

# Tracklet-based Video Anomaly Localization

## (Supplemental material)

Ashish Singh<sup>1,2\*</sup>   Michael J. Jones<sup>2\*</sup>   Erik G. Learned-Miller<sup>1</sup>  
<sup>1</sup>CICS, University of Massachusetts Amherst   <sup>2</sup>Mitsubishi Electric Research Labs  
ashishsingh@cs.umass.edu   mjones@merl.com   elm@cs.umass.edu

In this supplemental material, we will discuss the running time of our algorithm, discuss methods for speeding up the nearest neighbor search used for exemplar selection and anomaly detection, provide result videos showing anomaly detections for some of the test videos that we evaluate on, discuss our result videos, show some failure cases, discuss our tracking algorithm, provide more information about our model complexity (number of exemplars), give more details on the distance normalization constants, and finally give more details on the training examples used in our AppearanceNet.

### 1. Computational Analysis

We analyze runtime performance of our framework on the UCSD Ped2, CUHK Avenue and Street Scene datasets. We compute and report the processing speed for the anomaly detection stage for which the main computational expense is object detection and optical flow computation. We present our results in Table 1. We report our runtime speed in frames per second. We used a single NVIDIA GeForce RTX 3090ti GPU for feature extraction and Intel Xeon E5-2680 v4 @ 2.40GHz CPU for nearest neighbor computations.

For the anomaly detection stage the total time consists of the time to compute tracklets and the time for nearest neighbor search. The time to compute tracklets consists almost entirely of the time for object detection and optical flow inference. The time for motion segmentation and trajectory estimation (which is also part of tracklet computation) is negligible in comparison. Furthermore, the time for object detection in the table uses the fact that it is only run every 5th frame in our experiments. From the table it is clear that the total time is dominated by optical flow inference.

The time for the exemplar detection stage is almost identical to anomaly detection because they use the same pipeline.

The main computational bottleneck for our method is the optical flow computation and object detector inference.

Dataset	NN	Detection	Flow	Total
Ped2	1463fps	25fps	10fps	7fps
Avenue	1526fps	25fps	11fps	8fps
Street Scene	1277fps	25fps	9fps	6fps

Table 1. Computational speed for individual components of our pipeline. We report the speed in frames/second. (NN stands for nearest neighbor search.)

We use the DINO [7] detector with SWIN transformer [4] as backbone and RAFTv2 [6] for optical flow computation. We can improve the detector inference speed by using Resnet50 as the backbone which improves the detector inference speed by 50%. Furthermore, with computational deployment tools like [1], we believe the inference speed for such pre-trained models can be improved significantly. For our analysis, we report our runtime without the use of such tools.

Compared to other methods, we take significantly less time in model learning (training) because our method does not require any training of neural networks to learn a model of a new scene.

### 2. Efficient Nearest Neighbor Search

Our algorithm for selecting exemplars and our algorithm for computing anomaly scores for test tracklets are both based on nearest neighbor search among a set of tracklets. There are many, many algorithms for efficient nearest neighbor search in high-dimensional spaces. One possibility is to use ball trees [5] which generally works well with spatial data (an important component of a tracklet is its spatial location). For the results in this paper, we implemented a simpler method which also takes advantage of the spatial location of tracklets.

For a particular dataset with a certain frame size, the basic idea is to use a data structure consisting of a grid of regions of fixed height and width (we used height=width=60 pixels in our experiments) that tile a frame. Given a set of exemplars, which are represented as tracklets, each exem-

\*equal contribution, work done while A.S. was an intern at MERL

plar is stored in the grid region that contains the first coordinate of the exemplar’s trajectory. Then, in order to find the nearest exemplar given a test tracklet, we find which grid region the first coordinate of the test tracklet’s trajectory falls in and do a brute force search over all exemplars stored in that grid region as well as the 8 surrounding grid regions. We need to search the 8 surrounding grid regions as well because the coordinate could fall on the border of a grid region and have very small distance to a neighboring grid region.

This simple algorithm insures that we only search for exemplars that are reasonably close spatially which means that most of the exemplars do not need to be searched. This on top of the fact that our models have relatively few exemplars, makes our nearest neighbor search fast in practice. In the case that the number of exemplars in a model grows much larger (say tens or hundreds of thousands of exemplars), then a more sophisticated algorithm such as ball trees could be used instead.

### 3. Discussion of result videos

Our supplemental material includes 5 result videos showing the output of our video anomaly detection algorithm in various scenarios. Each video shows green bounding boxes around each normal object while objects detected as anomalous are shaded red. Because our tracklets are 10 frames long and we generate new tracklets (from new object detections) every 5 frames, tracklets often overlap. We use non-maximal suppression to remove overlapping bounding boxes with lower anomaly scores.

In SS\_Test023.mp4, a car making a u-turn is correctly detected as anomalous. Next, a bike being walked by a woman on the sidewalk is detected as anomalous for many frames. When the woman and bike get closer to the top of the frame, the bike is heavily occluded and is no longer recognized as a bike which causes it to no longer be detected as anomalous. A jaywalker is also correctly detected as anomalous. A false positive on a pedestrian briefly occurs due to a tracking error.

In the Ped1\_Test014.mp4 result video, four bikers are correctly detected as anomalous as well as a golf cart. Ped1 frames are low resolution (240x160 pixels) which results in pedestrians occurring in the top half of the frame (who are further from the camera) to be too small to be detected. This explains the lack of tracklets in the top half of the frame. Despite this our method is still able to detect part of all anomalous tracks with only a single false positive which is caused by a tracking error.

In the Ped2\_Test005.mp4 result video, an anomalous biker is correctly detected. This video gives a good idea of how our simple tracker works in crowded scenes. The vast majority of pedestrians are well tracked despite frequent occlusions. Occlusion does cause tracking to fail for a few



Figure 1. Visualization example for missed detection of a "biker outside lane" anomaly in Street Scene explaining why our method failed to detect it.

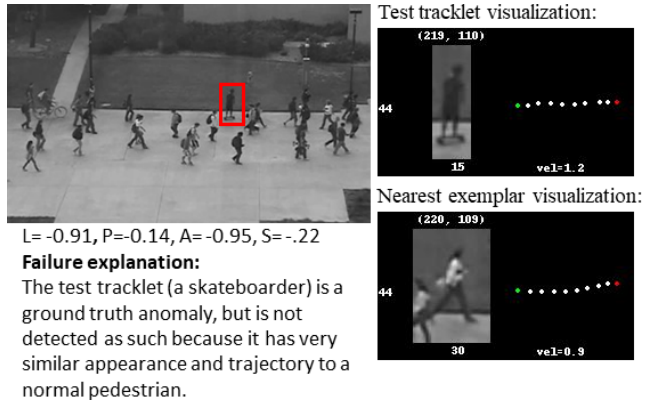


Figure 2. Visualization example for missed detection of a skateboarder anomaly in Ped2 explaining why our method failed to detect it.

frames on some pedestrians, but such pedestrians are usually tracked successfully again once the amount of occlusion lessens. Furthermore, these short-term tracking failures usually do not result in anomaly detection mistakes.

In Avenue\_Test006.mp4, a man who walks close to the camera is correctly detected as an anomaly as well as a backpack that is thrown into the air. This result video also shows bags and backpacks that people are carrying usually generate their own tracklets because they are detected by the DINO object detector.

In SHT\_Test001.mp4, a biker riding across the scene is correctly detected as anomalous.

### 4. Failure cases

Figures 1 and 2 show examples of missed anomaly detections by our method. In Figure 1, the anomaly is a biker riding outside of the appropriate bike lane. The only thing that is anomalous about the biker is their location. The test tracklet for the biker is visualized at the top, right of the figure. The closest exemplar tracklet is visualized at the

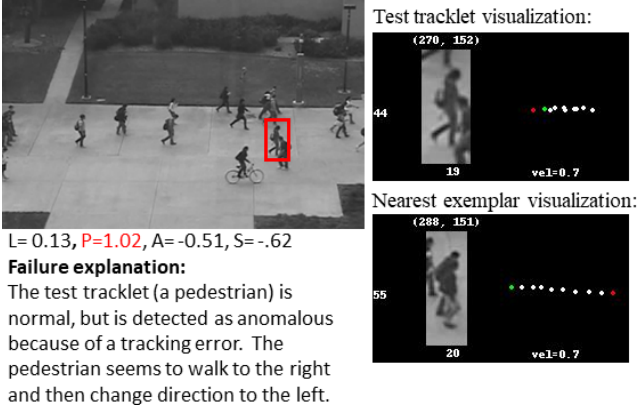


Figure 3. Visualization example for false detection of a pedestrian. A tracking error makes the pedestrian seem to walk to the right and then abruptly change direction and walk to the left.

bottom, right of the figure and is also a biker traveling with about the same trajectory as the test tracklet. The only difference is the location of the exemplar tracklet which is a few feet over and inside the bike lane. The location distance is small enough to be below the 1.0 anomaly threshold used in our results, so an anomaly is not indicated. Such anomalies are very difficult to detect because they are so similar to normal activity.

In Figure 2, the test tracklet (visualized in the top, right of the figure) is a skateboarder traveling at a similar speed to normal pedestrians. The skateboard itself only occupies a small number of pixels and therefore the appearance representation for the tracklet is very similar to a pedestrian. The closest exemplar tracklet (visualized in the bottom, right of the figure) is a normal pedestrian. All of the distances (location, appearance, trajectory and size) are far below the 1.0 anomaly threshold, so the test tracklet is judged to be normal. To detect such anomalies, one possibility would be to add more detailed motion information to the model, such as tracking a person’s skeletal pose over time which could distinguish normal walking pose changes from a skateboarder’s pose changes.

In Figure 3, an example of a false detection due to a tracking error is shown. The visualized test tracklet in the top, right of the figure shows a pedestrian initially walking to the right and then changing direction and walking to the left. (The visualization is difficult to interpret, but the green dot is the starting point and then subsequent points move to the right before ending up to the left of the starting point at the red dot.) The closest exemplar tracklet is a pedestrian walking to the right and the only attribute with a large distance is the trajectory. The tracking error was caused by another pedestrian walking in front of the initial pedestrian and the track switching to the second pedestrian. This error only persists for the 10 frames of this tracklet. Subsequent tracklets which are initialized from new object detections

do not perpetuate the bad track.

These failure examples are typical of our method’s failure cases and serve to give a better idea of the challenges remaining for our method.

## 5. Tracking discussion

Our tracking algorithm is motivated by a desire for simplicity and the fact that tracklets only require tracking for a small number of frames (10 frames in most experiments). Furthermore, because our method is already using optical flow to detect objects via motion segmentation, it is efficient to use optical flow for tracking as well. The point of this work is not to advance the state of the art in tracking, but rather to show the effectiveness of tracklet-based video anomaly detection even when a relatively simple tracking method is used. We could swap in a more sophisticated tracker instead, but we leave this idea for future work.

Our tracker handles static objects that are detected by the DINO object detector. The optical flow displacements on a static object will all be near zero, which will result in near zero displacements in the track. Thus static object tracking is handled very naturally by our tracker.

One issue that long-term object trackers need to handle is ID switches caused by an object moving behind another object. Because we are only tracking objects for a few frames and initializing new tracks every 5 frames based on object detections, the problem of ID switches is a minor issue. It is possible for our tracker to track an object for a few frames and then have it disappear behind another object at which point the track will switch to the other object. However, this tracking error will only persist for a few frames (because each track only lasts 10 frames). New tracks will be initialized from new object detections in subsequent frames. Our method does not try to associate track IDs from frame  $T$  to track IDs from frame  $T + 5$  or  $T + 10$ . etc. Thus, any ID switches caused by occlusion can only persist for a small number of frames.

In our method, a frame can have any number of tracklets in it (each initialized by a different object detection). Each tracklet is compared to all exemplars in the model to find the nearest neighbor. The distance to the nearest neighbor exemplar is the anomaly score for each tracklet. Thus, any number of tracklets in a frame can be handled.

Our simple tracker is not perfect (nor is it intended to be), but it works well in practice. Developing a new state-of-the-art tracker is not the point of this work. In our result videos, tracking mistakes can be found, and these sometimes result in VAD errors but these are infrequent enough that our method achieves excellent VAD accuracy on 5 different test sets.

## 6. Model complexity

For Singh et al., they report the total number of exemplars in the models they use for Ped1 and Ped2 are 4201 and 4270, respectively. Our models for Ped1 and Ped2 use 2133 and 673 exemplars, respectively. In the case of Ped2 this is over 6 times fewer exemplars. Furthermore, their exemplars consist of 513 floating point numbers, while ours consist of 152, mainly due to the much more compact motion representation (10 2-d coordinates in our case vs. 3 128-length feature vectors in their case).

## 7. More details on distance normalization constants

In practice, we found that some of the distance distributions have very long tails when computed over all possible pairs of normal tracklets, resulting in very large normalization constants which effectively underweights those distances. Just because two tracklets are normal does not mean they are similar. To avoid this, we use pairs of tracklets that are typically similar in order to focus on tracklet pairs with relatively small distances. We find such pairs by comparing tracklets that are close spatially since those tend to be similar in terms of appearance, trajectory and size as well. For each dataset, we use this strategy to estimate a set of distance normalization constants for that dataset using only the nominal (training) video.

## 8. More details on AppearanceNet training data

As discussed in the main paper, the training images used to train our AppearanceNet (a modified ResNext-50 architecture) came from multiple sources: MS-COCO [3] and VOC2012 [2] as well as publicly available surveillance video that we annotated. We used a variety of sources in order to create an object recognizer that would generalize to many different video anomaly detection datasets. We used five object classes for training: person, car, bicycle, dog, and fire hydrant. We did not use all examples for these classes that are available from MS-COCO and VOC2012, but rather manually curated a set of examples for which the desired class is clearly visible (some examples in these datasets include images for which the desired object is difficult to see). We will publicly release the 128x128 pixel example images that we used for training.

As discussed in the main paper, we also tested OpenCLIP features in place of our AppearanceNet features. OpenCLIP is trained on a much, much larger training set and includes a very large number of object classes. OpenCLIP also yields good video anomaly detection results, but not as good as AppearanceNet (see Table 6 in the main paper). We think this is most likely due to the fact that Ap-

pearanceNet is specialized to the object classes that are most commonly seen in video anomaly detection datasets.

## References

- [1] Nvidia tensorrt. <https://developer.nvidia.com/tensorrt>. 1
- [2] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015. 4
- [3] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 4
- [4] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 1
- [5] Stephen Omohundro. Five balltree construction algorithms. Technical Report TR-89-063, International Computer Science Institute (ICSI), 1989. 1
- [6] Deqing Sun, Charles Herrmann, Fitsum Reda, Michael Rubinstein, David J Fleet, and William T Freeman. Disentangling architecture and training for optical flow. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*, pages 165–182. Springer, 2022. 1
- [7] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M Ni, and Heung-Yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*, 2022. 1