Deep Affinity Networks for Multiple Object Detection

Darren Mei
Dept. of Computer Science
Stanford University
`dmei@stanford.edu`

Duncan MacWilliams
Dept. of Computer Science
Stanford University
`dmacwill@stanford.edu`

Aditya Khandelwal
Dept. of Computer Science
Stanford University
`akhand@stanford.edu`

Abstract

This paper solves multiple object detection and tracking using a Deep Affinity Network (DAN). Multiple object tracking builds off of object detection and assigns identities that are consistent throughout a video stream. We use the Multiple Object Tracking (MOT) dataset to evaluate our model. With revisions made to the original DAN architecture implementation, our model performs better and generalizes well to the test set. Additionally, it shows more robustness to occlusion-induced identity switching issues. Following an overview of prior efforts at solving this problem, we describe and evaluate baseline implementation of Simple Online, and Realtime Tracking (SORT) model, and use it to compare the results from the DAN implementation. We achieve an MOTA score of 51.05% and an MOTP score of 0.2175, both of which are slightly better than the original model.

1 Introduction

Commonly, in tasks that require object detection and tracking, detection is undertaken with the help of deep learning frameworks while tracking is accomplished using geometrical and mathematical analysis. In this project we are concerned with object tracking using deep learning. We use the datasets provided by the Multiple Object Tracking datasets available on the MOT Challenge website. The object tracking component assigns a unique id to each of the detected objects and keeps ids for unique objects consistent between frames of the video. In this task, the goal is to achieve high accuracy and find ways to handle multiple overlaps between objects as they cross paths across frames. Particularly, we are interested in applying deep learning techniques since they provide a non-linear model that could learn geometrical and mathematical approaches without any specific domain knowledge.

2 Related work

Most recent approaches for Multiple Object Tracking can be broadly classified into one of two categories: Detector Based Tracking (DBT) and Detector Free Tracking (DFT) (6). During DBT, objects are first located in individual frames and their trajectories are subsequently tracked. Given a sequence, type-specific object detection or motion detection, is applied to each frame to obtain object hypotheses, then (sequential or batch) tracking is conducted to link detection hypotheses into trajectories. On the other hand, DFT requires manual initialization of a fixed number of objects in the first frame, then localizes these objects in subsequent frames. Therefore, DBT algorithms usually track objects of a certain class, while DFT algorithms can track objects from multiple classes. Sequential methods (called online tracking algorithms) (10) (11) handle frames where only information up to the current time is available. Alternatively, offline methods employ both past and future frame knowledge to track objects. Kalman Filter (7) and Kernel tracking (8) algorithms like a mean shift tracker are examples of DBT trackers which require object detection in each frame. However, they are not precise enough to handle very dense frames with multiple objects simultaneously. Recent papers make use of contextual models to avoid losing track of the object (1). Finally, the current state-of-the-art model on the MOT17 dataset is LSST17 (5), which is a DBT model that uses a switcher-aware classifier for tracking objects.

3 Dataset and Features

We are using the Multiple Object Tracking Benchmark to train and evaluate our models, specifically the MOT17 dataset. In each of the videos there is a large amount of identity switching and overall pedestrian tracking, making these problems difficult for most tracking models. The training set for MOT17 originally comes with 21 videos of lengths varying from 30 to 60 seconds, and frame rates ranging from 14-30 frames per second. In order to tune our hyperparmeters and test our videos we divided the dataset into three parts: 19 videos for training, 2 videos for validation and 2 videos for testing purposes. Each video is sourced in a crowded area from various cities of the world. For each such scene, we have raw data, ground truths and detections available. The detection files consist of bounding box coordinates for each bounding box in each frame, along with an ID that identifies bounding boxes across frames.

4 Methods

4.1 SORT Baseline

SORT (Simple Online and Realtime Tracking)(3) is a barebones implementation of multiple object tracking that utilizes only state and data association. This is what we used for our baseline.

The displacements of each object between frames is approximated by a linear constant velocity model. For each object, the following are recorded for each frame: the horizontal and vertical pixel locations of the center of the object, and the area and aspect ratio of the object's bounding box. To assign detections to targets, intersection-over-union (IOU) distance is calculated between each detection and all bounding boxes that have been predicted for the current frame. Using only these metrics and no deep association whatsoever, SORT represents a minimal approach that works effectively as a baseline.

4.2 Deep Affinity Network

Our main approach to object tracking follows the architecture outlined in the Deep Affinity Network paper.(1) The Deep Affinity Network (DAN) aims to take a pair of video frames that are some parameter n frames apart, as well as the centers of the pre-detected objects in those frames. At a high level, the network then computes scores relating to the possible combinations of the centers of pre-detected objects in the earlier image to the centers in the later image. In doing so, it is able to gauge the "affinity" that an object in the later image has with an object in the earlier image and assign IDs accordingly.

4.2.1 Model Architecture

The first part of DAN consists of an architecture similar to the VGG16 Convolutional Neural Network model used for image classification.(2) This sub-network's primary purpose is as a feature extractor which provides comprehensive representations of the detected objects. Since there are two frames sent into the network the frames are sent through parallel feature extractor structures, as see in Figure 1. The reason this subnetwork is not identical to VGG16 is due to the input size into the network, as the frames are of higher resolution than the images that were input into VGG16 from ImageNet(4). Additionally, the fully-connected and softmax layers of VGG16 have been converted to convolutional layers, because the resulting feature maps of this sub-network are sent to another sub-network.

The next sub-network is denoted as the Extension network, because it has additional convolutional layers in order to reduce the input feature map size of 56x56 from VGG16 to an output feature map size of 3x3. The purpose of the Extension network is to create object feature maps at varying levels of abstraction. While the network uses 3x3 filter sizes to reduce the dimension, it also uses 1x1 convolutions in between each 3x3 convolution layer. This is because in order to learn over a large amount of feature maps, the dimensions need to be reduced at points.

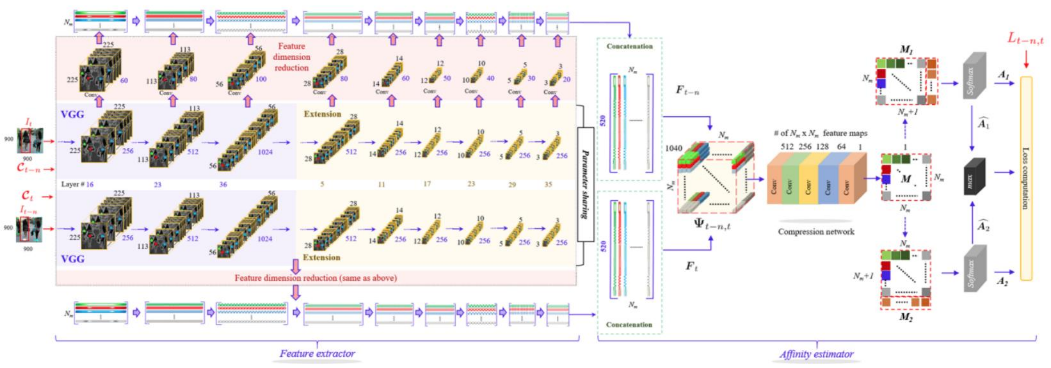

Figure 1: Architecture of the Deep Affinity Network(DAN)

During the VGG-like and Extension subnetworks, a selector goes along and picks 9 different feature maps of varying sizes. From this, the selector then concatenates these feature maps together to form a 520-dimensional vector for a detected object. This creates a feature matrix for the earlier video frame as well as the later video frame, because both were run through the same structure. These 2-dimensional matrices are then used to form a three-dimensional tensor of size (number of detected object center points) x (number of detected object center points) x (520 x 2). This occurs by arranging the two feature matrices in all possible permutations of the object ID correspondences.

This tensor is then run through the Final network consisting of five convolutional layers to form the affinity matrix M. M is then split into two cases, M_1 and M_2, where M_1 is M with an appended column and M_2 is M with an appended row. Finally, M_1 and M_2 are run through a softmax function to receive the matrices A_1 and A_2. These two matrices can then be used to compute the affinities between an object in the later image and an object in the earlier image. For example, the rows of A_1 encode probabilistic associations between an object in the earlier frame and all of the objects in the later frame. Similarly, the columns of A_2 encode the associations between an object in the later frame and all of the objects in the earlier frame.

4.2.2 Loss Function

The loss function used in the Deep Affinity Network is the average of four different loss functions. These loss functions represent the forward, backward, consistency, and assemble loss. Forward Loss, denoted as L_f, calculates the loss based on identity association from the earlier frame to the later frame. Backward Loss L_b is similar but from the later frame to the earlier frame. Consistency Loss is represented as L_c and maintains similarity between Forward and Backward loss. Lastly, Assemble Loss is a form of non-max suppression for forward and backward associations. In these equations below, $\mathbf{L}_1$, $\mathbf{L}_2$, and $\mathbf{L}_3$ correspond to trimmed versions of the difference between the ground truth annotation and the predicted matrices A_1 and A_2. In particular, $\mathbf{L}_1$ corresponds to trimming the last row, $\mathbf{L}_2$ trims the last column, and $\mathbf{L}_3$ trims both the last row and last column.

$$L_f(\mathbf{L}_1, A_1) = \frac{\sum(\mathbf{L}_1 \odot (-\log A_1))}{\sum(\mathbf{L}_1)}$$

$$L_b(\mathbf{L}_2, A_2) = \frac{\sum(\mathbf{L}_2 \odot (-\log A_2))}{\sum(\mathbf{L}_2)}$$

$$L_c(A_1, A_2) = ||A_1 - A_2||_1$$

$$L_a(\mathbf{L}_3, A_1, A_2) = \frac{\sum(\mathbf{L}_3 \odot (-\log(max(A_1, A_2))))}{\sum(\mathbf{L}_3)}$$

$$L = \frac{L_f + L_b + a_c + L_c}{4}$$

5 Experiments/Results/Discussion

5.1 Evaluation Metrics

We used the following metrics to evaluate our models:

MOTA (Multiple Object Trakcing Accuracy) is a metric that is evaulated from three error sources: false positives, missed targets, and identity switches.

$$MOTA = 1 - \frac{\sum_t (m_t + f_{p_t} + mme_t)}{\sum_t g_t}$$

where m_t, f_{p_t} and mme_t are the number of missed targets, false positives and identity switches respectively for time t.

MOTP is the precision metric that is evaluated from the misalignment between the annotated and predicted bounding boxes.

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t}$$

Where c_t is the number of matches found for time t, and for each of these matches $d_{i,t}$ is the distance between the object and its corresponding predicted box.

The IDF1 score is the ratio of correctly identified detections over the average number of ground truth and computed detections.

The MT metric represents mostly tracked targets. MT is the number of ground truth trajectories that are covered by a track prediction for at least 80% of their respective life spans.

5.2 Experiments

First, we ran our sort baseline on the MOT17 dataset to get preliminary results and see how our baseline performed on the test set. Following this, we ran our first unmodified DAN network, which took much longer to run given the introduction of deep metrics and many parameters. The parameters included the pixels of the input RGB training frames from the videos as well as the precomputed bounding box centers.

To modify the existing DAN architecture, we decided to add dropout to the convolutional layers themselves. Adding dropout with low drop probability in the convolutional layers can be beneficial to learning, so we added dropout with drop probability 0.1 to each of these layers.(13) We also noticed that the original architecture placed batch normalization layers before the ReLU activation layer, which we thought was curious. There is a lot of debate surrounding the order that these layers should be placed in, but we thought it made more sense to normalize after performing ReLU activation.(12) This way, the negative values that we throw out anyway with activation are not accounted for in the normalization. With these changes incorporated, we ran the modified model. We trained the model with batch size 4 and learning rate 0.01. We also used SGD with weight decay and ran the model for 40 epochs.

5.3 Quantitative Results

As seen in Table 1, the modified Deep Affinity Network architecture performs better than the original Deep Affinity Network architecture on the training, validation, and test sets by a small margin. Though both architectures perform slightly worse in MOTA than the SORT Baseline on the training and validation sets, they perform much better on the test set. MOTP also improves in the training and test set in the DAN implementations. Additionally, in all datasets the modified DAN architecture improves in IDF1 over the original Deep Affinity Network and SORT, showing that it correctly identifies more detections than the two other implementations. Finally, the number of mostly tracked targets is similar between the two Deep Affinity Network implementations, and both are higher in the validation and test set than the SORT Baseline.

One issue with our results is that the test set performance is significantly higher than the train and val set performance for the DAN architectures. Though we assumed the videos in the MOT17 training dataset were similar, since we used 2 videos for the val set and 2 videos for the test set it is possible

Dataset	MOTA	MOTP	IDF1	MT
SORT Baseline				
Train Set	41.75%	0.173	43.51%	13.765
Val Set	45.70%	0.189	8.25%	13.5
Test Set	38.65%	0.133	5.65%	10
Original DAN				
Train Set	39.82%	0.212	45.78%	13.706
Val Set	39.10%	0.1605	46.45%	18
Test Set	50.85%	0.217	50.30%	15
Modified DAN				
Train Set	39.88%	0.213	46.24%	13.765
Val Set	39.30%	0.1605	47.00%	17.5
Test Set	51.05%	0.2175	51.20%	15

Table 1: Implementation Results on MOT17 Data

that one or both of the videos in the test set contains an easier tracking problem. However, since the SORT Baseline did the worst on the test set, it is possible that the test set videos are ones that the DAN architecture performs better on.

5.4 Qualitative Results

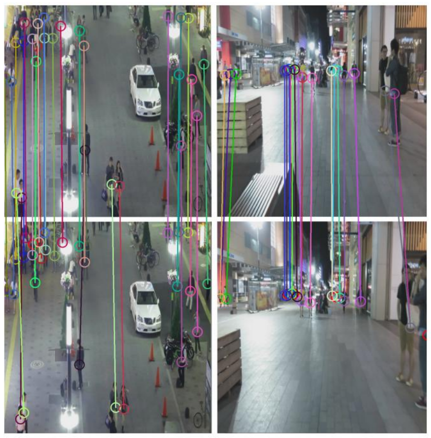

Figure 2: Output examples from the Modified Deep Affinity Network

As seen in Figure 2, the majority of detections in the earlier frame (top image in both examples) are correctly identified in later frames (bottom image in both examples). However, there are certain misidentifications which can be seen in the red circles that do not have a connection in the other frame. As seen in the right side of Figure 2, the person exiting the frame is not correctly identified in the later image, showing that the Deep Affinity Network does not perform as well with people entering and exiting the image. Additionally, this then causes the person next to them to be misidentified as both people in the later frame.

6 Conclusion/Future Work

In this paper, we introduced improvements to the Deep Affinity Network architecture for multiple object detection and tracking. From our results, it is evident that our revised version performs better on key metrics even when we reduce the training time of the model compared to the original model. Furthermore, the proposed DAN architecture was more robust at identifying and dealing with similar object tracking under greater occlusions. Given more time and computational resources, we want to try replacing the intro VGG-like subnetwork with ResNet. Additionally, we would like to train our model for more epochs on different dataset splits to better understand our results. Finally, we could utilize pyramid networks to extract the feature maps at varying scales.

7 Contributions

Duncan MacWilliams: Worked on getting the SORT baseline functional with our dataset and getting familiar with the code. Worked with Darren to analyze and make changes to the network architecture in the DAN model. Proposed adding dropout to each convolutional layer with small drop probability. Debugged DAN code to work with our new modified architecture. Created the poster with Darren. Wrote the following paper sections: SORT Baseline, Experiments, and Evaluation Metrics.

Darren Mei: Worked on configuring the Deep Affinity Network for training and evaluation and debugged how to run and save model checkpoints with the MOT17 dataset. Worked with Duncan on changing the network architecture, and proposed switching the Batchnorm and ReLU layers after literature review. Proposed the splits of the data for the train, val, and test set. Created the poster with Duncan. Wrote the following paper sections: Deep Affinity Network method overview, Quantitative Results, Qualitative Results.

Aditya Khandelwal: Worked on getting the SORT baseline functional with our dataset and getting familiar with the Github repository with Duncan. Evaluated and compiled results of the SORT baseline on the train, val and test sets. Trained the original DAN architecture implementation for up to 70 epochs, and the modified version of the DAN network as described by Darren and Duncan above for up to 40 epochs. Evaluated and compiled the results on the train, val and test sets for each model. Wrote the following paper sections: Abstract, Introduction, Related Work, Results Table, Conclusion/Future Work.

8 Acknowledgements

We would like to thank the following projects that helped us get going on our project: Alex Bewley for SORT: https://github.com/abewley/sort

And Sun et al. for their DAN implementation.

References

[1] Sun, S., Akhtar, N., Song, H., Mian, A., Shah, M. (2018). Deep Affinity Network for Multiple Object Tracking. *arXiv preprint arXiv:1810.11780.*

[2] Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

[3] Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B. (2016, September). Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing* (ICIP) (pp. 3464-3468). IEEE.

[4] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). IEEE.

[5] Weitao F., Zhihao H., Wei W., Junjie Y., Wanli O. (2019). Multi-Object Tracking with Multiple Cues and Switcher-Aware Classification *arXiv:1901.06129*

[6] Wenhan L., Junliang X., Anton M., Xiaoqin Z., Wei L., Xiaowei Z., Tae-Kyun K. (2017). Multiple Object Tracking: A Literature Review *arXiv:1409.7618*

[7] Isard, M., Blake, A. (1998). Condensation - Conditional Density Propagation for Visual Tracking. International Journal of Computer Vision *International Journal of Computer Vision, 29(1):5–28*

[8] Comaniciu, D., Ramesh, V., Meer, P (2003). Kernel-based Object Tracking. IEEE Trans. on Pattern Analysis and Machine Intelligence *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 25, NO. 5: 564-577*

[9] Comaniciu, D., Ramesh, V., Meer, P (2003). Single and multiple object tracking using log-euclidean riemannian subspace and block-division appearance model *IEEE Trans. Pattern Anal. Mach. Intel., vol. 25, no. 5: 564-577*

[10] W. Hu, X. Li, W. Luo, X. Zhang, S. Maybank, and Z. Zhang (2012). Kernel-based Object Tracking. IEEE Trans. on Pattern Analysis and Machine Intelligence *IEEE Trans. Pattern Anal. Mach. Intel., vol. 34, no. 12: 2420–2440*

[11] L. Zhang and L. van der Maaten (2013). Structure preserving object tracking in Proc. *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.: 1838–1845.*

[12] Han, D., Kim, J., Kim, J. (2017). Deep pyramidal residual networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5927-5935).

[13] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.