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

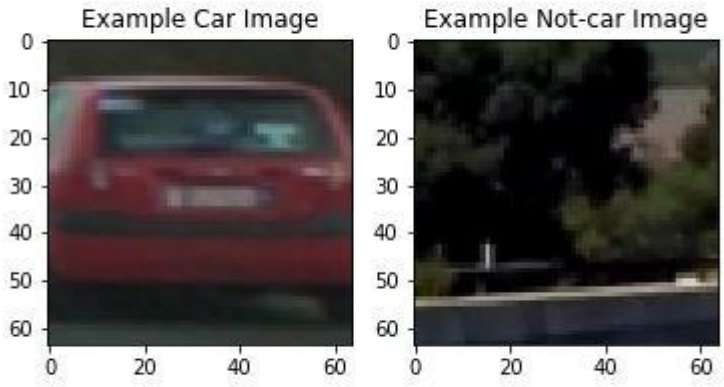
Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Histogram of Oriented Gradients (HOG)

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

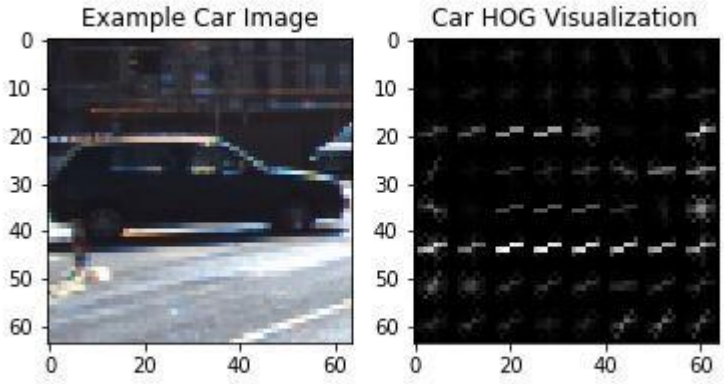
The code for this step is contained in the first and second code cell of the IPython notebook.

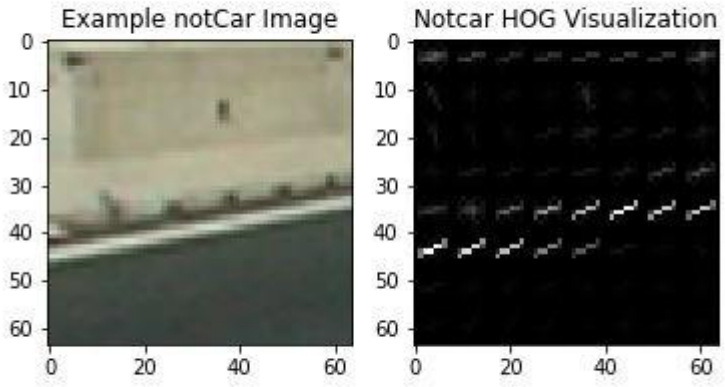
I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed 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=9`, `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, then I found the parameters listed below suit well on our dataset:

```
orient = 9
pix_per_cell = 8
cell_per_block = 2
```

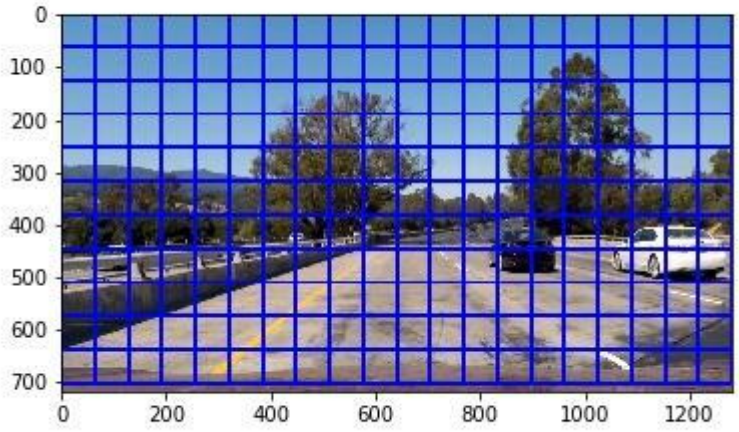
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

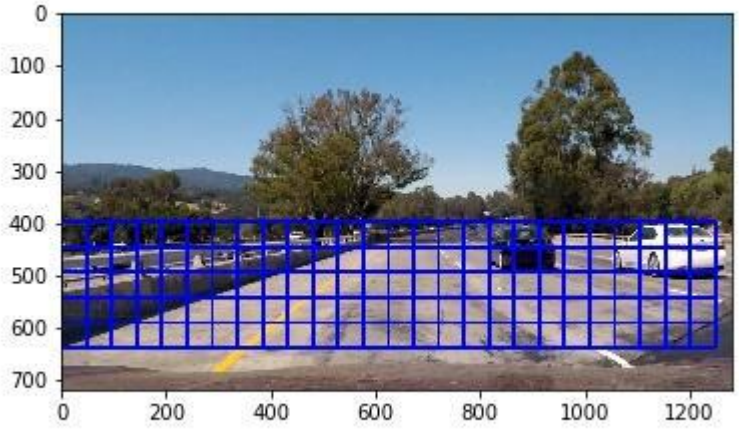
I trained a linear SVM using three kinds of features. One is binned color features, which defined as `bin_spatial()`. Another one is color histogram features and the function is `color_hist()`. The last but not the least is HOG features. I combined them by using a function`extract_features()`. Then, I can extract features of car dataset and notcar dataset. After feature extraction, I standardize the features and split up data into training and test sets so that the SVM classifier can be trained and tested. We achieve a test accuracy of SVC = 0.9876, which is a promising result for our task.

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 decided to search window positions at a certain scales all over the image. The position is located in `[400, 656]` in y-axis. This is based on the fact that the load and cars most likely appear on the bottom of images. Here are two images: one with full image search and one with certain position search.





Apparently, the second search method is better than the first. Additionally, I set the overlap windows as `xy_overlap=(0.5, 0.5)`.

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?

Ultimately I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:

To get a more efficient way to window search, I adopt a hog sub-sampling window search strategy. It is defined as a function `find_cars()`. Here is the window search result:

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.)

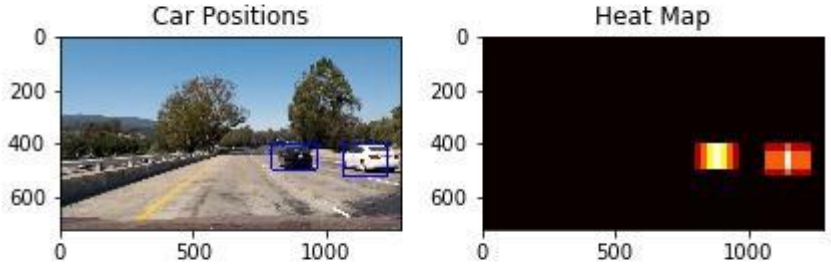
Here's a link to my video result

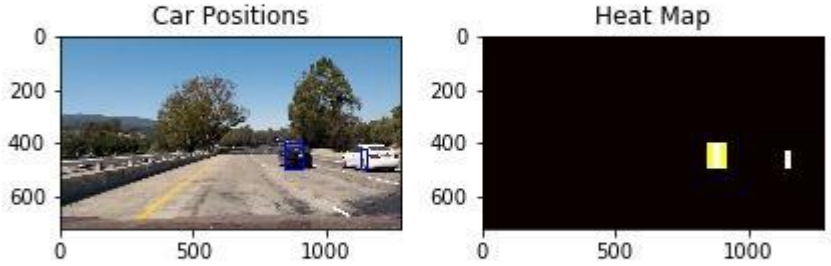
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

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.

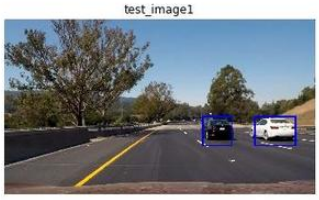
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the heat map with and without shreshold.





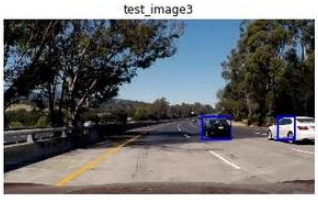
Here are six frames and their corresponding heatmaps:

test_image1



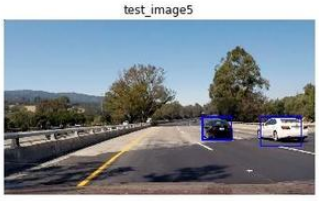
test_image2

test_image3

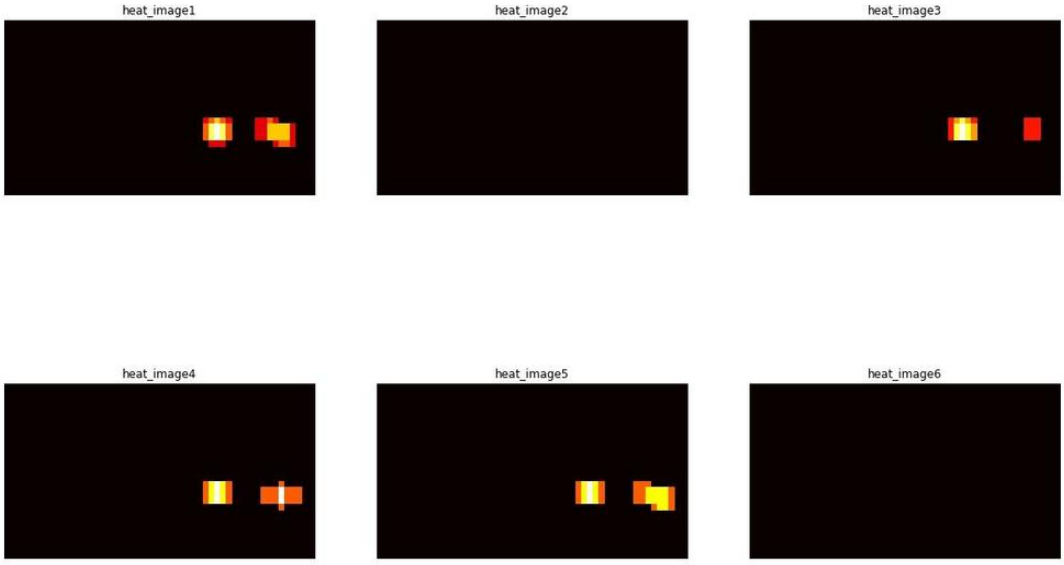


test_image4

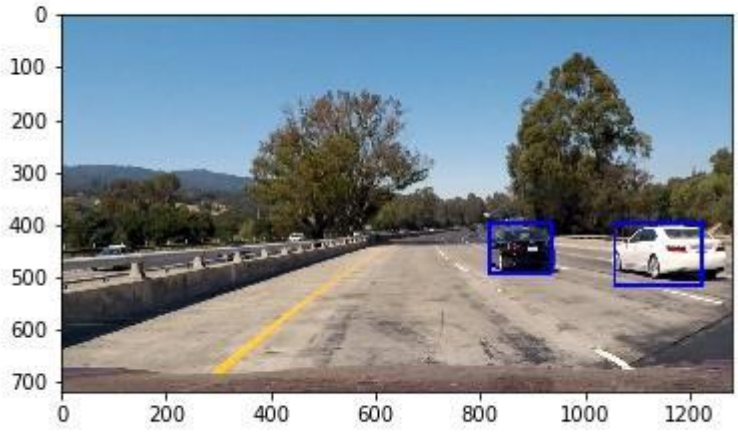
test_image5



test_image6



Here the resulting bounding boxes are drawn onto the test frame:



3. I finally combined vehicle detection and lane line finding result into one project. The output video is here project_video_final_result

---

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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.
The reaction of the model is a little bit slow. When the car head appears on your left, the model can't detect it immediately. In the future, we need to add more dataset to train the classifier or we can set some rules to recognize it.