Vehicle detection project

Udacity Self-Driving Car Nanodegree

Goals:

- 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.

Data exploration:

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:

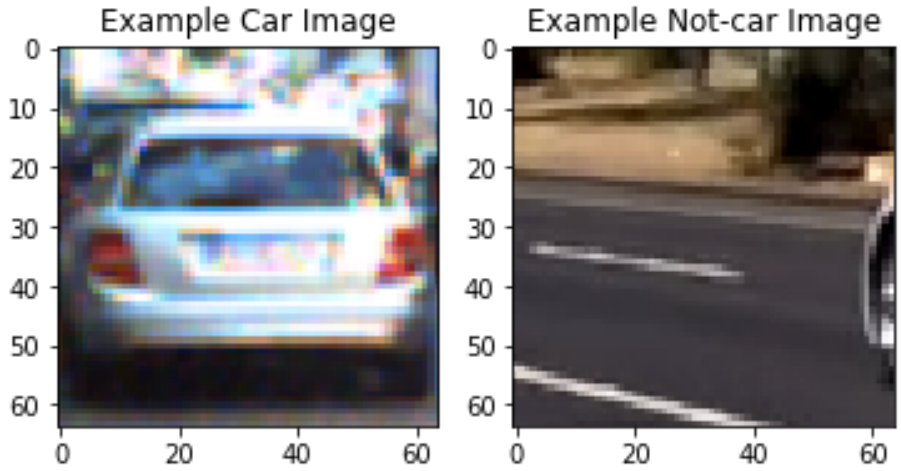

Illustration 1: Example of car and not-car of the dataset

Histogram of Oriented Gradients (HOG):

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

I 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)`:

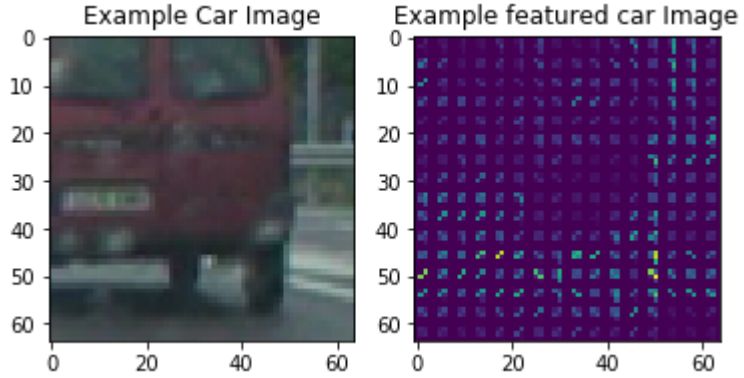

Illustration 3: HOG features of a car

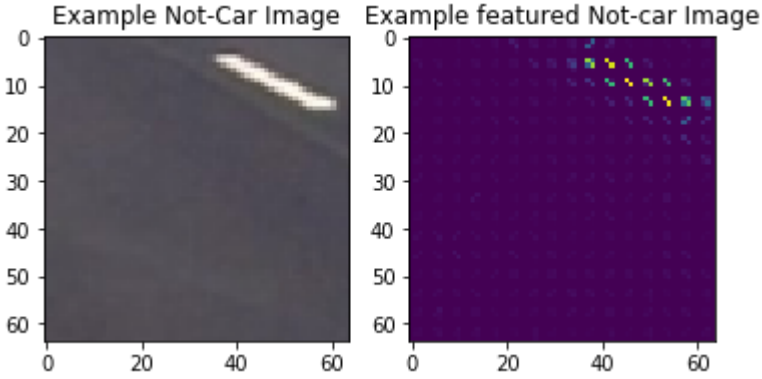

Illustration 2: HOG features of a non-car

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

The decision was based on try/error, I tried a lot of combinations and, at the end, I decided that YCrCb with that parameters was the best (accuracy/time)

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

I trained a linear SVM (4th cell in the Ipython notebook) with all the data processed, I split the data into test and training sets and I got a 99% of accuracy in the test set

4. Describe how you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I decided to search with three sliding-window all over the middle-bottom part of the image because I was using 8 * 8 pixels per cell, that was finding quite good the cars in different sizes, I decided to overlap 0.8 * 0.8 every window to find more hot windows and be able to play more with the heat threshold to reduce false positives

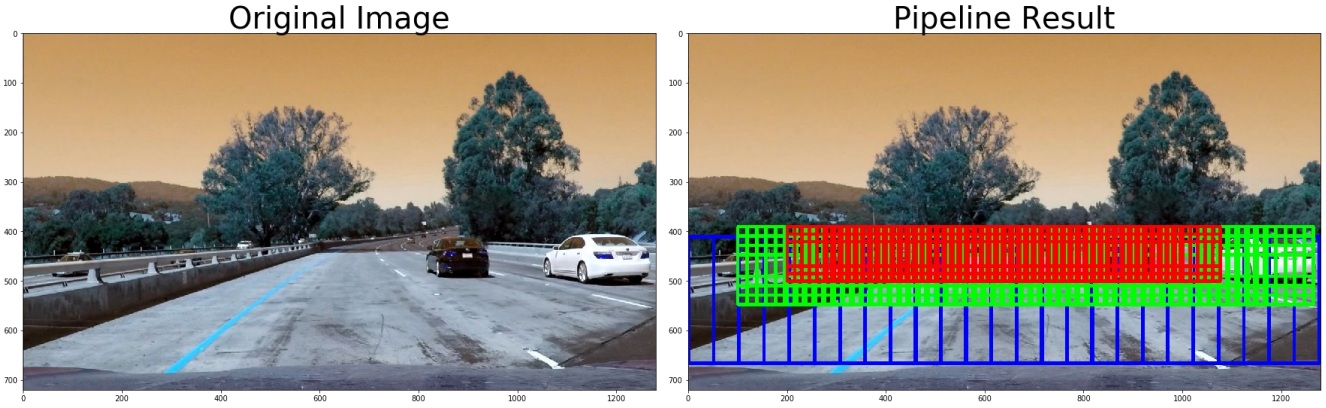

Illustration 4: All the sliding windows

The first possibility was to only one sliding-window, but that was not giving good results and often there was only one or two windows in the true positive, like this:

Illustration 5: With only one sliding window

With three sliding-window I had much more windows over every car.

Illustration 6: With three sliding-windows

At that point I decided that was better to do a better job with a lot of positives than tunning more the parameter in the dataset.

5. 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 is project.mp4, in the same folder as this writeup.

6. 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.

For dealing with false positives I used a heat map, because of the big number of positives windows obtained with three sliding-window I decided to tune the threshold parameter and increase it to 3.

8. 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?

There are no false positives in the video, but sometimes the car is not tracked and I think that it can be improved by tunning a bit the HOG parameters and by reducing the check frequency, (It looks for cars every 20 frames if a car is in the image, and 5 frames if there are no cars in the image) but changing those parameters will increase the time of the pipeline, so I decided to let it that way.