Vehicle Detection and Tracking Project

The goals / steps of this project are the following:

- 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.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

I included numbered sections in the code such that each rubic point can be easily identified. The jupyter notebook provided for this project has the following seven sections:

0. Importing relevant libraries from python, sklearn and other.
1. Loading the car and non-car data sets
2. Feature extraction (here you can find the HOG feature extraction) – in this section at the end I have done the splitting of the data into test and training and the normalization as well
3. Training a classifier (SVM)
4. Sliding window approach
5. Heat map generation
6. Image processing pipeline – this is the pipeline necessary for processing 1 frame
7. Process Video – at the beginning of this section a queue class is created and used for storing the hot boxes from the last 10 frames.

In the next parts I will answer the questions from the rubic. The answers can be identified in each section from the jupyter noebook provided.

A. Feature extraction and classifier training

```
color_space = 'RGB' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 16    # Number of histogram bins
```

The color space used in my project is RGB. I used this color code because I thought that 3 channels will give me more information about road objects. I used 9 orientation thinking that they are enough for describing an object. I generally experimented with several parameter combinations and the

ones I used, offered the best results. Other colour spaces and features might provide better results but for demonstration purposes I have just used the HOG with RGB. I have also used only hog because on my computer it took a lot of time to extract and train the classifier. In figure 1 we can see the result of the feature extraction

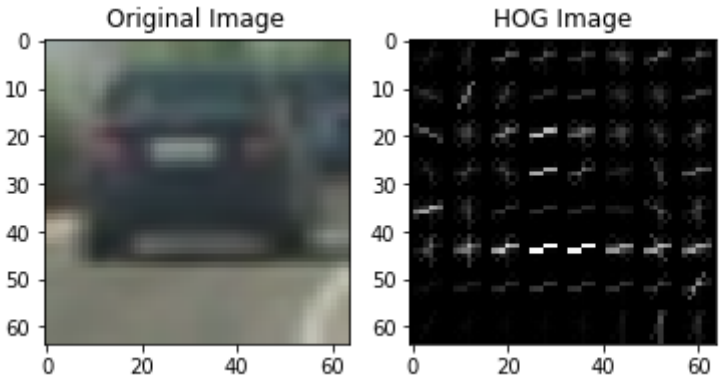

Figure 1. Here we have a snapshot of an image and the HOG descriptors with 9 orientations

B. Training Classifier

In cell number 6 I extracted features for car/not car and stored them in a feature vector X (which was then scaled). The data was split into randomized training and test sets - I made the training set 80% and the test set 20%. (These settings are visible in the last part from section 2 – cell 7).

I have used a support vector machine as a classifier. The classification begins in section 3 in cell 8. In this section I also compute the accuracy of the classifier (in order to see what effects my changes have). For viewing the accuracy I have a variable n_samples_predict which tells us what is the number of samples on which the accuracy is computed.

C. Sliding window

In section 4 I have implemented the sliding window technique. In cell 9 I introduced the helper functions from the course.

In cell 10 I have tested the algorithm and drawn the windows that were classified as cars. As we can see there are many misclassified regions.

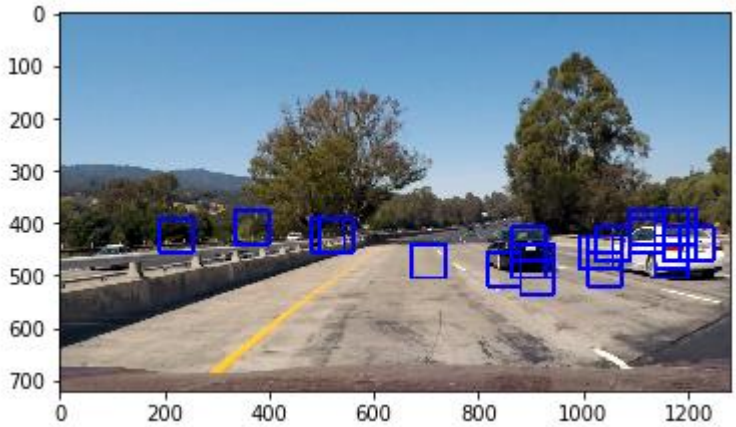

Figure 2. In this figure we illustrate the results of the sliding windows

To improve the result of the sliding window, I have split the image in 3 regions: near region, a middle region and a far region. For each of these regions I have generated windows of different sizes. I have created a function called generate_scan_boxes which creates sliding windows based on some parameters like the starting and ending limits of the window on x and y, overlapping percentage etc. I have called this function 2 times in order to generate more sliding windows with different dymensions. In figure 3 we see the result of the generated windows. I have set the overlapping percentage 50% and 75% in some cases. The reason for this is that a vehicle can be partially included in another scan box and may get misclassified for this reason.

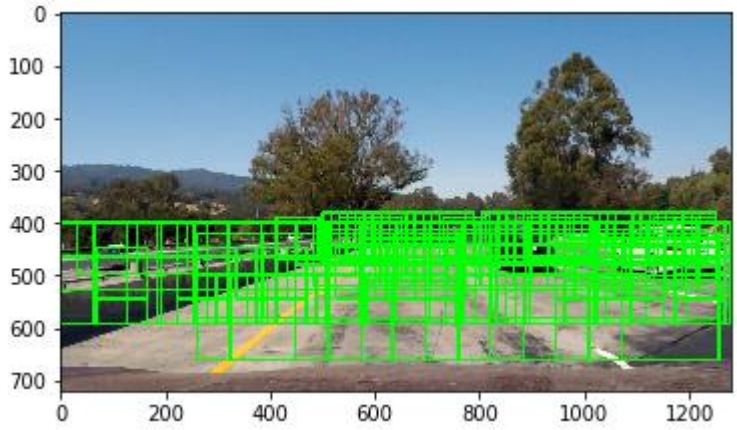

Figure 3. Generated windows

The final positive regions of interest are displayed in figure 4.

Figure 4. Final car detection result on one image

D. Heat map

In order to filter false positives I use a heat map (we add heat ,+1, to all pixels from a rectangle that overlap). We then apply a threshold to the map in order to reduce false positives). For the picture above the heat map is displayed in figure 5 below. The threshold introduced is has the value 2. Most of the code in this section has been taken from the Udacity course.

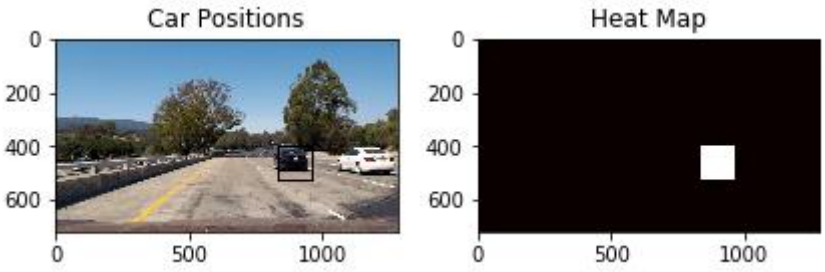

Figure 5. Filtered boxes and heat map.

E. Video Processing

In section 6 and 7 of the jupyter notebook I have implemented the general pipeline for processing a frame and a video(which is a sequence of frames). In cell 17 I created a class called BoxesQueue, in which I store the car boxes from the last 10 frames. When we will apply the heat map we will use these boxes with a larger threshold in order to better filter the false positives. I have also put an extra condition, to filter the bounding boxes. In case the dimension of the bounding box is smaller than a predefined number of pixels threshold, the box is not considered for future frames.

Discution

My main issues in the development of this project were the selection of the correct parameters for my algorithms. As we can see the overall pipeline is very sensitive to the number of search boxes generated, the size of these boxes, thresholds used and so on. Maybe a learning approach in which, depending on the environment, a neural network could tell us the optimal sizes of the generated scan boxes and their number , could be used. The parameters used affect the object segmentation. If I use fewer search boxes, the objects will get more fragmented, if I use more boxes, the objects can get joined together. Another major issue would be the classification accuracy. This issue could be solved by using more features (like good features to track, LBP or others), and making a more powerful classifier using ensemble methods like ada boost (building a stronger classifier by using several weak classifiers). Tracking the detected vehicles, using an extended Kalman filter, will also lead to an improvement of the algorithm.