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
- Apply a color transform and append binned color features, as well as histograms of color, to the HOG feature vector.
- For these first two steps features were normalized and a selection was randomized for training and testing.
- Implement a sliding-window technique and use trained classifier to search for vehicles in images.
- Run a pipeline on a video stream (starting with the test_video.mp4 and later implementing 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

Rubirc points are below considered and I describe how each was addressed the project implementation.

Writeup

Histogram of Oriented Gradients (HOG)

1. The code for this step is contained in the code cells 1, 3, 4 and 7 of the IPython notebook Vehicle-Detection.ipynb

All the `vehicle` and `non-vehicle` images were read in. Following is an example of one of each of the `vehicle` and `non-vehicle` classes; see "Car and not car" figure.

Different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`) were explored. Random images from each of the two classes were picked and displayed to get a feel for what the `skimage.hog()` output looks like.

Following is an example using the `YUV` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cell_per_block=2`; see "HOG example" figure.

I tried different color spaces, HOG parameters (orientations, pixels_per_cell). I found that the parameterss above worked well.

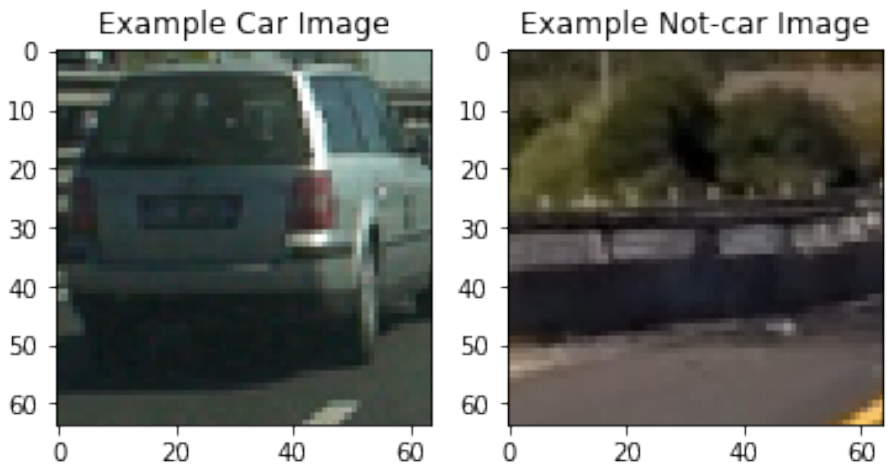

Figure 1: Car and not car

2. I trained a linear SVC classifier using my selected HOG and color features, see code cell 7, and scaled the features.

I divided the data after shuffling randomly into a 70% trainining set and a 30% test set. I thought that a larger test set than 20% would avoid overfitting, and that worked better in identifying less false positives.

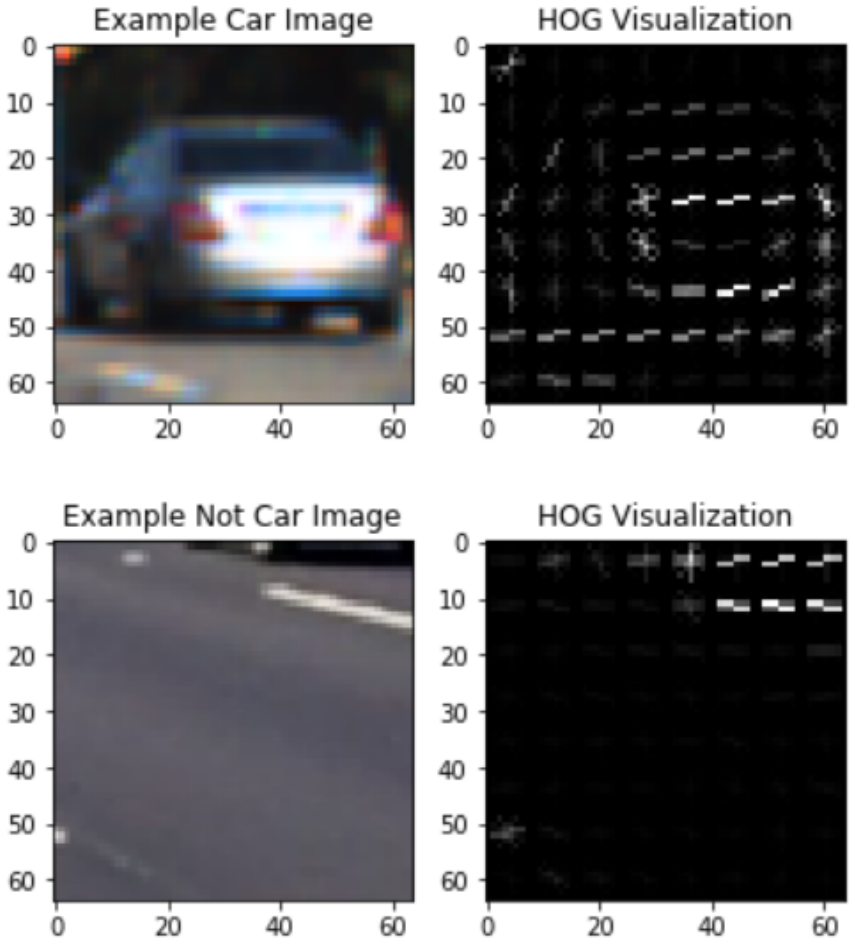

Figure 2: HOG example

Sliding Window Search

1. The code for this step is contained in the code cells 1, 3, 4, 6 and 8 of the IPython notebook Vehicle-Detection.ipynb.

I implemented a sliding window search using HOG-subsampling. My find_cars function extracts hog features once and then gets sub-sampled for the overlaying windows. A cells_per_step of 2 gave a search window overlap of 75%. I tried scale = 1.5 to 2 to search and decided on scale = 1.5 as higher values made it harder to find cars in the distance. I tried pixels_per_cell at 4 but decided on the default 8 when it didn't seem to make a difference, all other parameters being equal; see "sliding windows" figure.

Figure 3: sliding windows

2. Here are some examples of test images showing how the pipeline works.

To optimize the performance of my classifier, I used a higher ratio of test vs. training with test at 30%, using the large cars and noncars sets from Udacity.

I searched using YUV and all 3 HOG channels, with spatially binned color and histograms of color in the feature vector, which worked well to find cars and gave fewer false positives; see "sliding window" figure.

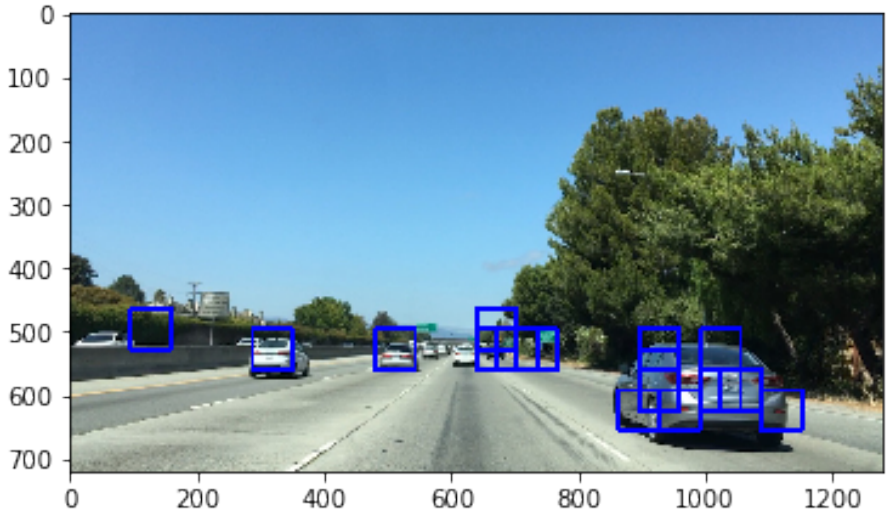

Figure 4: sliding window

Video Implementation

1. The pipeline in code cells 9 and 10 was run on the entire project video.

The result gives a bit of wobble and some instability of the bounding boxes, but identifies the vehicles most of the time with almost no false positives.

Link to video result.

2. I implemented the heat map filter for false positives and combined overlapping bounding boxes using labels, see cell 9.

The code keeps track of the positions of positive detections in each frame of the video using the most recent 16 bounding boxes, and a heatmap is created, then thresholded to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify separate regions from the heatmap, each corresponding to a vehicle. Finally the code constructs bounding boxes to cover the area of each region detected.

Below is an example giving the heatmap from a bunch of successive frames of video, the regions found from `scipy.ndimage.measurements.label()` and the drawn bounding boxes in the last frame of the series; see "border boxes and heat," "labels map" and "output border boxes" figures.

Figure 5: border boxes and heat

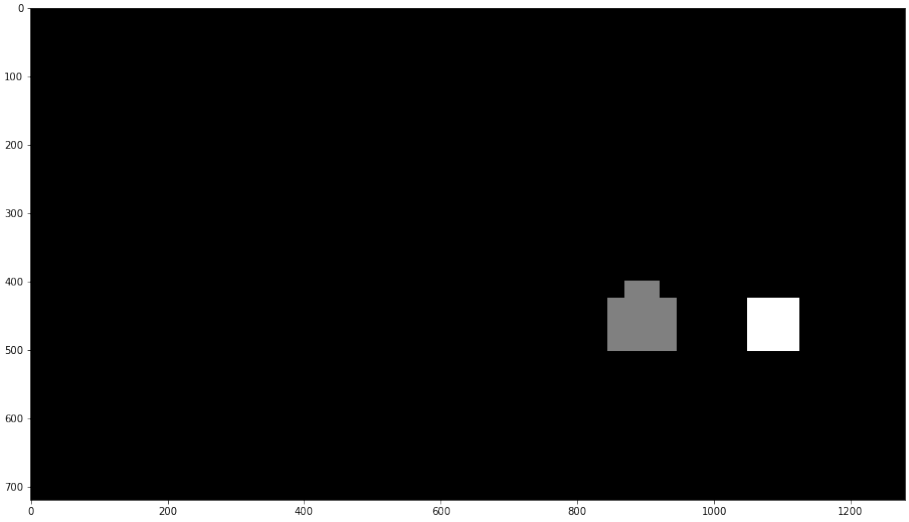

Figure 6: labels map

Figure 7: output border boxes

Discussion

In this project there were a number of parameters to tweak, and that made it somewhat long to try and cover them all.

My pipeline failed sometimes when it identified false positives that kept coming back in the same spot in successive or alternate frames, which was then not caught with heatmap thresholding.

Changing to another color space, using a higher heat map threshold and enough testing samples at 30% for the classifier fixed that for this video. The pipeline had more difficulty when cars were partially visible as they drive into view, when shading conditions changed and when cars were side by side, identifying them as overlapping.

I might improve this project further by using different parameters according to shading conditions and more training samples of cars and not cars, especially for the false positives.