AIC2018 Report: Traffic Surveillance Research

Tingyu Mao, Wei Zhang, Haoyu He, Yanjun Lin, Vinay Kale, Alexander Stein, Zoran Kostic
Columbia University

Abstract

Traffic surveillance and management technologies are some of the most intriguing aspects of smart city applications. In this paper, we investigate and present the methods for vehicle detections, tracking, speed estimation and anomaly detection for NVIDIA AI City Challenge 2018 (AIC2018). We applied Mask-RCNN and deep-sort for vehicle detection and tracking in track 1, and optical flow based method in track 2. In track 1, we achieve 100% detection rate and 7.97 mile/hour estimation error for speed estimation.

1. Introduction

In the context of autonomous driving and smart city management, it is beneficial to develop a cyber physical system to better monitor traffic situations. AIC2018 aims at dealing with three topics related to traffic surveillance based on camera data, including vehicle speed estimation, anomaly detection, vehicle re-identification. In this contest, we focus on the first two topics. The estimation of vehicle speed track is to estimate the speed of multiple vehicles simultaneously using a single bird's-eye view camera. This is different from conventional speed detection methods which always calculate speed based on time intervals and distance between two roadside cameras. The anomaly detection track is to automatically detect abnormal events including stalled cars as well as car crashes based on traffic flow. These applications will play an important role in reducing car accident rate and improving traffic situation.

To address these problems, it is required to integrate many computer vision techniques including object detection, multi-object tracking, camera calibration, optical flow tracking etc. The methods need to be robust to the variability of real scenarios. In this paper, we describe our methods for the first two tracks. The second section gives a brief review of related work. In the next three sections, we explain our methods and describe experiment details. Finally, we discuss the performance of our methods.

2. Related Work

2.1. Vehicle detection

Currently, with the advent of CNN networks, RCNN based methods have become the mainstream of object detection. They can be categorized into two different classes: single-stage and two-stage networks. The critical difference between them is the region proposal. Single-shot detectors do one-pass feature extraction and propose regions by a final regression layer, eg., you look only once (YOLO) [8] and single shot detector (SSD) [5]. Single-stage detectors are fast but less accurate. YOLO has scaling issues and cannot detect small objects accurately. In contrast, two-stage detectors are more accurate but slow. Jifeng Dai et al [2] proposed Region-based Fully Convolutional Networks (RFCN) which is fully convolutional with almost all computation shared on the entire image. An advanced Mask R-CNN [4] proposed based on RFCN which detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance.

2.2. Multi-Object Tracking

Multi-object tracking (MOT) is to track the trajectories of multiple objects simultaneously. With the progress in object detection, tracking-by-detection has become the leading scheme for MOT. The core of tracking-by-detection is to associate the detection bounding boxes across video frames correctly. In most cases, the metric of association is based on the appearance similarity as well as motion consistency. In terms of association policy, it can be divided into two categories: offline global optimization and online association. Offline global methods always use flow networks [7, 20] or probabilistic graphical models [18] to represent a MOT problem. This type of methods can achieve better tracking trajectories and are more robust to long-term occlusion. However, as they pursue a global opti-

mal solution of a time window, it is hard to implement it online, which limits its utility in some real-time applications. Whereas, online methods [1, 16, 15] are frame-by-frame based and focus on making correct association between each two frames, which makes it faster. Most of current online MOT methods are robust to false detection signal, short-term occlusion and missing objects. However, they cannot handle long-term occlusion very well.

2.3. 3D Coordinate Reconstruction

Projective Transformation: A projective transformation h can map line to line, ie., if x_1, x_2, x_3 are on the same line in the original plane, then $h(x_1)$, $h(x_2)$, $h(x_3)$ will remain on the same line after transformation. Such transformation can map the camera-view road back to the real road plane. It requires coordinates of at least four feature points from these two different coordinate systems to calculate a 3×3 matrix.

Camera calibration: To re-construct the 3D coordinate from a 2D video, Schoepflin et al.[10] use the vanishing points to estimate the calibration parameters of the camera. In their report, they compared three different calibration methods, with different required information. They found that the method, requiring two vanishing points, has less sensitivity on error. But the two vanishing points can not always be extracted from the video in our case since in some parts, like location 1 and 3, the second vanishing point does not exist or is very very far away. For the other two methods compared by the authors, both of them require additional information besides the video like the distance between the camera and the road which is, again, not available in our case.

Uncalibrated methods: Dailey et al. [3] proposed a method for vehicles speed measurement based on tracking of vehicle blobs and constraining them to move along a line. The blobs are detected as interframe differences followed by Sobel edge detector. The authors assume that the vehicles are moving towards or from the camera and use mean length of vehicles to obtain the scene scale. While theoretically the method is sound, we found that it had too many assumptions regarding the traffic setting (eg. the variability of camera angle to vehicle location is not accounted) and as a result, it does not produce intended results.

3. Track 1 Methods

To implement speed estimation, our method can be decomposed into three steps: (1) Vehicle detection and instance segmentation implemented by Mask-RCNN; (2) Online MOT model based on deep cosine metric which could generate track-lets from bounding boxes; (3) Speed estimation based on 3D coordinates reconstructed from vehicle segmentation via google map transformation matrix. Here a short-term accurate track-let will be sufficient for speed estimation, therefore we select a online MOT method, even though online methods are likely to cause more identity switches in a long run. In the next sections, more details about these steps will be introduced.

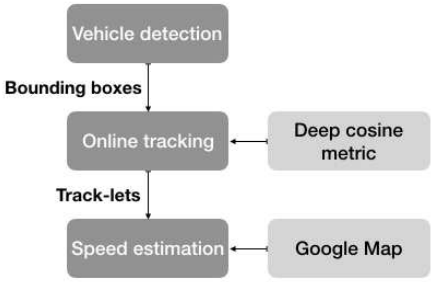

Figure 1. Track1 framework.

3.1. Mask RCNN

Amid different RCNN models, we select Mask-RCNN owning to its high accuracy. It uses RFCN for object detection and adds one more ROIAlign layer for instance-level segmentation, so that the segmentation provides more precise position information than bounding boxes, especially when the vehicle orientation is not aligned with the image borders. Besides, the network uses ResNet101 as a backbone and adds feature pyramid layers for multi-scale object detection, so that it can detect different scale vehicles.

3.2. Deep Cosine Metric Learning

To track the detected vehicles, the re-identification features for the deep-sort[15] are needed. The paper[14] modifies the standard softmax classifier, so that it can produce compact clusters in representation space, and calls it as cosine softmax classifier. It is defined as follows:

$$P(y_i = k|r_i) = \frac{\exp(\kappa \cdot \tilde{w_k^T} r_i)}{\sum_{n=1}^{C} \exp(\kappa \cdot \tilde{w_n^T})} \quad (1)$$

where κ is a free scaling parameter, $r_i = f(x_i)$ is the underlying feature representation of a parametrized encoder network which is trained jointly with the classifier. Here, l_2 normalization must be applied to the final layer of the encoder network to ensure the representation is

Table 1. Overview of the CNN architecture.

Name	Path Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 128$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 128$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 64$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 64$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 64$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 32$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 32$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 16$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 16$
Dense 10		128
l_2 normalization		128

unit-length and the weights must be normalized to unit-length. In training, the cross-entropy loss can be used in its standard form. This parameterization could enforce a cosine similarity on the representation space. By minimizing the loss function, samples in the same class are pushed away from the boundaries and toward the parameterized mean. The network architecture is shown in table 1.

3.3. Simple Online and Realtime Tracking with a Deep Association Metric

Similar to the method used in the paper[15], the association is solved by combining the Mahalanobis distance between predicted Kalman states and the newly arrived measurements, and appearance descriptor r_i, the output of the CNN in table 1.

The Kalman state is defined on an eight dimension space$(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$, where (u, v), γ, h are the center of the bounding box, aspect ratio and height respectively. $(\dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ are their respective velocities. The Mahalanobis distance is defined as:

$$d^{(1)}(i,j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i) \tag{2}$$

The projection of the i-th track distribution into measurement space is denoted as(y_i, S_i), and j-th detection bounding box as d_j.

The last appearance descriptors $\mathcal{R}_k = \{r_k^{(i)}\}_{k=1}^{L_k}, L_k = 100$ for each track k is kept. The appearance similarity is measured by the cosine distance between the i-th track and j-th detection in appearance space:

$$d^{(2)}(i,j) = \min\{1 - r_j^T r_k^{(i)} | r_k^{(i)} \in \mathcal{R}_i\} \tag{3}$$

To combine them, a weighted sum is used:

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{4}$$

where an association is admissible if it is within the gating region of both metrics:

$$b_{i,j} = \mathbb{I}[d^{(1)}(i,j) \leq t^{(1)}] \cdot \mathbb{I}[d^{(2)}(i,j) \leq t^{(2)}] \tag{5}$$

The matching cascade is shown in algorithm 1.

Algorithm 1: Matching Cascade

Input: Track indices $\mathcal{T} = \{1, ..., N\}$, Detection indices $\mathcal{D} = \{1, ..., M\}$, Maximun age A_{max}

Compute cost matrix $C = [c_{i,j}]$ using equation 4 ;
Compute gate matrix $B = [b_{i,j}]$ using equation 5;
Initialize set of matches $\mathcal{M} \leftarrow \emptyset$;
Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$;
for n *in* 1 *to* A_{max} **do**
  select tracks by ages $\mathcal{T}_n = \{i \in \mathcal{T} | a_i = n\}$;
  $[x_{i,j}] \leftarrow min\_cost\_matching(\mathcal{C}, \mathcal{T}_n, \mathcal{U})$;
  $\mathcal{M} \leftarrow \mathcal{M} \bigcup \{(i,j) | b_{i,j} * x_{i,j} > 0\}$;
  $\mathcal{U} \leftarrow \mathcal{U} \backslash \{j | \sum_i b_{i,j} * x_{i,j} > 0\}$;
return $\mathcal{M}, \mathcal{U}$

3.4. Speed Estimation

It is reasonable to assume that all vehicles are in the same horizontal plane. Thereby, to convert the pixel distance to real world distance, we could use the simple projection matrix based method. The transformation matrix of each video is calculated by taking 4 points (x_j, y_j), j=1,..4, on image. Their corresponding real-world coordinates points (u_j, v_j), j=1,...4, are measured using Google Map. Then a 3 x 3 projective transformation matrix is calculated from the four point pairs.

In order to apply projective matrix to recover 3d coordinates, all sample points of a vehicle should stay in the same horizontal plane. Therefore, we take advantage of the bottom part of vehicle's contour provided by Mask RCNN as shown in Figure 2 and consider the central position of contour points as the vehicle position. Then, given the projective matrix, the 3d coordinate of each vehicle is reconstructed from smoothed 2d coordinates. Velocity is estimated based on the difference between smoothed position coordinates.

4. Track 2 Methods

4.1. Fisher Vector Descriptor

When applying Fisher Vector, the papers [6][9] claim to use Gaussian Mixture Model(GMM) with K clusters with μ_k, Σ_k, π_k for each cluster where μ_k, Σ_k, π_k are the mean, variance and prior probability of k-th cluster.

Figure 2. Sample points extraction. The red curve is the orginal Mask-RCNN segmentation contour and the blue curve is of the select sample points.

Based on the paper [12], we use Fisher encoding to encode the vector by:

$$u_k = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ki}\left(\frac{x_i - \mu_k}{\alpha_k}\right) \tag{6}$$

$$v_k = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} q_{ki}\left[\left(\frac{x_i - \mu_k}{\alpha_k}\right)^2 - 1\right] \tag{7}$$

where N is the number of datapoints, q_{ki} is the posterior probability of i-th data in k-th cluster, x_i is the vector we intend to encode. By concatenating v_k and u_k, we form our Fisher vector with the length of $2D'K$ where D' is the reduced dimensionality after applying PCA.

4.2. Vector of Locally Aggregated Descriptor Encoding

VLAD encoding is treated as the simplified version of Fisher encoding. In the paper [17], a k-means clustering algorithm is first implemented to have 64 clusters. And it calculates the VLAD vector representation u_k by:

$$u_k = \sum_{i:NN(x_i)=c_k} (x_i - c_k) \tag{8}$$

where NN(x_i) indicates x_i's nearest neighbors among center c_k. The paper finds out that it achieves the best performance when K = 64.

In addition, other papers[9][19] also claim that Fisher encoding can be thought as the best image descriptor for image classification purpose, and VLAD can be treated as the best video representation vector. The only difference is that these two papers use different number of clusters with different dimensionality of data.

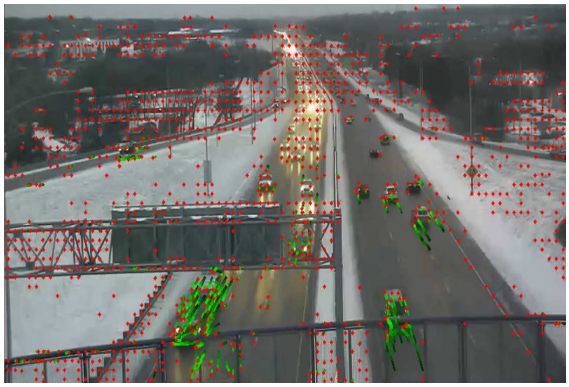

Figure 3. Visualization of iDT features

4.3. Sparse Tracking

We also try to apply sparse tracking algorithm to include temporal information. This algorithm uses VGG Net to generate features and a Siamese network to match objects. The reason that it is called sparse tracking is because we do not match every pixel between two consecutive frames. Instead, we only match the objects that potentially have moved between the two frames. Similarly, for the tracking system, we input $N = 50$ frames into system, and then have $N-1$ vectors to represent the sequence of moving. Each vector has the fixed length of 256 which describes the motion of all objects between two consecutive frames. Then, we apply Fisher encoding algorithm to encode 49 vectors into a fixed length of 32768. More experimental details will be shown in the later section.

4.4. Improved Dense Trajectory

In [11][12] iDT features have been proposed as the best motion descriptor and claimed to be invariant of camera motion. This is mainly because the proposed vector representation includes all the information from previous work of dense trajectories, Histogram of Gradient, Histogram of Optical Flow and Motion Boundaries Histogram. The key advantage of this method is to eliminate the motion made by camera. In Figure 3 we visualize the iDT algorithm by showing a resultant frame with markers where a green line points out the path and direction of a motion and a red dots marks the ending point of a particular motion.

4.5. Full Pipeline

For this specific case, we down-sample frame rate from input stream. In particular, we uniformly sample two frames per second. This is simply because the model we trained is based on low frame scenario rather

than 29 frames per second. Empirically, this down-sampling techniques shows a better result.

To build our full pipeline, we construct two separated branches to process raw data. The first branch serves as an image feature extractor such as VGG Net to extract futures from raw images. Then, the extracted features are encoded using VLAD encoding to form a fixed length vector. The second branch serves as a motion feature extractor such as sparse tracking or iDT from raw images. Similarly, we apply Fisher encoding on tracking vectors. At the end, we merge two branches together by concatenating two vectors and feed them into SVM to train six different classifiers.

We input training frames as a batch where each batch contains 50 frames. The resultant VGG Net image feature vector has the length of 4096 for each frame. After applying PCA, we can reduce this to 256. For GMM, we set K=64. Thus, Fisher encoding produces a vector of length $2 \times 64 \times 256$. Similarly, we use K = 64 coarse centers for VLAD encoding. As a result, VLAD encoding will lead to the size of 64×256 of the output vector. For the tracking part, sparse tracking will generate a vector of 256 for two consecutive frames. With a 256-clusters Fisher encoding, it produces a vector of length $2 \times 256 \times 256$. On the other hand, iDT tracking algorithm generates a vector of 426 in length on two consecutive frames. We also apply PCA to make it as short as 256. We apply Fisher encoding for a batch of frames, which makes it $2 \times 256 \times 256$ in length.

The full pipeline can be visually seen as Figure 4. It is necessary to point out that the final result of classification can be overlapping. That is, multiple events can happen at the same time. This is mainly because of the nature of data. For example, when an event of car accident occurs, the event of police arrival also often occurs at the same time. This can also be viewed from Figure 4.

5. Experiments

5.1. Deep Cosine Metric Training

We use UA-DETRAC dataset [13] to train the deep cosine metric. UA-DETRAC provides 60 training sequences and 83910 images in total. Considering the low resolution of images shot at night, we removed all night sequences during the training phase. Then we cropped each vehicle based on annotations and collected query images for 366 vehicles. The split ratio between training set and validation set used is 0.9/0.1. The training process is shown as in Figure 5. The final accuracy of validation set is about 91%. Later we apply this pre-trained deep cosine metric on vehicle tracking.

Table 2. Part of MOT evaluation results.

Seq id	20052	39861	40192	40871
Weather	sunny	night	cloudy	rainy
Type	road	T-junction	road	busy road
#Frame	692	739	2192	1718
#Car	43	14	313	37
#MT	21	10	145	18
#PT	19	2	159	5
#ML	3	2	9	14
MOTA	0.72	0.65	0.76	0.54
MOTP	31.26	49.64	23.54	8.19

5.2. UA-DETRAC Evaluation

Vehicle detection: The Mask-RCNN model is pre-trained on the COCO dataset and evaluated on the UA-DETRAC training dataset. UA-DETRAC consists of four different types of cars. However, as the purpose of competition is speed estimation, it is not necessary to distinguish different categories of cars. Therefore, all vehicles are classified into the same class "car". As a result, we get a Precision/Recall (PR) curve as shown as Figure 6 and the average precision (AP) is about 65%.

Multi-object tracking: Given the pre-trained deep cosine metric and detection responses, we further evaluate the MOT model. Part of evaluation results are shown in Table 2, where "mostly tracked" (MT) means that the object is tracked for at least 80 percent of its lifespan and "partly tracked" (PT) means the object is tracked between 20 and 80 percent of its lifespan otherwise the target car is "mostly lost" (ML). MOTA is the multiple object tracker accuracy while MOTP is the multiple object tracker precision. Figure 7 displays the corresponding scenarios. The tracking method achieves good accuracy on UA-DETRAC and become robust to variations on weather and scenes. In addition, from generated videos, it can be observed that the tracking method can handle the turning cars. Even though the orientation of a turning car has changed a lot, the tracker can still successfully track the car. According to the result of sequence 40871 where occlusion occurs more frequently, MOTA drops which means the tracking methods cannot handle occlusion very well.

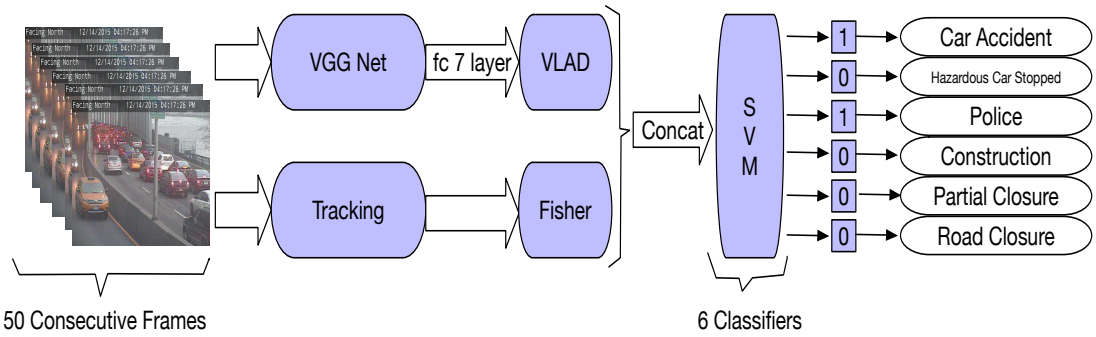

Figure 4. Full Pipeline of Proposed Algorithm

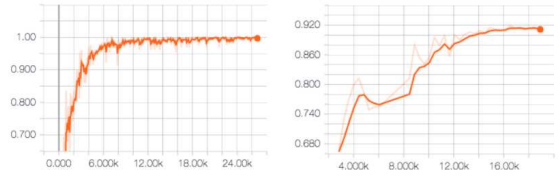

Figure 5. Training process of deep cosine metric. The left one is the classification accuracy of training batches. The right one is the top-1 validation accuracy during training.

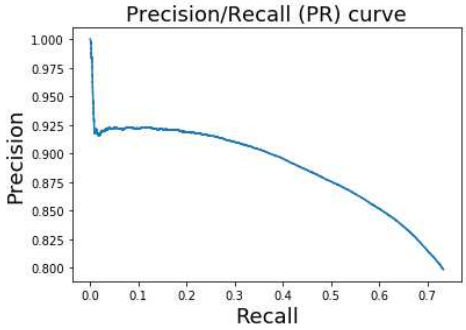

Figure 6. Precision/Recall curve of Mask RCNN.

Figure 7. Sample scenarios for MOT evaluation. The top-left red number is the sequence id (seq id)

6. Results

6.1. Track 1 speed estimation

Track 1 is evaluated on two aspects: detection rate and root of mean square error (RMSE) of speed estimation. According to the submissions, our detection rate is 100% and the overall RMSE is about 7.97 miles/hour. Next, we evaluate the speed estimation of different locations separately. Figure 8 visualizes the speed distribution of different locations. Average vehicle speed on highway (Loc1,2) varies from 60 to 70 miles/hour while average speed on intersection is from 9 to 12 miles/hour. In terms of mean speed value, our result is in a reasonable range. Table 6.1 provides a detailed description about the mean/median value of different locations as well as their RMSEs which are the results on the contest submission system. It can be observed that highway estimation is better than that of intersection. Based on our experiments, we guess that this is caused by three factors: more frequent occlusions at the intersection, unstable detections and shaking cameras. As our method relies on tracking accuracy, it is probably affected by the inaccurate detection of bounding boxes when two cars almost overlap with each other. For example, the bounding boxes will become larger than they should be. On the other hand, as 3d coordinates are reconstructed from a fixed projective matrix, then a shaky camera will affect the accuracy of 3d coordinates and further make speed estimation deviated from true value.

6.2. Track 2 anomaly detection

Track 2 is evaluated based on the detection performance which is based on F-1 score and event time difference judged by RMSE. Our goal is trying to predict the time stamp as accurate as possible without compromising too much on F-1 score. The distribution of the

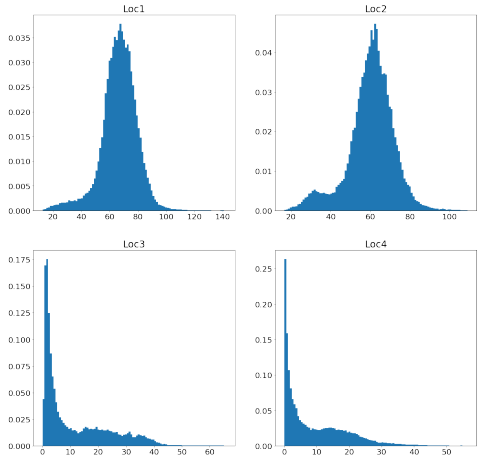

Figure 8. Histogram of velocity from four locations. The x-axis represents vehicle speed (miles/hour) while the y-axis is the number of vehicles within the corresponding speed range.

Table 3. Track1 speed estimation (mile/hour).

Location	Mean	Median	RMSE
1	66.64	67.31	9.61
2	60.15	61.20	10.20
3	11.54	5.27	6.50
4	9.27	5.48	5.50
Overall	-	-	7.97

Table 4. Track2 anomaly detection.

F1	RMSE (seconds)
0.7692	214.2712

predicted time stamps can be viewed in Figure 9. From the frequency plot, we can recognize how model performs on this testing dataset. Correspondingly, in Figure 10, we plot the confidence we have from the model. It is worth mentioning that we also perform a threshold cutting on the confidence score so that we can get as better F-1 score as possible. The final F-1 score and RMSE in seconds can be seen in Table 4. We think that part of reason of this high RMSE can be that we pay too much attention on the end time stamp of an anomaly event in training. Thus, the start time stamp cannot be fully trained. This is also an aspect on which we can improve on. Last but not the least, the sparsity of training data of anomaly can also be the reason for high RMSE.

7. Conclusions

We present our methods for the first two tracks of the challenge, ie., speed estimation and anomaly detec-

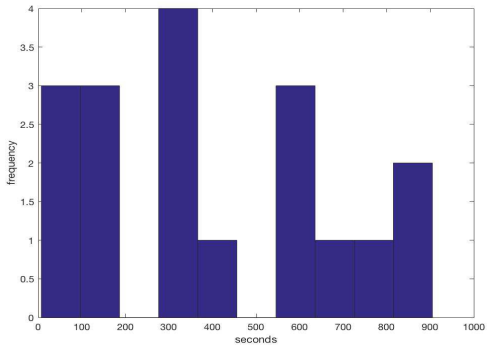

Figure 9. Distribution of Predicted Time Stamp

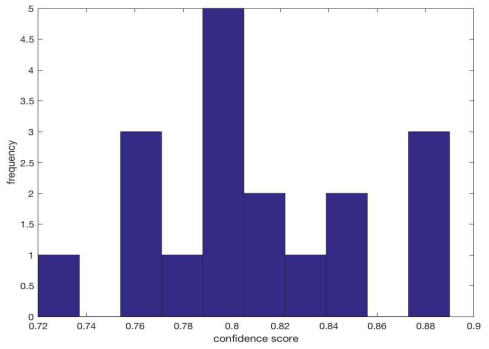

Figure 10. Distribution of Confidence Score

tion. For track 1, we propose a pipeline to estimate vehicle speed and evaluate the detection & tracking steps on UA-DETRAC. Due to the good accuracy in vehicle detection/tracking, our method achieves 100% detection rate, and the average estimation error is about 7.97 mile/hour. Our rank in track 1 is 5/13. For track 2, we develop an optical flow based method. By using the information from optical flow, we can include the temporal relationship between frames. We obtain 0.7692 F1 score and about 214 second estimation error in the contest. We apply a threshold cut on the confidence score to eliminate the unqualified judgment.

In the future, for the speed estimation problem, the first thing is to promote the accuracy by mitigating the aspects of frequent occlusions and increasing the displacement estimation via introduction of the 3D box estimation of vehicles. Secondly, it is essential to optimize our pipeline by speeding up detection response generation, which could be implemented by applying Mask RCNN on key frames and impute the missing frames by interpolation. For track 2, there are two main points that we want to improve on. First of all, we shoud do more fine-tuning to improve the feature extraction from

the VGG Net. In our scenario, we used the pretrained model weights directly for VGG Net, which might not perfectly fit in the case. Secondly, we should consider accumulating more data for the training purpose. Meanwhile, for this specific task, we should choose such an objective function so that only start time stamp matters.

References

[1] W. Choi. Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3029–3037, 2015. 2

[2] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016. 1

[3] D. J. Dailey, F. W. Cathey, and S. Pumrin. An algorithm to estimate mean traffic speed using uncalibrated cameras. *IEEE Transactions on Intelligent Transportation Systems*, 1(2):98–107, Jun 2000. 2

[4] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. 1

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. 1

[6] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. *Computer Vision–ECCV 2010*, pages 143–156, 2010. 3

[7] H. Pirsiavash, D. Ramanan, and C. C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1201–1208. IEEE, 2011. 1

[8] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 1

[9] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013. 3, 4

[10] T. N. Schoepflin, D. J. Dailey, and P. Briglia. Algorithms for estimating mean vehicle speed using uncalibrated traffic management cameras. Technical report, Washington State Department of Transportation, 2003. 2

[11] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011. 4

[12] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3551–3558, 2013. 4

[13] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu. Ua-detrac: A new benchmark and protocol for multi-object detection and tracking. *arXiv preprint arXiv:1511.04136*, 2015. 5

[14] N. Wojke and A. Bewley. Deep cosine metric learning for person re-identification. 2018. 2

[15] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. *arXiv preprint arXiv:1703.07402*, 2017. 2, 3

[16] Y. Xiang, A. Alahi, and S. Savarese. Learning to track: Online multi-object tracking by decision making. In *2015 IEEE international conference on computer vision (ICCV)*, number EPFL-CONF-230283, pages 4705–4713. IEEE, 2015. 2

[17] Z. Xu, Y. Yang, and A. G. Hauptmann. A discriminative cnn video representation for event detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1798–1807, 2015. 4

[18] B. Yang, C. Huang, and R. Nevatia. Learning affinities and dependencies for multi-target tracking using a crf model. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1233–1240. IEEE, 2011. 1

[19] J. Yuan, B. Ni, X. Yang, and A. A. Kassim. Temporal action localization with pyramid of score distribution features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3093–3102, 2016. 4

[20] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1