Vehicle counting, tracking and classification in highway roads with Day and Night environment

Quaglio Davide
davide.quaglio@studenti.unipd.it

Zanotti Federico
federico.zanotti@studenti.unipd.it

1. Introduction

Nowadays traffic problems are becoming more and more common in many cities and in order to face them, quality data is needed to make decisions on how to best manage the traffic. For the purpose of resolving this problem, vision techniques like vehicle detection, tracking and counting are becoming more and more advanced and thanks to them we are able to process and analyze a video and extract various types of information like number of vehicles, direction, speed and type.

In this report we mainly focused on vehicle counting, a technique used to fulfill the task of estimating the traffic density in some roads or intersections, in order to efficiently manage congestion in the roads.

In addition, detection and tracking of vehicles in traffic is very useful for traffic surveillance and it could be useful to automatically identify car accidents and intervene immediately.

For this project we decided first to build different systems that are able to detect and track vehicles in order to count them in order to compare the result and test them.
Secondly, we added a classification step on the vehicle, we detected the direction movement and we built a system that is able to detect license plates and perform OCR (Optical Character Recognition) on them. For the detection and tracking part we choose different approaches:

- YoloV3 [1] and Euclidean distances for object tracking
- Background subtraction [2] with OpenCV
- YoloV4 [3] with DeepSORT [4] for object tracking.

We tested all these methods also in a night environment, a challenging task that led to good results. We achieved the best results with Yolov4 DeepSORT, also in the night environment, and a good accuracy for Yolov3 with our tracking method both for day and night videos. The worst performance was obtained by the OpenCV method, especially in the night environment.

The most challenging task was the license plate recognition. The detection of the plate was obtained with a new training with Yolov4 on license plates, but reading plate characters turned out to be very difficult and the results are not very significant.

2. Related Work

Recently, advanced methods of object detection and tracking enable many smart applications to improve vehicle counting and self-driving cars.

Technically, the recent object detection methods can be classified into two approaches which are single-stage and two-stage detectors. Models in the R-CNN [5] family (Fast R-CNN [6] and Mask R-CNN [7]) are all two stage methods: (1) first, the model extracts region proposals from an image. The proposed regions are sparse as the potential bounding box candidates can be infinite. (2) Then a classifier is required to get the classification of the regions.

On the other hand, single-stage (e.g., SSD [8], YOLOv3, YOLO4, RetinaNet [9]) methods skip the region proposal stage and execute the detection directly over a dense sampling of possible locations.

Yolo is nowadays one of the “state of the art” models for object detection but we could also have used R-CNN which was a new way to do object detection compared to the old SIFT [10].

It’s also worth mentioning some newer and comparable alternatives like EfficientDet [11] which has a very similar AP, but is half as fast compared to Yolov4 as we can see in the image below.

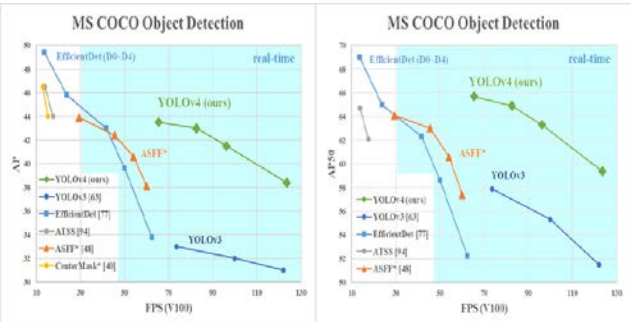

Figure-1: Yolo detection algorithms

EfficientDet is a type of object detection model, which utilizes several optimization and backbone tweaks, such as the use of a BiFPN, and a compound scaling method that uniformly scales the resolution, depth and width for all

backbones, feature networks and box/class prediction networks at the same time

Regarding the object tracking process, SORT is a well-known method which was ranked as the best open-source on the multiple object tracking (MOT) bench-mark [12]. Specifically, the algorithm implements a visual MOT framework based on rudimentary data association and state estimation techniques. Sequentially, DeepSORT, an extension of SORT, has proposed to improve the performance of the tracking by detection paradigm by incorporating deep features for tracking detected objects.

There exist alternatives to Deep Sort like Tractor++ and Track-RCNN [13]. Tractor++ and trackR-CNN are pretty accurate, but one big problem is that they are not viable for real-time tracking. Results show an average execution of 3 FPS for the first one and 1.6 FPS for the second one. DeepSORT is the fastest, with 16 FPS on average, with good accuracy.

3. Dataset

For this project we used different types of datasets, we used COCO[14] and Open Images Dataset V6[15] for the training of the networks, and then we acquired from the internet some videos to test our systems.

3.1. Coco Dataset

Both YoloV3 and Yolov4 were tested with pretrained weights based on the COCO dataset, that is composed of more than 200 thousand images labeled and of 80 different object categories of which 3 are useful for our task which are: car, truck and motorcycle. As we can see in Figure 2 we can have some close-up but also images of some details of the vehicle of interest and in this way we can improve the performance of the trained models.

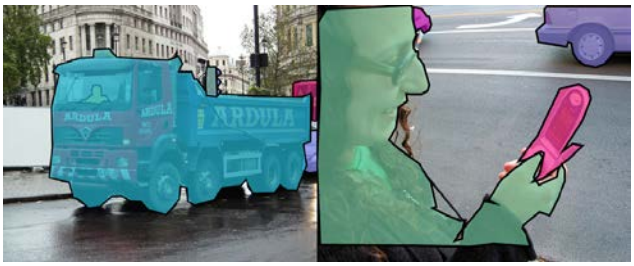

Figure-2: On the left we can see a fully truck depicted in detail, on the right only a wheel and a small part of the car body is visible

3.2. Open Images Dataset V6

We wanted to train our own YoloV4 model so that it would be able to recognize more types of vehicles and license plates, in this way we could acquire a good bounding box of it and then try to apply some OCR techniques to read the text. To do this we chose Open Image Dataset V6 which has all the useful categories contained in the Coco Dataset but also had the Van category and the License plate one. because of our hardware limitations we could select only a small subset for each needed category of 2500 images, gathering a total of 10000 images for the training dataset. To acquire these images and the related labels for the segmentations we used OIDVv4 Toolkit [16] which helped to download a perfectly tailored dataset for our needs.

3.3. Videos dataset

All the models use as input two videos, one during the day and the other one in the night environment, with a complete view from above of a traffic road recorded with a static camera as we can see in the pictures below.

Figure-3: On the left there is a screenshot of video in light environment, on the right the night environment

For the first video we analyzed 1 minute and 19 seconds and for the second one we used 49 seconds.

4. Method

To face the problem of vehicle counting we applied four different methods: Detection and Tracking with OpenCV, Yolov3 with our implementation of Euclidean tracking and Yolov4 with DeepSORT. Every method counts the vehicle if it passes through a virtual line plus or minus a certain offset, meanwhile in Yolov3 we also added a counting type by frame per vehicle type.

In Yolov4 with DeepSORT and in Yolov3 we also added the counting based on the direction of travel.

For detecting license plates, we used Yolov4 trained on our custom dataset and Pytesseract [17] to perform OCR on the license plates text.

4.1. Open CV

The principal component we exploited with OpenCV is the algorithm of Background Subtraction. This is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

BS calculates the foreground mask by performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in

general, everything that can be considered as background given the characteristics of the observed scene.

```
Data: detected = []; Vehicle counter = 0
while Video is opened do
    Convert frame to gray scale figure;
    Apply Gaussian BlurÀpply Background Subtractor to the modified
     frame;
    Find contours of regions highlited;
    Draw the line;
    for all contours in the frame do
        if contour is valid then
            Draw rectangle;
            Find center of rectangle;
            detected.append(center);
            for center in detected do
                if center is passed through the line then
                    Increment Vehicle counter;
                    detected.remove(center);
                else
                    go to the next center;
                end
            end
        else
            go to the next contour;
        end
    end
end
```

Algorithm 1: Open CV Method

Figure-4: OpenCV Method pseudocode

The pseudocode shown above analyzes frame by frame and tries to detect the contours of highlighted parts. These represent the movement that BS observed among two frames. The next step is to check if every contour identified is referred to a vehicle, so if they are above a certain threshold, they are discarded. If contour is valid, then the rectangle is drawn on the frame and the coordinates of the center are calculated. These are all the steps related to detection, while for the counting the method checks if the Y coordinate of the center is minor or greater than the Y coordinate of the line within a certain offset.

For what concerns the tracking part, we implemented an algorithm based on euclidean distance that we discuss in the next paragraph

4.2. Yolo

Yolo is the more recent algorithm approach used for object detection and it stands for “You Only Look Once”. It's an object detector that uses features learned by a deep convolutional neural network to detect an object, in particular it uses Convolutional Neural Network (CNN) to detect objects in real-time. The algorithm requires only a single forward propagation through a neural network to detect objects. This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

The YOLO algorithm consists of different variants. What we investigate in this report are Yolov3 and Yolov4 with two types of tracking: in the first one we implemented an algorithm based on euclidean distances, while in the second we used “the state of the art” DeepSORT. The pipeline in the figure below represents the process of counting vehicles used by these algorithms.

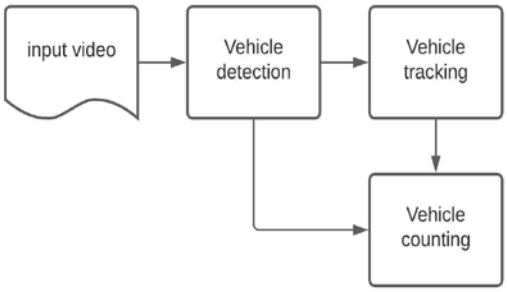

Figure-5: Pipeline

4.3. Yolo v3

Yolov3 is one of the latest versions and it improves the accuracy and especially the speed with respect to Yolov2. For the tracking phase we developed our own method based on euclidean distances of the centers of the bounding boxes, in the Algorithm 2 there is the description of our tracking, in which the tracking and counting part is highlighted.

First of all, the input is a Frame in which Yolo has found a detection with the Box coordinates. The main concept of this tracking is the update of the structure AllCenters. This is a Dictionary in Python (a general map of programming languages) where the key is the ID of the vehicle, while the value is the center coordinates.

After the data structure is initialized and we receive a new frame, the algorithm calculates the euclidean distance between the new center of the new bounding box found by Yolo and all the previously found centers and then we search for which vehicle ID has the minimum euclidean distance from the new one. If this value is below a fixed threshold, then we found a match with the previous frame and we know which vehicle’s center is referred to.

On the other hand, if the algorithm finds a new vehicle when the euclidean distance plus the threshold is too high to be a center already present in the data structure, it takes the last ID created (so the maximum ID) and increases it by one unit and it adds the new vehicle in memory.

```
Data: AllCenters = {ID:(CenterX, CenterY)}; ID=0; Counter=0;
      Threshold=Integer; passed=[ ]
INPUT: Frame; BoxCoordinates
OUTPUT: Frame
NewCenter = FindCenter(BoxCoordinates);
if First Frame then
 | Inizialize AllCenters
else
 | for Center in AllCenters do
 |  | EuclidianDistance = CalculateEuclidian(NewCenter, Center);
 |  | Min = minimum EuclidianDistance;
 |  | ID = ID of Center with minimum Euclidian Distance;
 | end
 | if Min < Threshold then
 |  | AllCenters.update(ID)=NewCenter;
 | else
 |  | ID = Last ID of AllCenters;
 |  | ID = ID + 1;
 |  | AllCenters.add(newID)=NewCenter;
 | end
end
if Center is passed through the line then
 | if ID not in Passed then
 |  | passed.add(ID);
 |  | Counter = Counter + 1;
 | else
 |  | Vehicle already passed;
 | end
end
Return Frame;
```

Algorithm 2: Euclidian Tracking

Figure-6: Euclidian Tracking Method pseudocode

4.4. Yolo v4

Yolov4 is another version that increases the accuracy with respect to Yolov3 maintaining similar speed.

With Yolov4 we used DeepSORT for the tracking phase, the most popular and one of the most widely used for object tracking. SORT means Simple Online Realtime Tracking and a key concept is the Kalman Filter [18]: it helps us factor in the noise in detection and uses prior state in predicting a good fit for bounding boxes.

For example, Yolov3 detects vehicles, but it's not so accurate and occasionally misses detections, like for example 1 misses every 10 frames. To effectively track and predict the next state of a vehicle, let us assume a **"Constant velocity model"**. Now, once we have defined the simple model according to laws of physics, given a current detection, we can make a nice guess on where the vehicle will be in the next frame.

Kalman Filter creates new bounding boxes with track, then, in order to associate new detections with new predictions, DeepSORT authors used squared Mahalanobis distance to quantify the association and the Hungarian method to associate the data.

Despite the effectiveness of the Kalman filter, it fails in many real-world scenarios like when there is occlusion or different point of views. To improve this the authors have introduced another distance metric based on the "appearance" of the object.

Authors first built a classifier over the dataset, trained it till it achieved a reasonably good accuracy, and then strip the final classification layer. Assuming a classical architecture, we will be left with a dense layer producing a single feature vector, waiting to be classified. That feature vector becomes our "appearance descriptor" of the object.

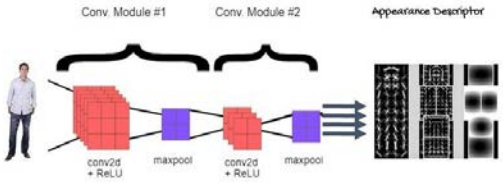

Figure-7: Appearance Descriptor

After the appearance descriptor is obtained, the authors use nearest neighbor queries in the visual appearance to establish the measurement-to-track association.

4.5. Yolo v4 license plate

A challenging task we faced was license plate recognition, and our goal was to detect the license plate in a vehicle and read it in real time.

To achieve our objective, we trained Yolov4 on a dataset that included license plates as category. For this purpose, we created a dataset from Open Image DatasetV6 and selected 5 categories (car, truck, van, motorcycle, plates) with 2500 images each.

With this dataset we trained the weights for our model starting from pre-trained weights on COCO dataset using a Darknet [19] reaching "mAP@0.50 = 77.67 %"

In order to read the license plate, we used PyTesseract on the bounding box of the license plate predicted by Yolo, with some basic steps of preprocessing like blur filtering and threshold and to further improve performance we try to find the contours of the characters and perform on each one of these OCR.

5. Experiments

We compared the goodness of the model with the accuracy criterion: we counted all the cars that passed through the line, every car missed or every car counted twice is an error and we divided all good counting by total number of vehicles. The results are listed in the table below:

	Accuracy	
	Day Video	Night Video
Open CV	69 %	NA
Yolov3	81 %	89 %
Yolov4	97 %	94 %

As we can see the best accuracy is achieved by YoloV4 both in the day and night environments but the algorithm struggles to detect motorcycles, on the contrary even if Yolov3 has lower accuracy than Yolov4, it recognizes almost all the motorcycles in the videos.

The method that uses only OpenCV reaches a sufficient but not very high accuracy that drops dramatically during the night environment.

5.1. OpenCV

The two figures below are screenshot taken by the two videos with OpenCV:

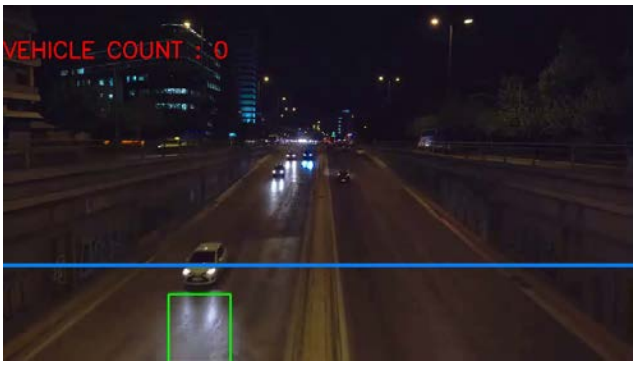

Figure-8: OpenCV in the night video

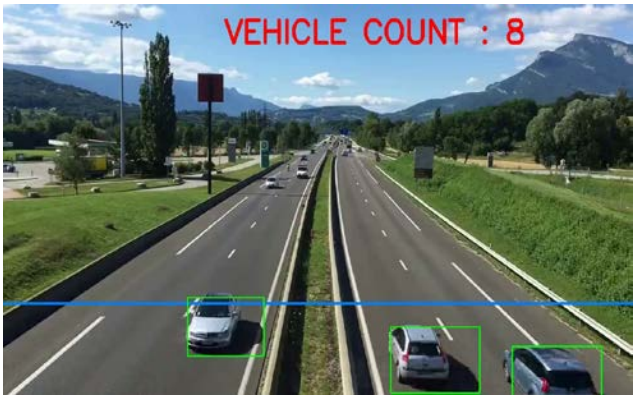

Figure-9: OpenCV in the light video

The cause of this bad accuracy for the night video is related to the fact that the regions highlighted by BS points to the light created by the cars. An example of this situation can be seen in Figure 10 where the most significant area between two frames is the one in front of the car.

Figure-10: BS highlighted regions in night video

5.2. YoloV3

Yolov3 reaches good accuracy, and the tracking algorithm based on euclidean distance works very well, because it identifies every vehicle with a distinct ID. In the figures below there are two examples

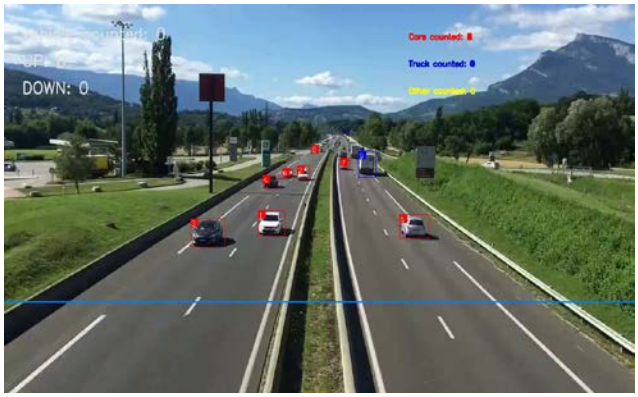

Figure-11: YoloV3 in the light video

Figure-12: YoloV3 in the light video

5.3. Yolov4

The best results are achieved with the state-of-the-art Yolov4 with DeepSORT and some examples can be seen at Figure 13 and Figure 14.

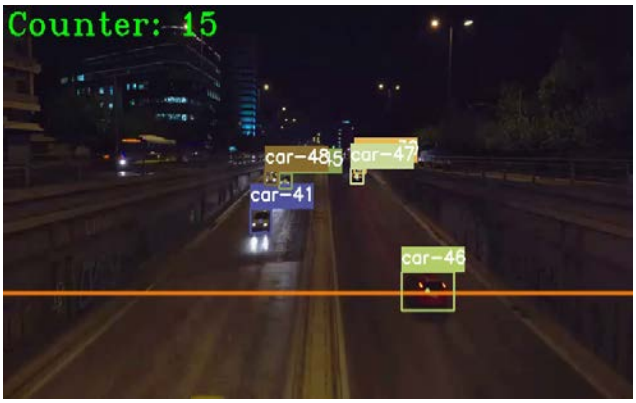

Figure-13: YoloV4 in the night video

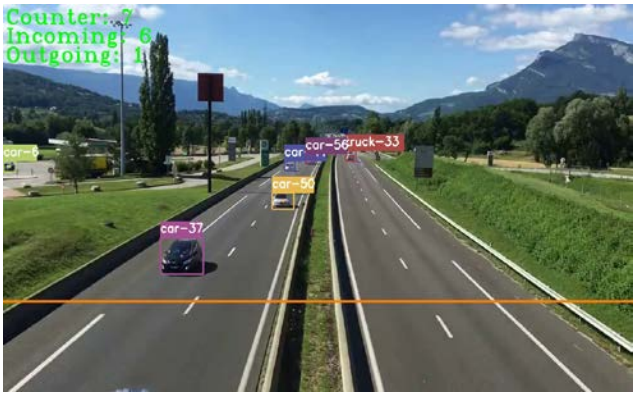

Figure-14: YoloV4 in the light video

As we can see Yolo is able to correctly identify even smaller and farther vehicles from the camera with good precision assigning to it the correct label and most of the time without losing the track of it. It seems like the weights are not trained enough to permit the network to recognize the motorcycles, but aside from it everything else is correctly trained.

5.4. YoloV4 License Plate

We can see in Figure 15 an example taken from running this model and the model loses a lot of accuracy in the recognition task since a lot of vehicles are not detected at all. This is caused by the relatively small dataset that we had to use and by the time constraint to train the network.

As we can see from Figure 2 the license plate number is not right (it's HFK65 and the real number is HFK9765), and everytime the illumination changes the characters read change.

Figure-15: YoloV4 with LP detection and recognition

To improve the performance of the model we tried to make some changes on the model, for example we changed the input size from 416 to 640 since with bigger images the accuracy should increase, but give that the video was not HD and the license plate is a very small part of it the bounding box for it was still small and hard to infer the text on it.

6. Conclusion

In this report we presented a vehicle counting task implemented in three different methods. The aim was to compare the state of the art algorithms and test the tracking systems in order to achieve more precision. The last one visually works very well, but it reaches a lower accuracy with respect to the "state of the art" Yolov4 with DeepSORT. Despite this, the result of our Euclidean algorithm has a good final accuracy. From our point of view, it could be improved with the addition of a Deep Learning part as it was done with DeepSORT.

Another part that could be upgraded is the YoloV4 with license plate training, because taking more images for more categories would improve the accuracy of the weights giving better bounding boxes to work on.

For what concerns the characters recognition on the license plate itself, probably a CNN approach trained to recognize the characters given a dataset of license plates with the correct golden labels would have been able to give better results.

This work could be further extended adding more features to the pipeline, for example we could add the possibility to recognize the color of the vehicles, the speed and predict the direction that they would take on an intersection, giving as result a system with every information needed to take a decision on how to better manage the roads and theirs traffic. Moreover, for the future we could test our methods on AI city challenge [20] datasets to compare our results with the participants.

7. References

[1] Joseph Redmon. “YOLOv3: An Incremental Improvement”, arXiv:1804.02767

[2] OpenCV . (2016). OpenCV : Background Subtraction, https://docs.OpenCV .org/3.2.0/db/d5c/tutorial_py_bg_subtraction.html

[3] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”, arXiv:2004.10934

[4] Nicolai Wojke, Alex Bewley, Dietrich Paulus. “Simple Online and Realtime Tracking with a Deep Association Metric”, arXiv:1703.07402

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. “Rich feature hierarchies for accurate object detection and semantic segmentation”, arXiv:1311.2524

[6] Ross Girshick. “Fast R-CNN”, arXiv:1504.08083

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick. “Mask R-CNN”, arXiv:1703.06870

[8] Liu et al., “SSD: Single Shot MultiBox Detector”, arXiv:1512.02325

[9] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár. “Focal Loss for Dense Object Detection”, arXiv:1708.02002

[10] Lowe, David G. (1999). “Object Recognition from Local Scale-Invariant Features”. doi:10.1109/ICCV.1999.790410

[11] Mingxing Tan, Ruoming Pang, Quoc V. Le. “EfficientDet: Scalable and Efficient Object Detection”, arXiv:1911.09070

[12] Anton Milan, Laura Leal-Taixe, Ian D. Reid, Stefan Roth, and Konrad Schindler. MOT16: “A benchmark for multi object tracking.” CoRR, abs/1603.00831, 2016.

[13] Bing Shuai, Andrew G. Berneshawi, Davide Modolo, Joseph Tighe. “Multi-Object Tracking with Siamese Track-RCNN”. arXiv:2004.07786

[14] COCO dataset, https://cocodataset.org

[15] Open Images Dataset V6, https://storage.googleapis.com/openimages/web/index.html

[16] OIDVv4Toolkit, https://github.com/EscVM/OIDv4_ToolKit

[17] Pytesseract, https://github.com/madmaze/pytesseract

[18] R.E.Kalman. https://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[19] Alexey.Bochkovskiy.Darknet, https://github.com/AlexeyAB/darknet

[20] AI City Challenge 2021, https://www.aicitychallenge.org/2021-data-and-evaluation/