Writeup

Vehicle Detection 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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

You're reading it!

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images.

All my code can be found in the Jupyter Notebook "Vehicle_Detection_and_Tracking.ipynb".

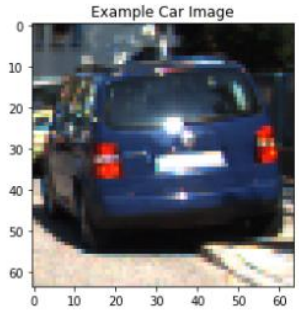
To get HOG features, I defined a function get_hog_features(), that uses skimage.feature.hog from the scikit-image library. The function can be passed a grayscale image and some parameters like number of orientations, pixel per cell, cells per block etc., and it outputs the hog features and optionally a bitmap visualizing the gradients.

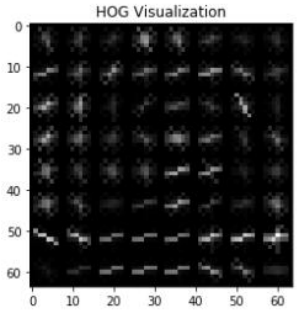
```
hog_features, hog_image = get_hog_features (gray, orient,
                                            pix_per_cell, cell_per_block,
                                            vis=True, feature_vec=True)
```

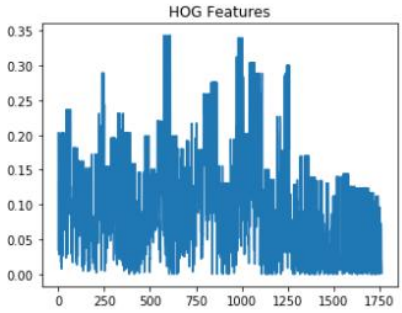
2. Explain how you settled on your final choice of HOG parameters.

I tried various combinations of parameters and finally used 9 orientations, 8 pixel per cell and 2 cells per block.

Below you can see one of the given training data images and the corresponding HOG visualization and HOG features.



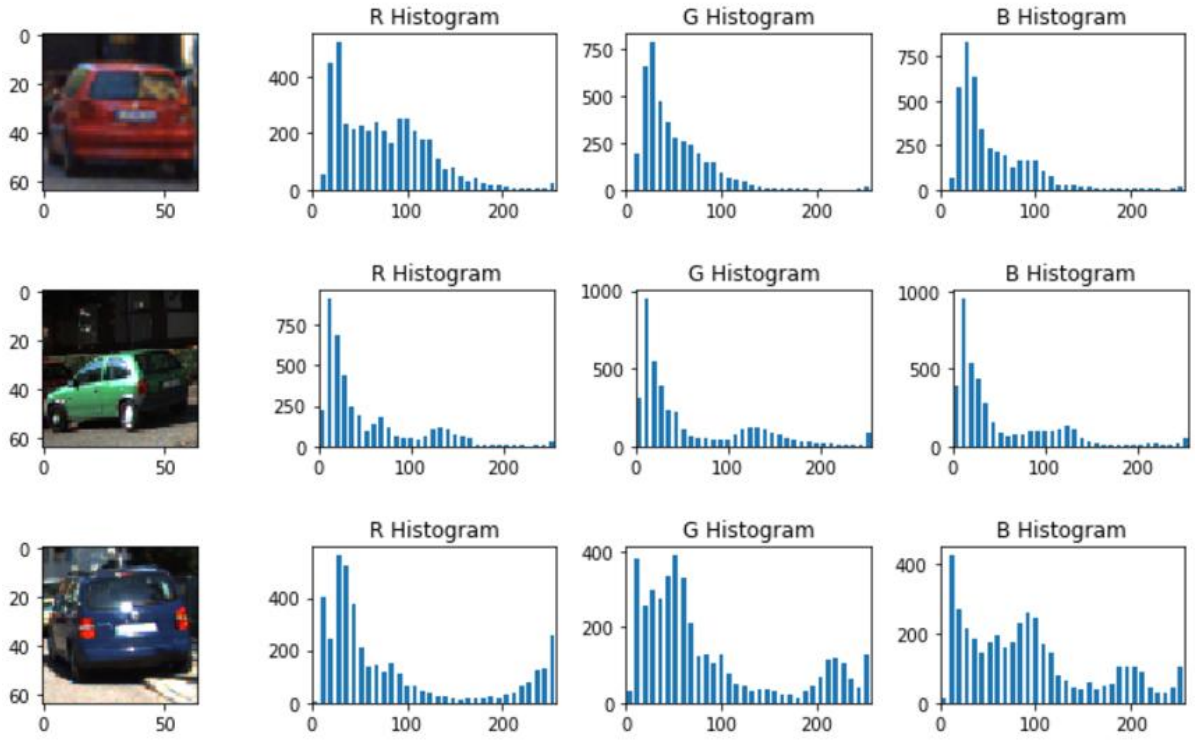




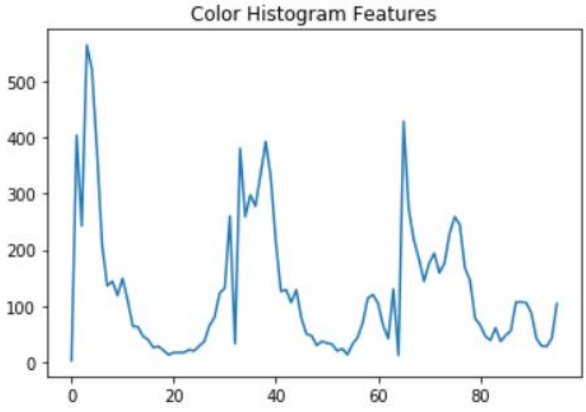
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

Training the classifier, I also used color features and spacial binning.

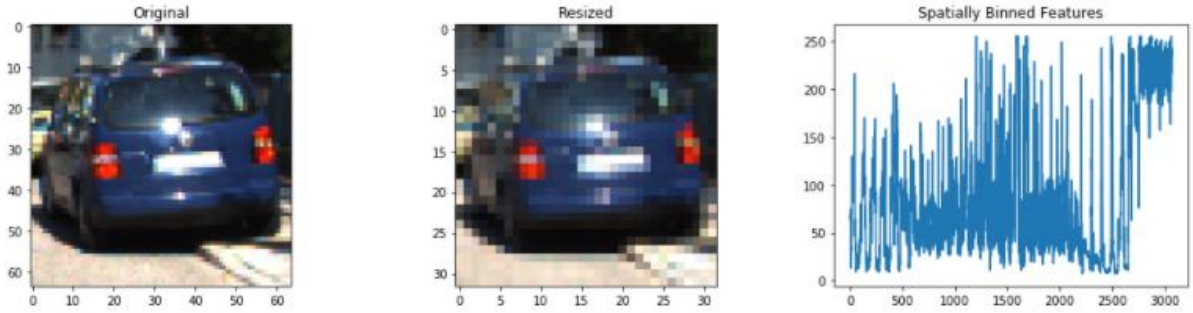
I used the function color_hist() to get color histogram features. Below, you can see the histograms of the 3 color channels for 3 cars of different color (red, green, blue). For the red and blue car, you can see a rise of values around an intensity of 100 in the corresponding channel (red or blue).



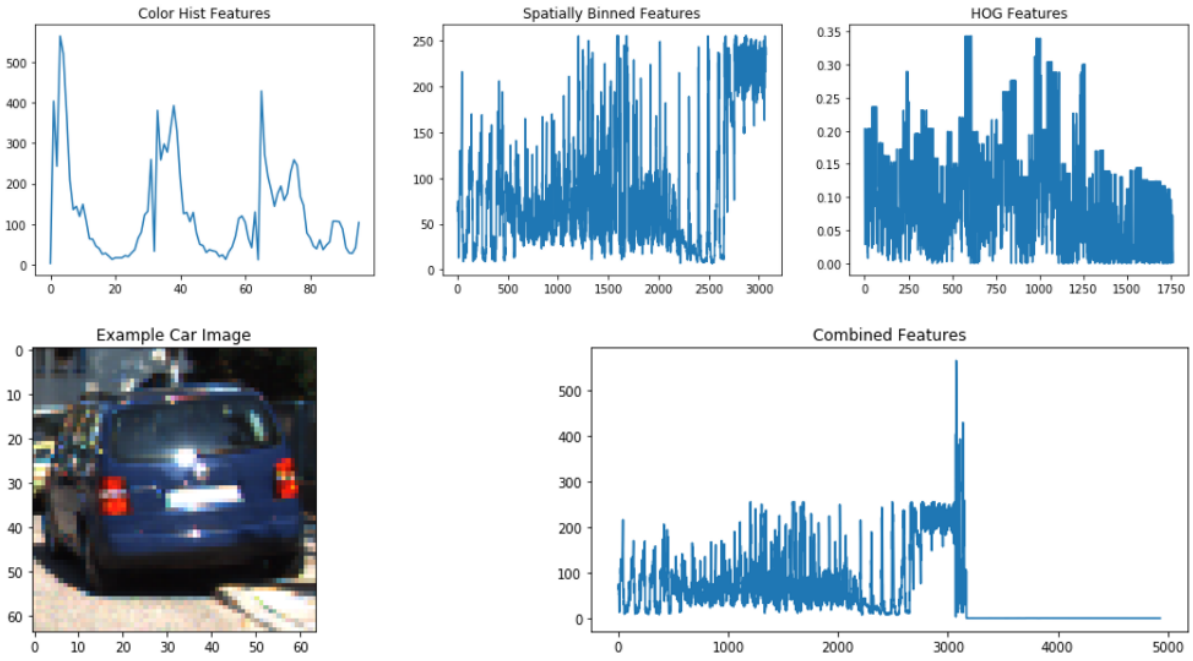
The color features of the 3 color channels get concatenated. Here is an example for the blue car (last row in the figure above).



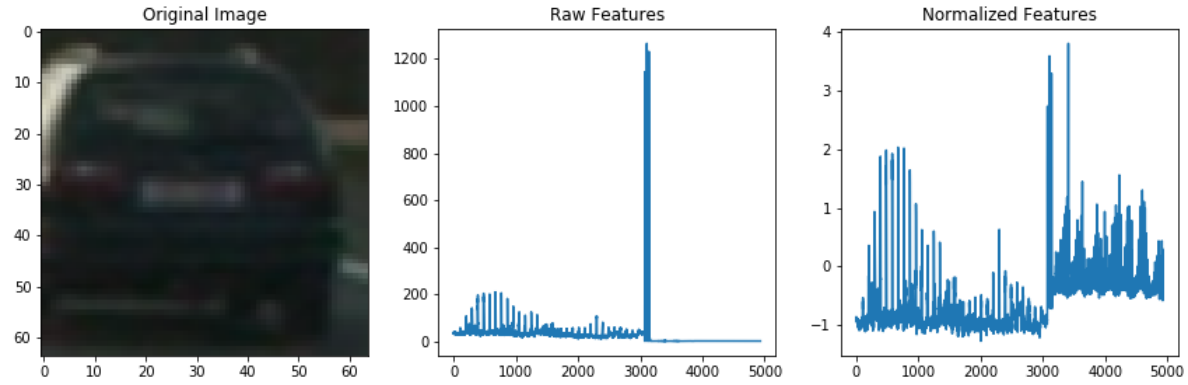
I used the function bin_spatial() to get spacial reduced features. In the following example, I resized the input image (64x64) to 32x32 pixels and then flattened the pixel array to a feature vector.



Then I combined all features to a single feature vector.



Before training the classifier, I normalized the feature vector using StandardScaler from sklearn.preprocessing.



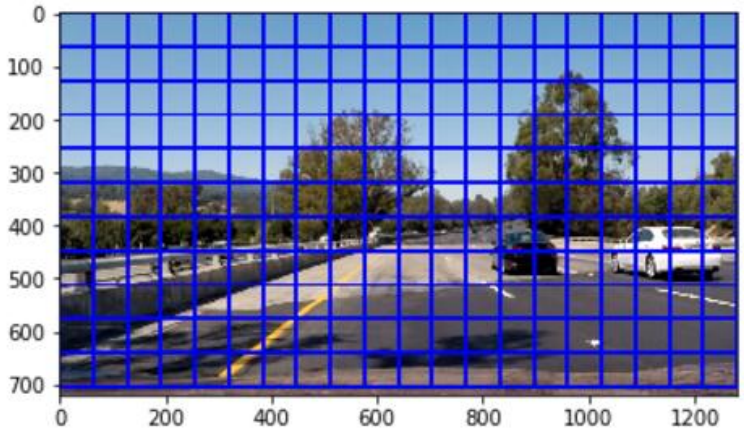
I used a Linear SVM classifier (LinearSVC from sklearn.svm). I split the given data in a training (80%) and a test set (20%). Accuracy of the trained classifier was 98%.

Sliding Window Search

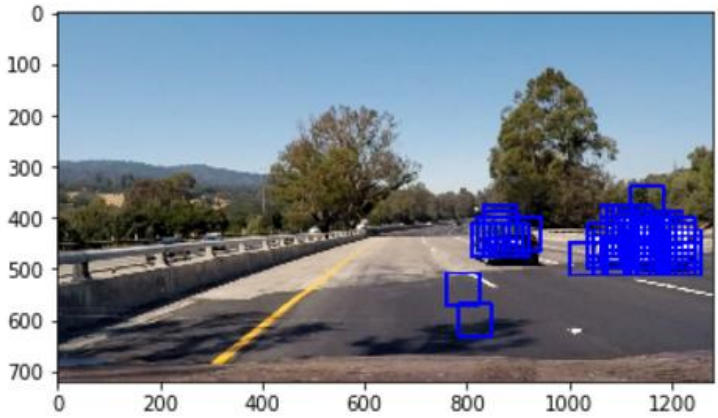
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I used a sliding window technique to search for cars in the images.

The function slide_window() divides an image into sliding and overlapping windows.



The function search_windows() examines the windows returned from slide_window() and returns an array of all windows where the classifier found a car.



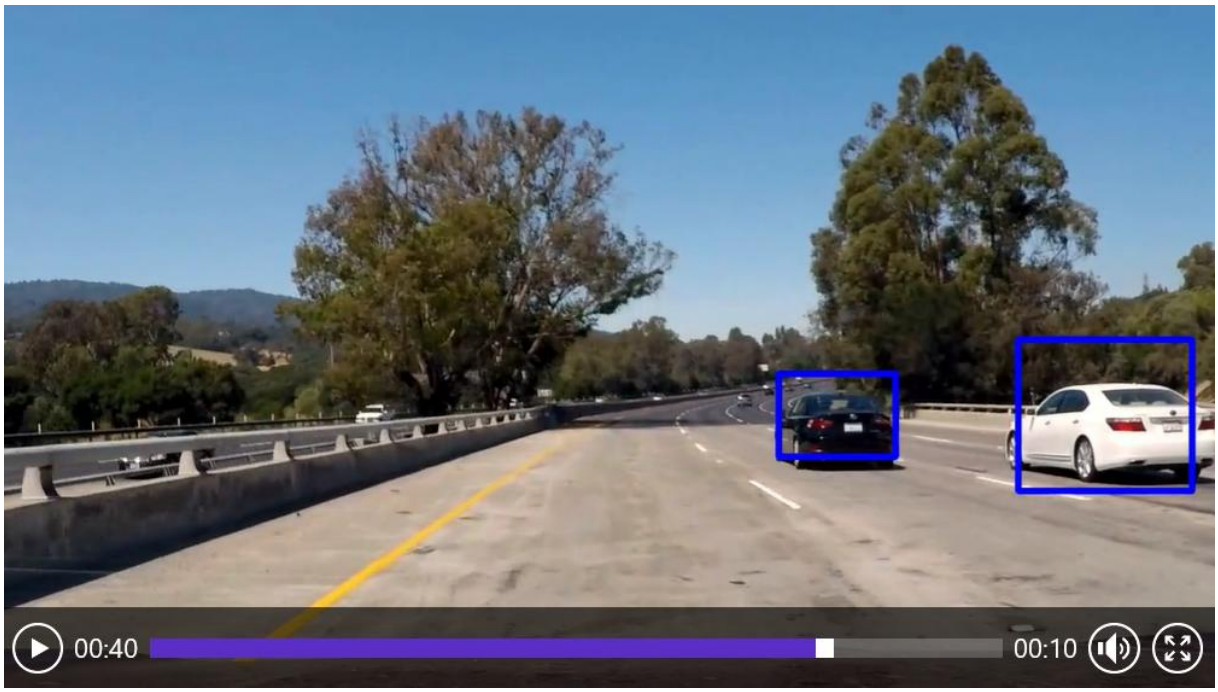
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

To improve performance, I used a HOG sub-sampling technique, where the HOG features are calculated only once for the entire image, and then, the sliding windows use the pre-calculated values. The HOG sub-sampling is implemented in the function find_cars().

Video Implementation

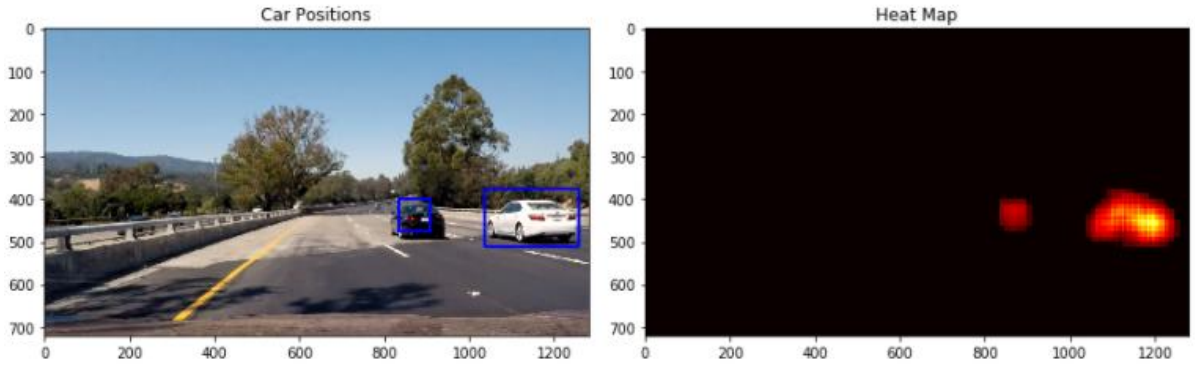
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

The final result can be found in the video " project_video_processed.mp4".

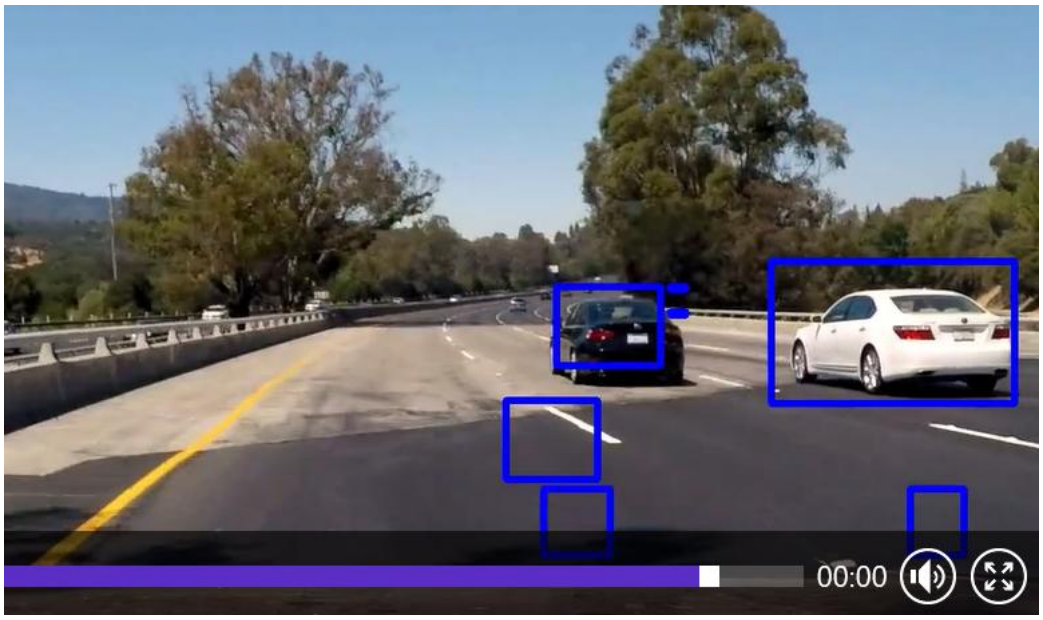


2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

To avoid multiple detections and false positives I used a heat map technique to count overlapping boxes and apply a threshold on the count value.



The first pipeline used only sliding windows of one size and showed some false positives.



To increase performance, in the second pipeline, I used HOG sub-sampling.

In the 3rd and final pipeline, I used multi scale search windows and increased the heat threshold.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

The result of the final pipeline still shows false positives. To improve the pipeline further, I would store the windows found for a number of frames and then calculate the heat map over these frames. Furthermore, I would restrict the sliding window search only to regions where a car was detected in the last frames.