Vehicle Detection, Tracking, and Speed Estimation using YOLO-Based Architecture with Road Line Detection and Spatial Analysis

Matteo Bergamaschi
matteo.bergamaschi@studenti.unipd.it

Simone De Renzis
simone.derenzis@studenti.unipd.it

Davide Varotto
davide.varotto.2@studenti.unipd.it

Abstract

This project focuses on the task of vehicle detection in images and videos, which has been extensively studied in computer vision due to its wide-ranging applications. Recognizing vehicles accurately is a challenge, primarily because of the variations in vehicle sizes. To address this, a recent technique based on the YOLO architecture has been developed.

In this project, the YOLO architecture is employed to detect and classify vehicles in both images and videos. Building upon this, the project goes a step further by incorporating a real-time vehicle counting mechanism. Additionally, the project aims to estimate the speed of the vehicles using an algorithm that is independent of environmental factors and can be universally applied on any road. The proposed method retains the key advantages of YOLO, namely its speed and effectiveness.

The experimental results validate the effectiveness of the proposed approach in tackling these tasks.

1. Introduction

Computer vision is an interdisciplinary field that has been gaining a lot of interest in the last years. An integral part of computer vision is object detection.
Object detection is the task of locating objects in an image with bounding boxes and assigning a class label and confidence for each detection. Hence, an object detection system takes as input an image, with one or more objects, and returns a set of bounding boxes and the corresponding class labels. This is the main difference between object detection and classification algorithms: the first one try to draw a bounding box around the object of interest.
In Deep Learning domain, the detection method can be divided in two category. A two-stage method is used to first generate a candidate box of the object using different algorithm, and then to classify the object using Convolutional Neural Network. On the other hand, the one-stage method directly convert the positioning problem of bounding box into a regression problem for processing. YOLO belongs to the One-stage methods: it divides the image into a fixed number of grids and then these grids are responsible for detecting the objects whose center are inside them [4]. In this project, YOLO is used to detect and classify different vehicles in a image or video. Then, we improve this task adding a *Tracker* that track the position of any vehicles, to count the number of vehicles that cross the road. Furthermore, a experimental algorithm is used to detect the speed of each vehicle, without using any real reference in the video. In order to evaluate the performance of this model, vehicle detection and counting have been tested with different videos of road traffic, available in the GitHub repository at `https://github.com/DavideVarotto/vehicle-detection`.

2. Related Work

A complete analysis on vehicle detection techniques can be found in [6], we are going to present only the major contributions.

2.1. Vehicle detection

R-CNN architecture

Region-based Convolutional Neural Network (R-CNNs) are a family of machine learning models used in image processing and computer vision. Based on Two-stage detector, the original goal of any R-CNN is to detect objects in an image defining bounding box around it.
This is how it works: an input image given to the R-CNN goes through a technique called selective search, to extract information related to the region of interest (ROI). There can be over than 2000 ROI, depending on the scenario. These ROI are fed as input to a Support Vector Machine (SVM) that classifies the presence of the object within that region.[3]

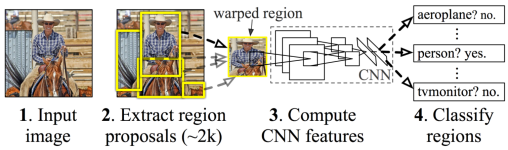

Figure 1. R-CNN architecture [3]

Several challenges are associated with the current models. First, training the neural network is time-consuming due to the need to classify 2000 region proposals per image. Additionally, the models are not suitable for real-time tasks as the processing time for each image is approximately 47 seconds. Another limitation is that the selective search algorithm employed is fixed and lacks learning capabilities, potentially resulting in the generation of poor candidate region proposals. These issues highlight the need for improvements in training efficiency, real-time processing speed, and the development of more adaptive and learning-based algorithms for region proposal generation.

To deal with these issues, others versions of R-CNN have been implemented, in particular Fast R-CNN, which is a faster version of R-CNN that solves problem related to the speed of inference, and Faster R-CNN, which also uses a different algorithm, with respect with selective search, to learn the region proposals.

YOLO

You Only Look Once is a method belonging to the One-stage family: it just goes through the entire image just once. YOLO uses an other point of view for solving object detection: instead of dealing with this task as a classification problem, it treats it as regression one by spatially separating bounding boxes and associating probabilities to each of the detected images using a single Convolutional Neural Network. There are different version of YOLO, starting from the *YOLO_v2* to the actual *YOLO_v8*, with different properties but same architecture, as described in [7]:

- Residual Block: divide the image in a NxN grid cells, each cell is responsible for localizing and predicting the class of the object that it covers.
- Bounding box regression: determine the bounding boxes which correspond to rectangles highlighting each objects in the image. There can be as many bounding boxes as there are objects within a given image.
- Intersection Over Union: a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU is to discard such grid boxes to only keep those that are relevant.
- Non-max Suppression: setting a threshold for the IOU is not always enough because an object can have multiple boxes with IOU over the threshold, and leaving all those boxes might include noise. NMS is used to keep only the boxes with the highest probability score of detection.

R-CNN vs YOLO comparison

These two architecture are the state of art in object detection, in particular in vehicle detection. Same purpose but different methods: which is better?

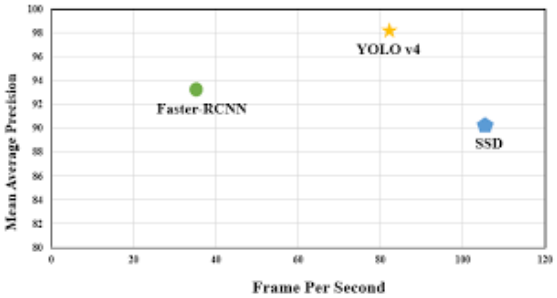

Figure 2. Comparison between R-CNN and YOLO in terms of effectiveness and speed [6]

Based on this paper [6], YOLO seems to produce an higher accuracy and also an higher frame rate, resulting perhaps in the best choice. R-CNN excels in the recognition of small objects, where YOLO still struggles, due to the structure of the model. Same purpose, different methods: there is no a better choice, it depends on the domain in which we apply these architectures.

2.2. Speed estimation

A video source provides images captured at a particular frame rate. It is possible to calculate the speed of a moving vehicles with the $\frac{s}{t}$ formula, where

- t is the time the vehicle takes to navigate from two points in the image: provided a way to track the moving vehicle (YOLO in our case), this is easily inferred from the video stream and its framerate;
- s is the distance between those same two points.

Most of the approaches that are currently used for obtaining the distance in a 2D image rely on one of these principles:

- Using knowledge from manual measurements in the actual environment, such as laser measurements or GPS information. This approach requires an actual interaction with the environment and this can be infeasible or costly in many cases;

- Exploiting the spatial information provided my two cameras: having more cameras pointed in the same direction but from a slightly shifted point of view allows to infer stereo information and reconstruct a 3D model of the environment. Then it is possible to infer the distance between two arbitrary points. This approach requires the presence of two cameras [5];
- Exploiting the knowledge of the dimensions of some standard objects (for example number plates) or of the vehicles themselves to automatically infer the spatial information of the environment. The first approach was unfeasible in our case, as the number plate is a relatively small detail in an often wide angle and low resolution view of traffic; the second one was applied in aerial view, which is not our use case [2].

Our approach is designed to require a single camera, no actual environment measurements and to provide a fairly accurate distance estimation from every scene, exploiting the average measures of vehicles.

3. Dataset

The YOLO model has been trained with MS COCO dataset: it's a Microsoft Dataset used for objects recognition, segmentation, key-point detection, and captioning dataset [9]. The dataset consists of 328.000 images with 80 classes, from various domains, like vehicle, animals, people, and others.
For the purpose of our tasks, we could use only vehicle samples, in particular cars (12786 samples), motorbike (3661 samples), bus (4141 samples) and truck (6377 samples). Since we are not going to train our model, but instead load the weights from a pre-trained version of our, the model used has been trained with the entire MS COCO dataset. The weights can be downloaded from `https://github.com/AlexeyAB/darknet`

4. Method

4.1. Vehicle detection and counting

Our first task is to detect vehicles and count how many of them pass through each lane of the road. By lanes, we mean the sections of the road where vehicles travel in opposite directions.

Vehicles detection

There are 4 types of vehicles that we deal with: car, motorbike, bus and truck. To perform the counting, we need to keep track of the position of each vehicle, in different frame. To do this we make use of:

- the x coordinate, which is used to place the vehicle inside the right line of travel direction;
- the y coordinate, which is used to keep track of the same vehicle over the different frames, in order to count each object just one time. This idea is expanded in the following section, where the Tracker mechanism is explained.

The vehicles are detected, for each line, when they pass trough a region delimited by 2 horizontal lines.

Road line detection

To identify the roads regions, we propose the following pipeline:

1. Identify road lines in the image with *Canny edge detector* and *Hough Line Transform*. Canny edge detector is an edge detection algorithm that uses a multi-stage process to find the edges in an image[1]. Road lines, thanks to their high contrast with the road, respond well to this kind of detection. A minimum threshold in terms of number of detected lines is set, and the two parameters for the hysteresis procedure are iteratively lowered until the threshold is reached. This accounts for images with different brightness conditions. Hough Line Transform is a feature extraction technique that uses a voting procedure to find straight lines in an image[10].

2. The techniques described in step 1 are utilized on the initial n frames of the video, where n is a predetermined fixed number. We accumulate all the lines detected in different frames to make the detection more robust to temporary perturbations of the images (for example, long trucks that easily respond as straight line).

3. K-means clustering is performed to identify k road lines out off all the lines accumulated in step 2. Each line is viewed as a data point described by 4 features (the 2D coordinate of the starting and ending points) and the whole set of lines is fed to K-means, which identifies k lines (the centroids). The parameter k is chosen automatically using the Silhouette coefficient [11].

Vehicles line counting

One of the tasks in road analysis involves monitoring the vehicles within each lane of a divided road. In this task, the goal is to track the position of all the vehicles within the detection area in each frame. The position data of these vehicles is then used to control their movement relative to the coordinates of the outermost points that define the boundaries of the different travel lanes.

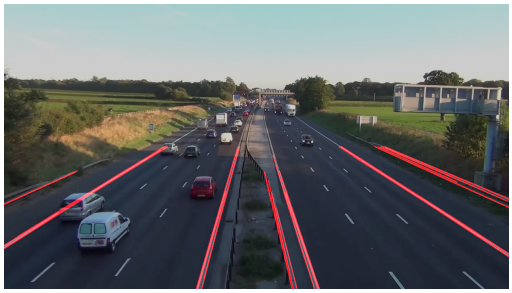

Figure 3. The road lines detected and accumulated in the previous frames

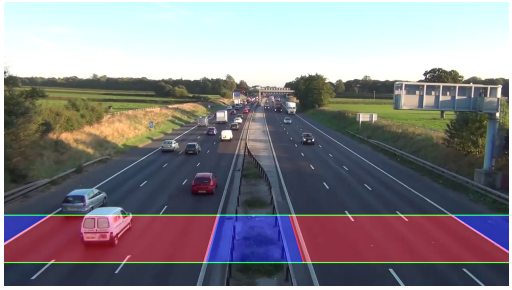

Figure 4. K-means identified 4 road lines out of those accumulated. Road lanes have been coloured by filling the space between every adjacent road line

Tracker

The Tracker is used to track the position and assign same unique ID for each vehicle in different frame, in order to count them just one time. This is how it works: for each frame, the tracker lists all detected vehicles. For each of these, we calculate the distance from the (current) position at time t, and the position of all vehicles at time $t-1$: if one of them is lower than a threshold (25 pixels in our case) it means that we are dealing with the same vehicle: we track the ID and change its position. Instead, if there is no distance lower than threshold, we assign a new ID and start to keep track of this new vehicle. This may seem prone to inaccuracies, particularly when dealing with close proximity between small objects. However, in the context of tracking vehicles within the immediate vicinity of the camera, where the vehicles appear larger, this concern becomes less significant. Therefore, the approach of tracking vehicle positions within the region of detection becomes a reliable and effective method for accurately monitoring the vehicles in question.

4.2. Speed estimation

As mentioned in 2.2, we estimate the speed of vehicle the by estimating the distance between two points and the time that the vehicle takes to travel from one to the other. While time estimation is easily derived by the framerate of the video, distance is inferred by using vehicle dimensions as reference.

Figure 5. Length of the vehicle obtained by intersecting the edges with a segment

The idea is to determine how long is a portion of road by comparing it to the length of the vehicles that travels in it. Since we know how long a vehicle is (at least, an average), it will then be possible to estimate the length of the portion of road by making a proportion that takes into account the length of the vehicle in real life (meters), the length of the vehicle in the image (pixels), the length of the road in the image (pixels) to obtain the length of the road in real life (meters).

Car length detection

1. The first step consists in being able to identify, in the image, a segment that describes the length of a vehicle. To do this, we apply *Canny edge detector* to each vehicle image detected by YOLO, and intersect the edges with a straight line. The result is shown in image 5

2. The intersection segment, which defines the area where the edges meet, is derived by intersecting the vanishing point of the overall image with the slightly lowered center of the car picture. To find the vanishing point we used the RANSAC algorithm applied to the road lines detected in section 4.1. The RANSAC algorithm is a method to estimate parameters of a model from a set of data that contains outliers. It randomly samples subsets of data and evaluates the model fit using a voting scheme[8]. One application of RANSAC is to detect the vanishing point in one-point perspective images by finding the intersection of straight lines. The idea is explained in 6

Distance: proportions

$$gap\_image : car\_image = gap\_real : car\_real$$

To estimate the distance between the two red lines, which we will use later to calculate the distance, we use the proportion between the length of the cars and the length to be measured. In particular, as we can see in the figure 7, we find the proportion between the green line, which is length of the car, and the yellow line, which is the distance we want to measure. We can assume that the estimates will be sufficiently accurate for two reasons:

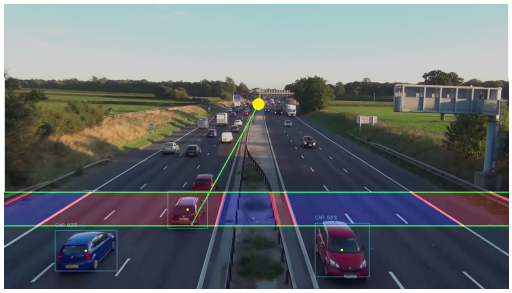

Figure 6. After applying the RANSAC algorithm, the vanishing point is identified and visually highlighted in yellow. A green line is then drawn, originating from the vanishing point and extending towards the center of the vehicle image. The intersections of this green line with the edges of the car determine the length of the vehicle.

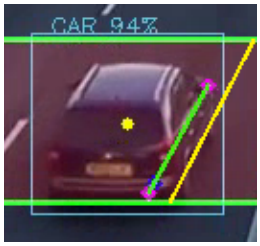

Figure 7. Proportions

1. Since the car is going in the same direction as the road, the two lines will be parallel, so the distortion given by the perspective, which given the proximity of the camera is small anyway, will be the same for both distances, assuming that the measurements will be taken evenly over the entire distance to be measured;

2. we can use the average length of a car as the length of the green line, provided that the measurement is taken using the average of a sufficient number of measurements.

Vehicle speed

$$speed = (gap\_real/(nFrame/fps)) \cdot 3.6$$

The formula provided calculates the speed of a vehicle based on the real distance between two consecutive lines on the road, the number of frames it takes for the vehicle to pass through that distance, and the frame rate of the captured or processed frames. By dividing the number of frames by the frame rate, the time taken to traverse the distance in seconds is determined. Multiplying this time by 3.6 converts the speed from meters per second to kilometers per hour. Thus, the formula enables an estimation of the vehicle's speed based on the measured distance and time.

Our detection mechanism consists of two distinct phases: convergence and detection.

During the convergence phase, two main processes are executed:

- In road segmentation, lanes are detected to identify the road structure. Among these lanes, only the subset containing vehicles passing through them is considered "active." Then, the two horizontal lines defining the region for vehicle detection are calibrated. The upper line is iteratively adjusted to extend beyond a specified threshold distance, ensuring a slightly longer area for more accurate estimations.

- To estimate the speed of vehicles, the fixed positions of the two lines determined during the convergence phase are utilized. The distance between these lines is estimated, and using this distance, the speed of the vehicles passing through is calculated. Note that this initial speed estimation is not the final value and will be further refined in subsequent iterations to determine the accurate distance.

Once the convergence phase concludes, the detection phase begins, utilizing the precise distance obtained from the convergence phase. This value remains unchanged throughout the detection phase.

5. Results

We tested our framework on four videos. These videos each have different characteristics, so we can observe the behaviour of the algorithm in a wide range of situations. Each video was cut to 20 seconds for a total of 600 frames each.

5.1. Vehicle detection and counting

Video 1 and 3 present only two roadways in different traffic situations, both during the daytime, and as we could expect on them the framework obtains excellent results, as we can see respectively in tables 1 and 4. These results are also confirmed in Video 4, the particularity of which is that it is a night-time video. We can also add that the few errors on this video are caused by the proximity of two additional roads on either side, which disturb the detection process. We can see the results in table 3. The most challenging video for our framework is Video 2, which has as many as four carriageways, as well as a more distant shot, which in particular makes it difficult to distinguish between large cars and trucks. Despite this, the results, shown in table 2, remain good.

Finally, in table 5 We can see the aggregated results. In particular, We can see that on the car class the framework is really precise. We can also identify two most common types of errors:

	Lane 1	Lane 2
Cars	20 (20)	7 (7)
Truck	1 (1)	10 (7)

Table 1. Video 1 results

	Lane 1	Lane 2	Lane 3	Lane 4
Cars	30 (33)	2 (5)	6 (8)	43 (39)
Truck	3 (0)	3 (0)	2 (0)	4 (2)
Motorcycle	0 (0)	1 (1)	0 (0)	0 (0)

Table 2. Video 2 results

	Lane 1	Lane 2
Cars	19 (19)	17 (17)
Truck	4 (6)	4 (4)
Bus	2 (0)	0 (0)

Table 3. Video 3 results

	Lane 1	Lane 2
Cars	17 (15)	22 (20)
Truck	1 (1)	0 (0)
Motorcycle	0 (0)	1 (1)

Table 4. Video 4 results

- big cars are sometimes mistaken as small trucks,
- some trucks are mistaken as buses.

In general the results on detection are very good, since the framework didn't miss any object, while it can be improved at the classification stage.

	Detection	Ground Truth
Cars	183	183
Truck	32	21
Bus	2	0
Motorcycle	2	2

Table 5. Aggregated results

5.2. Speed estimation

To verify the accuracy with which the framework estimates speed, we directly checked the distances calculated through satellite measurements (Google Maps). Since the speed is estimated from these distances, the error should have the same magnitude. We can see our results in the table 6

	Model estimation	Measured distance	% error
Video 1	32.63 m	30.50 m	6.98%
Video 2	29.95 m	28 m	6.96%
Video 3	24.89 m	24.50 m	1.59%
Video 4	23.55 m	20.50 m	15.88%

Table 6. Distance estimation results

5.3. Video ouput

Our work produces a source video with an informative overlay that includes vehicle detection and speed estimation. To indicate the detection of each vehicle, a bounding box surrounds it. Adjacent to the bounding box, a number displays the vehicle's current speed in kilometers per hour (km/h) after estimation. If the speed is below our predefined threshold of 130 km/h, the text appears in green, indicating compliance with the speed limit; otherwise, it is displayed in red.

Additionally, the overlay features detected lanes on the road. Each lane is delimited by red lines, and the area within each lane is filled with different colors. To note that only the "active" areas are filled.

Another notable feature is an overlay grid positioned in the top left of the image. This grid consists of rows representing each detected road lane. The columns within the grid display the vehicle types (Car, Moto, Bus, Truck) and accumulate the count (Tot) of vehicles transiting in each specific lane. An additional column (HighSpeed) tallies the number of vehicles in each lane that are traveling above the speed limit.

6. Conclusions

In this paper, we proposed a method to perform vehicle detection and counting that is based on the YOLO architecture while incorporating a Tracker mechanism to track the vehicles' positions as they traverse the road. To detect the road lines, the approach utilizes the Canny edge detector and Hough Line Transform, followed by employing K-means clustering for road line identification. Subsequently, the YOLO architecture is employed to detect vehicles within image or video frames, and the Tracker is utilized to monitor their positions. The counting of vehicles is accomplished by considering their location within predetermined regions of interest on the road. Evaluation of the method using various videos has shown its effectiveness in accurately detecting and counting vehicles.

For speed estimation, the method relies on calculating the distance between two points along a vehicle's trajectory and determining the time taken by the vehicle to travel between those points. The time is derived from the video frame rate, while the distance is computed based on the vehicle's position within the image. Notably, this speed estimation algorithm does not depend on external measurements or stereo vision, yet it delivers a reasonably accurate estimation of a vehicle's speed.

You can find the code for vehicle detection, counting, and speed estimation on GitHub at the following repository: `https://github.com/DavideVarotto/vehicle-detection`.

References

[1] John Canny. A computational approach to edge detection. https://ieeexplore.ieee.org/document/4767851/authors#authors 1986.

[2] Daphna Weinshall Chaim Ginzburg, Amit Raphael. A cheap system for vehicle speed detection. https://arxiv.org/abs/1501.06751 2015.

[3] Rohith Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e 2018.

[4] Haoxiang Liang Huansheng Song. Vision-based vehicle detection and counting system using deep learning in highway scenes. https://etrr.springeropen.com/articles/10.1186/s12544-019-0390-4 2019.

[5] Eun-Ju Lee Chi-Hak Lee Young-mo Kim Inwon Lee, Jong-pil Ahn. The vehicle speed detection based on three-dimensional information using two cameras. https://link.springer.com/chapter/10.1007/978-3-642-41671-2_30 2014.

[6] Se-ho Park Jeong-ah Kim, Ju-Yeong Sung. Comparison of faster-rcnn, yolo, and ssd for real-time vehicle type recognition. https://ieeexplore.ieee.org/document/9277040 2020.

[7] Zoumana Keita. Yolo object detection explained. https://www.datacamp.com/blog/yolo-object-detection-explained 2022.

[8] Jean-Charles Bazin Marc Pollefeys. 3-line ransac for orthogonal vanishing point detection. https://www.researchgate.net/publication/261495656_3-line_RANSAC_for_orthogonal_vanishing_point_detection 2012.

[9] Microsoft. Coco dataset. https://cocodataset.org/#home 2014.

[10] Peter E. Hart Richard O. Duda. Use of the hough transformation to detect lines and curves in pictures. https://www.cse.unr.edu/~bebis/CS474/Handouts/HoughTransformPaper.pdf 1971.

[11] Peter J. ROUSSEEUW. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. https://wis.kuleuven.be/stat/robust/papers/publications-1987/rousseeuw-silhouettes-jcam-sciencedirectopenarchiv.pdf 1986.