# Countor: count without bells and whistles

Andres OSPINA
CSAI Engie
4 Rue Josephine Baker 93240 Stains - France
gm5734@engie.com

Felipe TORRES
Universidad de los Andes
Cra 1 Nº 18A - 12, Bogotá - Colombia
f.torres11@uniandes.edu.co

## Abstract

*The effectiveness of an Intelligent transportation system (ITS) relies on the understanding of the vehicles behaviour. Different approaches are proposed to extract the attributes of the vehicles as Re-Identification (ReID) or multi-target single camera tracking (MTSC). The analysis of those attributes leads to the behavioural tasks as multi-target multi-camera tracking (MTMC) and Turn-counts (Count vehicles that go through a predefined path). In this work, we propose a novel approach to Turn-counts which uses a MTSC and a proposed path classifier. The proposed method is evaluated on CVPR AI City Challenge 2020. Our algorithm achieves the second place in Turn-counts with a score of 0.9346.*

## 1. Introduction

In modern cities, the abundance of data sources, if exploited well, allows for the improvement of the quality of life. One of the most important sources of information is video cameras that are present around the city. This data could be used to solve many problems such as, but not limited to, traffic estimation, prediction and management. Therefore, different computer vision tasks have been defined to address those problems. Between those tasks are: Vehicle Re-Identification (ReID), multi-target single camera tracking (MTSC), multi-target multi-camera tracking (MTMC).

This paper deals with the one of tasks proposed by the AI City Workshop at CVPR 2020. The task we decided to tackle is:

- Challenge Track 1: Vehicle Counts by Class at Multiple Intersections.

For the Challenge Track 1, the goal is to count vehicles that go through a predefined path or movement ID, as exemplified in Figure 1. The algorithm should count four-wheel vehicles and freight trucks. The vehicle should be counted when it leaves the region of interest (ROI) of each camera which is defined in advance. Moreover, the algorithm should be efficient in terms of program execution. The organizers of the AI City Workshop provided the dataset, which contains 31 video clips captured from 20 unique cameras.

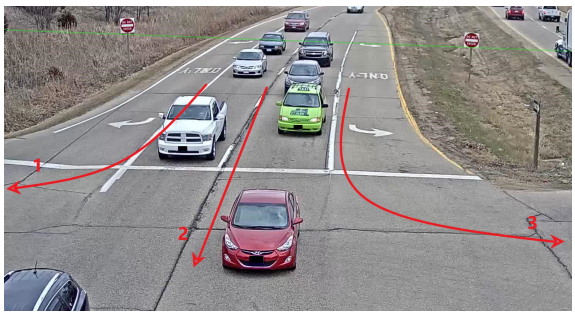



Figure 1. Example of Vehicle Count in one intersection.

It should be noted that the organizers do not allow using any external data for either training or validation if the participant wants to be listed in the public leader board and win the challenge. For further information about the data or the challenges, refer to the organizers [9].

## 2. Challenge Track 1

In this section, we focus on developing a review of the state-of-the art on vehicle counting followed by a description of our solution. For this work we decided to develop a system based on vehicle tracking along with an algorithm of line-crossing. Therefore, we present and explain the tracking and line-crossing algorithms. Afterwards we present the classification of four-wheel vehicles and freight trucks. Finally, we present the results of our experimentation.

### Contributions

The following are the main contributions of this paper regarding Vehicle Counts by Class at Multiple Intersections:

- We created an algorithm that effectively counts vehicles and is efficient during inference. The efficiency and the effectiveness were tested during the challenge,

our algorithm achieved second place in the ranking at $0.9346$ with score ($-0.43\%$ from first place).

- The algorithm exploits a multi-target single camera tracking (MTSC) which can be used for tracking and counting vehicles.
- The algorithm uses a convolutional neural network (CNN) that is trained using the COCO dataset. Therefore, the algorithm can be easily implemented and exploited for other categories.
- We propose a line-crossing algorithm capable of distinguishing multiple movements of the objects by just using the initial and final position of the vehicle.

## 2.1. Related Work

The most commonly implemented solution to count vehicles is using special sensors that are placed directly on the road. The main problem with this solution is that the sensors are not designed to stay on the road. By using cameras, the system can stay in place and will not be not damaged by the vehicles. In this regard, the advantages of using cameras instead of sensors could be significant.

First, we compare the given dataset called Track1 to the state of the art data sources. There are many publicly available datasets for vehicle counting or tracking such as [23, 19, 16, 14, 6]. However, most of those datasets such as [14, 6, 23] are based on the quantity of vehicles present in the pictures and the scope is to detect the vehicle in crowed situations. The datasets [19, 16] that were developed for vehicle tracking and re-identification, still do not have the annotations to count the directions of the vehicles (the dataset [16] is part of the challenge being the Track 3). There is no other dataset to our knowledge that allows to test the same tasks as this challenge.

Moving forwards we review the methods for counting vehicles, that go through a predefined path. First we study the surveys [4, 18, 2] in intelligent transportation systems (ITSs). In [18], a general architecture is presented for video surveillance systems. In this architecture, the vehicle counting is part of "Behavior Understanding" that sits on top of "Extraction of Dynamic and Static Attributes", that contains the detection and tracking of vehicles. This classification helps divide the current task in two depending tasks: Behavior Understanding and Attributes Extraction.

### 2.1.1 Attributes Extraction

This is normally composed of the detection, classification and tracking of objects. Specifically for car counting, this task must run as fast as possible (in order to be used in real time). Therefore, many algorithms still use classical algorithms such as background extraction (ex. [12, 10, 13, 21]) to detect the vehicles. Other authors such as [7, 11] use deep learning detectors such as YOLO and tiny-YOLO that are known for their fast inference. All theses approaches also have a tracking algorithm that allows them to not count the same car multiple times. Consequently, since most algorithms have a tracking part, we decided to review multi-target single camera tracking (MTSC). The review started on the different benchmarks for MTSC such as [16, 8, 19]. The criteria is that the algorithm should be simple to implement in the short period of time available for this challenge, to be fast at inference and have good performance. Some options were DeepSORT [20], TC [17] and MOANA [15]. Those algorithms always have two stages: detection then tracking. We chose to implement Tracktor from [3] (current state of the art in the benchmark [8]) because this algorithm uses the detector to do the tracking, which makes it faster at inference stage.

### 2.1.2 Behavior Understanding

This task, behavior understanding, consists of using the extracted attributes to count vehicles that go through a predefined path. Object counting can be divided in three categories. The first is to count how many objects are in the image. This category enters into crowed object detection (ex. [22]). The second is to count how many objects cross a line. This is the most frequently implemented solution [7, 11, 13]. The last category is to count how many objects go through a predefined path. We did not find an implemented solution for complex paths as the ones presented in the challenge. Our solution is based on line crossing and explained further in this paper.

## 2.2. Countor: Counting system

The system developed for this challenge is called Countor, because it is based on the "Tracktor" from [3]. In the "Tracktor" project, they present different trackers based on the same tracking algorithm. The best tracker that they presented had two add-on modules: re-identification (reID) and camera motion compensation (CMC). We chose not to use the reID in order to reduce the inference time. Also, we do not use the CMC as our cameras are not moving. A diagram of our implementation is shown in Figure 2.

To explain the algorithm, we start by the following definitions: Each tracked object is identified with a number $k$. The trajectory of the object $k$ is defined as an ordered list as $T^k = \{b^k_{t_1}, b^k_{t_2}, \cdots\}$ where $b$ is a bounding box and is defined as $b^k_t = (x, y, w, h, s)$. The elements of the bounding box $b^k_t$ are the coordinates in pixels of the top corner, left corner, the width, the height and a score between 0 and 1 respectively. Also, $t$ is the time of the frame where the bounding box was detected or tracked. The list of active trajectories is defined as $G_t = \{T^{k_1}, T^{k_2}, \cdots\}$, this list

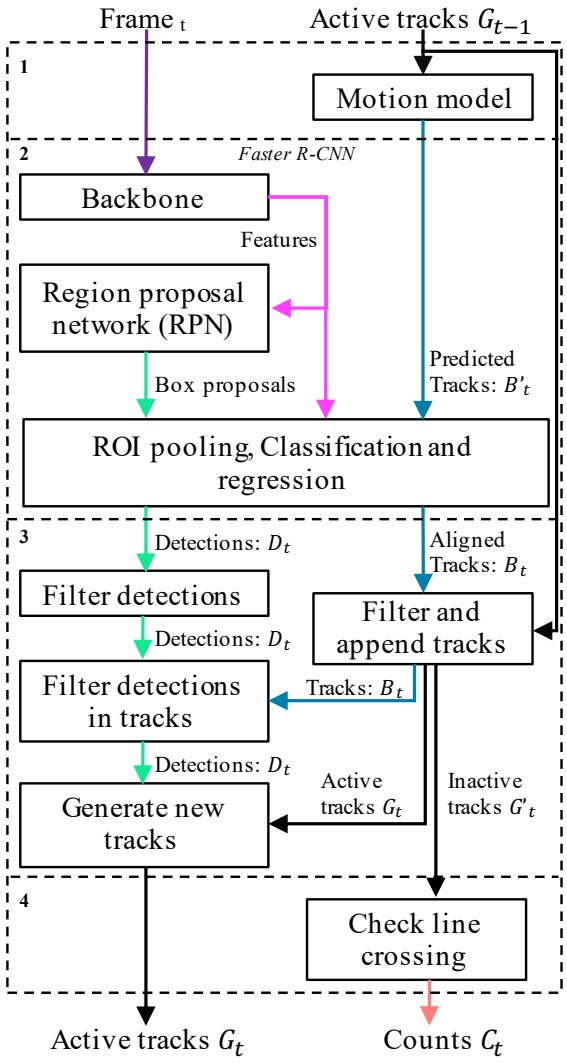



Figure 2. Diagram of the counting algorithm

contains the trajectories of the objects that were not lost up to the time $t$. The list of bounding boxes at time $t$ is defined as $B_t = \{b_t^{k_1}, b_t^{k_2}, \cdots\}$. It should be noted that the list $B_t$ can be extracted from $G_t$. The list of bounding boxes from the detections at one frame in time $t$ is defined as $D_t = \{b_t^1, b_t^2, \cdots\}$. Each time that a track $T^k$ is lost (no longer in from $G_t$) the algorithm generates the counter element $c^k = (t_f, m_{id})$, $t_f$ is the time where the track is lost and $m_{id}$ is the movement ID of that track (remainder the movements ID are exemplified at Figure 1). The list of counter elements is defined as $C_t = \{c^{k_1}, c^{k_2}, \cdots\}$.

The input of the algorithm is the frame at time $t$ and the list of active trajectories $G_{t-1}$. At $t = 0$, the $G_t$ is empty. The output is the list of active trajectories $G_t$ and the counted vehicles $C_t$.

To have a better explanation, the algorithm is divided in four parts (the indexes in the diagram 2): Motion model, Tracktor, Filtering and Track generation, and Line crossing.

### 2.2.1 Motion model

For each track in the list $G_{t-1}$ the motion model is applied to estimate the current position at the time $t$ of each track. In this case, a simple linear movement is applied. The estimated bounding box $b_t'^k$ maintains the width and height $b_{t-1}^k$ and the position changes by adding the distance between the center of the last two bounding boxes $b_{t-1}^k$ and $b_{t-2}^k$. This is exemplified in Figure 3. The output of this model is the list of predicted bounding boxes $B'_t = \{b_t'^{k_1}, b_t'^{k_2}, \cdots\}$

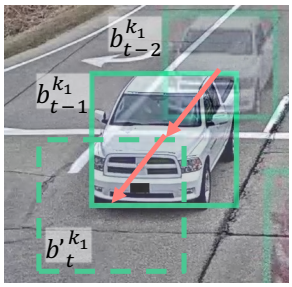


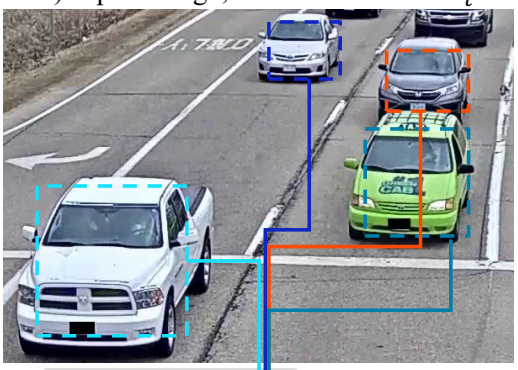

Figure 3. Motion model example for one object $k_1$ to estimate the bounding box $b_t^{k_1}$

### 2.2.2 Tracktor

This module detects the objects in the image and predicts the new positions of the previous tracks. As presented in [3], Tracktor "tackles multi-object tracking by exploiting the regression head of a detector to perform temporal realignment of object bounding boxes" i.e. a detector as the Faster R-CNN can be divided in: backbone (feature extractor), region proposal network and ROI pooling. The outputs of the region proposal network are bounding boxes that are aligned and classified by the ROI pooling. The tracking is done by using the tracks from the previous frame as box proposals. The assumption is that the target has moved only slightly between frames and/or the motion model has already corrected the box proposals. The outputs are the sets of detections $D_t$ and tracks $B_t$. All this is shown in Figure 4.

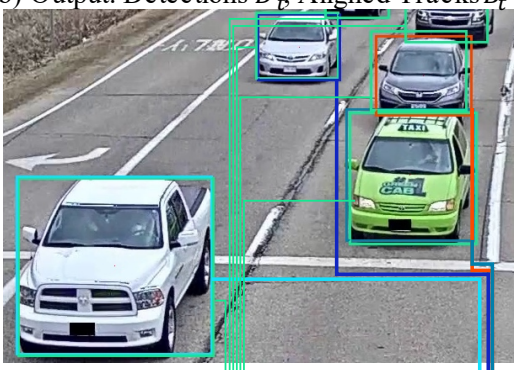
In our implementation, we use a Faster-RCNN pretrained in COCO (directly from torchvision). No further training was done. That detector can detect 91 different classes, therefore, the counting and tracking could be easily applied to people, or other COCO classes. In this case, we use the detentions and classification for three COCO classes: cars, trucks and busses.

a) Input: Image, Predicted Tracks $B'_t$



$B'_t = \{b'^{k_1}_t, b'^{k_2}_t, \ldots\}$

b) Output: Detections $D_t$, Aligned Tracks $B_t$



$D_t = \{d^1_t, d^2_t, \ldots\}$ $B_t = \{b^{k_1}_t, b^{k_2}_t, \ldots\}$

Figure 4. Input and output example for the Tracktor

#### 2.2.3 Filtering and Track generation

As explained in the description of the challenge, the vehicles should be counted when they exit the region of interest (ROI). Therefore, any detected or tracked object outside of this region is filtered out. To achieve this, a filter that computes the percentage of the bounding box inside of the ROI is created, if the percentage is inferior to a defined threshold, the box is filtered out.

**Append and filter tracks**

The list of tracks $G_t$ is created by updating the list of tracks $G_{t-1}$ with the list of bounding boxes $B_t$. Then, the list is filtered by: the ROI filter, minimum score, non max suppression (in the case of overlapping tracks) and minimum area (the area is the height times width of the bounding box). The group of tracks that did not pass the filter are listed as $G'_t = \{T^{k_1}, T^{k_2}, \cdots\}$ and called inactive. The tracks that pass the filter are called active and defined as $G_t$.

**Filter detections**

The detections $D_t$ are filtered by: the ROI filter, minimum score, non max suppression (because we have multiple classes that we suppose as one), minimum area and maximum area.

**Filter detections in tracks**

The detections that are already in the tracked objects are filtered. To achieve this, the list $B_t$ is recreated from $G_t$ the last added bounding box. Then, the score of each bounding box is temporally set to 2. This is set to 2 because the maximum value of the score in the detections is 1. Next, the lists $D_t$ and $B_t$ are jointed and non max suppression is applied. Finally, any bounding box with the score of 2 is removed from the list. Any remaining detections are considered as new tracks.

**Generate new tracks**

For each detection that passed all the filters, a new ID $k$ is created and the track is added to the list $G_t$.

The parameters in the presented filters impact directly the behaviour of the system. In the experiments, we will show how these parameters were chosen.

#### 2.2.4 Check Line crossing

To classify the movement of the vehicle $k$, with the trajectory $T^k$, the center of the first and last bounding box are computed and used as a vector $\vec{V_k}$. As $T^k$ is an ordered list, to generate the vector the first and last element of this list are selected. This representation is shown in Figure 5.a.

The developed algorithm to classify the movements is based on the intersection of vectors. The vector intersection has been largely studied and the solution to this problem can be found in the textbook [5]. The line-crossing algorithm as presented in [5] is capable of differentiating the orientation of the intersection of the vectors.

**Line crossing:** A brief explanation of the line crossing algorithm from [5] is presented. First, The function $F(p,q,r)$ that is the orientation of an ordered triplet of points in the plane is defined as:

$$\begin{aligned} z(p,q,r) &= ((q_y - p_y) * (r_x - q_x)) \\ &\quad -((q_x - p_x) * (r_y - q_y)) \end{aligned}$$

$$F(p,q,r) = \begin{cases} 0 & \text{if } z(p,q,r) > 0 \\ 1 & \text{if } z(p,q,r) < 0 \\ -1 & \text{if } z(p,q,r) = 0 \end{cases} \quad (1)$$

where $p$, $q$, $r$ are a pair of coordinates in a plane as $(x, y)$. The output is interpreted as 0:clockwise,

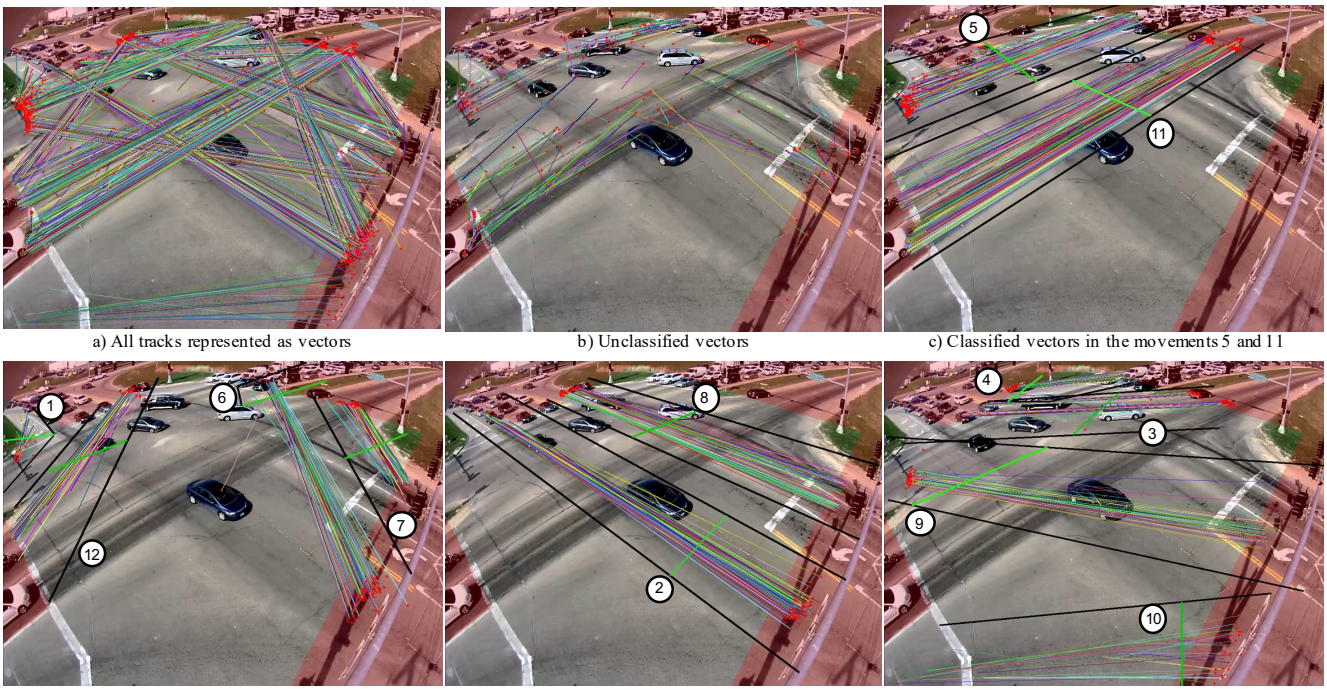



Figure 5. Vectors representing the paths of the vehicles across one camera. Each line represent a different vehicle $k$. The direction of the vector is given by the red crosses. The zone that is not red is the ROI

1:counterclockwise and $-1$:co-linear. By [5]: in general two vectors $\vec{V_1} = (p_1, q_1)$ and $\vec{V_2} = (p_2, q_2)$ intersect if $F(p_1, q_1, p_2)$ and $F(p1, q1, q2)$ have different orientations and $F(p2, q2, p1)$ and $F(p2, q2, q1)$ have different orientations. There exist other cases of co-linear vectors, but in our case we consider that they do not intersect. The intersection function is defined as:

$$\begin{aligned} o_1 &= F(p_1, q_1, p_2) \\ o_2 &= F(p_1, q_1, q_2) \\ o_3 &= F(p_2, q_2, p_1) \\ o_4 &= F(p_2, q_2, q_1) \end{aligned} \tag{2}$$

$$H(\vec{V_1}, \vec{V_2}) = \begin{cases} (1, o_1) & \text{if } o_1 \neq o_2 \wedge o_3 \neq o_4 \\ (0, 0) & \text{otherwise} \end{cases}$$

The output of the crossing line function is 1 if the vectors intersect and $o_1$ the orientation is 1 or 0. We define these variables as bool (1:TRUE and 0:FALSE). We also define the calibration vectors as explained in Figures 5.c to 5.f. The black vectors should not be crossed by the vector $V_k$. In the other hand, the green vectors should be crossed in the right direction. For each movement and each camera we then define a logical table as explained in table 1.

Finally, for every element in the inactive list of tracks $G'_t$ the vector $V_k$ is generated. In order to filter noise tracks, any vector $V_k$ with norm inferior than 50 pixels is ignored.

| Movement id | Intersection function | Intersection | Orientation |
|---|---|---|---|
| Movement 1 | $H(\vec{V_1}, \vec{V_k})$ | TRUE | FALSE |
| | $H(\vec{V_2}, \vec{V_k})$ | FALSE | X |
| | $H(\vec{V_3}, \vec{V_k})$ | FALSE | X |
| Movement 2 | $H(\vec{V_4}, \vec{V_k})$ | TRUE | TRUE |
| | $H(\vec{V_5}, \vec{V_k})$ | FALSE | X |
| | $H(\vec{V_6}, \vec{V_k})$ | FALSE | X |

Table 1. Example of the logical table about the intersection on the vector $V_k$ and the calibration vectors $V_1, V_2, \ldots$

Next, the vector $V_k$ is intersected with every calibration vector $V_1, V_2, \ldots$. That result is compared with the logical table of the camera to classify the object $k$. For each $V_k$ a counter element $c^k = (t_f, m_{id})$ is generated with the $t_f$ that is the last frame the track was followed and $m_{id}$ is the movement ID of that track. Finally The list of counter elements is generated as $C_t = \{c^{k_1}, c^{k_2}, \cdots\}$.

## 2.3. Vehicle classification

The detector we used is not able to differentiate four-wheel vehicles from freight trucks. Because we cannot train in external data, we decided to detect every car, truck, bus or freight trucks as vehicles. To classify the freight trucks, we know that they are bigger than the most four-wheel vehicles (except from buses). Then, we added a module that computes the mean area of each tracked object $k$ (the area of the

bounding box). As the size of the cars depends how far is from the camera, we do the classification by movement ID. Then, for each movement ID, we calculate the average area of the 90% of the smallest vehicles. The assumption is that the bounding box of freight trucks is at least 3 times bigger than the average vehicle, so any vehicle with a mean area bigger than 3 is considered as a freight truck.

### 2.4. Experiments

The experimentation is divided in 2 stages: Finding the correct parameters of the tracker (the filter parameters) and calibrating the cameras. All the experimentation was done is a DGX-2, a computer with 16 GPUs tesla V100 with 32Gb each of memory and a Dual Intel Xeon Platinum 8168, 2.7 GHz, 24-cores.

The first set of parameters were selected from the original tracktor [3]. However, the cameras that are far from the objects as the cameras 1,2,3 and 7 had poor tracking performance. Consequently, we made some exceptions and manually tuned these cameras by visualising the results in the videos. With those parameters, a first iteration of camera calibration was made (by creating the vectors to classify the movements). Once we were able to count the vehicles, we implemented an optimization method. We created a loss function defined by the number of not counted vehicles minus 5 times the number of counted vehicles. Our reasoning was that we wanted more vehicles to be counted but we also wanted to decrease the number of not counted vehicles. By using the skopt library [1], we created a function that computed the first 2000 frames of each video and returned the loss. Then, we ran the optimization of the parameters for 100 iterations, which calibrated the cameras and improved the tracking parameters.

### 2.5. Results

The evaluation for this challenge is defined by the organizers and is a weighted combination between the efficiency score and the effectiveness score. The details could be found in [9]. The results of the challenge are shown in Table 2. We achieved second place. It should be noted that the system without the classification of four-wheel vehicles and freight trucks achieved a score of $0.9010$, which means that if placed in the final ranking, it would only be at fourth place.

| Rank | Team ID | Team Name | Score |
|---|---|---|---|
| 1 | 99 | Everest | 0.9389 |
| **2** | **110** | **CSAI** | **0.9346** |
| 3 | 92 | INF | 0.9292 |
| 4 | 26 | Orange-Control | 0.8936 |

Table 2. Score on Track 1. Our team is shown in bold

## 3. Conclusions

In this paper, we present our approache for the track 1 of the AI City Workshop at CVPR 2020.

For the track 1, a novel approach for counting vehicles that go through a predefined is presented. From our experiments, the proposed method is efficient and effective. We achieved second place with an score of 0.9346%. In perspective the neural network will be trained in different datasets in order to improve the performance. In the same way most of the operation of the system will be integrated to create end to end system.

## References


[1] scikit-optimize: sequential model-based optimization in Python — scikit-optimize 0.7.4 documentation.

[2] Ma'moun Al-Smadi and Rosalina Abdul Salam. Traffic Surveillance : A Review of Vision Based Vehicle Detection , Recognition and Tracking, 2016.

[3] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixé. Tracking without bells and whistles. *CoRR*, abs/1903.05625, 2019.

[4] N. Buch, S. A. Velastin, and J. Orwell. A Review of Computer Vision Techniques for the Analysis of Urban Traffic. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):920–939, Sept. 2011.

[5] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms, 3rd-edition. *The MIT Press*, 2009.

[6] Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez, Roberto López-Sastre, Saturnino Maldonado-Bascón, and Daniel Oñoro-Rubio. Extremely Overlapping Vehicle Counting. In *Pattern Recognition and Image Analysis*, pages 423–431. Springer, Cham, June 2015.

[7] Jia-Ping Lin and Min-Te Sun. A YOLO-Based Traffic Counting System. In *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 82–85, Taichung, Nov. 2018. IEEE.

[8] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, Mar. 2016. arXiv: 1603.00831.

[9] Milind Naphade, Rama Chellappa, David Anastasiu, Anuj Sharma, Ming-Ching Chang, Xiaodong Yang, Shuo Wang, and Zheng Tang. AI CITY CHALLENGE. aicitychallenges@gmail.com.

[10] Joao Neto, Diogo Santos, and Rosaldo J. F. Rossetti. Computer-vision-based surveillance of intelligent transportation systems. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–5, Caceres, June 2018. IEEE.

[11] G. Oltean, C. Florea, R. Orghidan, and V. Oltean. Towards real time vehicle counting using yolo-tiny and fast motion estimation. In *2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pages 240–243, 2019.

[12] Leonel Rosas-Arias, Jose Portillo-Portillo, Aldo Hernandez-Suarez, Jesus Olivares-Mercado, Gabriel Sanchez-Perez, Karina Toscano-Medina, Hector Perez-Meana, Ana Lucila Sandoval Orozco, and Luis Javier García Villalba. Vehicle Counting in Video Sequences: An Incremental Subspace Learning Approach. *Sensors*, 19(13):2848, Jan. 2019.

[13] Nilakorn Seenouvong, Ukrit Watchareeruetai, Chaiwat Nuthong, Khamphong Khongsomboon, and Noboru Ohnishi. A computer vision based vehicle detection and counting system. In *2016 8th International Conference on Knowledge and Smart Technology (KST)*, pages 224–227, Chiangmai, Thailand, Feb. 2016. IEEE.

[14] Corey Snyder and Minh Do. STREETS: A Novel Camera Network Dataset for Traffic Flow. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 10242–10253. Curran Associates, Inc., 2019.

[15] Zheng Tang and Jenq-Neng Hwang. MOANA: an online learned adaptive appearance model for robust multiple object tracking in 3d. *CoRR*, abs/1901.02626, 2019.

[16] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. CityFlow: A City-Scale Benchmark for Multi-Target Multi-Camera Vehicle Tracking and Re-Identification. *arXiv:1903.09254 [cs]*, Apr. 2019. arXiv: 1903.09254.

[17] Z. Tang, G. Wang, H. Xiao, A. Zheng, and J. Hwang. Single-camera and inter-camera vehicle tracking and 3d speed estimation based on fusion of visual and semantic features. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 108–1087, 2018.

[18] Bin Tian, Brendan Tran Morris, Ming Tang, Yuqiang Liu, Yanjie Yao, Chao Gou, Dayong Shen, and Shaohu Tang. Hierarchical and Networked Vehicle Surveillance in ITS: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):557–580, Apr. 2015.

[19] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking. *arXiv:1511.04136 [cs]*, Sept. 2016. arXiv: 1511.04136.

[20] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017.

[21] Honghong Yang and Shiru Qu. Real-time vehicle detection and counting in complex traffic scenes using background subtraction model with low-rank decomposition. *IET Intelligent Transport Systems*, 12(1):75–85, Nov. 2017.

[22] Shanghang Zhang, Guanhang Wu, Joao P. Costeira, and Jose M. F. Moura. FCN-rLSTM: Deep Spatio-Temporal Neural Networks for Vehicle Counting in City Cameras. pages 3667–3676, 2017.

[23] Shanghang Zhang, Guanhang Wu, João P. Costeira, and José M. F. Moura. Understanding Traffic Density from Large-Scale Web Camera Data. *arXiv:1703.05868 [cs]*, June 2017. arXiv: 1703.05868.