HoG and Yolo for vehicles counting

Authors	Contacts
Forin Paolo	paolo.forin@studenti.unipd.it
Guizzaro Chiara	chiara.guizzaro@studenti.unipd.it
Visonà Matteo	matteo.visona.1@studenti.unipd.it

Abstract

Vehicle detection, tracking and counting applications play an important role for the society such as in surveillance of traffic and urban traffic planning. This paper compares the classical approach Yolo to another approach HOG. We use these methods to perform vehicles detection, instead for tracking we employ a centroid tracking and for counting we implement a function that counts how many vehicles through a line. Moreover, for Yolo we do not do a train, but we used the network pre-trained on CoCo dataset, while for HOG we train a SVM. We compare accuracy and speed. To have a complete overview, we tried many parameters for HOG features and we used multiple versions of Yolo. The results are quite interesting.

1. Introduction

In the last two decades the fast progress in computer vision, camera technologies and the advent of neural networks and big data have raised the interest in video-based traffic surveillance applications. The application of computer vision techniques in the field of traffic surveillance plays an important role to build intelligent transportation systems (ITS). These kinds of systems make use of vehicle detection, classification and tracking to help with several tasks such as traffic measurement and planning, traffic crime prevention, incident detection and behaviour analysis and understanding. In this paper we concentrate into the traffic measurement task. Traffic statistics are of considerable significance to intelligent traffic management and control of the highway. We propose a traditional approach capable of detecting and counting vehicles near real time on videos taken from a camera located in a highway. Moreover we show, also, a comparison with a modern deep-learning object detection framework.
The remaining of this paper is organized as follows. In the next section, we provide a review of the state of the art research on vehicle detection, vehicle tracking and counting. In section III, we discuss the datasets used for the realization of the project. In section IV, we discuss in details the pipeline we proposed. A detailed discussion of the experiments is then presented in section V. Finally, section VI summarizes and concludes this paper.

2. Related works

2.1. Vehicle detection

Object detection in images is a well studied problem.
Cars are widespread object of interest due to their common presence in everyday scenes, simple structure and low variance between different type of cars. The first approaches are based on statistical methods like Support Vector Machine (SVM) [12], [16] and Neural Network [11], [6].
A step froward to real time object detection was brought by Viola and Jones cascade classifier [19] who proposed a method for training a sequential classifier with Haar-like features and showed its real-time application on face detection problem. Afterwards, this method was also tested with other objects, including cars [15]. To get better real-time performances other algorithms exists, but their assumptions about the real-world scenes are simplistic and do not hold in general, for example when there are shadows or vehicles symmetry [18], [5], [17], [10].
In this phase we use Yolo that is a Convolutional Neural Network (CNN) and we compare it with Histogram of Oriented Gradients (HOG) + Support Vector Machine (SVM), it is explained in section 4.1.

2.2. Vehicle tracking

Object tracking is the task of automatically identifying objects in a video and interpreting them as a set of trajectories with high accuracy. Many tracking algorithms exist and each of them has one or another limitation. Two popular and simple algorithms are Kernelized Correlation Filter (KCF) and Multiple Instance Boosting (MILBoost). The first one is based on the idea of correlation filter and kernel trick, the major advantage of KCF is the computational efficiency (it works in the Fourier domain), however it is not so robust against occlusions [9]. Moreover, there are linear and non-

linear kernel versions of the tracker.
The latter uses the idea of MIL (Multiple Instance Learning) to train a cascade of classifiers. MIL is characterized by bags, that can be positive (at least one element inside is positive) or negative (all elements inside are negative) and all samples in the bag share the same label [20].
In our application we decided to implement a very basic and simple object tracking called centroid tracking, it is simpler than the previous ones (see section 4.2).

2.3. Vehicle counting

Vehicle counting is an important aspect of the project and there are many ways to perform that. A simple method is to count a vehicle when the related detected box passes through a line [6] or when the related virtual vision of it passes through a delimited area [10].
In our project the counting is divided into two different phases: one used to decide which element we will count, and the other to count the vehicles that go from top to down or viceversa (section 4.2). With this approach our project is more general because we do not use the information related to the camera position.

3. Dataset

For vehicles detection and counting purposes, several sources of images taken from different points of view are available online. Usually such dataset have photos taken from the back, while very few are available from the front. Therefore we have looked also for these last ones and merged all images together.

The final dataset used for training our methods is composed by all images of Kaggle dataset [3] and CompCars [1] using only frontal photos. This gives us 8792 images of vehicles from the back or slightly side-back angles, 439 from the front and 8968 of non-vehicles, for a total amount of 18 199 samples of size (40x40).

Other datasets, taken into account but discarded at the end, are GTI vehicle image database [2] and a combination of this last one with KITTI vision benchmark suite, both considered with and without the addition of front images. GTI dataset has vehicles taken from a left, right, middle-close and far points of view, but the images as a whole are really similar with just slightly differences in translation or angle of view, causing overfitting while training. Similar reasons are for the combined dataset. Instead regarding other possible choices available online, they have not been taken under consideration because too general, having low quality or images with different perspectives from the ones of the task of interest.

Moreover, after an initial training process we have tried to use the 'hard negative mining' technique, unfortunately without getting improvements on the final results quality. The process consisted in taking several videos [4] and then for each one of them, every 60 frames only one is picked. In total we get 1517 additional images from which we can extract false positive samples (regions that are classified as vehicles but they are not) to add at the initial training set. Again this technique has not be helpful, because it caused overfitting.

A final consideration is regarding the selective searching process that we plan to use. Often the initial dataset is taken and then a new dataset is generated after the application of selective search. This may be effective on having images, translated or with different perspectives, to train a classifier used to the selective search behavior. Nevertheless we have decided to use the same training dataset both for HOG with sliding window and selective search in order to perform a fair comparison between the two.

Note that everything said before is about HOG's training, while Yolo versions are used with weights trained on MS COCO dataset.

Lastly, a set of 17 images taken from Google and 2 videos from YouTube (whose duration is 15 and 53 seconds) are used for testing.

4. Method

In tasks such as traffic surveillance it is important to have a good compromise between detection speed and accuracy, however a slight delay is allowed because the goal is to calculate average traffic statistics rather than instantaneous ones and it is realistic to think that the traffic situation cannot change suddenly.
Moreover, the detection system should be able to work even in low-cost devices, with limited computational resources, such as an Arduino. For all these reasons we have decided to tackle this task using a pipeline based on HOG-SVM. Among the variety of object detection frameworks HOG-SVM is a popular one. It has been originally proposed by Dalal and Triggs [7] and their approach consisted in using HOG descriptors in combination with a sliding window approach and a linear SVM to perform human detection.
HOG is a powerful descriptor, it is excellent at representing local appearance and in general it is very accurate for object classification, and even if it is not the faster algorithm out there (in fact an Haar cascade is usually faster) it offers a good balance between speed and accuracy. It is in general more accurate than a simple Haar cascade and less computationally demanding with respect to modern deep learning end to end approaches. Since that time, and with the advent of neural networks and big data, more advanced regions proposals algorithms have been proposed, such as Selective search. It has been proposed by Uijlings et al [8] and it employs image segmentation techniques in order to return only a selection of proposals that could contain an object among all the possible regions in the image.

We wanted to try out if the combination Selective

search/HOG/Linear SVM could effectively outperform the approach with sliding window. There is a poor documentation and usaget for this strategy online, we were not sure whether this is because they can not work together or because all the projects and documentation we can find online refers to the original Sliding window/HOG/SVM paper, therefore we have tested both approaches in our project to see which one performs better.

Moreover, we have compared these approaches with Yolo, a more advanced deep-learning object detection algorithm, that is the state of the art in this field.

Beyond that and considering that our final goal was processing a video stream, we had also considered Background Subtraction as a pre-processing step in our pipeline. More in particular we wanted to build a binary mask, where blobs in this mask corresponded to potential vehicles, and then use it to discard in advance some useless sliding window proposals. (e.g. if the proposal had intersection lower than a certain threshold w.r.t. the binary mask, it would have been rejected). We soon discarded that idea for the following reasons:

- Vehicles may be stationary in heavy traffic conditions. If we had used background subtraction here, we would not have been able to detect vehicles because there would have been very little changes among consecutive frames. Moreover, it would not have worked very well in the case of adverse weather conditions.
- The OpenCV implementation of the function *detectMultiScale(..)* (used for the sliding window and for the detection) is very limited, it works like a black box and it does not allow to specify conditional constraints regarding the proposals. We have also tried to implement a sliding window algorithm by ourselves, but it was performing worse w.r.t. the OpenCV implementation, maybe because the latter is optimized and the code is parallelized. For the same reason we could not try other preprocessing techniques (Saliency Maps, Edge segmentation ...) to reduce the number of proposals at the beginning.

The two pipelines we have built will be described in the following sections. It is important to note that the pipelines we are going to described works in a single image, indeed they are applied to every single frame of a video stream. Two schematic illustrations can be found in Figures 1 and 2.

4.1. Vehicle detection

In this sub-section we are going to present the core of our pipeline, the vehicle detection.

4.1.1 HOG + SVM

Sliding window For each image we resize down the image maintaining its aspect ratio, to reduce the area to be examined from sliding window, and hence the number of proposals. The sliding window algorithm and the subsequent detection are both implemented by OpenCV in the function *detectMultiScale(...)* inside *HogDescriptor* class. This function internally applies a pre-processing (gamma correction for hog descriptors and filtering with Gaussian filter), implements a multiscale sliding window and performs the classification for each proposal. Finally, the function returns a list that contains the coordinates of the bounding boxes that has been classified as vehicles and a list containing the classification confidences. We consider only the detections that have distance from the SVM hyperplane greater than 0.15. *NonMaximaSuppression* algorithm is then applied to suppress overlapping bounding boxes. In the end we calculate the centroid position for each bounding box to update some information useful for vehicle tracking and for the subsequent counting. Vehicle tracking and counting steps will be described in detail later in this paper.

Selective search For each image we resize down the image maintaining its aspect ratio, to speed up selective search algorithm. We apply a median filter, which we have found out speeds up the process without losing much accuracy, indeed in some cases it helps removing some false positives. Then we apply Selective Search algorithm in 'fast' mode since we noticed that it gives good enough proposals and it is way faster than 'quality' mode. Each region of interest is then resized to a prefixed size of 40x40. By doing this, we loose the aspect ratio of the original regions, but this is a collateral effect that we must accept. We convert each region to grey-scale color space, and compute its HOG descriptor that will later be passed to the SVM classifier for classification. Only the proposals that have distance from the hyper-plane greater than 0.15 are classified as vehicles. *NonMaximaSuppression* algorithm is then applied to the positive proposals to suppress overlapping bounding boxes. As did before, we calculate centroids positions to update vehicle tracking information and for the subsequent counting.

4.1.2 Yolo

We have implemented both YoloV2 and YoloV3 (YoloV4 is still not supported by OpenCV) pre-trained on the COCO dataset. This pipeline is straightforward. First, as we did in the previous methods, we downscale the original image to (416, 416) in order to fasten the subsequent detection. We have chosen the size (416, 416) because it is a good trade-off beetween speed and accuracy. Then we run Yolo detection on each image. It returns a list of bounding boxes,

a list of classification confidences and a list containing the class IDs to which the detections correspond to. Yolo is a multi-class object detection algorithm and it can detect a great variety of objects. Since we are only interested in detecting vehicle, we consider only the bounding boxes where the respective ID is in the range [2,9], the range of vehicle classes. Then we run *NonMaximaSuppression* to suppress overlapping boxes, and in the end we calculate centroids positions to update vehicle tracking information and for the subsequent counting.

4.2. Vehicle tracking and counting

We need to implement tracking to keep track of the vehicles, because in this way we do not count the same vehicle two or more times.
For tracking the vehicles in video we implement a centroid tracker, that is a simple tracker. A centroid is a point that is placed in the center of the bounding box detected with the previous methods. So it is calculated with a simple mathematical formula, because we have the top left coordinates, the width and the height of the box. Every centroid has an id, hence every box have a centroid and an id. Important aspects to understand are when update the centroid position, when there is a new centroid, and when delete an old one. For updating and insertion we calculate the centroids in the current frame and we see the Euclidean distance with all the centroids in the previous frame, then we update the centroid position with the nearest current frame centroid if the distance is less than a threshold. If in the current frame there are some centroids that are not connected with an older one, then it is a new centroid with a new id. Moreover, a centroid with its id is deleted when it is not updated for some consecutive frame.
For counting we use two line and the area between them, the position of these two line depends on the method used for the detecting and on the frame prospective. Every centroid that stays in the area delimited by the two line are saved in a dictionary, then for every successive frame we update the position of the centroid also in this dictionary if the position stays in the area of interest. The counting occurs when a centroid leaves the delimited area. Furthermore, this phase is divided in two counting, one for the Vehicles from top to bottom and another for bottom to top.
A problem that can happen with tracker is when two cars are close and in a frame one of them is detected and in the following frame the other car is detected so there is one centroid and one id for the two cars.
Another problem that can happen is that for one vehicle there can be more ids, because it is detected in one frame and it is detected again after some frames, hence it does not respect one of the two constraints, because the distance between the two centroids is greater than a threshold or it is detected again after too many frames.

5. Experiments

The proposed method with sliding window and selective search variants has been compared with YoloV2 [13] and YoloV3 [14]. Everything has been implemented using Python programming language and OpenCV 4.5.3 version library. First of all, we describe some initial assumptions that has been made in this work:

- bad weather conditions and night surveillance are not considered;
- objects might be overlapped and more bounding boxes can appear near each other;

moreover for videos:

- no sudden changes of direction and car accidents are expected;
- it may happen that a certain vehicle does not appear on all frames (changing ids);
- video clips are captured from a point of view above the roadway surface.

In order to get the best results possible, a lot of hyperparameters and pre/post processing phases have been tested.

Firstly all RGB images are loaded and converted in grayscale photos, then, when computing HOG descriptors a gamma correction and a Gaussian filtering are applied respectively in order to enhance objects from the background and remove eventual noise. Further experiments with other color spaces selecting only one specific channel such as saturation for the HSV color space has not improved the detection process, thus they have been discarded.

Moreover we have seen that the best parameters for our task are the ones in table 1. In specific winStride, padding, scale and threshold T_S impact the most and thus they had been looked carefully during the tuning process. The value of T_S is used to remove all bounding boxes during Non-maximum Suppression (NMS) that do not reach a certain minimum score.

Also the choice of the dataset to use to train the model is of the most importance. Indeed the final result quality is directly influenced by how similar the training images are with respect to the task of interest. If we look at the results obtained on a set of 17 test images of different roads (some examples are in Figure 3), HOG with sliding window outbests HOG with selective search. The number of vehicles counted is clearly superior even if there may be some false positives (regions that are classified as vehicles when actually they are not). Selective search does not work so well with HOG features due to the deformations caused by the resizing process of the regions of interest, which can be very large areas, in small windows. Thus the aspect ratio is lost. We remember that also HOG is not invariant to deformations. Nevertheless the resizing process has been taken

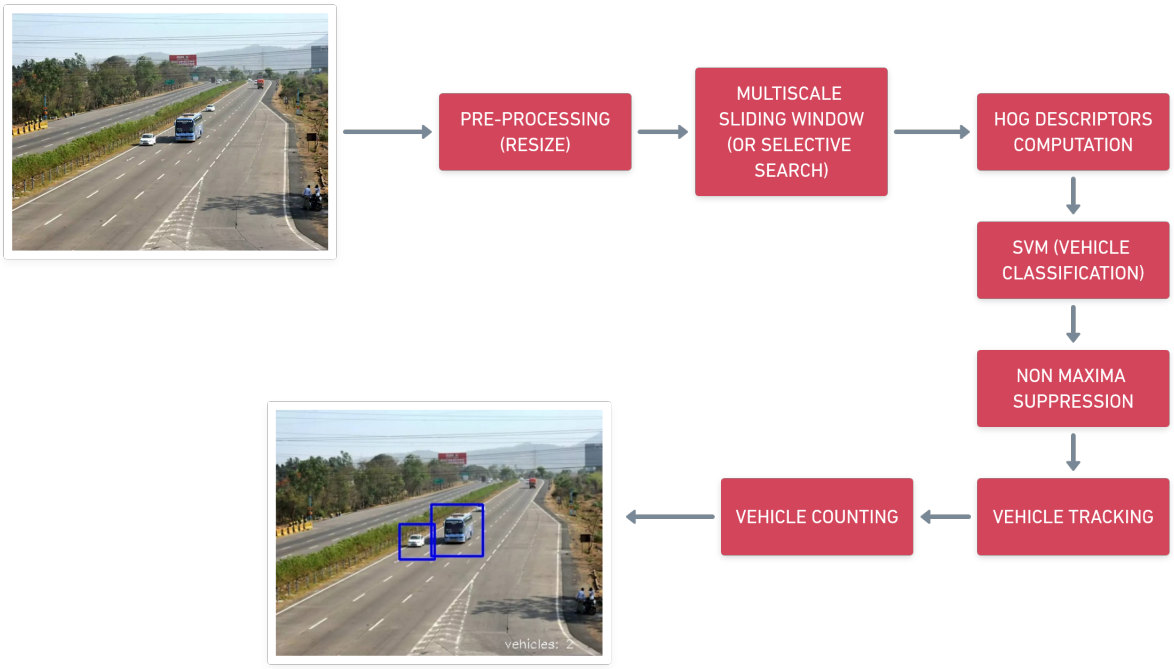

Figure 1. HOG/SVM pipeline for a single video frame (sliding window and selective search)

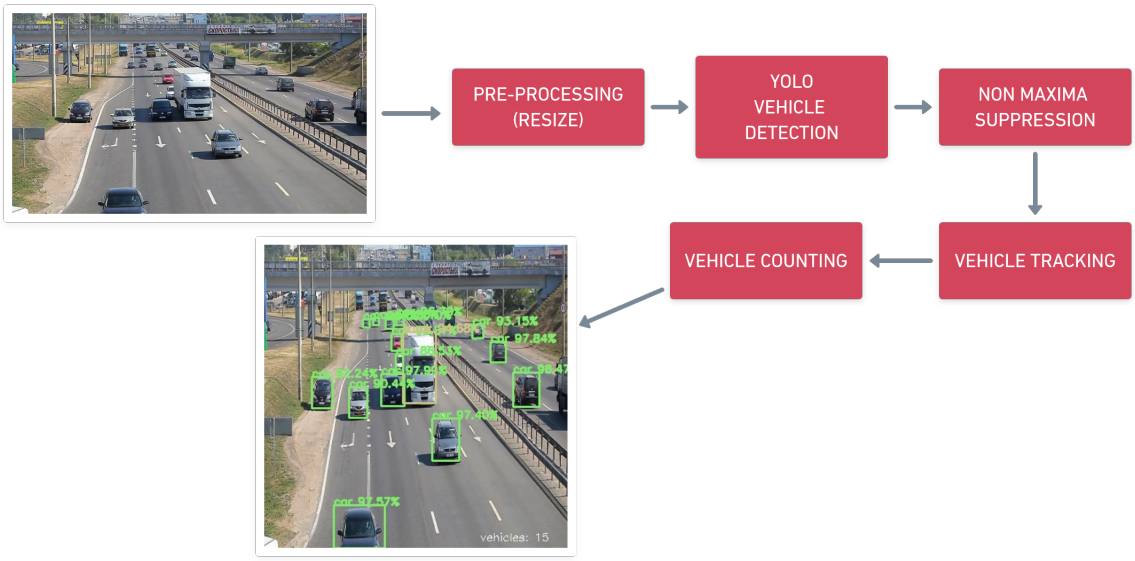

Figure 2. Yolo pipeline for a single video frame

into account to speed up the detection process especially for the sliding window approach where it does not damage too much the general accuracy.
We want to mention that also the usage of bilateral and median filters has been tried in order to remove some possible noise. Both of them has not been helpful for sliding window, while the median filter has turned out to be useful for having a quicker selective search process.
Now if we compare HOG with Yolo we can see that in general HOG has bigger bounding boxes given some vehicles, while Yolo has tighter ones. Furthermore HOG with selective search and YoloV2 do not detect many vehicles and in some cases nothing at all. On the contrary HOG with sliding window detects correctly much more vehicles, even if

HOG parameters	
winSize	(40,40)
blockSize	(16,16)
blockStride	(8,8)
cellSize	(8,8)
nbins	9
derivAperture	1
winSigma	24.
histogramNormType	0
L2HysThreshold	2.0000000000000001e-01
gammaCorrection	1
nlevels	64
winStride	(2, 4)
padding	(8, 8)
scale	1.25
threshold T_S	0.05

Table 1. HOG parameters used on OpenCV.

the best results are for YoloV3 where the missing cars are very few. We want to mention that while HOG is trained only to detect vehicles and in specific cars and trucks without classifying them, the same can not be said for YoloV2 and YoloV3 which are trained on MS COCO dataset and can detect and classify 8 kind of vehicles: 'bycicle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck' and 'boat'.

Looking at table 2 we have a conformation of what said before. HOG with sliding window (SW) has a higher Mean Average Precision (mAP) on vehicles class than HOG with selective search (SS) and YoloV2, while clearly the best is YoloV3. Also on one hand, changing the value of Interception Over Union (IOU) for which the metric mAP is calculated does not change much the results for both versions of Yolo, for HOG decreasing the value of IOU increases the mAP obtained. This is because as already said HOG tends to have larger boxes given some vehicles, thus the interception with the ground truth boxes is smaller.

Following what already said before, for our vehicles counting implementation on videos between the HOG version available only the one with sliding window approach will be taken into account. Thus about the videos performances we can make another consideration regarding the trade-off between accuracy and speed. Looking at the values of the mAP metric and the respective frames per second (fps) achieved, even if YoloV3 has the best mAP value, it is also the slower analysed method. Only YoloV2 and HOG with sliding window have fps greater than 11.
For videos we tried a lot of parameters for the max distance for tracking and max disappear for deleting centroids and we saw that a good trade-off depends on the frame prospective. Also, the position of the lines for counting depends on the same thing, in our examples they are in the middle or at three-quarters of the frame. Also the space between the two lines depends on the prospective frame.
We use as test two different videos, the first of 15 sec with a lot of cars and overlapping, the second of 53 sec and with few overlapping. These are used to compare especially HOG with sliding window and YoloV3, which are respectively the best versions of HOG and Yolo analysed.
The parameters used by HOG are:

	First video	Second video
maxDisappear	40	40
maxDistance	40	40
lines position	three-quarters	center
space between lines	20	20

The parameters used by YoloV3 are:

	First video	Second video
maxDisappear	40	40
maxDistance	75	75
lines position	three-quarters	center
space between lines	30	30

Video results are:

	down to top		top to down	
HOG	1^{st} video	2^{nd} video	1^{st} video	2^{nd} video
	20	20	18	9
Yolo	1^{st} video	2^{nd} video	1^{st} video	2^{nd} video
	22	22	21	6

6. Conclusion

In images as we can see in figure 3 YoloV3 detects more cars than any other methods and HOG is the second one. However HOG is faster than YoloV3. Fruthermore, if we need speed, we accepts very few false positive and we do not have a GPU, HOG is a valid option. Instead, if we have a GPU and we need very high precision and accuracy the best option is YoloV3.
In the video of 15 seconds the correct number of vehicles from bottom to top is 22 and from top to bottom is 25, while in the video of 53 seconds the correct numbers are 25 and 8 respectively.
As we can see in the experiments analysis, both HOG and YoloV3 are good for counting. YoloV3 results for the first video are 22/22 and 21/25, for the second are 22/25 and 6/8. On the contrary HOG has for the first video 20/22 and 18/25, for the second 20/25 and 9/8. This results show that YoloV3 is more accurate, while HOG in the second video counted one vehicle twice. Instead in speed valuation HOG is better than YoloV3. Another important aspect HOG velocity shows is that it can work well on a CPU while for YoloV3 to have a good speed it needs a powerful GPU.
In general, some possible suggestions for further experiments and for having quicker computations are: the usage of C++ as programming language, which is notoriously fast, more GPUs for Yolo and for SVM the application of some pre-processing based on edges to eliminate some proposals.

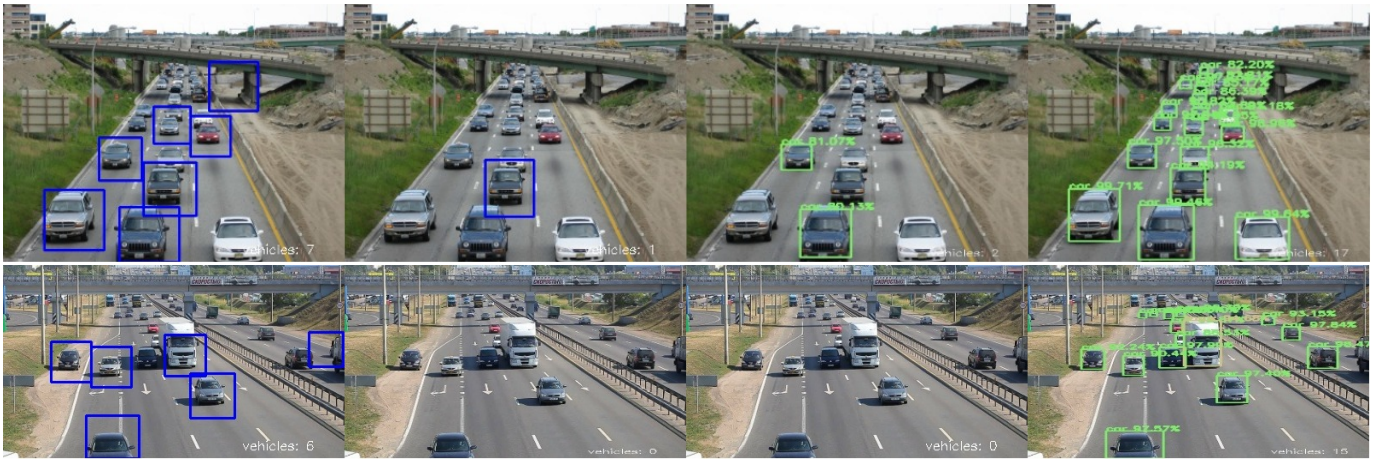

Figure 3. Examples of vehicles detection and counting. The first column is for HOG with sliding window, the second column is for HOG with selective search, the third column for YoloV2 and the last one is for YoloV3.

	mAP 0.5	mAP 0.4	mAP 0.3	mAP 0.2	mAP 0.1
HOG SW	**7.08%**	**15.52%**	**27.15%**	**31.76%**	**31.76%**
HOG SS	0.83%	2.86%	2.86%	3.37%	3.37%
Yolo v2	4.14%	4.14%	4.14%	4.14%	4.14%
Yolo v3	71.40%	72.04%	72.04%	72.04%	72.04%

Table 2. Mean Average Precision (mAP) metric with different values of Interception Over Union (IOU) considered. In detail the IOUs are 0.5, 0.4, 0.3, 0.2 and 0.1. Note mAP is calculated only for the vehicles class.

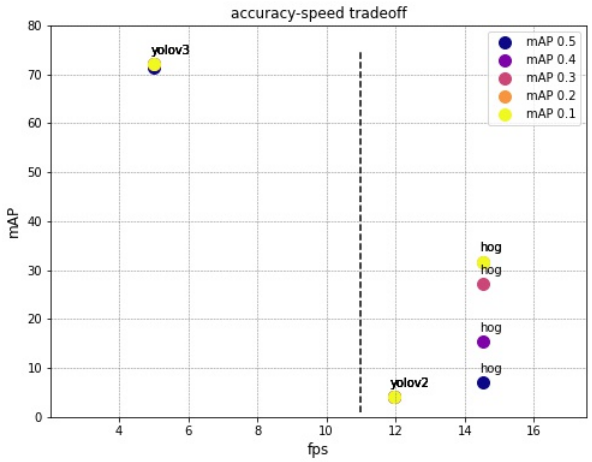

Figure 4. Accuracy and speed trade-off on HOG sliding window, YoloV2 and YoloV3 models.

References

[1] *CompCars*, 2021. https://drive.google.com/drive/folders/18EunmjOJsbE5Lh9zA0cZ4wKV6Um46dkg.

[2] *GTI dataset*, 2021. https://www.gti.ssr.upm.es/data/Vehicle_database.html.

[3] *Kaggle dataset*, 2021. https://www.kaggle.com/brsdincer/vehicle-detection-image-set.

[4] *Videos*, 2021. https://www.youtube.com/watch?v=UM0hX7nomi8, https://www.youtube.com/watch?v=d_yZz10mI8k, https://www.youtube.com/watch?v=GCj8x7ou-U8, https://www.youtube.com/watch?v=lbYKYpqVEmw, https://www.youtube.com/watch?v=Ina7KMV2OEI.

[5] Alberto Broggi, Pietro Cerri, and Pier Antonello. Multi-resolution vehicle detection using artificial vision. 07 2004.

[6] Zhe Dai, Huansheng Song, Xuan Wang, Yong Fang, Xu Yun, Zhaoyang Zhang, and Huaiyu Li. Video-based vehicle counting framework. *IEEE Access*, 2019.

[7] Dalal and Triggs. Histogram of oriented gradients for human detection. 2005.

[8] Uijlings et al. Selective search for object recognition. 2012.

[9] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *CoRR*, 2014.

[10] Manchun Lei, Damien Lefloch, Pierre Gouton, and Kadder Madani. A video-based real-time vehicle counting system using adaptive background method. 2008.

[11] D. CHARNLEY N.D. MATTHEWS, P.E. AN and C.J. HARRIS. Vehicle detection and recognition in greyscale imagery. 1995.

[12] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 06 2000.

[13] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[14] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[15] Sayanan Sivaraman and Mohan Manubhai Trivedi. A general active-learning framework for on-road vehicle recognition and tracking. *IEEE Transactions on Intelligent Transportation Systems*, 2010.

[16] Zehang Sun, G. Bebis, and R. Miller. On-road vehicle detection using evolutionary gabor filter optimization. *IEEE Transactions on Intelligent Transportation Systems*, 2005.

[17] Zehang Sun, G. Bebis, and R. Miller. On-road vehicle detection: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.

[18] Christos Tzomakas and Werner von Seelen. Vehicle detection in traffic scenes using shadows. 1998.

[19] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. 2001.

[20] Paul Viola, John Platt, and Cha Zhang. Multiple instance boosting for object detection. *Advances in neural information processing systems*, 01 2005.