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.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. [Here](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

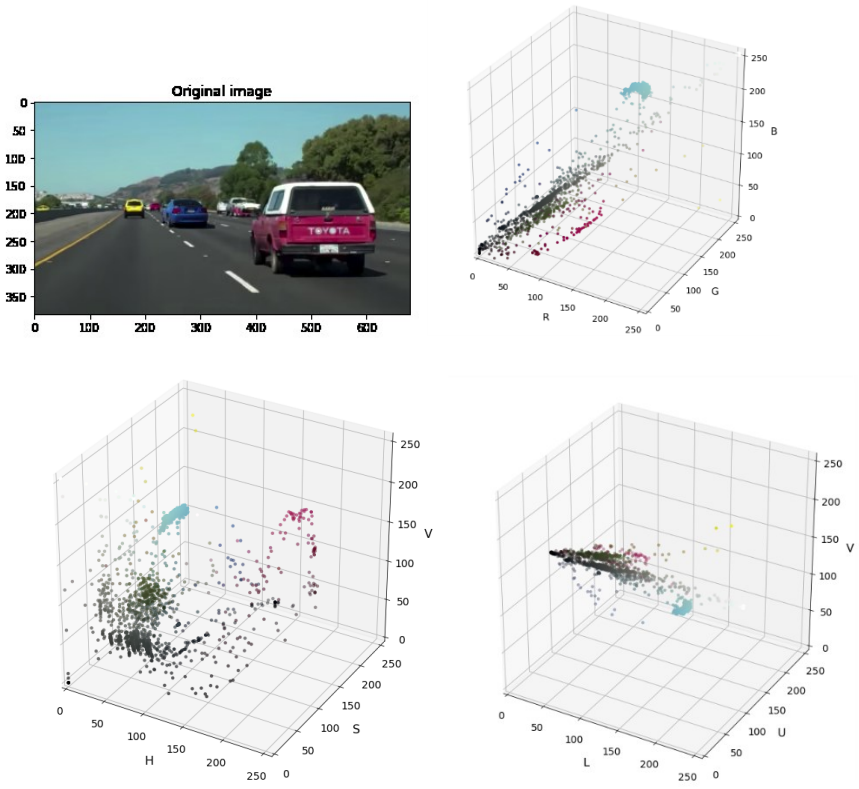
You're reading it!

Histogram of Oriented Gradients (HOG)

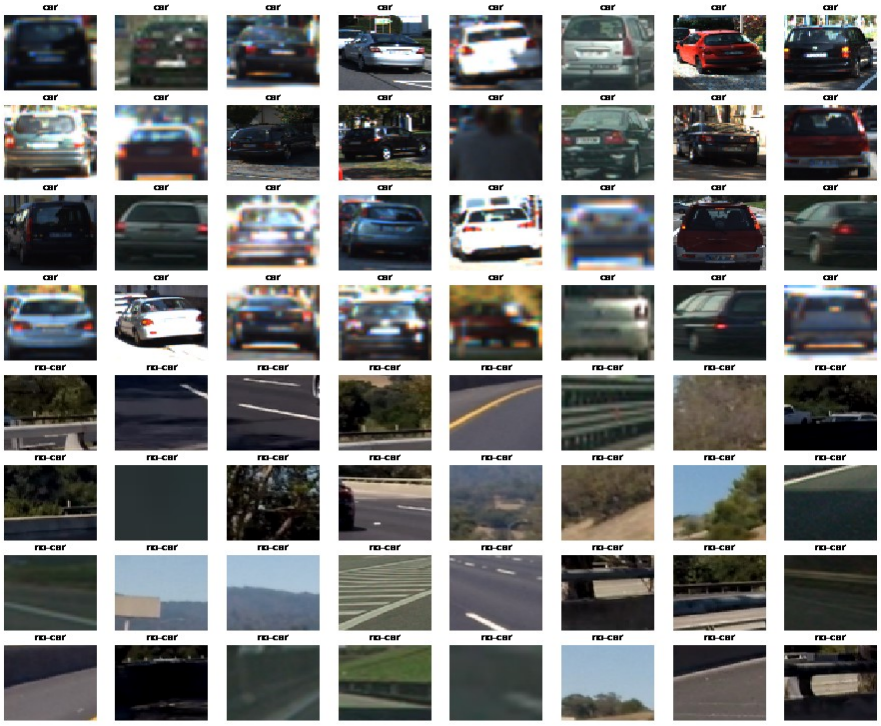
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the get_hog_features() function in the file “Vehicle Detection.ipynb”

I started first by exploring the color spaces. Saturation proved to be one characteristic that most cars shared.

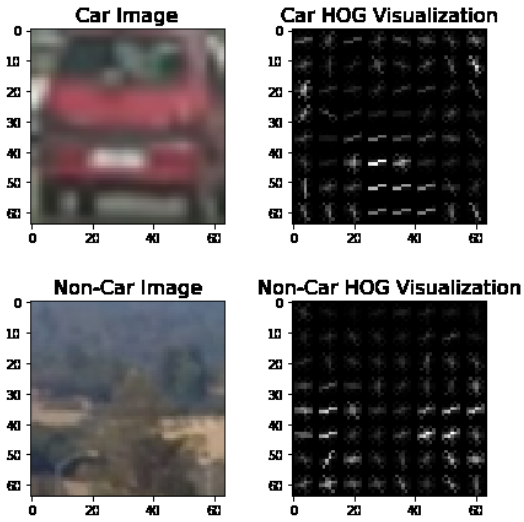


Then I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of each of the `vehicle` and `non-vehicle` classes:

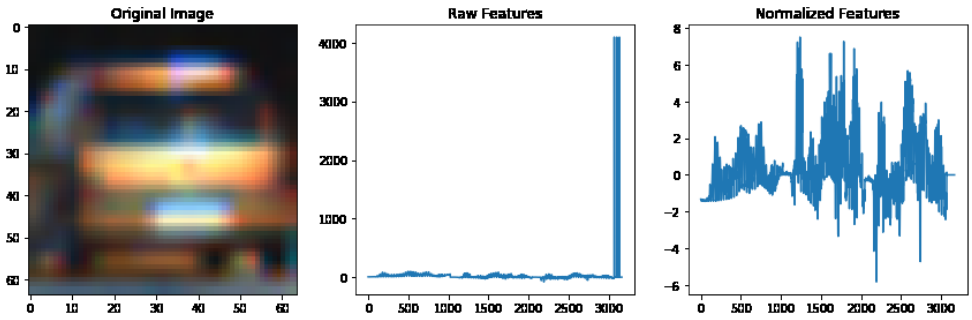


I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



I also explored on combining the features and then normalizing them as explained in the lesson.



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

I tried various combinations of parameters and settled on final choice of HOG parameters based upon the performance of the SVM classifier produced using them. I considered accuracy with which the classifier made predictions on the test dataset. Finally, I got:

```
88.89 Seconds to extract HOG features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 5292
59.5 Seconds to train SVC...
Test Accuracy of SVC =  0.9806
```

```
My SVC predicts:  [0. 0. 0. 1. 0. 1. 1. 1. 0. 0.]
For these 10 labels:  [0. 0. 0. 1. 0. 1. 1. 1. 0. 0.]
0.98096 Seconds to predict 10 labels with SVC
```

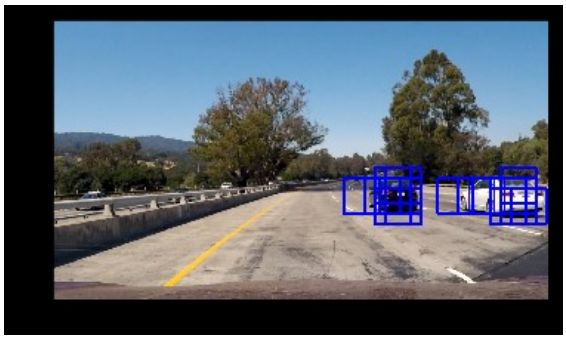
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVM with the default classifier parameters and using HOG features alone and was able to achieve a test accuracy of 98.06%. I also trained color feature based classifier and obtained a test accuracy of merely 90.4%. Refer to code block 9 and 10 (color classify & HOG classify) respectively.

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?

In the code block 11 (under the title - Hog Sub-sampling Window Search) - takes in an image and performs a sliding window search of the normalized combination of features on it. As a result it returns a list of bounding boxes for the search windows, which then I pass to draw draw_boxes() function.



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 only, which provided a nice result. Here are some example images:

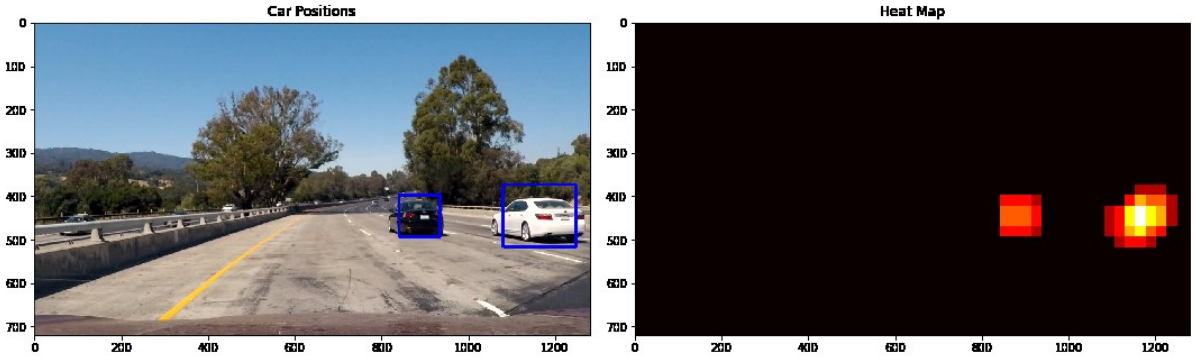
Video Implementation

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 video file is attached - project_video_result.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.

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. If the classifier is working well, then the "hot" parts of the map are where the cars are, and by imposing a threshold, I could reject areas affected by false positives. Here is an example of the output and the heatmap applied to an image:



Refer to code block #14 for the implementation in “Vehicle Detection.ipynb”

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?

For this project, I used a step by step approach mostly referring from each of the lesson videos and quizzes which helped me understand each part of the pipeline very well. This was very helpful when things didn't go exactly as expected. Debugging was bit of tricky and most of the time times my machine used to freeze during the training time (probably due to low configuration). At the end I put together a pipeline by simply calling the functions I implemented along the way. The pipeline is probably most likely to fail in cases where vehicles don't resemble those in the training dataset, but lighting and environmental conditions might also play a role.

I could think of improving the pipeline by averaging the detected boxes by apply the heatmap on more than one frame instead of each individual image. Also, exploring CNN would be worthy exercise to improve the accuracy !