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 can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

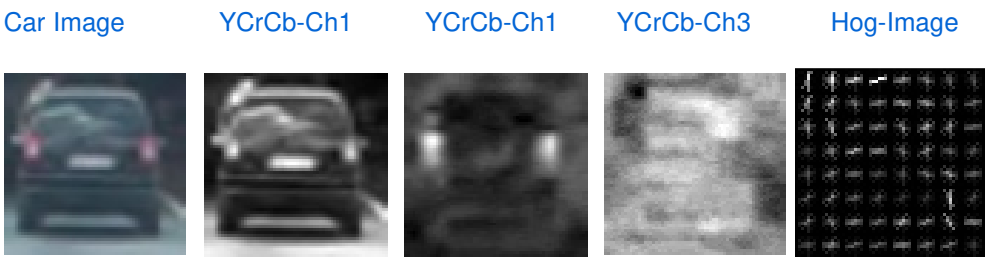
You're reading it!

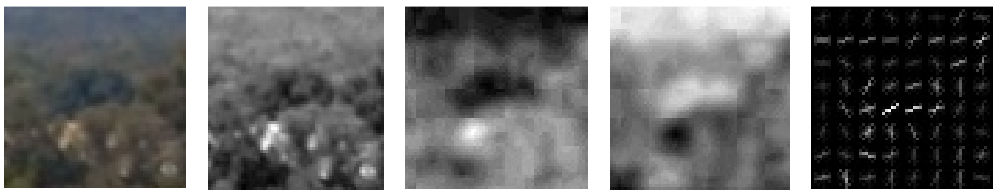
###Histogram of Oriented Gradients (HOG)

####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 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.

Before applying HOG, I transformed the image to YCrCb color space as it maintains the car features (i.e. shape) in its channels. Then I scaled the image between 0 and 1.

I settled down to the following mentioned HOG parameters they parameters were able to show clear distinction between a car & a non-car images.

HOG orientations = 9

HOG pixels per cell = 8

HOG cells 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 with the following features

- YCrCb Color space features
- HOG feature all 3 channels
- Spatial features with 16 histogram bins

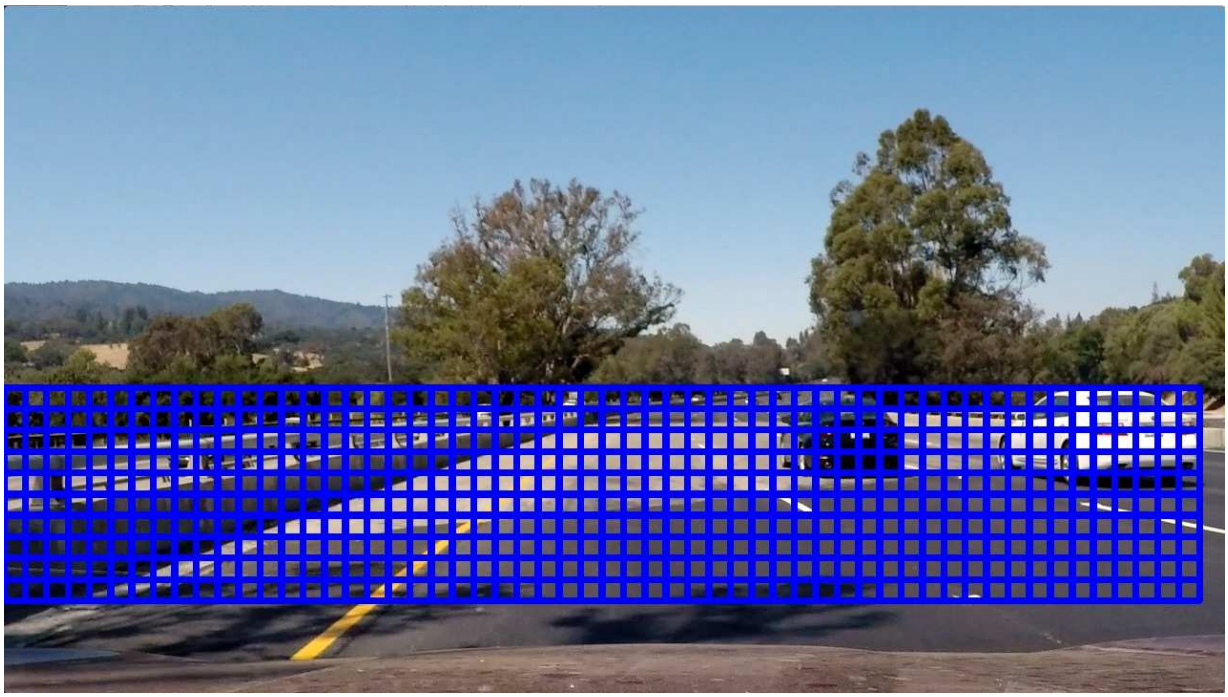
Before training, I scaled the features to zero mean and unit variance using standard scalar function.

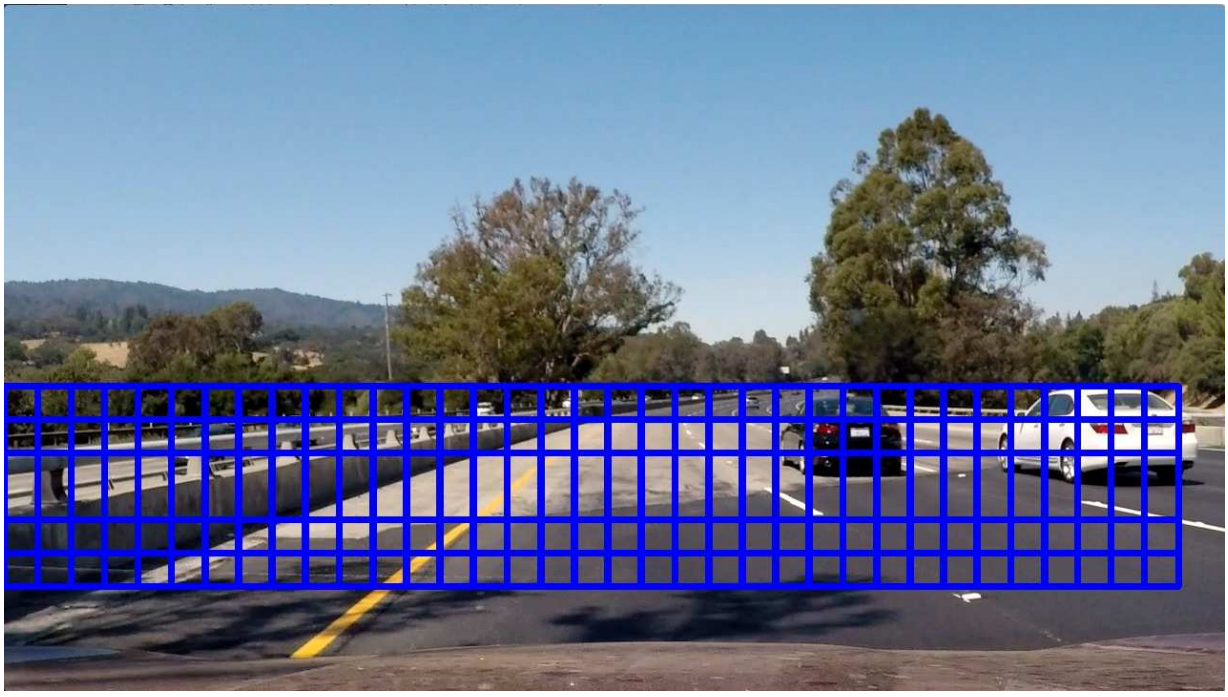
The code for the training is in 3rd cell of the jupyter notebook

###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?

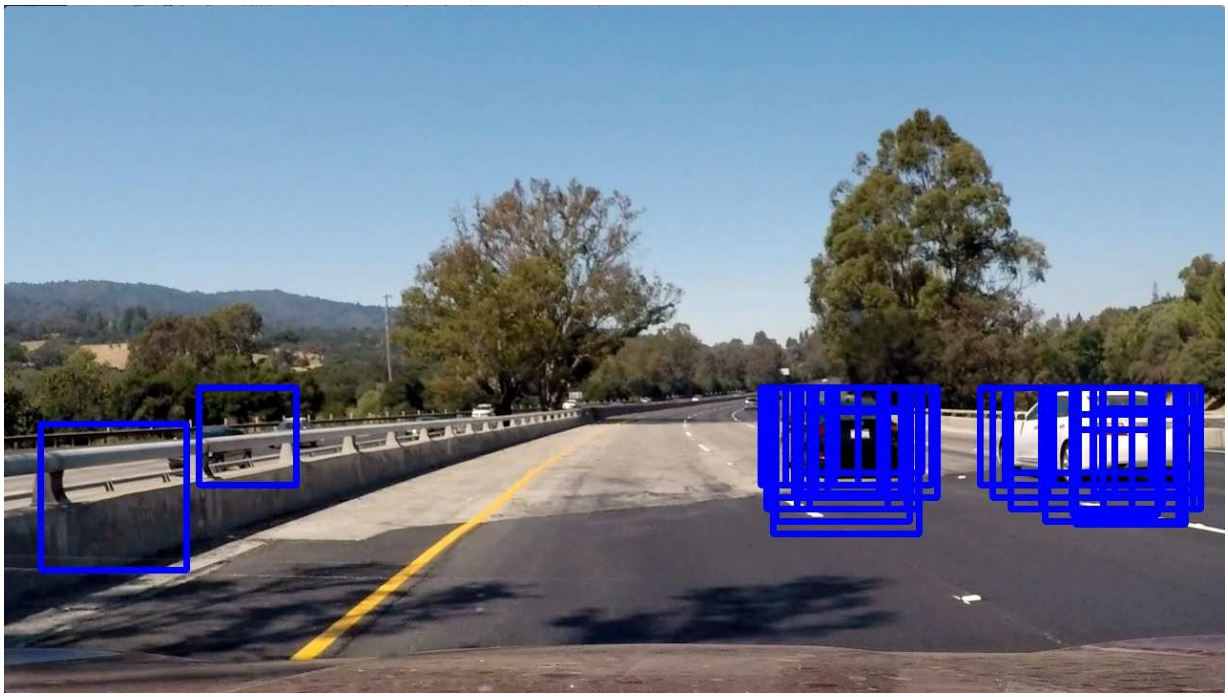
- The sliding window search is coded in the 4'th cell of the jupyter notebook.
- I applied hog transform on the complete image with the same hog parameters as used for training
- Then I defined my sliding window size as 8x8 cells.
- I used a parameter "cell_per_step" to define the overlap during the sliding. A value of 2 will result in 75% overlap in x & y direction
- I used a scaling factor to resize the image, so that the features are extracted at various zooming levels of the car. Scaling factor of 1 represents the same image size, whereas higher scaling factors reduces the size of the image which helps in extracting features at higher levels.
- I settled for scaling factors starting from 1.4 until 2.4 with 0.2 steps (i.e 6 multiple scaled windows). Attached are the example images of multiple scale windows with overlap.





####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 six 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:



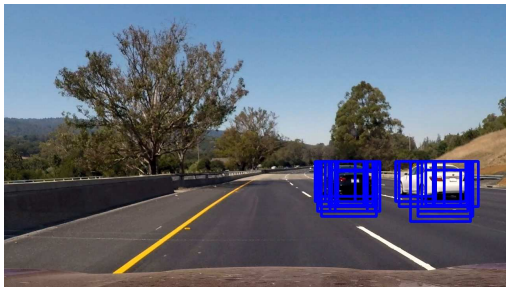
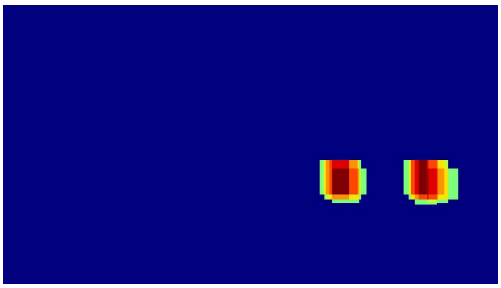
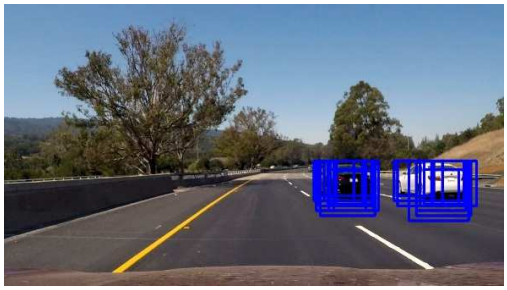
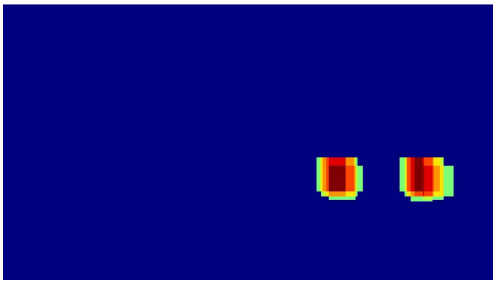
Video Implementation

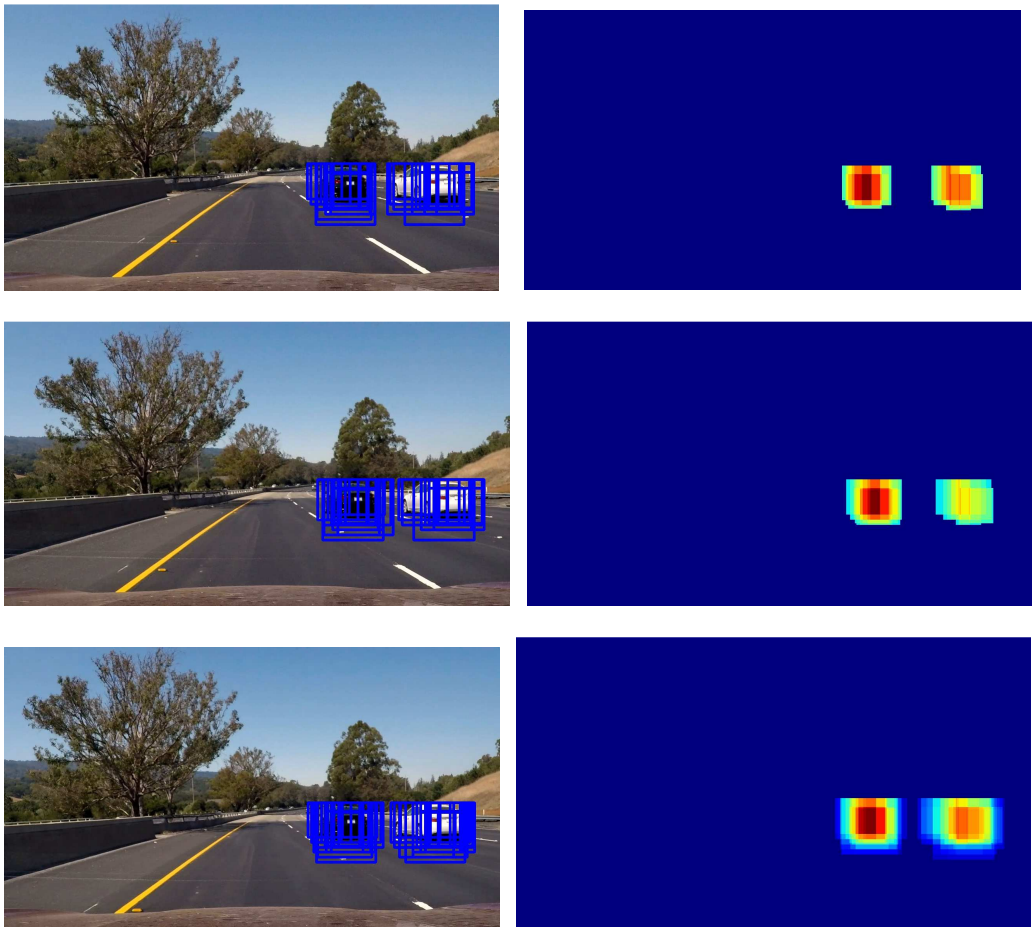
####1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.) Here's a link to my video result

####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 bounding boxes then overlaid on the last frame of video:

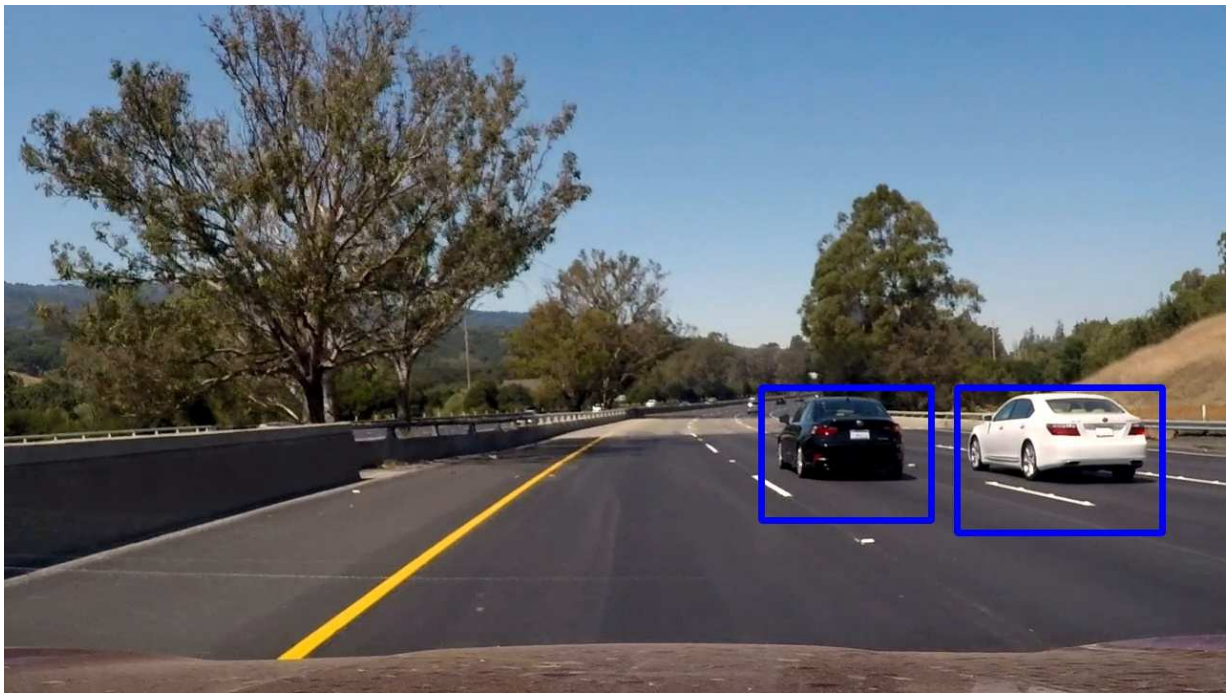
Here are six frames and their corresponding heatmaps:



Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames:

Here the resulting bounding boxes are drawn onto the last frame in the series:



###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?

I tried to apply all the best best practices as mentioned in the project tips and tricks. But still I would like to reduce the false positives by applying better filtering techniques.