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)

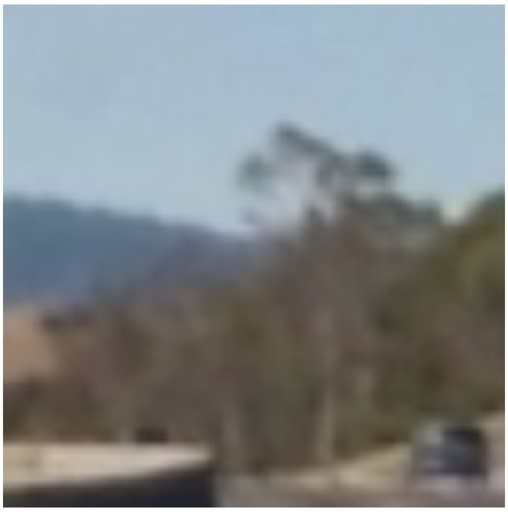
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is in `utils/vehicle_functions.py` file in `extract_features_image` function, lines 124 through 160. It's possible to use HOG of all channels or single channel of image (`VehicleDetectorOptions.hog_channel` class line 11). I use `skimage.feature.hog` in `get_hog_features` function lines 96 through 104 to compute HOG and other functions use this function.

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:

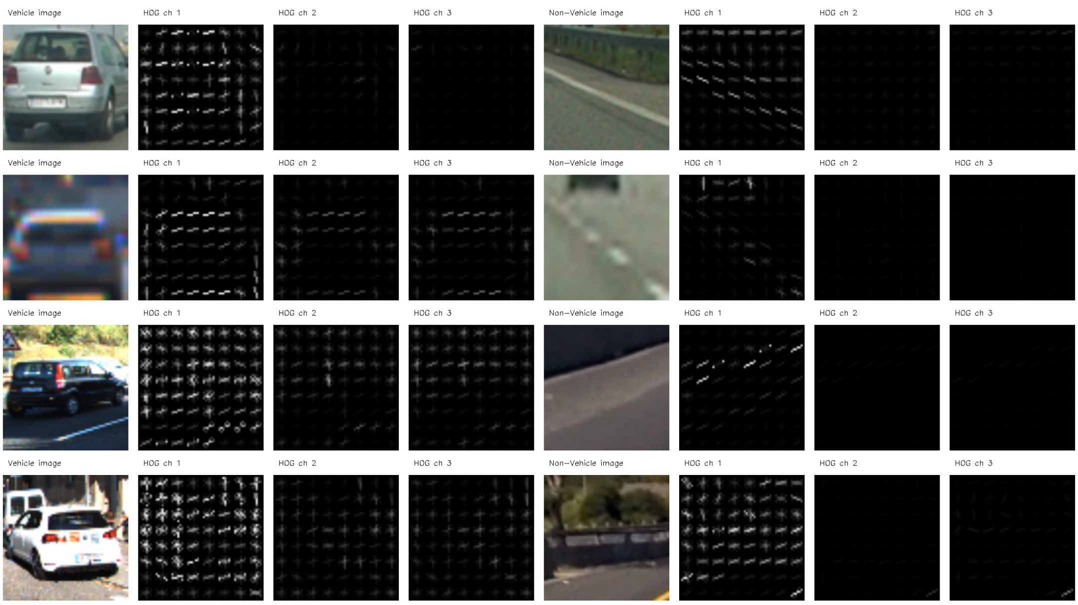
Vehicle sample

Non vehivle sample



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=11, 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 then trained the svm model for them as you can see in vehicle_detection.py file in hyper_train function, lines 72 through 92. Results can be seen in hyper-train-results.csv file and here you can see top 10 results ordered by test accuracy:

color space	orient	pix/cell	cell/block	channels	test accuracy
YCrCb	11	8	2	ALL	0.973254505
LUV	9	8	2	ALL	0.96875

YUV	8	8	2	ALL	0.968468468
LUV	11	8	2	ALL	0.966779279
LUV	10	8	2	ALL	0.965653153
YCrCb	10	8	2	ALL	0.965653153
YUV	10	8	2	ALL	0.964808559
YUV	11	8	2	ALL	0.964808559
YCrCb	9	8	2	ALL	0.964245495
LUV	8	8	2	ALL	0.961993243

Finally I chose 11 as HOG orientations, all channels for HOG and YCrCb as color space. Then retrain the svm using color histogram so I achieve 0.9809 test accuracy.

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).

The code for this step is in utils/vehicle_detector.py file in train method of VehicleDetector class, lines 25 through 67. Selection of HOG features and color features can be done using VehicleDetectorOptions class in utils/vehicle_functions file. I did these steps in train method:

1. Find all train images in vehicle and non-vehicle directories (Lines 27-32)
2. Extract features for the train images (Lines 35-38)
3. Stack vehicle and non-vehicle features in one numpy array (Line 40)
4. Normalize the features using StandardScaler (Lines 43-48)
5. Build label array with ones and zeros for vehicle and non-vehicle features respectively (Line 50)
6. Shuffle features and labels and then split them into train and test features and labels (Lines 53-59)
7. Build an instance of LinearSVC model and fit it on the train features. I used default model parameters (Lines 65-68)
8. Test the model on test features (Line 73-74)

Sliding Window Search

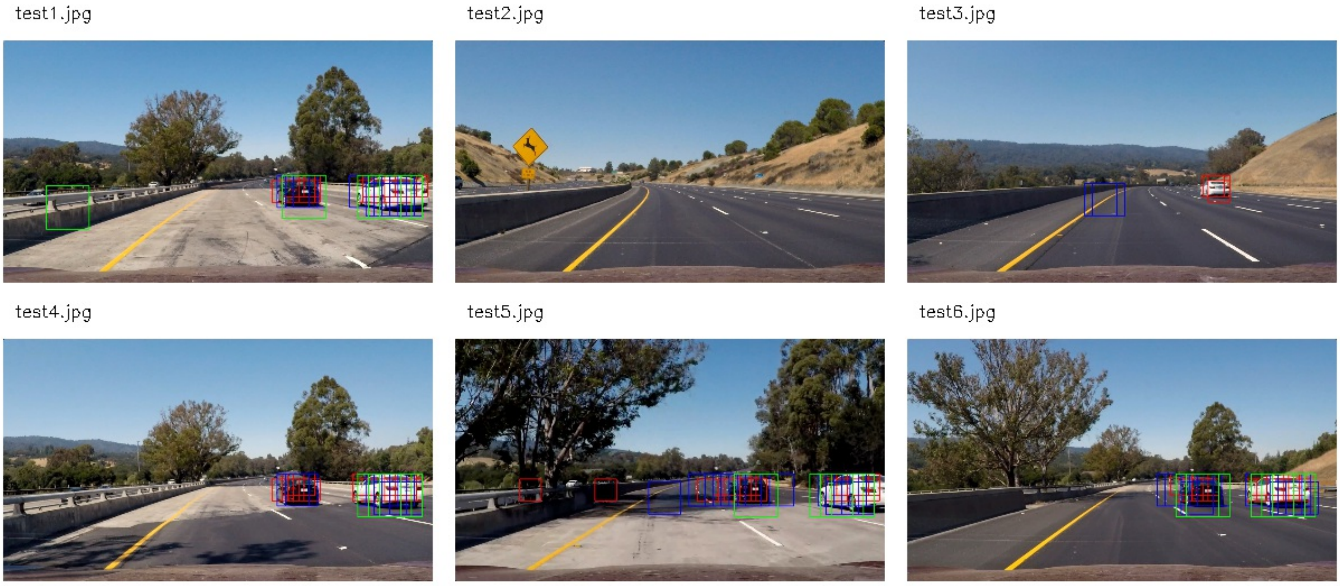
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The code for this step is in utils/vehicle_detector.py file in find method of VehicleDetector class, lines 91 through 147. I decided to up scale the image itself in different scales (1, 1.5, 2.0 and 3.0) and search with fix window size(64 pixels with 2 pixel overlap). The code for finding cars on a scaled image is in utils/vehicle_functions.py file in find_cars function, lines 178 through 260.

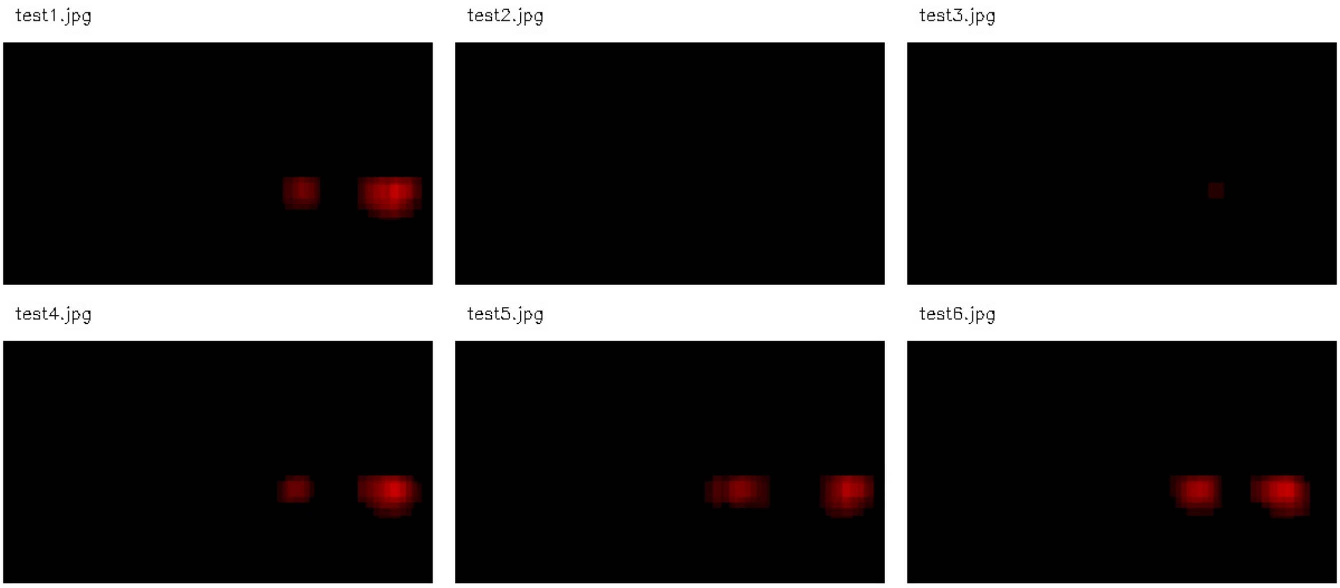
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on four scales of image using YCrCb color space, all channels HOG features plus histograms of color in the feature vector, which provided a nice result. Here are pipline example images:

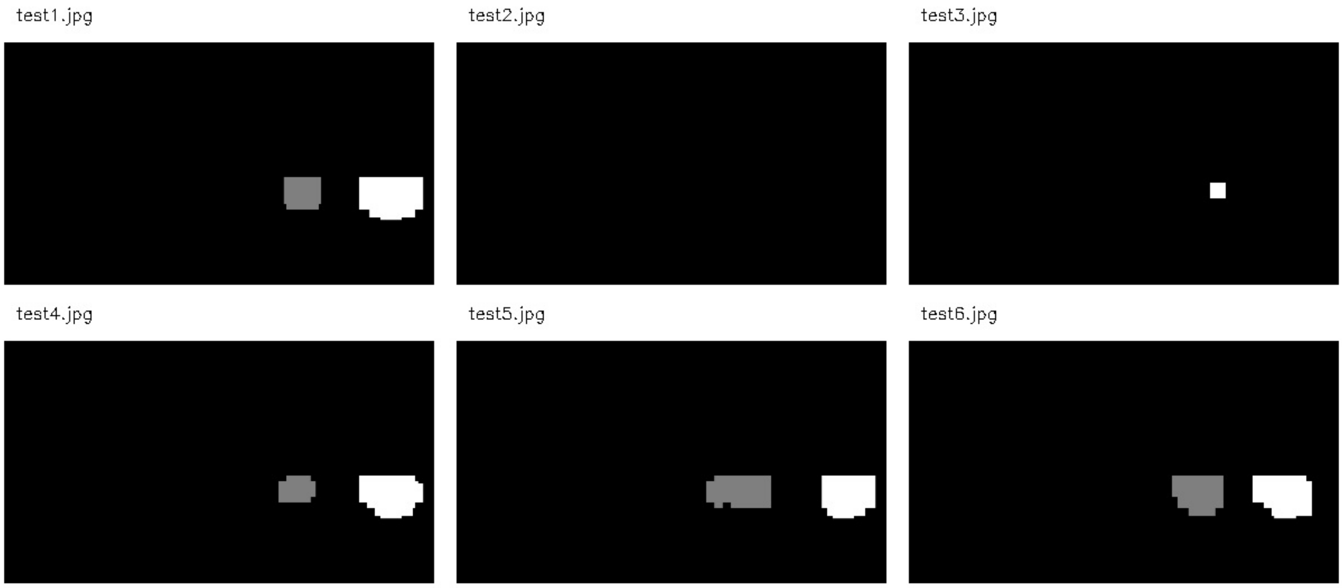
1. Find cars in multi-scale image (utils/vehicle_detector.py, VehicleDetector.find, lines 94-110)



2. Build heatmap (utils/vehicle_detector.py, VehicleDetector.find, lines 111-123)



3. Find connected components using skimage.measure.label (utils/vehicle_detector.py, VehicleDetector.find, lines 124-135)



4. Find final bounding box (utils/vehicle_detector.py, VehicleDetector.find, lines 136-147)

Video Implementation

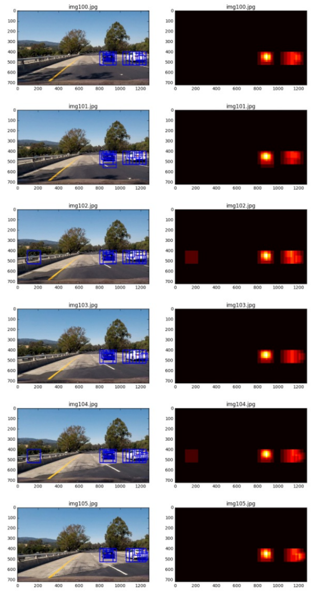
####1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.) Here's a link to my video result

####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.

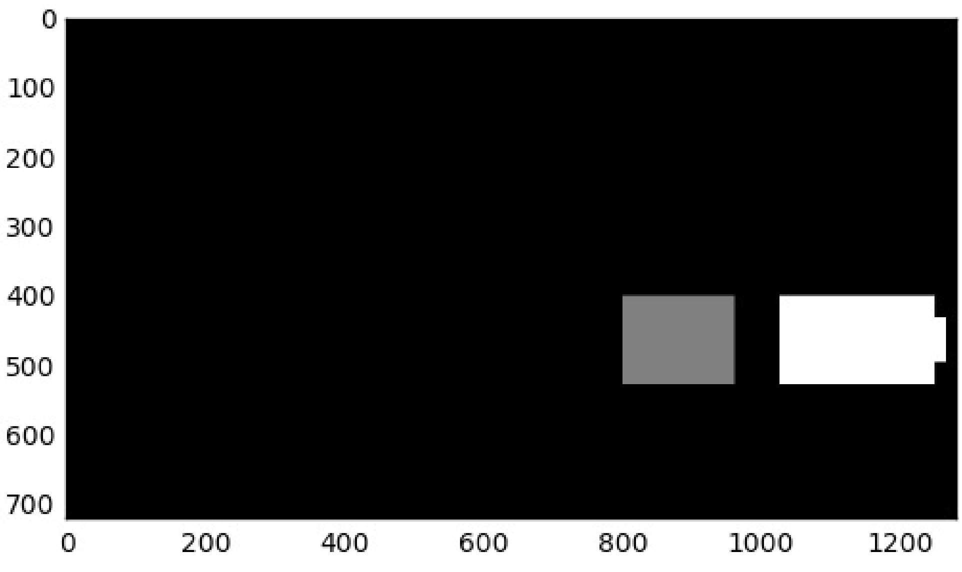
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. I then assumed each blob 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?

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.

There is another place in this file in find_car function lines 150 through 160 where I extract HOG from multi-scaled image.