A Unified Pipeline for Multiple Object Tracking

Wu Yue  
ventus@umich.edu

Dong Zhengyuan  
dongzy@umich.edu

Abstract

In this project, we work on the multiple object tracking (MOT) problem. We aim to reproduce and improve the frame given in the paper Joint Detection and Online Multi-Object Tracking [4], which illustrates a unified pipeline consisting of multiple methods like Kalman filter, Single Shot MultiBox Detector, RNN, etc. To improve the performance, we review literature on MOT in the last decade and focus on the paper, Simple Online and Real-time Tracking with a Deep Association Metric (Deep SORT) [8], which utilizes a cosine metrics together with the traditional filter method to achieve better association result. We do experiments on different networks for the detection part and compare multiple association methods. For the last part, we combine the deep SORT method with the joint detection frame to produce a unified pipeline.

1. Related Works

1.1. introduction

As a long-existing and eye-catching topic, tons of work has been done and remarkable results were achieved over the decades.

Generally, models towards the problem of multi-object tracking (MOT) can be categorized in two ways: online tracking vs. offline tracking regarding scope of optimization, and Detection-Based Tracking vs. Detection-Free Tracking [6], regarding whether the initialization is manual or automatic.

For the task of real-time MOT, although an offline method with batch technique (optimizing on a batch of previous frames) can emulate the behavior of an online method, it still need a significantly longer runtime to reach the same level of performance given by online methods. Additionally, considering that more and more online methods utilize the power of deep networks and achieve higher performance on benchmarks, we choose to focus on building an online tracking system that evolves and learns as it takes new inputs.

We browse through papers from top computer vision conferences over the past few years. Based on our understanding of the problem, we selected the following three models as our foundation:


1.2. Deep Sort

The deep SORT method is an improvement of the authors’ previous work, Simple Online and Real-time Tracking [9]. We will utilize it to solve the association part later in our pipeline. It consists of the following steps:

1. Train a Faster Region CNN (FrRCNN) to detect object, which has been pre-trained to discriminate pedestrians on a large-scale person re-identification dataset.

2. Use a linear constant velocity model to describe each target. In [9],

\[ x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \]

in [8]

\[ x = [u, v, \lambda, h, \dot{x}, \dot{y}, \dot{\lambda}, \dot{h}]^T \]

3. Each target’s bounding box is given a cost matrix, which is computed as the intersection-over-union distance between each detection and all predicted bounding boxes from the existing targets.

4. Use cosine distance to measure the appearance feature and detection feature. \( d^{(2)}(i, j) = \min_{\lambda} |\lambda_d^{(1)}(i, j) - \lambda_d^{(2)}(i, j)| \) where \( i \) is a feature vector with a dimension of 128. Train it on a convolutional neural network.

5. Combine the cosine metric \( c_{i,j} = \lambda d^{(1)}(i, j) + (1 - \lambda)d^{(2)}(i, j) \) with the result given by Kalman filter. If the value is smaller than some threshold, labeled it as tracked.

6. Use cascade match (in a series of matched detection) to solve blocking issue.

7. Apply intersection over union (IOU) assignment. View every tracked object within one frame to be the candidate for IOU and the others to be not. Then calculate the IOU distance. With the primitive matching result given by cascade matching, we get rid of the ones that are smaller than a threshold value.
Algorithm 1 Cascade Matching

\[
\begin{align*}
&\tau = 1, \ldots, N, \text{ Detection indices } D = 1, \ldots, M, \text{ Maximum age } A_{\text{max}} \\
&\text{Compute cost Matrix } C = [c_{i,j}] \\
&\text{Compute gate matrix } B = [b_{i,j}] \\
&\text{let the set of matches } M \rightarrow \emptyset, \text{ and the set of unmatched detections } \mu \rightarrow D \\
&\text{for } n = 1, \ldots, A_{\text{max}} \text{ do} \\
&\quad \text{Select track by age } \tau_n \rightarrow i \in T, a_i = n \\
&\quad [x_{i,j}] \rightarrow \text{mincostmatching} \\
&\quad M \rightarrow M \cup \{(i,j) | b_{i,j} \cdot x_{i,j} > 0 \} \setminus \{j : \sum b_{i,j} \cdot x_{i,j} > 0\}, \mu \rightarrow \mu \setminus \{j : \sum b_{i,j} \cdot x_{i,j} > 0\} \\
&\text{end for} \\
&\text{return } M, \mu
\end{align*}
\]

Compared with the original SORT algorithm, Deep SORT reduces ID switches for instance, a person reappear after being blocked by other pedestrian.

The key components of deep SORT is its usage of a feature vector as additional information. One situation that a Kalman filter would fail would be when a object gets blocked by someone else and it reappear after a period of time. The feature vector can help us predict if the object would reappear. It combines the cosine metrics which evaluate the deep learning result and the filter result, which proves to be effective on multiple benchmarks.

Another advantage of deep SORT is its Speed.

<table>
<thead>
<tr>
<th>Method</th>
<th>MOTA</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>POI [10]</td>
<td>66.1</td>
<td>10Hz</td>
</tr>
<tr>
<td>MCMOT [5]</td>
<td>62.4</td>
<td>35Hz</td>
</tr>
<tr>
<td>LMP [3]</td>
<td>71.0</td>
<td>0.5hz</td>
</tr>
<tr>
<td>Deep SORT [8]</td>
<td>64.4</td>
<td>40Hz</td>
</tr>
</tbody>
</table>

Table 1. Tracking results on the MOT16 [7]

We can see that it outperforms many other online methods in the runtime while achieving similar level of accuracy.

We are particularly interested in this paper because the techniques like cascade matching and cosine distance would be potential helpers to improve the performance, especially for association.

1.3. Deep Affinity Network

The paper Deep Affinity Network for Multiple Object tracking focuses on tracking the detected objects based on affinity, proposing a deep network that learns compact yet comprehensive features at several levels of abstraction, and performs a dense pairing of features between any two frames to infer object affinities. Before being fed into the network, the data goes through three augmentation steps, namely photometric distortions, frame expansion and cropping.

The network has two main components, a feature extractor and an affinity estimator. The feature extractor processes the input by two streams with shared parameters through an extended VGG-like network, and concatenate their output to obtain the feature vectors. The affinity estimator then densely permutes these vectors to form a 3D tensor and runs through a compression network to map it back to 2D. Extra rows/columns are appended to account for multiple identities leaving and entering between the frames, and the resulting tensor is compared with ground truth to compute the loss.

We find this paper interesting in that it substitutes traditional hand-crafted association constraints like appearance, motion, spatial complexity and grouping with association metrics learned by deep networks. This effectively makes the problem end-to-end and significantly reduces tedious hyper-parameter tuning. Also, the dense feature matching between every two frames leads to more reliable association, which contributes to the ideal performance across MOT dataset.

1.4. Joint Detection

Our proposed model would mainly be constructed on the basis of the paper Joint detection and online multi-object tracking [4]. The fundamental idea of this paper is that both detection and tracking can benefit from each other. The authors proposed a method that jointly performs both tasks in a single neural network architecture.

By training both parts together, the model can use optimized parameters instead of heuristic decisions over the track lifetime. The paper adopts an RNN architecture which keeps a fixed number of tracks and takes single-shot-detection (SSD) results (class scores and bounding box) as its input at each step. An association metric is calculated between the SSD results and current tracks. Through non-maximum suppression, bipartite graph matching solved by Hungarian algorithm and confidence correction, the new score for each track (including existing and new tracks) is calculated, and old tracks gets replaced by new tracks with higher scores to maintain a fixed number of tracks.

This method can best several of the state-of-the-art methods on MOT16 dataset, though not all of them. We find this paper inspiring in that it proposes a unified approach to the detection and tracking problem where parameters are shared by the two parts, and integrates multiple filtering and track correction techniques into a single pipeline, providing a flexible ground for optimization on each step and investigating their significance.
2. Developments and Experiments

2.1. Dataset Selection

Here is a list of datasets we found for MOT tasks.

- PTES 2009: for multi-person tracking
- MOT: for multiple object training.
- KITTI-Tracking: for multi-person or multi-car tracking dataset.
- UA-DETRAC: for multi-car detection and tracking.
- PoseTrack: for multi-person pose tracking.
- JTA Dataset: for pedestrian pose estimation and tracking in urban scenarios (created by exploring Grand Theft).
- PathTrack: for multi-person tracking.

As MOT (Multiple Object Tracking Benchmark) being the most popular and well-constructed dataset, we would mainly focus on it. Here is a list of some of the metrics the MOT benchmark evaluates.

- MOTA: Combines false negatives, false positives and mismatch rate.
- Recall: Ratio of correctly matched detection to ground-truth detection.
- Precision: Ratio of correctly matched detection to total result detection.
- IDS: Number of times that a tracked trajectory changes its matched ground-truth identity.
- MT: Percentage of ground-truth trajectories which are covered by the tracker output for more than 80 percent of their length.
- ML: Percentage of ground-truth trajectories which are covered by the tracker output for less than 20 percent of their length.

2.2. Experiments

Through literature reviews, we can see that online MOT is composed of two main parts: Detection and Association.

To begin with, we experimented with Deep SORT using the open-source implementation on the website. We run the tracker on the test set and generate detection file with standard format:

- frame: id, bb left, bb top, bb width, bb height, conf, x, y, z

And here is a result we get from the devkit.

<table>
<thead>
<tr>
<th>VGG</th>
<th>Resnet 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDF1</td>
<td>49.5</td>
</tr>
<tr>
<td>Rcll</td>
<td>44.2</td>
</tr>
<tr>
<td>MT</td>
<td>120</td>
</tr>
<tr>
<td>FP</td>
<td>4103</td>
</tr>
<tr>
<td>FN</td>
<td>52322</td>
</tr>
<tr>
<td>MOTA</td>
<td>49.0</td>
</tr>
<tr>
<td>MOTP</td>
<td>86.5</td>
</tr>
</tbody>
</table>

Table 2. Tracking results on the MOT17, using RE-ID baseline

We can see that ResNet performs better on Completeness (including MT, GT), Accuracy (MOTA), Robustness (IDF, etc) and only slightly lower on precision (MOTP). Therefore, we would adopt Resnet50 for our Frcnn detector.

Another question we have on the detector is that how a deeper pre-trained network on a more general dataset like ImageNet would perform compared with the network trained on datasets like YACVID for pedestrian. We use the model provided by the link. And the result is the following.

<table>
<thead>
<tr>
<th>ImageNet</th>
<th>Pedestrian Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDF1</td>
<td>44.3</td>
</tr>
<tr>
<td>Rcll</td>
<td>46.2</td>
</tr>
<tr>
<td>MT</td>
<td>254</td>
</tr>
<tr>
<td>MOTA</td>
<td>54.2</td>
</tr>
</tbody>
</table>

Table 3. Tracking results on the MOT17, with network trained on YACVID and ImageNet.

We can see that the tradeoff (actually one of the disadvantage for all learning methods) is that although the pre-trained model achieved better result on the benchmark, it may not generalize well to a wider range of objects in reality. But since we only focus on MOT dataset (which mostly track pedestrian), we would use the detector trained on pedestrians in our implementation.

Second, we went through different association methods. Apart from Deep SORT, DAN applies a CNN (VGG) to deal with the features it extracts. The source code can be found on the website. We modify it so that the network setup for detectors is similar to...
the one Deep SORT uses. We notice that the author did not show their result on MOT16, so that we decide to compare the result of DAN and deep SORT on MOT16 and the result is as follows:

<table>
<thead>
<tr>
<th></th>
<th>DAN</th>
<th>Deep SORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOTA</td>
<td>69.2</td>
<td>66.1</td>
</tr>
<tr>
<td>ML</td>
<td>0.32</td>
<td>0.208</td>
</tr>
<tr>
<td>FP</td>
<td>10221</td>
<td>8698</td>
</tr>
<tr>
<td>FN</td>
<td>59370</td>
<td>63245</td>
</tr>
</tbody>
</table>

Table 4. Tracking results on the MOT16 [7] challenge

We can see that the DAN achieves higher accuracy while Deep SORT performs better on precision and completeness. We speculate that techniques like cascade matching and cosine association metrics will help to reduce ID switches, which results in lower false positive value. But more experiments are needed to prove this.

Third, in the joint detection paper, the author does not mention what kind of CNN networks they use (code not published).

We used the Deep SORT setup (with the original FrCNN implementation) and obtained the following result on the MOT16 dataset [7]

<table>
<thead>
<tr>
<th></th>
<th>Joint Detection</th>
<th>Deep SORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOTA</td>
<td>39.1</td>
<td>66.1</td>
</tr>
<tr>
<td>ML</td>
<td>0.41</td>
<td>0.208</td>
</tr>
<tr>
<td>FP</td>
<td>9411</td>
<td>8698</td>
</tr>
<tr>
<td>FN</td>
<td>99727</td>
<td>63245</td>
</tr>
</tbody>
</table>

Table 5. Tracking results on the MOT16 [7] challenge

Here the joint detection score is given by the paper [4]. We can see that the joint detection method does not achieve a high score except on completeness. What inspires us is its methodology, as it provide us a fully integrated pipeline so that other ones can modify it and assemble different methods. We will show how we implement it in the next section.

2.3. Proposed Model and Implementation

The model we choose to implement is based on the model proposed in paper Joint detection and online multi-object tracking. The detailed pipeline is as follows, where the following data classes are defined and used in our implementation.

- **Detection**:
  - frameidx: Integer denoting frame number
  - tlwh: Vector of float (x, y, width, height) denoting the bounding box, (x, y) being the top-left corner
  - size: Vector of float (width, height) denoting the size of the bounding box (inferred from tlwh)
  - center: Vector of float (x, y) denoting the center of the bounding box (inferred from tlwh)
  - confidence: Float denoting the confidence of detection, which is the difference between the highest class score and the background score.
  - feature: 128-d vector of float denoting the appearance feature of the detection, used for calculating affinity

- **Track**:
  - track_id: ID of the track, used for identifying ID switches
  - delta_max: Integer denoting the maximum number of frames of detections stored in the track
  - det_history: List of Detections storing the past (up to delta_max) frames of detections in the track, from newer to older.
  - flags: Float denoting the confidence of detection, which is the difference between the highest class score and the background score.
  - track_score: List of bool denoting whether a detection was associated to the track in the past (up to delta_max) frames.

2.3.1 Detection

The MOT-16 dataset contains detections generated by Single-shot Multibox Detector model. Each detection consists of frame index, bounding box top-left corner and size, confidence and appearance features. On the top level, our model is an RNN that keeps a list of Tracks as its hidden state and takes a list of Detections (with the same frame index) at each time step as its input. The forward pass is implemented in our model.py and explained below in details.

2.3.2 Affinity Measure

Starting with list of Detections, for each detection in the list, we compare it with each detection in the detection history of each track, in terms of feature, center and size, and obtain Euclidean distances $a, b$ and $f$ accordingly. In order to let the same distance have a greater influence on smaller bounding boxes, we correct $b$ and $f$ by multiplying them with $r = R(w, h)$ of both detections, where $R(\cdot)$ is a multi-layer perceptron with two hidden layers (the perceptrons mentioned later in the model will also have two hidden layers). The corrected distances $b'$ and $f'$, together with $a$,
are fed into another perceptron \( M(\cdot) \) to obtain the affinity score \( m = M(a, b', f') \). We have a different \( M(\cdot) \) for each frame offset (the difference between the frame index of the two detections compared), with different parameters. For each new detection \( s \) and each existing track \( d \), the affinity measure \( m_{s,d} \) is the maximum value of \( m \) among \( d \)'s history where a detection is present. If no detection is present in the entire history, a dummy value is set for \( m_{s,d} \).

2.3.3 Non-maximum Suppression

For each detection \( d \), we choose among \( m_{s,d} \) for the highest affinity measure among tracks \( m_d \). We then feed \( m_d \) and track confidence \( c \) into a perceptron \( C(\cdot) \) to obtain an updated confidence \( c' = C(c, m_d) \). We conduct non-maximum suppression with this updated confidence, first filtering detections with a low confidence, then removing all detections overlapped (IOU greater than 0.85) by another detection with a higher confidence.

2.3.4 Association Measure

Instead of comparing each detection with each detection in track history, we compare each detection with each track as a whole for association measure. For a track \( s \) and a detection \( d \), we feed the current track score \( l \), updated track confidence \( c' \) and affinity measure \( m_{s,d} \) obtained previously into a perceptron \( O(\cdot) \) to obtain association measure \( o_{s,d} = O(l, c', m_{s,d}) \). Then we solve the assignment problem using Hungarian algorithm to make an initial assignment of new detections to tracks. We then examine the assignments and remove those with a negative association measure and put together a list of unmatched detections and a list of unmatched tracks.

2.3.5 Track score

For track \( s \) with an assigned detection \( d \) in the previous step, we feed the previous track score \( l \), track confidence \( c \) and affinity score \( m_{s,d} \) into a perceptron \( L(\cdot) \) and obtain the updated track score \( l' = L(l, c, m_{s,d}) \). For tracks without an assigned detection, we update the track score to a dummy value. We also create a new track for each unmatched detection and set its track score to another dummy value. Note that both dummy values are learned through training.

2.3.6 Track management

We rank old and new tracks according to their track score and keep a fixed maximum number of tracks with the highest scores. We update their flags, inserting a True at the front of flags if the track has an associated detection or is a new track, and a False if the track is an existing one and does not have an associated detection. Finally, we output these tracks to finish a forward pass of the network.

2.3.7 Training and evaluation

The training process is done separately for the SSD detector and the RNN tracker. We focus on the latter and implemented it in \texttt{train.py}. We define our loss for a single frame as the sum of four sub-losses: the loss for the affinity metric \( L_m \), the loss for the association metric \( L_a \), the loss for the new detection score \( L_c \) and the loss for the track score \( L_t \).

Generally, we adopt a cross-entropy form of \( L = -\frac{1}{n} \sum_{i=1}^{n} p_i \log \hat{p}_i + (1 - p_i) \log (1 - \hat{p}_i) \), where \( p_i \) is the binary ground truth that is 1 when the IOU between the bounding boxes of a matched detection and a ground truth is sufficiently high (\( >0.85 \)) and 0 when not. \( \hat{p}_i \) varies for different kinds of losses. For \( L_m \), \( \hat{p}_i = \sigma(m_{s,d}) \). For \( L_a \), \( \hat{p}_i = \sigma(o_{s,d}) \). For \( L_c \), \( \hat{p}_i = \sigma(c) \). For \( L_t \), \( \hat{p}_i = \sigma(l) \). Additionally, for \( L_m \) and \( L_a \), the averages are taken separately for positive (\( p_i = 1 \)) and negative (\( p_i = 0 \)) samples and then added.

During training, we sum up the losses across a batch of 16 frames and do a weight update through back-propagation at the end of every batch. Defining an epoch as one pass through the whole training set, we choose to train for 50 epochs with a learning rate of 0.0005.

For evaluation, we implemented the Multiple Object Tracking Accuracy (MOTA) metric in \texttt{main.py}. MOTA compares the matched detections with ground truth detections at each frame \( t \), counting the number of missed detections \( m_t \), false positives \( f_p_t \) and mismatches \( m_m_t \) such as ID switches. The final score is given by

\[
MOTA = 1 - \frac{\sum_t (m_t + f_p_t + m_m_t)}{\sum_t g_t}
\]

where \( g_t \) is the number of ground truth detections at frame \( t \).

Due to limited time, we did not manage to train the network and tune its hyperparameters. There are also many other metrics for the MOT problem such as ID switches, Recall, Precision, MT and ML that we did not have time to implement and evaluate on.
3. Results and Conclusion

The following is a sample output from one of the videos given by the MOT16 dataset.

Figure 1. Sample detection result

Figure 2. Sample tracking result

These two pictures refer to the same frame of the video. The red boxes are the result given by the detector, while the colored ones are given by the tracker after association and were labeled in their unique identity. We can see how the associate adjust around the initial detected object.

The following pictures are given by the tracker on eighth video of MOT16 test set, with each one 20 frames away from each other(approximately a second).
We can see that our tracker works well under various conditions. For instance, take a closer look at target 5 and 8 (the aged couple) in the first four images. They first are blocked by target 4 (the man with beg) for a second. When they reappear, our tracker does not lose track of them, and their IDs do not switch.

Our tracker would also fail under many circumstances. For instance, when the number of targets increases (or the target get smaller), the result get worse. The following is tested on a video on a crowded square.

If we look at the two targets pointed out by the red circle, we can see that the target 26 and 123 are not labeled correctly. We can observe the 123 is labeled mistakenly as 251 in the fourth frame. We try the other two models and the similar mistakes still occurs. A possible explanation is
the training set for the cosine metric features is too small, and it is not sufficient enough to overcome the deficiency.

In summary, we propose a unified pipeline with shared parameters for both detection and association to utilize the power of traditional filtering method and deep learning, so as to try solving the multiple object tracking problem.

References


