Vehicle Detection Project

The goals / steps of this project are the following:

1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
2. 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.
3. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
4. 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.
5. Estimate a bounding box for vehicles detected.

Provide a Writeup / README that includes all the rubric points and how you addressed each one.

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.

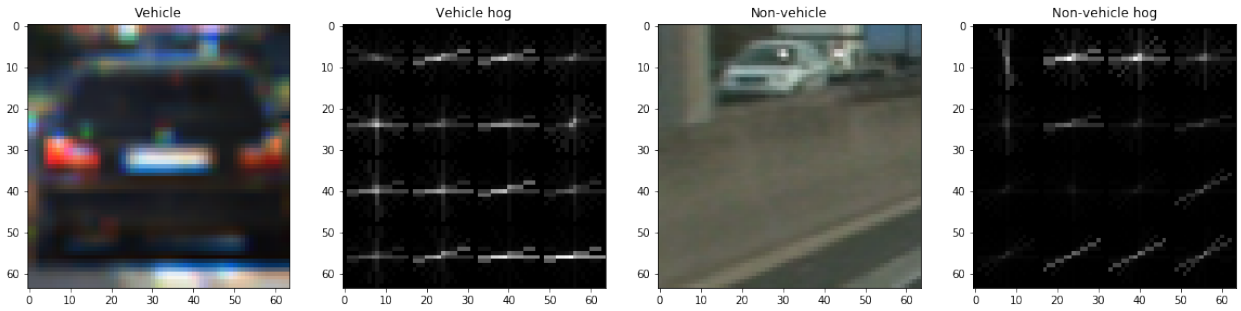
The code for this step is contained in the section named **"PROFILE DATA"** of the IPython notebook
I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

Vehicle

Non-vehicle

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=12`, `pixels_per_cell=(16, 16)` and `cells_per_block=(2, 2)`:



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

I started by fixing **cells_per_block = (2, 2)** since that's seemed to be standard. Then I started with color spaces (**HSV, LUV, HLS, YUV, YCrCb) (except RGB)** since by keeping **orientations = 8** & **pixels_per_cell = 8**. Each time I recorded accuracy which was well over 98% BUT the accuracy of the classifier is judged on using the *sliding windows function* (explained below) on the test images and its ability to detect cars ONLY. I tried various combinations of parameters and settled on the **YCrCb** color space which yielded better results than other color spaces with orientations values = **12** and **pixels_per_cell = 16**. Using all the Hog channel was also found to be better than just using one of the channels.

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 color features as well as HOG features. Using 32 spatial bins and HOG features specified above. The classifier used the labeled data for vehicle and non-vehicle examples but because of the time-series images present, it did not yield satisfactory results. Hence, I downloaded the smaller subset of the vehicle data given in the lesson and appended the data to the existing dataset. It improved the results. I think the quality of the data in the smaller subset is better.

Sliding Window Search

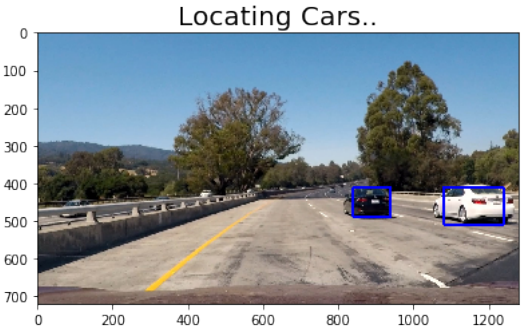
Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

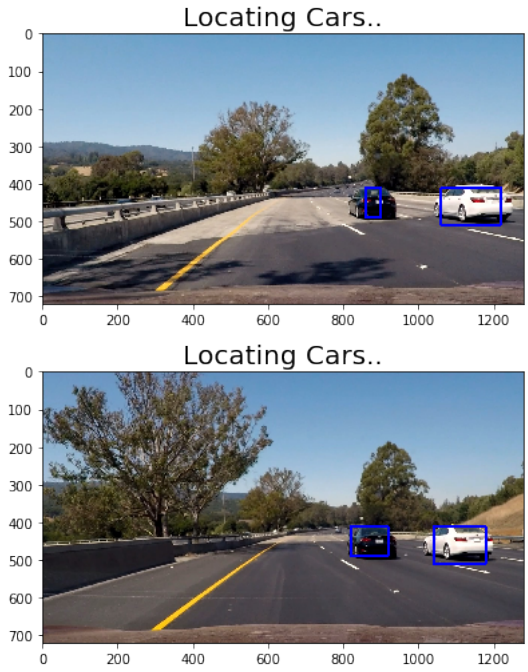
I decided to start with 50% overlap and (64, 64) windows size (since the images we have trained on were 64x64). The window we decided to search between the Y values = (350, 656). This skips the top half of the image and only search a portion of the image. Then I tested on 6 given test images. Gradually, increasing the overlap from 50 – 85%

to get the best results. I kept the scales constant at 1.25. There were many false positives still.

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

I finally arrived at the following results. This was made possible by using heat-map from detections in order to combine overlapping detections and remove false positives. The threshold was set to 3.





Video Implementation

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.)

In the submission folder titled **project_video_op.mp4**

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.

The same technique used to remove false positives on the images was used in the video. However, to account to false positives threshold was increased to 5.

Discussion

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.

1. The pipeline still was identifying the cars on the left-hand side (cars coming on the other way). We can perhaps put a X axes filter (along with Y axes filter)

2. Cars moving further away was not detected as well. It could be because of the scaling. Perhaps we can implement dynamic scaling to handle this.

3. Last but not the least, we can start eliminating the time-series natures of the vehicle dataset to only select quality images for classifier

EXTRAS

I took the video from PROJECT 5 and fed the same to pipeline function from PROJECT 4 to combine the 2 projects together.