Self-driving car Nanodegree (Term 1). Project 5: Vehicle detection and tracking

Ruggero Vasile

June 26, 2017

Abstract

Report for the fifth project in the self-driving car nanodegree program, term 1. In this project the aim is to detect vehicles in images and to track them in a video frame as the car move.

1 Introduction

This project can be essentially divided into two parts. In the first part a standard machine learning pipeline is built, consisting of feature engineering on a set of inputs, training and validation phases to build a robust classifier and testing on new images. In the second part we deal with larger road images and we need to identify the positions of cars on the road. To do so, different parts of the image will be scanned and resized to appropriate size to be fed into the classification model built, which will tell us how many vehicles appear in the image and their position.

The submission comprise this report, a jupyter notebook with relevant steps, three scripting files (*utils.py*, *learning.py* and *preprocessing.py*) that contain the functions and classes used for this project, a set of 6 processed test images with the requirements fullfilled, and a video clip showing the application of the pipeline to a video stream.

2 Data

To build the classifier we have in hand a dataset of car images and non car images which can be used as raw data. In Fig. 1 one can see examples of both vehicle and non-vehicle images. All the images in the set are already scaled to 64×64 pixels and contain 3 channels in RGB color space. There are in total 8792 samples of vehicle images and 8968 non vehicle images, therefore the dataset looks to be balanced between the two classes.

Together with this standard dataset, we have been given access to a bigger set collected by Udacity. From this dataset it is possible to extract other images from vehicle, e.g. cars, and augment the previous dataset. I followed the choice of a partial dataset augmentation adding 500 samples not to unbalance too much the whole dataset. Examples of such images can be found in Fig. 2.

3 Feature extraction

In this project I build a classifier using as input features coming from standard computer vision techniques. The features considered are:

Figure 1. (Above) Two images containing a vehicle. (Below) Two images not containing a vehicle.

Figure 2. Two examples of images coming from the Udacity dataset. We can see that here there is more variability, with some samples showing cars pictures taken from a different angle.

- HOG features: I extract them choosing 9 gradient orientations, 2 cells per block and 8 pixels per cell for each of the three channels of the input image. Moreover I found that better results are obtained when the input RGB image is first transformed into the $YCrCb$ color space (similar results also for the YUV color space). In total from each image a feature vector of lenght 5292 is extracted. (See Figs. 3 and 4 for examples of such features).
- Color channel Histogram features on the input RGB images using 32 bins, i.e. a total of 96 features.
- Spatial bin features with size equal to 32, i.e. a total of 3072 features.

After extraction these features are aligned on a single feature vector of length 8460.

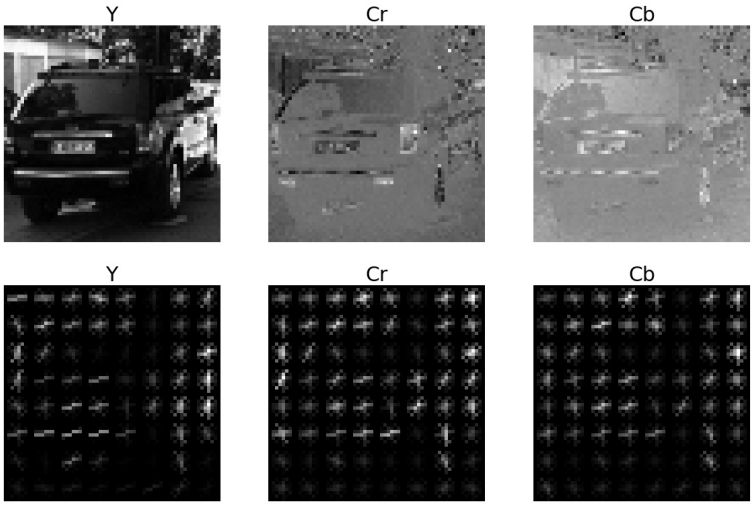

Figure 3. Examples of HOG features extracted from the three channels of color space YCrCb for a vehicle sample.

4 Training a classifier

The feature vector extracted will be used to train a classifier for car detection. Before the training phase, it is practice to normalize the feature to a common interval. I use here the standard Scaler from sklearn to normalize each feature in the interval $[0, 1]$.

I try different classifiers before settling with LinearSVM. Especially I was interested to see if ensemble methods like RandomForests or Adaptive Boosting where performing good enough. However in those cases I found that the validation error was always well below the training error. Hence I settled for Support Vector Machines, and in particular for Linear SVM. Even if the SVM with radial basis function kernel could achieve better results, the training and testing times are much larger, and accuracy improvement is actually very limited.

The full training phase consisted in the following steps:

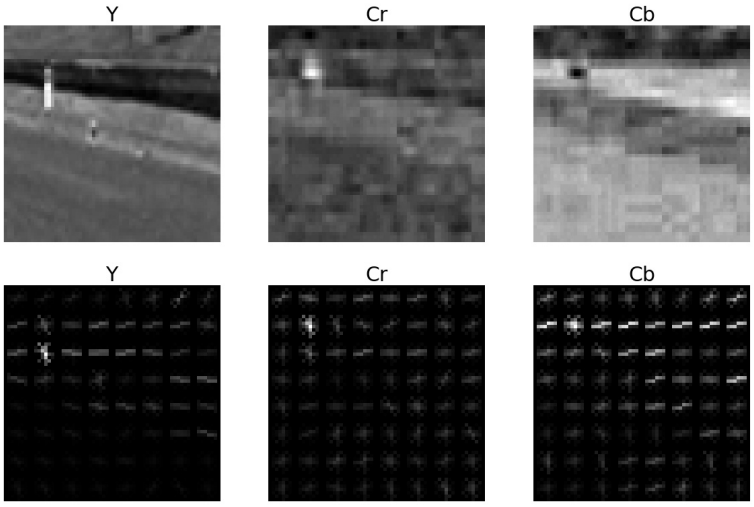

Figure 4. Examples of HOG features extracted from the three channels of color space YCrCb for a non-vehicle sample.

- First I perform a GridSearch with 5-fold crossvalidation to identify the best combination of hyperparameters. For a linear SVM the main parameter to tune is C and I settled for $C = 0.1$ which was giving the best cross-validation error.
- Once the hyperparameters are chosen, I perform a training using the whole training set as input.
- The trained model will be then used for prediction of other images.

Different accuracies are achieved when using the augmented dataset and the non augmented dataset. For the non augmented dataset the validation accuracy reaches 99%, while our best result with the augmented dataset is about 97%. Despite this difference I chose to train the model with the augmented dataset because it was performing better in recognising vehicles in real images and videos. The reason is that the augmented dataset contains also vehicles pictures from side view, as they may appear in real images. Instead the standard dataset mostly contains images of cars taken from behind.

5 Sliding Windows Extraction

We now deal with real images and videos that cannot be directly fed into the classifier for image classification. Each image will have to be first divided in different sub-windows that after appropriate preprocessing steps can be used to feed the classifier.

It is here adopted a sliding window approach. A window box is defined as a rectangular area with horizontal and vertical sizes. Its position is varied within certain limits on the image. Depending on its position, different parts of the image can be cropped and preprocessed for classification.

In order to find the vehicles on the image we need to scan its area by moving the sliding window around:

- Window boxes can have different sizes. Here I consider only 64×64 and 96×96 sizes. These values seem to be good enough to identify all the cars in the test images and video.
- There is no reason to scan the whole image, because cars can appear only on the road (lower image part). Therefore I put some limits to the exploration. The smaller window will scan the image in the portion $[350, 550] \times [0, 1280]$. The bigger window will scan the vertical portion $[400, 650] times [0, 1280]$. Due to perspective issues closer cars appear in fact bigger and to be detected need a bigger window size.
- The windows slide in the designated area with a certain overlap. For the smaller window I choose to scan the designated area horizontally and vertically by sliding the window with a step of 10 pixels. The value 20 pixels is chosen for the bigger window.

For each window position and size the image will be cropped, preprocessed and its class will be predicted. Here are the fundamental steps for correct classification using a cropped image through the sliding window:

- The window selects an area of the image that is cropped.
- The selected area is resized to a 64×64 pixel size, as required by the preprocessing pipeline.
- HOG features, spatial bin features and color channel features are extracted.
- The feature vector is normalized using the same scaler used for preprocessing the training set.
- The normalized feature vector is fed into the trained model that will classify it as a vehicle/non-vehicle sample.

The protocol described may be computationally demanding. In fact the higher the number of windows the larger the computational time. Clearly at this stage, due to limited computational power, choice of programming language and not optimal preprocessing and classification scheme, a full scan of an input image takes about 6 seconds. Alternative method will need to be provided for real time detection where the frame rate f would require the pipeline to be processed in less than $1/f$ seconds.

6 Sliding Windows Postprocessing

The result of the pipeline is to provide a set of overlapping cropped image parts that contain a vehicle according to the classification strategy defined. Once this set is extracted from each image or video frame, further postprocessing is required for dealing with: 1) False positive classification, 2) Window clustering to extract a unique window that encapsulates a vehicle.

The false positive elimination is here not addressed because no false positive has been detected neither in the images nor in the video. The window clustering can be achieved using two steps:

- A heatmap of observation is first evaluated. The method is the one suggested in the lesson of Udacity, i.e. each pixel belonging to a selected window will be given a value of 1. Summing up all the contribution for all pixels and windows will generate a heat map images (see third panels in the next figures).
- From the heatmap an estimated number of labels can be extracted using the *scipy.ndimage.measurements.label* function. For each of these labels an overall bounding box is generated and assigned. Each bounding box locates the presence of a vehicle in the image. (See fourth panels in the next figures).

In the next section I show the result of these steps on the test figures.

7 Results test images

In the next six figures (5, 6, 7, 8, 9 10) I show the results obtained with the previous pipeline on the test images. In all of them I successfully identify the vehicle present.

Figure 5. (Above left): Input image. (Above right): The sliding windows whose cropped image pieces resulted positive in the vehicle classification. The green rectangles correspond to windows of size 96×96, the red rectangles to windows of size 64×64. (Below left) The heat map corresponding to the windows found. (Below right). The final rectangles identifying the single cars vehicles in the images.

8 Results video

Please refer to the attached file.

Figure 6

Figure 7.

Figure 8.

Figure 9.

Figure 10.