Multi-Class, Multi-Movement Vehicle Counting on Traffic Camera Data

Vineet Shenoy, Pirazh Khorramshahi, and Rama Chellappa
Johns Hopkins University

{vshenoy4, pkhorra1, rchella}@jhu.edu

Abstract

Multi-Class, multi-movement vehicle counting is essential for determining turning behavior in city-scale intelligent transportation. Understanding traffic patterns and vehicle actions is critical to proper timing of signals for congestion mitigation and usage along certain corridors. This paper presents an end-to-end pipeline for real-time multi-class, multi-movement vehicle counting. We match the tracking results from off-the-shelf trackers to movements represented as parametric curves, outputting results in an online fashion. Our novel contribution is the definition of vehicle actions and the projection of vehicle trajectories in image space onto potential actions. The code can be found at `https://github.com/vineetrshenoy/VehicleCounting2021`

1. Introduction

In recent years, the proliferation of traffic cameras has produced a constant stream of vehicle-related data that can be used for intelligent transportation systems. This data can be used for tasks such as speed estimation, dynamic traffic routing, multi-camera tracking, and traffic anomaly detection. In addition, with the recent advances of Deep Convolutional Neural Networks (DCNNs) [5], significant improvements in classification, detection, and tracking have led to corresponding improvements in traffic analytics.

Given raw traffic camera videos and metadata describing potential vehicle actions, multi-class multi-movement vehicle counting seeks to detect, classify, track, and assign vehicles to pre-defined movements as they exit a scene. Potential actions can include passing through an intersection, making right or left turns, and U-turns; finding robust mathematical representations of these actions is critical. Accurate matching involves handling uncertainty and unusual patterns such as extended stopping at signals and occlusions and re-appearances in a scene. It also includes maintaining performance under adverse lighting and weather conditions such as rain and snow. The AI City Challenge dataset [8] provides this challenging data through video feeds captured at twenty different intersection and highway scenes. This dataset only provides raw video and the description of potential movements at each scene – detection, tracking, and and counting must be independently defined.

We propose an algorithm that efficiently classifies movements and vehicles on this dataset. Our contributions include:

- Creating an end-to-end pipeline that ingests raw video and classifies various movements via a trajectory matching module.
- Novel representations of a vehicle movements (i.e. “left-turn”, “right-turn”) using parametric curves in image space.
- Simple and fast matching algorithms that associate the trajectory of a vehicle with a movement.

We build this pipeline in three stages – a detection stage, a tracking stage, and a counting/matching stage.

The rest of the paper is organized as follows: Section 2 discusses related works for vehicle counting. Section 3 discusses our representation for vehicles movements. Section 4 discusses the algorithm and experiments. Finally, we conclude in Section 5 and discuss future work.

2. Related Works

Successfully matching vehicles to movements requires detection, tracking, and matching modules. Previous works in this area are described below.

Detection: Detection is a critical first step for most algorithms. The AI City evaluation metric, described in section 4, measures run-time performance of algorithms, so previous works [1], [3] chose the YOLOv3 detector [10].

Though the YOLOv3 detector does not achieve the state-of-the-art mAP scores on datasets like MSCOCO [6], it trades accuracy for speed as it is a one-stage detector (i.e. does not use region proposals and detects objects in a single pass). Other detectors, such as [7] [9] used two-stage networks like Faster R-CNN [12]. These networks act in two stages: in the first stage, feature maps are passed to a Region Proposal Network which identifies and crops feature maps where potential objects may be located, and in the second stage classification and bounding box regression occurs. The Feature Pyramid Network incorporated into Faster-RCNN combines feature maps from different backbone levels for better localization at a various scales.

Tracking: Another fundamental task is tracking; this is critical for correct and robust matching. Many works such as [1], [3] used Deep Simple and Online Realtime Tracking (Deep SORT) [15], which integrates both appearance and motion information, as opposed to SORT [2], which only uses motion information. Both algorithms use bounding box information from previous video frames to estimate the motion of objects in the scene; Deep SORT also considers feature descriptors of objects across frames to associate global IDs to unique vehicles, and uses the cosine distance to match the feature descriptors for each vehicle. While appearance information may improve the accuracy of the tracker, it does pay a penalty in terms of speed.

Both trackers described above are considered to be "two-stage" trackers i.e. detections from a previous stage are associated to track objects globally. Other work such as [13] [17] create one-stage trackers which perform detection and tracking simulataneously. Note that these trackers require additional training, which is outside of the prescribed rules for the AI City Challenge.

Another major problem in tracking is identity switching, which occurs when a single vehicle is given two different object identities. Accounting for the variety of ID switches when occluded, the authors in [7] improve their detections with a re-match and a single object tracking strategy.

Matching: Previous works such as [1] [4] used the concept of "entry polygons" to classify movements. After detection and tracking, a vehicle entering the movement's "entry polygon" was classified as that movement. If a vehicle was tracked but for some reason did not enter the movement polygon, then a movement-based k-NN classifier would assign a movement to that vehicle. Similarly, [3] used the notion of "distinguished regions" pairs, two linked regions for each movement, to reduce identity switches and occlusions that frequently result in erroneous movements. As opposed to using polygons, [9] used "line crossings" which classified movements as vehicles crossed starting and ending lines. Using a vector from the center of the first and last tracked boxes, the authors in [9] obtain intersection and orientation information that contributes to their matching algorithm.

Several previous works have used trajectory-based methods. The authors in [7] used computed and manually drawn trajectories as ground-truth movements, and then matched the vehicle tracklets to trajectories. As a major addition, they modified the SORT matching algorithm's distance metric, the Mahanoblis distance, to account for the non-linear motion by adding an identity matrix to the covariance term. During the counting stage they considered the vehicle track as points in space and use the Hausdorff distance to match the points to a trajectory.

3. Approach

In this section we present our proposed approach, **Pa**rametric **C**urve **T**racking (PACT), for Multi-Class, Multi-Movement Vehicle Counting. The pipeline consists of three modules: detection, tracking, and counting modules.

3.1. Detection

A requirement of the 2021 NVIDIA AICITY challenge was to use no external datasets to improve detectors. Therefore, we used off-the-shelf detectors trained on the MS COCO [6] dataset, specifically Faster-RCNN [11]. This detector is a two-stage detector, which generates regions of interest from feature maps using a Region Proposal Network, and then sends these regions downstream for classification and bounding box regression. We obtain detections per-frame, and retain only those that belong to the classes of "car", "bus", and "truck".

3.2. Tracking

All detections for a single video stream are input to the tracking algorithm; we use SORT [14]. Assuming a constant velocity model, a Kalman Filter handles the motion prediction, while the Hungarian Algorithm handles the data association problem. Successful tracking outputs a series of bounding boxes for each frame of the video along with a vehicle identifier consistent for all frames in which the vehicle appears. We compare SORT to the Jointly learned Detector and Embedding (JDE) tracker in [13], which uses visual features and a deep association metric along with motion information to track vehicles. The results from both trackers are presented in Table 1.

3.3. Counting

We propose a novel counting method using parametric curves. As compared to methods such as [1] [3] [4] which

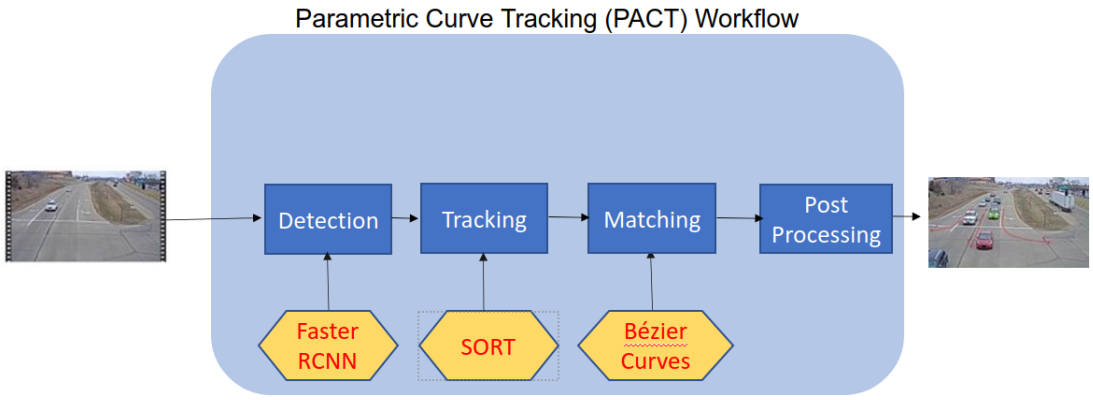

Figure 1. The pipeline for multi-class, multi-movement vehicle counting

only account for a vehicle's starting and ending location, we can estimate a vehicle's *entire* trajectory and encode a notion of *direction*.

3.3.1 Movement Definition

The first step is to mathematically define a certain movement, such as a "left turn" or a "right turn". To do this, we use Bézier curves. Give points $\mathbf{P_1}, \mathbf{P_2}, \ldots \mathbf{P_n}$ a Bézier curve is defined by the polynomial

$$\mathbf{B(t)} = \sum_{i=1}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i, \quad 0 \leq t \leq 1 \qquad (1)$$

The curve begins and ends at point $\mathbf{P_1}$ and $\mathbf{P_n}$, respectively, while points $\mathbf{P_2} \ldots \mathbf{P_{n-1}}$ serve as "control" points, determining any "change of direction" of the curve. For instance, a first-order Bézier curve (a line) has no change in direction, a second-order curve has one change of direction, and a third-order curve has two changes of direction. For the matching module, only a first order curve

$$C(t) = (1-t) \cdot P_1 + t \cdot P_2 \qquad (2)$$

and a second order curve

$$C(t) = (1-t)^2 \cdot P_1 + 2 \cdot (1-t) \cdot t \cdot P_2 + t^2 \cdot P_3 \qquad (3)$$

for $0 \leq t \leq 1$, are used. First-order curves are used for any "straight through" movement, while second-order curves are used for turns. After choosing point P_i appropriately in image coordinates, we can closely match the trajectory of movements, described as metadata in the dataset, as shown in Figure 2.

3.3.2 Movement Matching

The goal of matching is to find the parametric curve that closely resembles the trajectory of a vehicle. For each

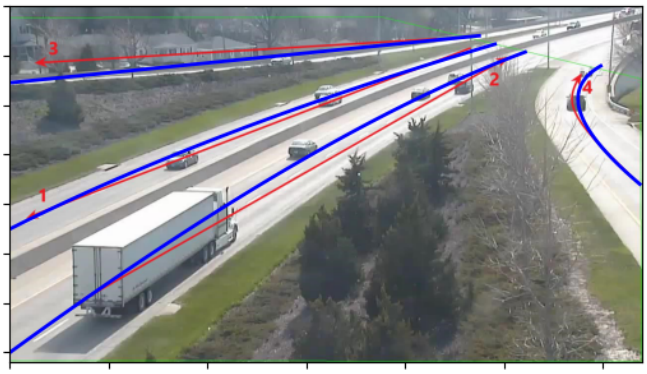

Figure 2. Parametric movement curves (blue) overlayed on camera scene and pre-defined movement curves and number (red). The green polygon is the region of interests and signals when a vehicle should be counted (upon exit). Movement 3 uses a first-order curve while movement 1,2, and 4 use second-order curves

bounding box center $\langle x_i, y_i \rangle$ for $i = 1 \ldots n$ from the tracker, where n is the number of frames in which the vehicle is tracked, the closest point t_i to Bézier curve j, $C_j(t_i) \quad \forall j$ can be found, and a distance d_i from the tracker to a point on the curve is obtained; summing d_i over each curve j results in a cumulative distance from the trajectory to the curve. The curve which has the smallest cumulative distance to the tracker points can be assigned. The full algorithm is defined in Algorithm 1, and a visual representation is in Figure 3. We describe below how to find the closest points from the tracked vehicles to the curve for first and second order curves.

First-Order Curve: Both the tracker output, a set of bounding box centers $\langle x_i, y_i \rangle$ for $i = 1 \ldots n$ per unique vehicle, and the curve itself can be viewed as a series of vectors; the vector on curve $C_j(t_i) = \langle x_i, y_i \rangle$ at time i is the tangential direction of the curve at point t_i. Using an orthogonality argument, the point $t_{i,j}$ closest to the

curve $C_j(t_i)$ from the tracked vehicle vector $\langle x_i, y_i \rangle$ is that point for which the distance vector is orthogonal to the direction of the curve. Letting $(\bar{x}_1, \bar{y}_1)$ $(\bar{x}_2, \bar{y}_2)$ be our curve points, and re-writing equation 2 in vector notation as $C_j(t_i) = \langle \bar{x_1} + t_i \cdot (\bar{x}_2 - \bar{x}_1), \bar{y_1} + t_i \cdot (\bar{y}_2 - \bar{y}_1) \rangle$, we solve

$$\begin{aligned} &\langle \bar{x}_1 + t_i \cdot (\bar{x}_2 - \bar{x}_1), \bar{y}_1 + t_i \cdot (\bar{y}_2 - \bar{y}_1) \rangle \cdot \\ &\langle (\bar{x}_1 - x_i) + t_i \cdot (\bar{x}_2 - \bar{x}_1), (\bar{y}_1 - y_i) + t_i \cdot (\bar{y}_2 - \bar{y}_1) \rangle = 0 \end{aligned} \tag{4}$$

The first term in the dot product is the tangent vector of the curve. The second term is the offset vector from the point to the curve. See figure 3 for a visual understanding of the algorithm. This can easily be solved for t_i in the first-order case as

$$t_i = \frac{-1 \cdot ((\bar{x}_1 - x_i) \cdot (\bar{x}_2 - \bar{x}_1) + (\bar{y}_1 - y_i) \cdot (\bar{y}_2 - \bar{y}_1))}{(\bar{x}_2 - \bar{x}_1)^2 + (\bar{y}_2 - \bar{y}_1)^2} \tag{5}$$

Second Order Curve: The second order case proceeds similarly; given that we have curve and control points $(\bar{x_1}, \bar{y_1}), (\bar{x_2}, \bar{y_2}), (\bar{x_3}, \bar{y_3})$, we find the dot product of the tangential vector (pink arrows in figure 2) and the vector from the point $\langle x_i, y_i \rangle$ to the curve. After algebraic manipulation, we solve a third-order polynomial $0 = at_i^3 + bt_i^2 + ct_i + d$ and ignore roots that are out of the range [0, 1] and select the root which results in the shortest distance vector to $\langle x_i, y_i \rangle$. We find that the our coefficients are

$$\begin{aligned} a &= (\bar{x}_1^2 + 4\bar{x}_2^2 + \bar{x}_3^2 - 4\bar{x}_1\bar{x}_2 + 2\bar{x}_1\bar{x}_3 - 4\bar{x}_2\bar{x}_3) + \\ &\quad (\bar{y}_1^2 + 4\bar{y}_2^2 + \bar{y}_3^2 - 4\bar{y}_1\bar{y}_2 + 2\bar{y}_1\bar{y}_3 - 4\bar{y}_2\bar{y}_3) \\ b &= (-3\bar{x}_1^2 - 6\bar{x}_2^2 + 9\bar{x}_1\bar{x}_2 - 3\bar{x}_1\bar{x}_3 + 3\bar{x}_2\bar{x}_3) + \\ &\quad (-3\bar{y}_1^2 - 6\bar{y}_2^2 + 9\bar{y}_1\bar{y}_2 - 3\bar{y}_1\bar{y}_3 + 3\bar{y}_2\bar{y}_3) \\ c &= (3\bar{x}_1^2 + 2\bar{x}_2^2 - 6\bar{x}_1\bar{x}_2 + \bar{x}_1\bar{x}_3 - x_i\bar{x}_1 + 2x_i\bar{x}_2 - x_i\bar{x}_3) + \\ &\quad (3\bar{y}_1^2 + 2\bar{y}_2^2 - 6\bar{y}_1\bar{y}_2 + \bar{y}_1\bar{y}_3 - y_i\bar{y}_1 + 2y_i\bar{y}_2 - y_i\bar{y}_3) \\ d &= (-\bar{x}_1^2 + \bar{x}_1\bar{x}_2 + x_i\bar{x}_2 - x_i\bar{x}_2) + (-\bar{y}_1^2 + \bar{y}_1\bar{y}_2 + y_i\bar{y}_1 - y_i\bar{y}_2) \end{aligned} \tag{6}$$

Note that only coefficient c and d depend on the the tracked vehicle's coordinates $\langle x_i, y_i \rangle$. Solving this cubic equation can be done using any off-the-shelf solver.

The above procedure is performed for all curves $C_j(t)$, and for each curve, a vector $\mathbf{t_j}$ and $\mathbf{d_j}$ corresponding to the points on the curve and shortest distance to each point is obtained. The distance vector with the smallest cumulative sum corresponds to the movement. See algorithm 1 for a pseudo-code implementation.

Algorithm 1: Vehicle Movement Matching

Result: Movement ID
Input:

- N Tracker Locations, labeled $(x_i, y_i), i = 1 \ldots N$ for vehicle k.
- Control Points for Curves C_j

foreach *Curve* C_j **do**
  $\mathbf{t_j} \leftarrow$ Closest point t_i on C_j to (x_i, y_i), aggregated;
  $\mathbf{d_j} \leftarrow$ Distance d_i from C_j to (x_i, y_i), aggregated;
  if t_j *decreasing* **then**
    $d_j \leftarrow$ MAXINT;
  end
end
mvt-id $= arg\min d_j$

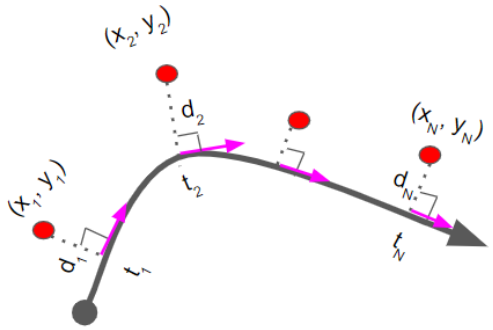

Figure 3. Visualization of the matching algorithm. The gray curve is the Bézier curve for a certain movement. For the for each point from the tracker (red), the closest point on the curve is found, as well as the corresponding distance from the point to the curve. The purple arrows are vectors show the direction of the curve at each point.

3.4. Post-Processing

The majority of movements can be classified based solely by choosing the movement with the smallest cumulative distance. However, the associated vector $\mathbf{t}$ provides valuable information that can be used for matching as well. Firstly, all values of t_i in $\mathbf{t}$ must be between zero and one; a large proportion out of this range signifies an incorrect match to that movement. Secondly, elements of $\mathbf{t}$ must be increasing, usually monotonically. Checking this condition can be especially important for parallel paths in which a vehicle's trajectory may be closer to an incorrect movement (due to jittering of the tracker or imprecise curve definition). Consider movements 1 and 2 in figure 2, and a vehicle's tracked trajectory that corresponds to movement 2 (ground truth). Due to noise in the tracker, the aggregated

distance may be smaller for movement 1 versus movement 2; however, $\mathbf{t}$ will clearly be decreasing from one to zero for movement 1 and increasing from zero to one for movement 2, unambiguously classifying this movement correctly. Clearly, $\mathbf{t}$ encodes a notion of *direction*. Other behavior corresponding to incorrect movements was observed; for example, elements of the $\mathbf{t}$ vector increased to a certain point, then only decreased. This helps eliminate unlikely paths.

Analysis of the $\mathbf{t}$ vector can provide insights beyond the scope of the challenge; for example, an approximate speed can be determined by differentiating the vector with respect to time. Displacement can also be measured. Stationary or stopped vehicles can also be identified by viewing the highly oscillatory behavior of the $\mathbf{t}$ vector, which has applications in anomaly detection. When designing curves on roads, civil engineers can use this information to understand how closely vehicles "hug" the intended trajectories. Entry/exit region-based methods provide no such information.

4. Experiments and Results

4.1. Dataset

One dataset for this problem has been developed as part of the AICITY 2021 workshop [8]. The vehicle counting dataset consists of twenty different camera views from traffic cameras in Iowa, with varying lighting conditions (i.e. dawn, mid-day, etc), as well as different weather conditions (i.e. rain and snow). The majority of scenes focus on intersections, with a limited number of highway scenes for capturing on-ramp and off-ramp movements. The number of movements to classify per-scene varied from as little as two different movements to as many as twelve different movements for a busy intersection. The movement themselves included "right turn", "left turn", "no turn", "highway on-ramp", and "highway off-ramp". Vehicle classification was split into two classes – "car" which was assigned to many smaller vehicles such as cars, minivans, pick-up trucks and mail trucks, and "trucks", which encompassed 18-wheelers, garbage trucks, and many other large vehicles. A region-of-interest (ROI) was also defined to indicate when teams should start and stop tracking of vehicles. A total of nine hours of video was captured at a resolution of 960 pixels or better and the majority of videos were captured at 10 frames per-second. As a part of the challenge, algorithms were evaluated for both "effectiveness" and efficiency; seventy-percent of the score was attributed to "effectiveness" while thirty percent of the total score was obtained from the speed of the algorithm.

Tracker Type	$S1$	$S1_{effectiveness}$	$S1_{efficiency}$
SORT [14]	0.5350	0.7553	0.0209
JDE Tracker [13]	0.4155	0.5683	0.0591

Table 1. The results on the AI City Challenge

4.2. Evaluation Criteria

The algorithms were evaluated for both performance/accuracy as well as the run-time. A combined $S1$ score was obtained a the sum of the performance score and efficiency scores:

$$S1 = \alpha S1_{efficiency} + \beta S1_{effectiveness} \tag{7}$$

where $\alpha = 0.3$ and $\beta = 0.7$. The $S1_{efficiency}$ score was governed by the the execution time (in seconds) and the efficiency base factor, a number encapsulating the performance characteristics of the system:

$$S1_{efficiency} = max(0, 1 - \frac{time \times base\_factor}{1.1 \times video\_total\_time}) \tag{8}$$

The effectiveness score was computed as the weighted average of a normalized weighted root mean square error (nwRMSE):

$$wRMSE = \sqrt{\sum_{i=1}^{k} w_i(\hat{x_i} - x_i)^2} \tag{9}$$

where $w_i = \frac{2i}{k(k+1)}$. Teams were scored based on the $S1$ score only.

4.3. Result

We report our scores in Table 1. We used the Faster-RCNN detector [16] as for all experiments but varied the tracker. The first tracker was SORT [14] and the second tracker was the JDE Tracker (appearance matching portion only) [13].

We found that the appearance descriptor used in [13] suffered greatly from ID switching. In the counting algorithms, this usually resulted in double counting vehicles or not counting vehicles at all due to many broken trajectories. The SORT algorithm was more consistent in this scenario, which resulted in better performance. Using techniques from [7] for more robust trajectory matching is a potential step to further improve performance.

5. Conclusion

In this paper, we have proposed an end-to-end pipeline that ingests raw video, detects vehicles of different classes,

tracks each unique video through the video frames, and assigns a movement to the tracked vehicle. We tested our implementation on the 2021 AI City Challenge [8] and presented the results. Our novel contribution involves defining vehicle movements using Bézier and matching a vehicle's trajectory to that of the curve. In addition, our algorithm quantifies a notion of direction that can unambiguously classify movements in the presence of noise. This work also sets the foundation for work in related tasks, such as vehicle speed estimation and anomaly detection. Future work would include improving the tracking results by limiting ID switching and stitching broken trajectories.

References

[1] Awad Abdelhalim and Montasir Abbas. Towards Real-time Traffic Movement Count and Trajectory Reconstruction Using Virtual Traffic Lanes. *CVPR Workshop*, pages 2527–2533, 2020.

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[3] Khac-hoai Nam Bui, Hongsuk Yi, and Jiho Cho. A Vehicle Counts by Class Framework using Distinguished Regions Tracking at Multiple Intersections. pages 2466–2474, 2020.

[4] J. Folenta, J. Špaňhel, V. Bartl, and A. Herout. Determining vehicle turn counts at multiple intersections by separated vehicle classes using cnns. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2544–2549, 2020.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[6] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer, 2014.

[7] Zhongji Liu. Robust Movement-Specific Vehicle Counting at Crowded Intersections.

[8] Milind Naphade, Shuo Wang, David C. Anastasiu, Zheng Tang, Ming-Ching Chang, Xiaodong Yang, Yue Yao, Liang Zheng, Pranamesh Chakraborty, Christian E. Lopez, Anuj Sharma, Qi Feng, Vitaly Ablavsky, and Stan Sclaroff. The 5th ai city challenge, 2021.

[9] Andres Ospina, Rue Josephine, Baker Stains, and Felipe Torres. Countor : count without bells and whistles.

[10] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. 2018.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[13] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking, 2020.

[14] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.

[15] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *Proceedings - International Conference on Image Processing, ICIP*, 2017-Septe:3645–3649, 2018.

[16] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[17] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points, 2020.