Towards Real Time Vehicle Counting using YOLO-Tiny and Fast Motion Estimation

Gabriel Oltean
Bases of Electronics Department,
Technical University of Cluj-Napoca, Romania
Gabriel.Oltean@bel.utcluj.ro

Camelia Florea
Communications Department,
Technical University of Cluj-Napoca, Romania
Camelia.Florea@com.utcluj.ro

Radu Orghidan
SC Mrobots SRL,
Cluj-Napoca, Romania,
radu@orghidan.eu

Victor Oltean
Computer Science Department,
Technical University of Cluj-Napoca, Romania
oltean.victor10@gmail.com

***Abstract*—Real-time vehicle detection, tracking and counting from surveillance cameras is a main part for many applications in smart cities. Usually, this task encounters some problems in practice, like the lack of real-time processing of the videos or the errors in detection and/or tracking. In this paper we propose an approach for real time vehicle counting by using Tiny YOLO for detection and fast motion estimation for tracking. Our application is running in Ubuntu with GPU processing, and the next step is to test it on low-budget devices, as Jetson Nano. Experimental results show that our approach achieves high accuracy at real time speed (33.5 FPS) on real traffic videos.**

***Keywords*— vehicle counting; Tiny YOLO; motion estimation; GPU processing; real time processing.**

I. Introduction

Real time traffic management systems, having vehicle counting as an important part, has become popular recently due to the availability of low-cost surveillance cameras, the powerful mobile devices for video analysis and the cloud infrastructure.

The purpose of this work is to detect and track vehicles on a video stream and count those going through a defined ROI (Region of Interest). Our approach is based on YOLO (You-Only-Look-Once) [1, 2] for object detection and fast motion estimation for tracking and counting purpose.

The state-of-the-art shows that the subject is still in today's topics [3-9], considering both: vehicle counting problem, and using Deep Learning Methods/YOLO for real time applications.

There are several very recent works that deal with traffic management, to estimate the road traffic density and assess the traffic conditions for intelligent transportation systems. In the work of Hardjono et. al [3], vehicle counting is investigated using various machine methods on four datasets. Through numerous simulations, F1 scores ranging from 0.32 to 0.75 have been successfully obtained for one low resolution dataset, using Background Subtraction and Viola Jones methods, on existing CCTV data. It has been found also that Viola Jones method can improve F1 score, by about 39% to 56%, over Back Subtraction method. Furthermore, the use of Deep Learning especially YOLO has provided good results, with F1 scores ranging from 0.94 to 1 and its precision ranges from 97.37% to 100% involving three datasets of higher resolution.

In the work [4] Ferero and Calderon propose an algorithm for the classification, tracking and counting of vehicles and pedestrians in video sequences. Their detection method is based on YOLO too. They trained and evaluated the performance with a set of more than 50000 labels, which were made available for their use.

The work of Asha et. al [5] proposed a video-based vehicle counting method in highway traffic using handheld cameras. The video processing part is achieved in three stages: object detection by YOLO, tracking with correlation filter and counting. Their proposed method can detect, track and count vehicles accurately but they don't specify the processing time, and we consider it is not a fast-enough solution for real time processing on edge devices as Jetson Nano or Raspberry Pi.

In the work of Silva et. al [6], they consider that even if YOLO v3 works very well on computer it could not be executed on low-budget devices. Therefore, they propose to use for detection an adaptation of the Viola and Jones framework with which they achieved processing rates between 8 and 10 FPS on the Raspberry Pi 2 model.

YOLO [1, 2] is a well-known Deep Neural Network and powerful object detection algorithm with real-time performance on a computer with GPU (Graphics Processing Unit). YOLO has been trained using COCO (Common Objects in Context) dataset and can predict 80 classes such as car, bus, truck, motorbike, person, animals, food, etc. In our implementation we consider 3 classes: car, truck and bus, and we treat them as vehicles. The presence of a graphic card is a real demand to increase the speed of object detection using the pre-trained YOLO models.

Considering the pre-trained models provided by the YOLO web site [1, 2], due to the real time demands of our application we chose to use YOLOv3-tiny version. YOLOv3-tiny assure the proper balance between accuracy and speed. YOLOv3-spp is more accurate but too slow for real time processing, even on

GPU. Since we use videos in our application, we solve the accuracy problem of YOLOv3-tiny by the fact that every vehicle is for sure detected in at least one of the frames (passing through the active ROI). Once we detect a vehicle in the current frame, we track it by checking if it appears detected in the next frames, or, if not detected, we estimate its position in the frame taking in account its last positions and the velocity and direction of movement.

The implementation is done on Ubuntu OS, with GPU on the GeForce GTX 950M graphic card. We tested the proposed method with several videos, with very good results: a counting accuracy of 100% and an average speed up to 33.5 FPS, qualifying our implementation for real-time operation.

II. Proposed Method

A. Application flowchart.

The application flowchart is presented in Fig.1. The very first step is to create two configuration files in which we write the most important settable parameters. First configuration file is for defining the regions of interest. Usually we have two regions – for the two traffic directions. Once the regions are set the processing is done just inside of ROIs. The second configuration file contains parameters needed for object detection and vehicle counting. Also a The Registered Vehicle Buffer (RVB) is generated and initialized.

For each ROI the object detection method, based on YOLO, is applied. All the vehicles from the ROI of the current frame are stored in a buffer RVB (Registered Vehicle Buffer). At each frame we check if a detected vehicle is already registered in RVB. If not registered, it is a new vehicle, so we register it in RVB and count it. If the vehicle is already registered, we update the information about vehicle position and movement. If a registered vehicle is not detected in the current frame, we keep the vehicle in RVB, and predict its future position at the most likely position, considering just the previous positions and its dynamic average movement.

B. YOLO-Tiny based object detection

YOLO is a powerful object detection algorithm with real-time performance on a computer with GPU. Even if YOLOv3-spp is much more accurate, it is too slow for real time processing, even on GPU. Therefore, we choose to use YOLOv3-tiny, which assures the proper balance between accuracy and speed. In our implementation we use the pre-trained model and consider just 3 classes: car, truck and bus, and we treat them as vehicles.

The vehicle detection result, generated by YOLO-Tiny, is possible to have some artifacts due to the reduced performance of Tiny YOLO. One of artifacts is that usually a car is detected twice. The second artifact is some lack of detection of a vehicle.

The version YOLO-Tiny [2] predicts bounding boxes using anchor boxes and multi-label classification, therefore, for the same object we can have different bounding boxes detected, or we can have the same object classified in two different classes (with different scores). If in the current frame a vehicle is detected twice, we filter the resulted bounding boxes to obtain unique identified vehicle.

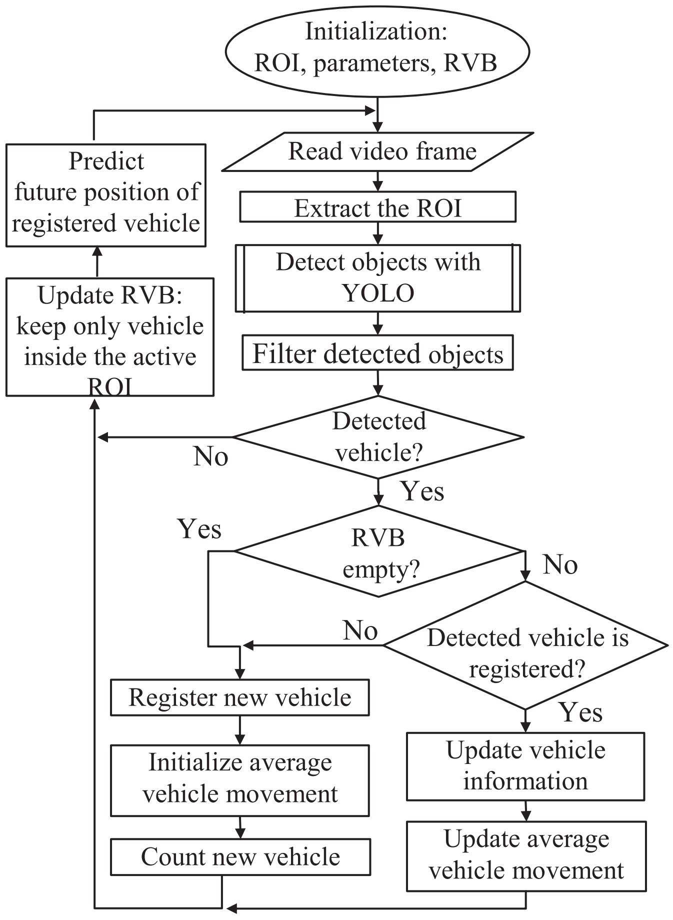
Initialization: ROI, parameters, RVB
Read video frame
Extract the ROI
Detect objects with YOLO
Filter detected objects
Detected vehicle?
No
Yes
RVB empty?
Yes
No
Detected vehicle is registered?
No
Yes
Register new vehicle
Initialize average vehicle movement
Count new vehicle
Update vehicle information
Update average vehicle movement
Update RVB: keep only vehicle inside the active ROI
Predict future position of registered vehicle

Fig. 1. Application flowchart.

If we are in the case of the lack of detection of a vehicle in a certain frame this is solved by doing a prediction taking in consideration the last known or previously estimated position.

For the tracking purpose, once a car is registered in RVB, it receives a unique identifier. When the car goes out of the ROI, its identifier is released and can be reused for a new vehicle.

C. Fast Motion Estimation

The motion estimation is done after each ROI processing. We consider all the present vehicle and estimate the most likely position in the next frame, considering just the previous positions and its dynamic average movement.

When we are checking if the detected vehicle is registered, we consider the minimal distance of this vehicle to all vehicles in RVB. The distance is computed considering the estimated location of vehicles for the current frame, not the real location where the vehicles were at the previous frame detection. If we detect that the current vehicle appears in the RVB we update the vehicle information, modifying the estimated location with the real one. We update the average vehicle movement for a more accurate prediction in the next frame.

The vehicle detection, correct prediction, tracking and counting is maintained as the vehicles moves on the road.

III. Impmelentation and Results

Our application is developed in Python and implemented on a computer running Ubuntu OS and uses the GPU on the GeForce GTX 950M graphic card (650 CUDA core, compute capability 5.0). The presence of the graphic card is a real demand to substantially increase the speed of operation, especially for the object detection using the pre-trained YOLOv3-tiny model. We tested our implementation with several movies containing real traffic videos. Some experimental results are presented and discussed in the following.

First, we consider a test movie, with real traffic, but we will treat only vehicles traveling in one direction. As a consequence, the ROI was selected to include two or more lanes with one-way traffic in the video stream. Fig. 2 contains a collection of four screenshots with the results corresponding to four different frames in the first test movie. In each image, the left window represents the ROI and the results of detection provided by YOLO. Each detected object is characterized by a bounding box (the green rectangle) describing the object location, a label, and a confidence score. The version 3 of YOLO used here, predicts bounding boxes using dimension clusters as anchor boxes and multi-label classification (for the same bounding box we can have different detected objects, with different scores). Also, it is possible to obtain multiple detections for a certain object, meaning that more bounding boxes, with their associated confidence scores, can be generated.

The right window in Fig.2 of each image contains the same ROI but with the final results. It contains uniquely identified objects with the corresponding bounding box. For the tracking purpose, once a vehicle is registered in RVB, it receives a unique identifier, graphically represented by the colored dot placed in the center of the bounding box. In the top left corner of the window, the total number of counted vehicles (Count:) and the number of current video frame (Frame:) are displayed. It is worth mentioning that even in a case where the YOLO misses an object for detection (no bounding box in the left window), due to the prediction, a predicted bounding box appears in the right window, and the vehicle is still tracked.

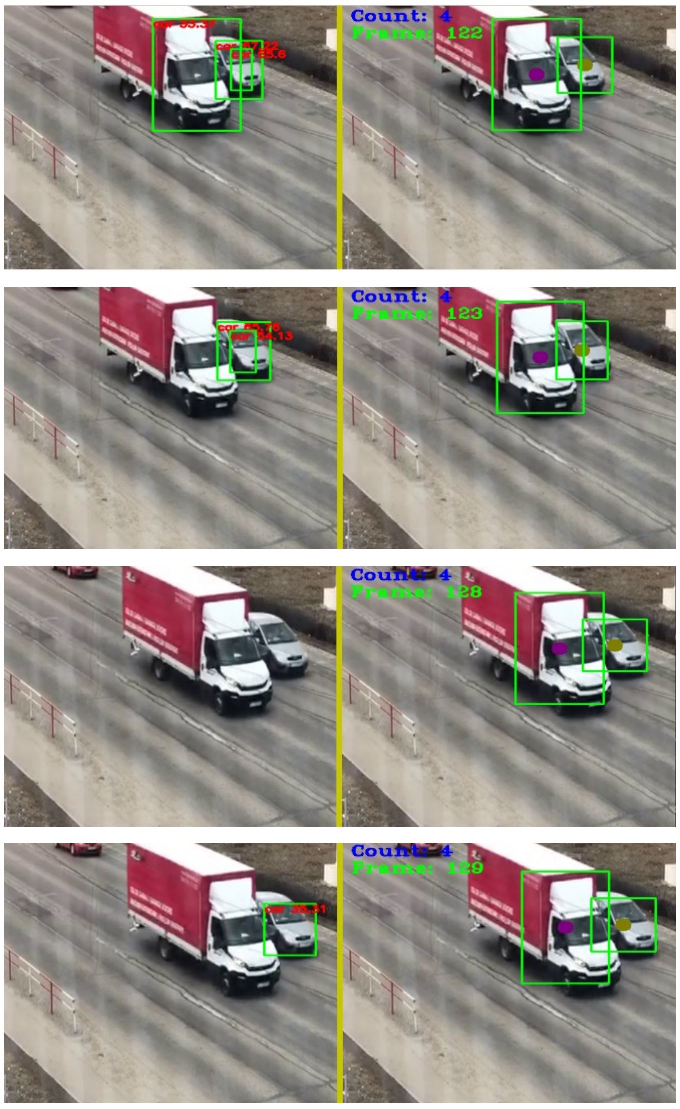

Fig. 2. A collection of four screenshots, illustrating the dynamics of prediction, tracking and vehicle counting for the first test movie.

Going back to the results in Fig. 2, the first image refers to the frame 122, where the detection found three objects. In the case of the big vehicle, its front part is detected as car with a confidence score of 33.37. The small vehicle is detected twice: once with the larger bounding box with a confidence score of 67.22, and second with a smaller bounding box with a confidence score of 65.6. The filtering operation does its job, in the right window only the two existing objects being kept, marked with a purple dot for the big vehicle and with an olive dot for the small vehicle. The total number of cars counted up to the present time moment is 4.

The correct prediction, tracking and counting is maintain as the vehicles moves on the road, as one can see in the rest of images in Fig. 2, that stands for frames 123, 128, and 129, both vehicles keeping their identifier (the color of the dot marking the center of the bounding box), and not increasing the vehicle counter. Due to the robust prediction mechanism, the big vehicle is correctly tracked, even if it is not detected in these frames.

Note also that, even if the small vehicle is not detected in frame 128 (no bounding box for it in left window, third image), but it is re-detected in the frame 129, it is correctly treated as an already registered car, keeping its initial identifier and not increasing the car counter.

As a continuation, Fig. 3 presents the results for the same test movie as in Fig. 2, but for the frame 433 and frame 454. There are three vehicles in the scene, all of them with double detection (see the left window of the image), but with uniquely identification after the filtering operation (right window). They are correctly registered (different color for their graphical identifier), tracked and counted. The total number of unique counted vehicles is now 12. Let us mention that even if the color of identifier for the big vehicle in Fig. 2 and the red vehicle in Fig 3 have the same color (purple), the algorithm operation is correct, because when the big vehicle left the ROI, its identifier was released and so, it can be reused for a new vehicle.

A second test movie is now considered, but the ROI is

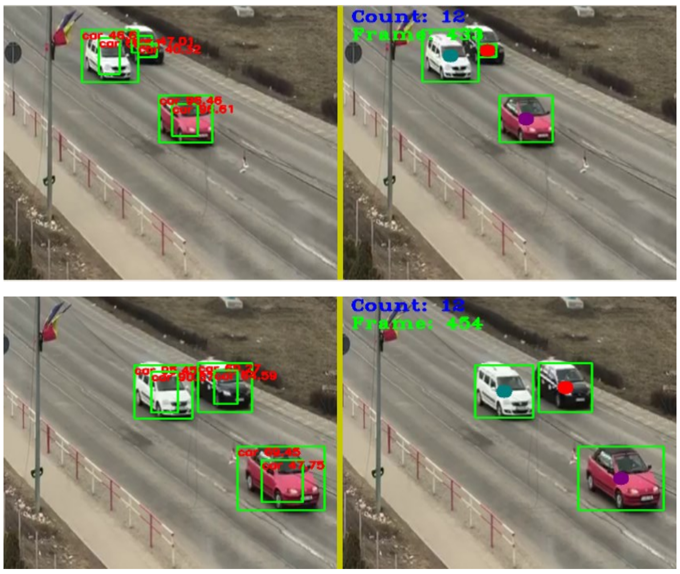

Fig. 3. Two screenshots with the results for the first test movie, frame 433 and frame 454.

defined so that both directions of travel are contained inside. A single counter is used, to count all unique vehicle, regardless their movement direction. Fig 4 shows the results for two frames, 415 and 417. There are four vehicles, three of them going from left to right, and one going from right to left. All vehicles are detected by YOLO. In the frame 415 (top image in Fig. 4), after the filtering operation only three of them remains.

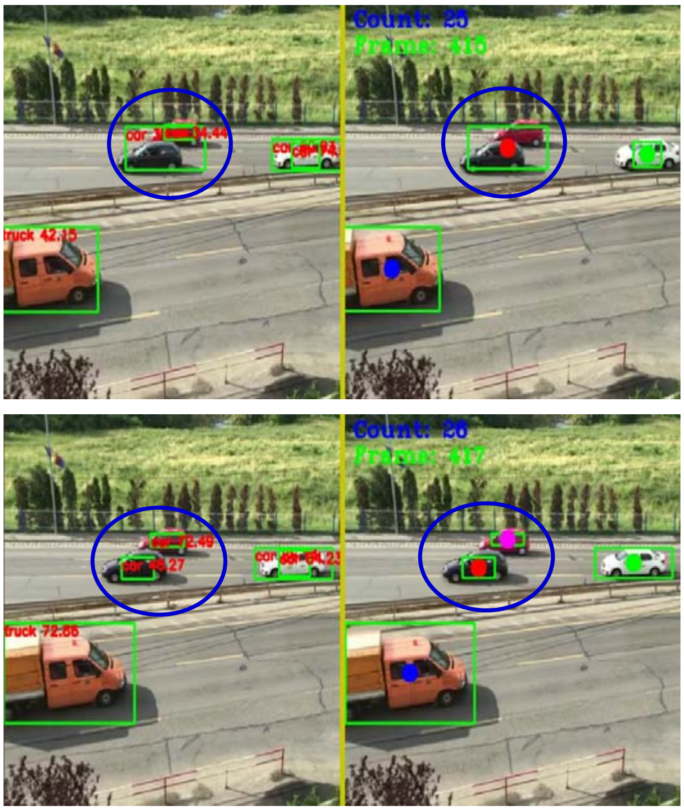

Fig. 4. Two screenshots with the results for the second test movie.

This happens because the bounding boxes for two cars (marked with a blue circle) are superimposed, so the IoU method eliminates the smallest bounding box. As a consequence, only the car marked with a red dot (left window of the image) is counted; at that moment there are 25 vehicles counted. Anyway, the small red car is not definitively lost. Two frames later (frame 417) it is correctly processed and counted, the counter indicating now 26 vehicles. All the other vehicles are correctly tracked and counted during their transit to the ROI.

IV. CONCLUSION

This paper describes a system for vehicle detection, tracking and counting. The vehicle detection uses the pre-trained YOLOv3-tiny model that assures the best trade-off between speed and accuracy. A robust tracking mechanism, based on next position prediction is implemented, for correct vehicle counting. We tested our system using few test movies with real traffic. The system operation is correct both for one way traffic, and two way traffic. A vehicle is counted only once, even if it is detected in multiple frames, or if there is a lack of detection in some intermediate frames, and then the vehicle is detected again. With our Python implementation, under Ubuntu OS, running on a computer with GeForce GTX 950M graphic card, the average speed was 33.5 FPS, qualifying our implementation for real-time operation.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 779–788 [Online 25.05.2019] https://pjreddie.com/darknet/yolo/

[2] J. Redmon, A. Farhadi, “YOLOv3: An Incremental Improvement’, Tech Report , arXiv:1804.02767, 8 Apr 2018, [Online 20.08.2019], https://arxiv.org/pdf/1804.02767.pdf

[3] B. Hardjono, H. Tjahyadi, M. G. A. Rhizma, A. E. Widjaja, R. Kondorura, A. M Halim, “Vehicle Counting Quantitative Comparison Using Background Subtraction, Viola Jones and Deep Learning Methods”, 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 1-3 Nov. 2018;

[4] A. Forero, F. Calderon, “Vehicle and pedestrian video-tracking with classification based on deep convolutional neural networks”, 2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA), Bucaramanga, Colombia, 24-26 April 2019.

[5] C. S. Asha, A. V. Narasimhadhan, "Vehicle Counting for Traffic Management System using YOLO and Correlation Filter", 2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 16-17 March 2018;

[6] L. F. V. Silva, D. R. C. Bandeira, B. M. Carvalho, “A Low-Budget Approach for Vehicle Detection and Occlusion Removal on Traffic Videos”, 2019 International Conference on Systems, Signals and Image Processing (IWSSIP), Osijek, Croatia, Croatia, 5-7 June 2019, pp: 77-82.

[7] G. Natanael, C. Zet, C. Foșalău, “Estimating the distance to an object based on image processing”, 2018 International Conference and Exposition on Electrical And Power Engineering (EPE), 18-19 Oct. 2018, Iasi, Romania.

[8] J. P. Lin, M. T. Sun, “A YOLO-Based Traffic Counting System”, 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI), Taichung, Taiwan, 30 Nov.-2 Dec. 2018;

[9] W. Chen, C. K. Yeo, „Unauthorized Parking Detection using Deep Networks at Real Time”, 2019 IEEE International Conference on Smart Computing (SMARTCOMP), Washington, DC, USA, 12-15 June 2019.