d) t). The predicted bounding-boxes are shown in blue, the ground-truth are shown in green and the gradient direction is visualized using red arrows.

ment accuracy. We visualize the relation between WA and the input matrix size in Fig. 4. For validation, we generate square matrices with sizes ranging from $[2, 300]$. Precisely, we generate $\mathbf{D}$ with a uniform distribution $[0, 1)$ and use the Hungarian algorithm implementation from [2] to generate assignment matrices $\mathbf{A}^*$. For each size, we evaluate 10 matrices, which gives us 2,990 matrices in total. As can be seen in Fig. 4, (i) the proposed **seq_gru** consistently outperforms the alternatives. (ii) The assignment accuracy of DHN and its variants decreases with the growth of the matrix size. Moreover, we observe a performance drop for very small matrices (*i.e.*, $\mathrm{M} = \mathrm{N} \leqslant 6$). This may be due to the imbalance with respect to the matrix size during the training.

C. Training Gradient Visualization

The negative gradient should reflect the direction that minimizes the loss. In in Fig. 5 we plot the negative gradient of different terms that constitute our DeepMOT loss w.r.t. the coordinates of each predicted bounding box to demonstrate visually the effectiveness of our DeepMOT. In this example, we manually generated the cases which contain the FP, FN or IDS. We observe that the negative gradient does encourage the tracks' bounding boxes to be close to those of their associated objects during the training.

D. MOT15 Results

We summarize the results we obtain on MOT15 dataset in Tab. 3. Our key observations are:

(i) For the MOT-by-SOT baseline, we significantly improve over the trainable baselines (SiamRPN and GOTURN). DeepMOT-SiamRPN increases MOTA for $+2.3\%$, MOTP for $+0.7\%$ and IDF1 for $+2.0\%$. Remarkably, DeepMOT-SiamRPN suppresses 2,416 FP and 143 IDS. We observe similar performance gains for DeepMOT-GOTURN.

	Method	MOTA ↑	MOTP ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDS ↓
2D MOT 2015	DeepMOT-Tracktor	**44.1**	**75.3**	46.0	17.2	26.6	6085	26917	1347
	Tracktor [1]	**44.1**	75.0	46.7	18.0	**26.2**	6477	**26577**	1318
	DeepMOT-SiamRPN	33.3	74.6	32.7	9.3	43.7	7825	32211	919
	SiamRPN [12]	31.0	73.9	30.7	12.6	41.7	10241	31099	1062
	DeepMOT-GOTURN	29.8	**75.3**	27.7	4.0	66.6	3630	38964	524
	GOTURN [7]	23.9	72.8	22.3	3.6	66.4	7021	38750	965
	AP_HWDPL_p [3]	38.5	72.6	**47.1**	8.7	37.4	**4005**	33203	586
	AMIR15 [16]	37.6	71.7	46.0	15.8	26.8	7933	29397	1026
	JointMC [9]	35.6	71.9	45.1	**23.2**	39.3	10580	28508	457
	RAR15pub [5]	35.1	70.9	45.4	13.0	42.3	6771	32717	**381**

Table 3. Results on MOTChallenge MOT15 benchmark.

(ii) DeepMOT-Tracktor obtains results, comparative to the vanilla Tracktor [1]. Different from MOT16 and MOT17 datasets, we observe no improvements in terms of MOTA, which we believe is due to the fact that labels in MOT15 are very noisy, and vanilla Tracktor already achieves impressive performance. Still, we increase MOTP for 0.3% and reduce FP for 392.

References

[1] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixé. Tracking without bells and whistles. *ICCV*, 2019. 1, 2, 4

[2] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: The clear mot metrics. *JIVP*, 2008:1:1–1:10, 2008. 1, 4

[3] Long Chen, Haizhou Ai, Chong Shang, Zijie Zhuang, and Bo Bai. Online multi-object tracking with convolutional neural networks. *ICIP*, 2017. 4

[4] 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

[5] Kuan Fang, Yu Xiang, Xiaocheng Li, and Silvio Savarese. Recurrent autoregressive networks for online multi-object tracking. *WACV*, 2018. 4

[6] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. *CVPR*, 2008. 1

[7] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. *ECCV*, 2016. 4

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[9] Margret Keuper, Siyu Tang, Bjorn Andres, Thomas Brox, and Bernt Schiele. Motion segmentation & multiple object tracking by correlation co-clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2018. 4

[10] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. ICLR, 2015. 1

[11] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942*, 2015. 1

[12] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High performance visual tracking with siamese region proposal network. *CVPR*, 2018. 4

[13] Anton Milan, Laura. Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. MOT16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016. 1

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *NIPS*, 2015. 1

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *MICCAI*, pages 234–241. Springer, 2015. 3

[16] Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. Tracking the untrackable: Learning to track multiple cues with long-term dependencies. *ICCV*, 2017. 4

[17] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012. 1

[18] Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. *CVPR*, 2016. 1