Vehicle Detection Project Writeup

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

Writeup/README

1. Provide a Writeup
This document

Preliminaries

The project has ultimately been merged with the preceding "*Advanced Lane Lines*" project, which is why the respective code and classes are included in the IPython notebook.
The respective code is located in code cells 1 to 4.

HISTOGRAM OF ORIENTED GRADIENTS (HOG)

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

The code for this step is contained in the fifth code cell of the IPython notebook .
I started by reading in all the *vehicle* and *non-vehicle* images. Since initial results left room for improvement, I set up a pipeline storing false positives in a directory which I manually inspected, moving false positives in the respective category (i.e. vehicle vs

non-vehicle). Once completed I re-trained the classifier, hereby increasing detection rate and, perhaps more useful, reducing false positives.

Here is an example of one of each of the vehicle and non-vehicle classes:

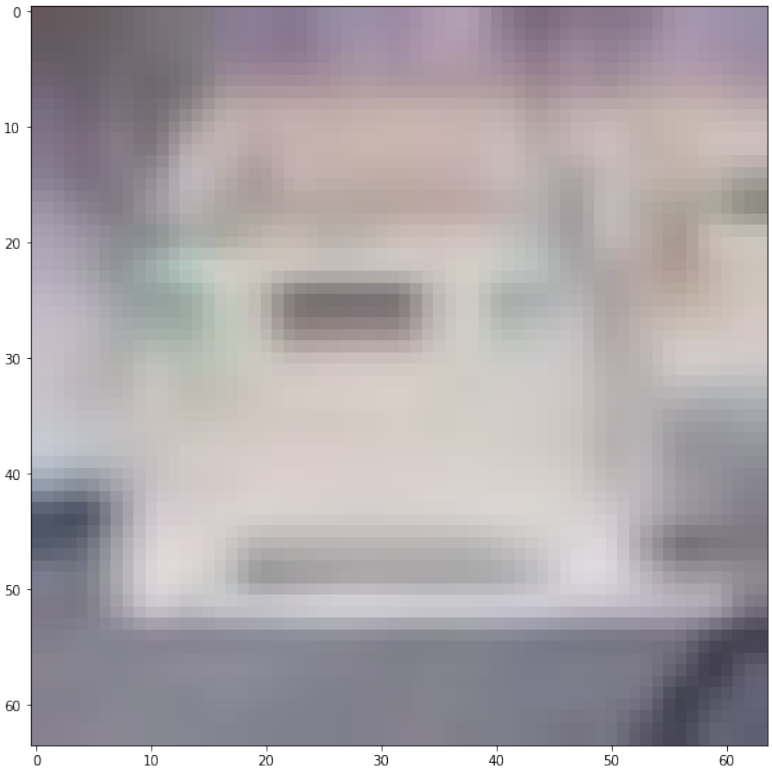


Fig (left): A car (scaled to reflect different color spaces between *cv2* and *matplotlib*)

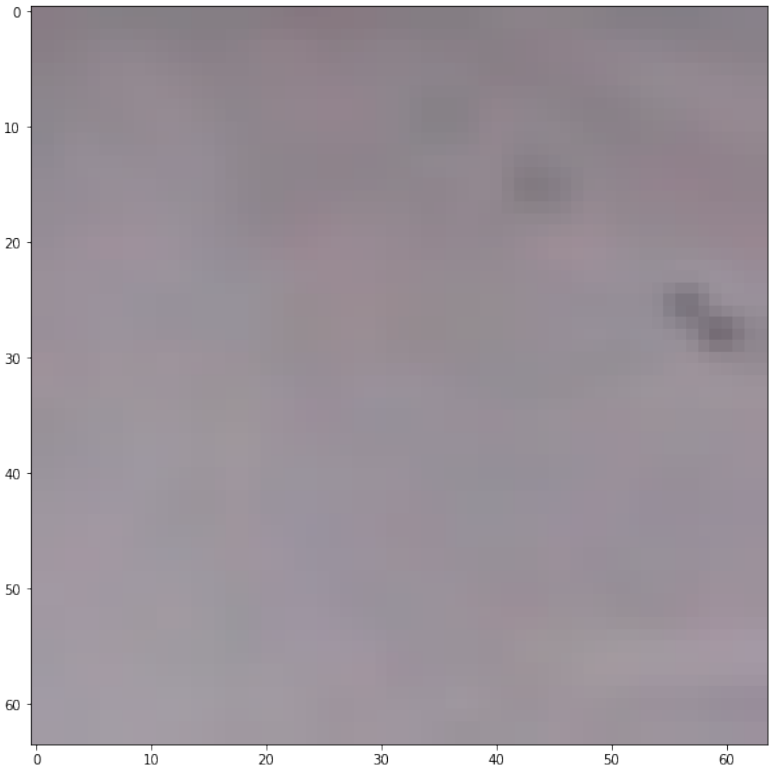
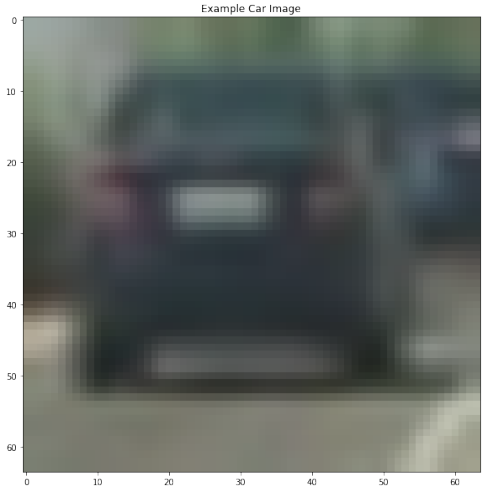
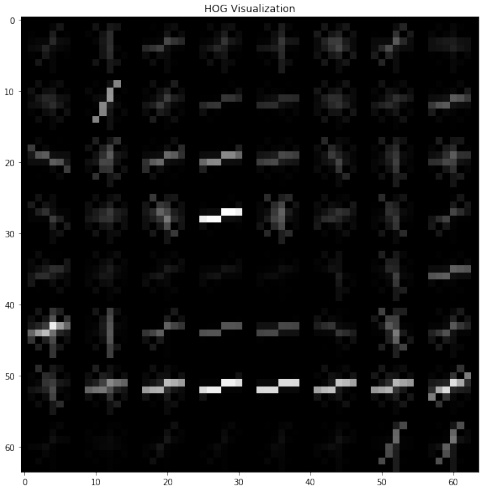


Fig (left): No car, again scaled

I then explored different color spaces and different *skimage.hog()* parameters (orientations, pixels_per_cell, and cells_per_block). I used random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like. Here is an example using the YCrCb color space and HOG parameters of orientations=8, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):





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

I tried various combinations of parameters and finally settled with parameters that appeared to work well

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).

I trained a linear SVM using HOG features (3 color channels), spatial and color histogram features. The features are extracted and concatenated using class methods (*extract_features(), extract_image_features(), get_hog_features()*). Since the training data consists of PNG files, and because of different ways of reading PNG files, image data is scaled up to 0-255 (see code cell 1) in a respective method.
Extracted features are combinded and normalized (code cell 5, method *combine_and_normalize()*).
Finally, a LinearSVC classifier is trained (code cell 5, method *train()*), splitting the data into a training set and a test set (20 % of the data).

Classifier output (code cell 6):

```
Using: 32 orientations 32 pixels per cell and 2 cells per block
Feature vector length: 8460
Saving Classifier...
Classifier Saved.
39.36 Seconds to train SVC...
Test Accuracy of SVC =  0.9815
My SVC predicts:  [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  1.  0.  0.  0.
0.  0.  0.  0.  0.
  0.  1.]
For these 20 labels:  [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  1.  0.  0.
0.  0.  0.  0.  0.  0.
  0.  1.]
0.00184 Seconds to predict 20 labels with SVC
```

As can be seen, the classifier accuracy is about 98.1 %

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 the code sample as given in the course material as template for my code (code cell 5, method *find_cars()*).
The function takes the image to work on as well as a scale factor as inputs.

The top and bottom y-range was found on a trial-and-error base, as was the scale used. Different scales were explored and the following set was eventually selected based on classification performance.

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.)

A sample output (*project_video_out.mp4*) is included in the submission

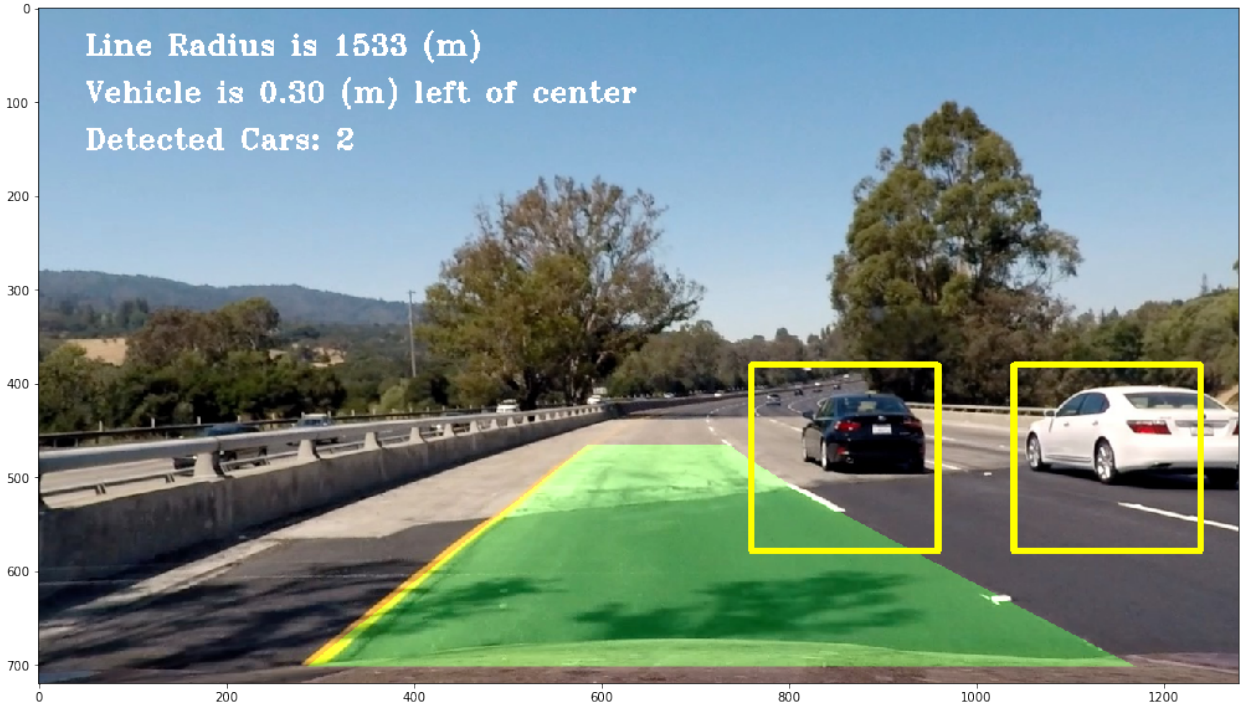
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used *scipy.ndimage.measurements.label()* to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

To smooth out detections, I stored each frame's heatmap in a list and averaged the heatmaps over 60 frames

Here's an example result showing the heatmap from a frames of video, the result of scipy.ndimage.measurements.label() and the bounding boxes then overlaid on the last frame of video. See example below.

Finally, the output images have been augmented with the output of the *Advanced Lane Line* project pipeline, giving the bounding boxes, lane lines, and respective information in text form.
An example output of a processed frame:



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 pipeline still detects a inconveniently high number of false positives. I am not positively sure on why that is, I suspect it has to do with color spaces.
Speaking of, the most elaborate part of this project proved to be the classifier, which seemed to work fine - albeit, in the sliding window / HOG subsampling part the classifier seemed to produce a much higher number of false positives.
As it turned out this was due mismatches in color spaces - which is why I now consequently scale images accordingly.

As said, the detection rate, both in terms of detections and, to a higher degree, false positives, still leaves room for improvement.
I suspect more experimentation in particular with different color spaces (HSV, perhaps) might prove helpful