Vehicle Detection

Project goals:
* Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
* Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
* Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
* Run pipeline on a video stream
* Create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
* Estimate a bounding box for vehicles detected.

Pipeline Overview

In order to meet the previously mentioned goals, the following steps have been taken:
1. Read in a set of training data
2. Extract selected features (as HOG, colour histograms and spacial bins)
3. Train a classifier based on extracted features from the data set
4. Apply a sliding window that scans a given image for the trained features
5. Collect the regions matched
6. Apply a heat map to display only one region
7. Repeat steps 4 - 6 for every image in a video and store the resulting image in a new video file

Remarks: since the processing pipeline passes each image through different libraries, and since the training and test data are used from different sources, it is important to verify that the interpretation of the images is done identically. My observations regarding the processing chain are stated in the following.

1. Image scaling
   - mpimg.imread(file) returns an scaled from 0 to 1
   - moviepy fl_image returns an image with value range 0 - 255

—> This implies scaling the image in the find_cars method

2. Colorspace of mpimg.imread(file) and moviepy fl_image is RGB

3. Scaling of feature vector for training and for video creation is done using StandardScaler().fit()

Training Data

The provided training data included the following 8792 images labeled as car and 8968 images labeled as “nocar”. An example of an image labeled as “car” and one labeled as “notcar” is provided in figure 1.

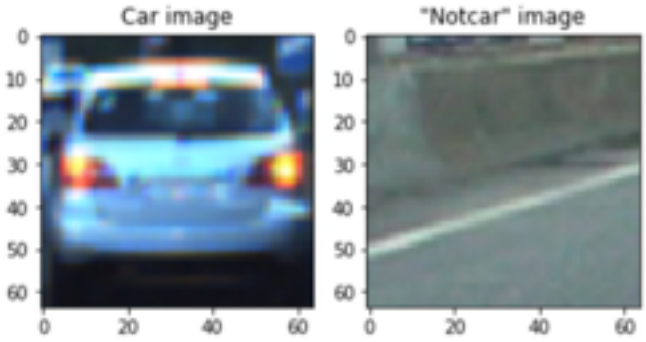

Figure 1: Example of “car” and “notcar” data

Feature Extraction

I have experimented with different colourspaces and my observations indicated that the YCrCb colourspace (used for the analogous television signal) was the best choice; although the visualisation of each single channel seems to imply redundancy, the usage of each single channel was necessary for the task. The amount of HOG orientations was selected to be 9, pixels per cell to 8 and cells per block to 2 as this implied a tradeoff between performance and results.
From all the features used, I found the HOG features to be the most important ones for the classification task. However, I have also used spacial bins and histograms as their usage led to an enormous decreased in the false positive rates.

An example of HOG features in YCrCb colourspace of a random car-labeled image is shown in figure 2.

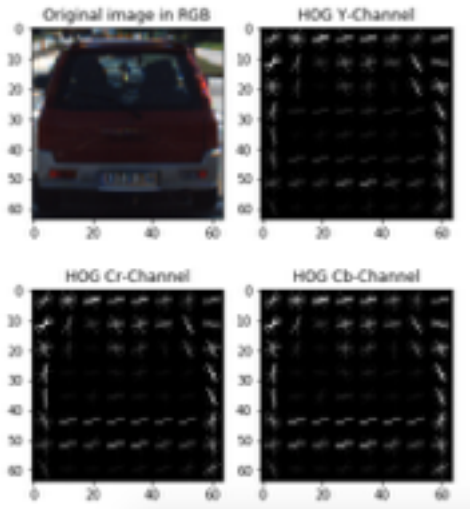

Figure 2: HOG features example in YCrCb

Classifier Selection

I have experimented with a few classifiers.

* DecisionTree
* NaiveBayes (BernoulliNB and GaussianNB)
* LinearSVM
* SVM with rbf Kernel

The NaiveBayes Classifiers had too many false positives. Same applied to the DecisionTree classifier; which was slow in addition.
The best classifier was the SVM with a rbf kernel. Even in most complex test images, it seemed to have no false positive classifications. Since however, the classifier was the slowest, I have the standard LinearSVM (as a tradeoff for performance and runtime). Example results with a DecisionTree classification are show in figure 3 and with a LinearSVM are shown in figure 4.

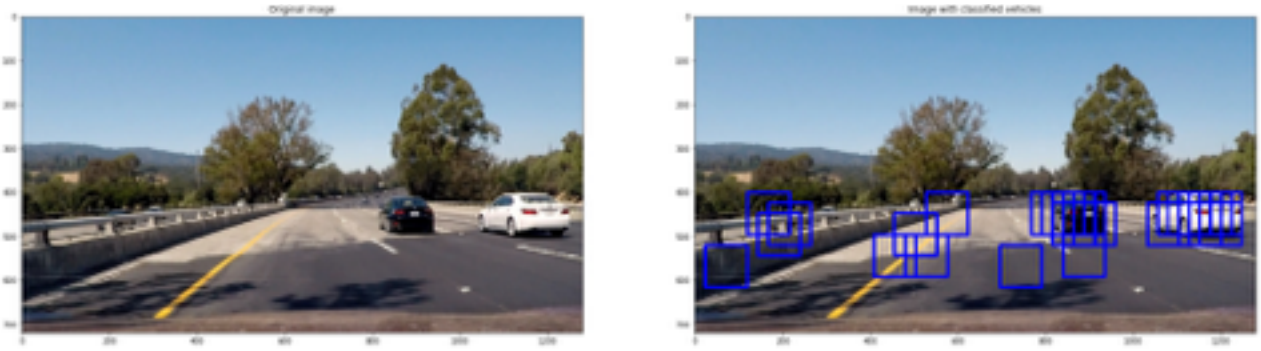

Figure 3: Classification results using a DecisionTree classifier

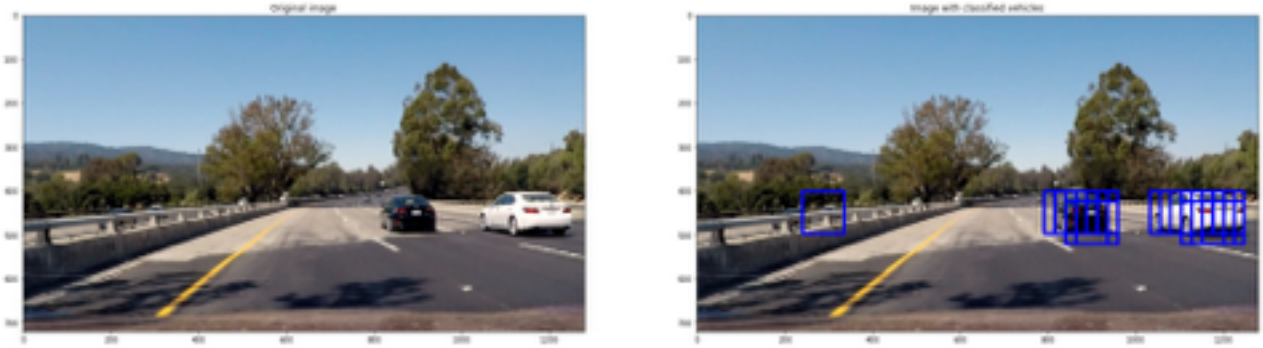

Figure 4: Classification results using a LinearSVM classifier

Improving the Search Grid

I have encountered a lack in computational speed by using a simple grid based search approach. This certainly arises from the fact, that the (first) sliding window implementation provided in the tutorial was searching the region of interest with one grid cell format and was extracting the HOG features per cell.

The second proposal was based on a scaling factor that I have used to divide the image in two parts by using a bigger search window at the bottom of roi and one additional small search window at the upper part of the roi. The reason for the overlap were issues in classifications of cars at the upper part.
An example is shown in figure 5 where the smaller search windows are displayed in blue and the bigger ones are displayed in red.

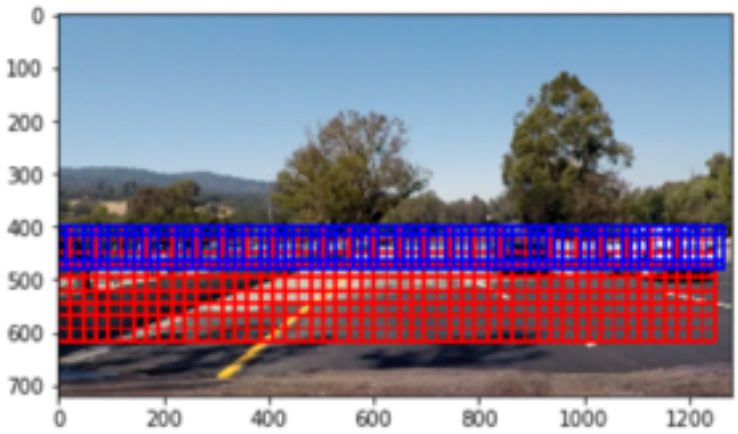

Figure 5: Search grid illustration

Reduction of False Positives

In order to reduce the false positive rate as well as the amount of displayed classifications, I have taken over the heatmap approach from the tutorial where a threshold for multiple classifications with the scipy label operator was described.
Figure 6 shows an example of unfiltered and filtered classifications. Note that for the filtered classification a heatmap threshold of 5 was applied.

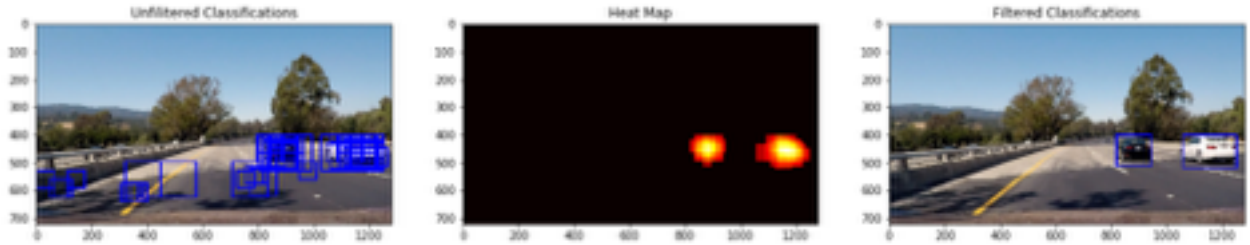

Figure 6: Illustration of classification filtering using a heatmap

Improving the Visual Representation

With the above mentioned steps, the classification rate was already around 99%. The rectangles identifying the classifications were very shaky, however. In order to smoothen the projection of the rectangles, I have introduced global memory for detected cars (in form of a class) that tracks classified cars over multiple frames. The class includes the centre of a detected vehicle which is derived from positions over the last 30 iterations (that should correspond to one second).

Furthermore I have introduced the method “track_vehicles_from_labels” that identifies if a classification corresponds to an already known vehicle. The identification is based on the centre of previous rectangles within a radius of 120px (refer figure for an illustration). The rectangles are calculated from the mean of the last 30 iterations. This enables a smoothly moving box around the classifications.
Rectangles are only drawn if these have been present over the last 5 iterations. In a way this is similar to the heatmapping approach.
Loss in classifications are handled by keeping all previously drawn rectangles for at 15 iterations after the loss.

Figure 7: Radius of rectangles for mapping of new classifications to existing ones.

Discussion

The major issue in my implementation is the speed of the processing chain. Since the detection of vehicles in real world would require to track vehicles in real time, the presented algorithm would not be feasible.
My approaches for further improvement are listed in the following.

1. Programming language: although Python is very powerful, the choice for a real world scenario would be C++ or even C as these languages are not interpreted and have a clear focus on speed.
2. More training data: based on my observation, the data provided did not seem to include many images of vehicles taken from the left or right hand side. This is also the reason why my classifier did not identify the white car at the middle of the project video (which is also the reason for the second search grid).
3. Perspective Transform: in order to overcome issues resulting from lack of training data, I would try to use perspective transform on vehicle that have already been classified.
4. Reduction of search space: in order to speed thing up, a dynamic search grid is necessary. It should not scan occluded spaces and for instance.
5. Two-staged classification: NaiveBayes, as mentioned above, had the fastest training and classification time but seemed to have too many false positives. Keeping this in mind, I would run a search on a dense search grid with the NaiveBayes and one other search with a bigger scaled search grid using the linear SVM. Using heat maps, this could combine the strengths of both classifiers.
6. Deep NN: a Deep Neural Network with several convolutional layers is most likely to identify shapes of vehicles better than HOG.