Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

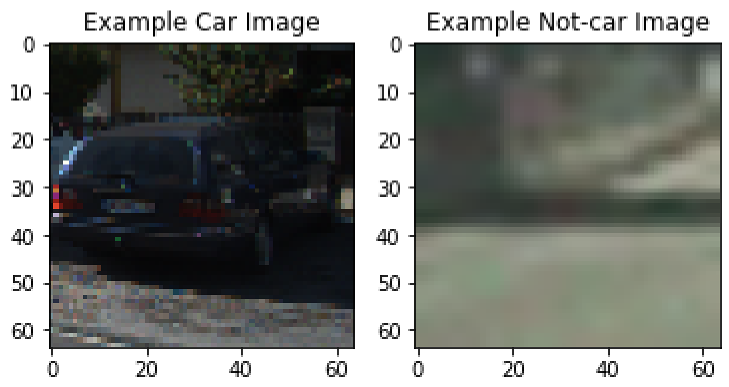
You're reading it! All of the code for the project is contained in the Jupyter notebook ·Vehicle*Detection*Code.ipynb`

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 'HOG Feature' code cell of the IPython notebook

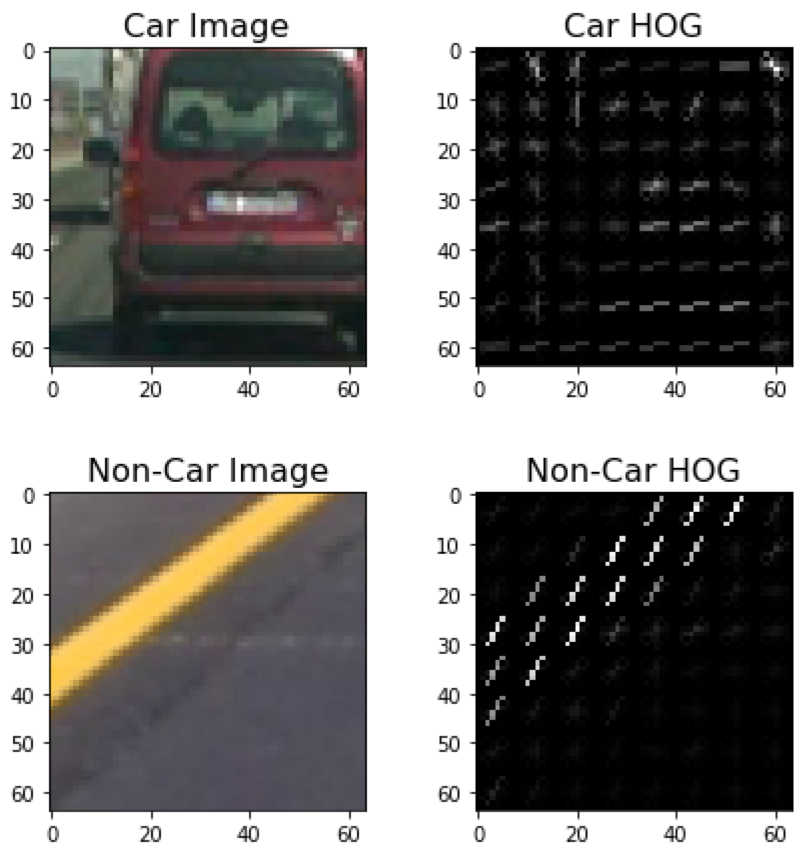
First of all, I loaded all of the `vehicle` and `non-vehicle` image paths from the provided dataset. The figure below shows a random sample of images from both classes of the dataset.



The method `extract_features` in the section 'Combine Color and HOG Features' accepts a list of image paths and HOG parameters, then it calls `img_features` function, which process a single image. Finally, `extract_features` produces a flattened array of HOG features for each image in the list.

Next, I define parameters for HOG feature extraction and extract features for the entire dataset. These feature sets are combined and a label vector is defined (1 for cars, 0 for non-cars). The features and labels are then shuffled and split into training and test sets in preparation to be fed to a linear support vector machine (SVM) classifier.

Here is an example using HOG parameters of `orientations=9`, `pixels_per_cell=8` and `cells_per_block=2`:



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

I tried various combinations of parameters and I finally choose the following parameters:

`spatial_size = (32, 32)` `hist_bins = 32` `cspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb` `orient = 15` `pix_per_cell = 8` `cell_per_block = 2` `hist_range = (0, 256)` `hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"`

I mainly focused on accuracy, but also I take into consideration the speed at which the classifier is able to make predictions. Then if the detection pipeline were not performing satisfactorily, I come back and tweak the parameters.

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).

A `LinearSVC` was used with default classifier parameters as a classifier. The training process was included in the section titled "Extracting Features". I used HOG features, spatial intensity features and channel intensity histogram features all together. By tuning the parameters above, the length of feature vector changes. The longer the feature vector is the more features available for classifier to use, usually this means better accuracy. However, if too much features are available may also cause overfitting and the runtime of extracting features will also increasing significantly. As a result, I choose chosen the parameters I think give a balance between accuracy and runtime.

I also performed normalization of the features and random shuffling of data before training classifier to improve the robustness and performance of the classifier.

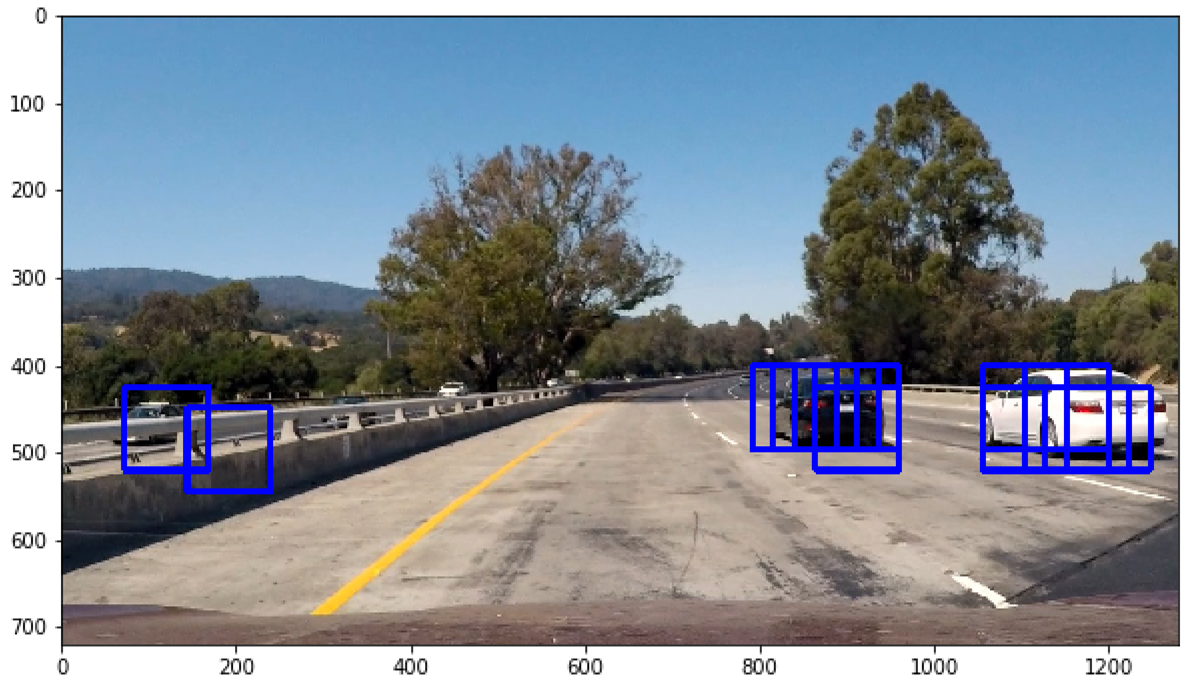
My classifier was able to achieve a test accuracy of 98.11%.

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?

In the section 'Hog Sub-sampling Window Search', I adapted the method `find_cars` from the course material. As mentioned in the class, The method combines HOG feature extraction with a sliding window search, but rather than perform feature extraction on each window individually which can be time consuming, the HOG features are extracted for the entire image (or a selected portion of it) and then these full-image features are subsampled according to the size of the window and then fed to the classifier. The method performs the classifier prediction for each window region and returns a list of rectangle objects corresponding to the windows that generated a positive ("car") prediction.

The image below shows the first attempt at using find_cars on one of the test images, using a single window size:



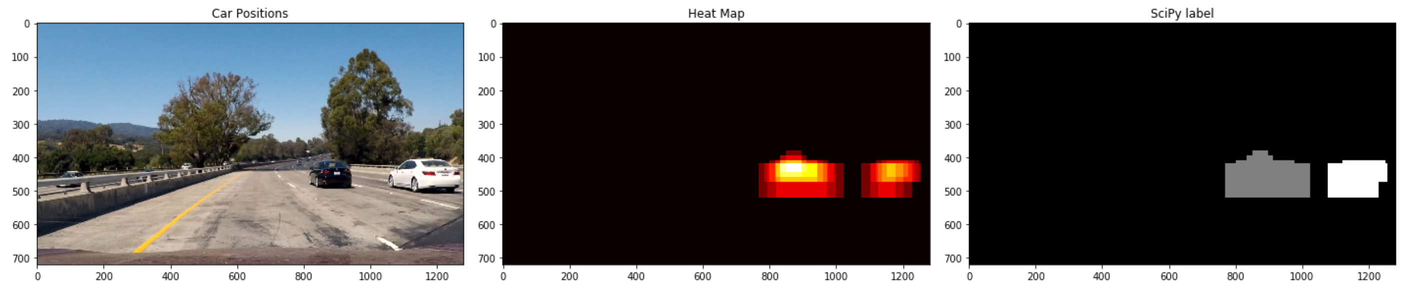
In section 'Multiscale Search' I applied `find_car` with different scale and Ystart and Ystop several times. In 'Show All Potential Search Areas' I explored different combinations of these parameters, and test on different test images. The main strategy is smaller range for smaller scales to reduce the chance for false positives in areas where cars at that scale are unlikely to appear.

Finally I choose the following list of parameters:

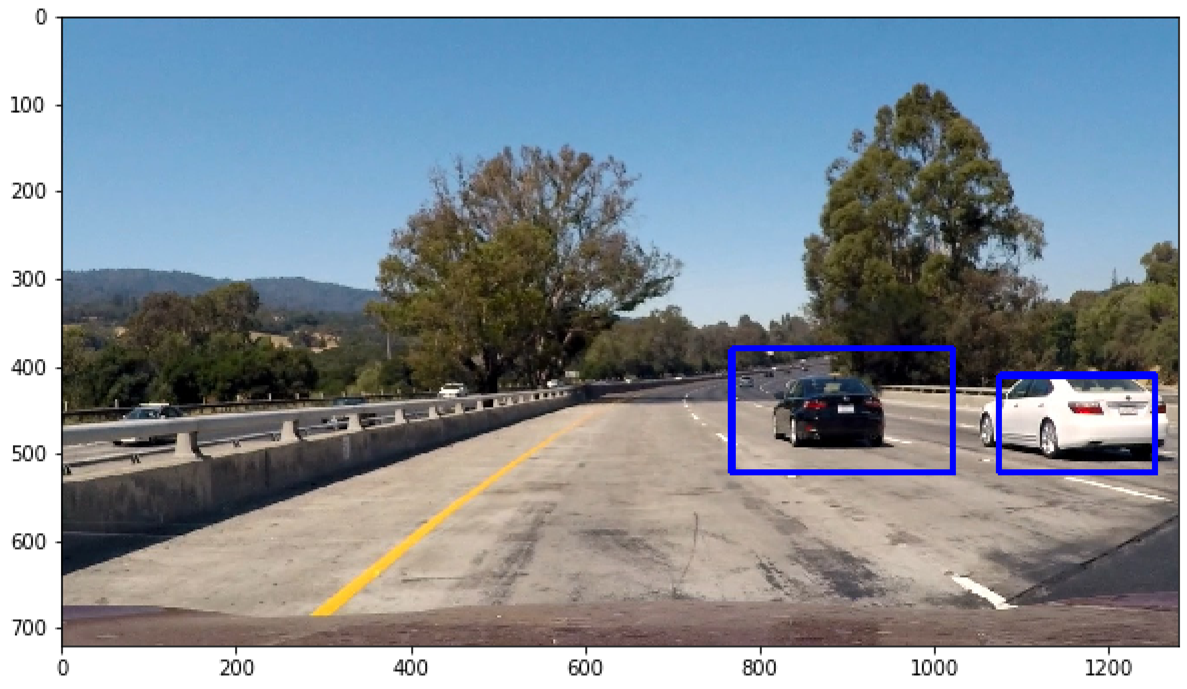
`ystart_list = [380, 410, 420, 430, 400, 500]` `ystop_list = [490, 490, 556, 556, 556, 656]`
`scale_list = [1.0, 1.0, 1.6, 2.0, 2.2, 3.0]`

Then in section 'Multiple Detections & False Positives' I applied Heatmap with threshold to improve the performance. `add_heat` and `apply_threshold` functions are adapted from course material to reduce false positives. The add_heat function increments the pixel value (referred to as "heat") of an all-black image the size of the original image at the location of each detection rectangle. Areas encompassed by more overlapping rectangles are assigned higher levels of heat.

The following image is the resulting heatmap from the detections in the image above:



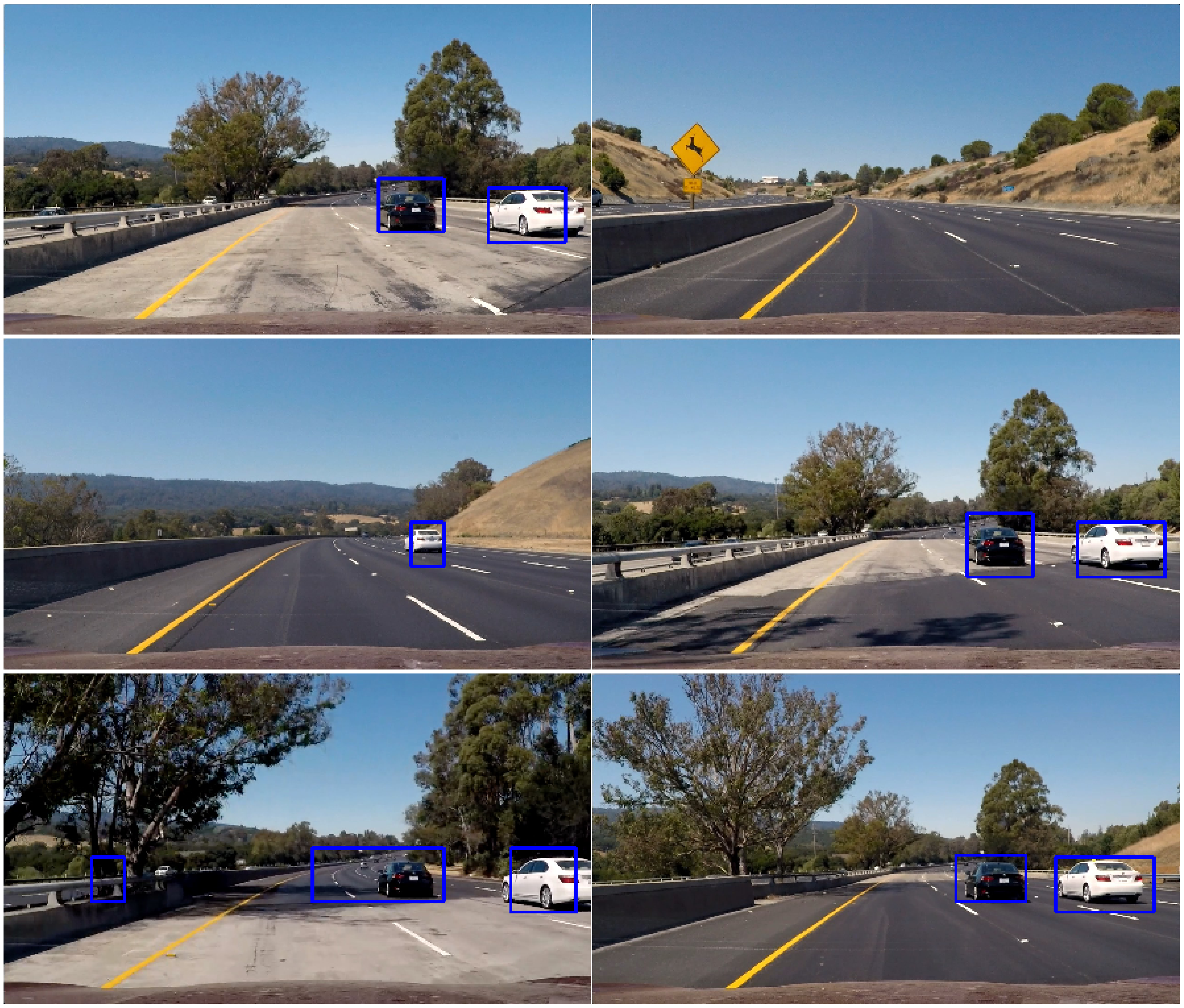
And the final detection area is set to the extremities of each identified label:



Compared with the result in the previous section, we can see that at first one car on the other side of the road and one false positive were detected at first, then after applied heatmap which was then thresholded, the false positive detection was removed.

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?

he results of passing all of the project test images through the above pipeline are displayed in the images below:



Note that There is still false positive detection on test5.jpg. However, I tried different paramters. This is still the best result I got so far.

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.

The code for processing frames of video is contained in the cell titled "Define Final Process Pipeline" and is identical to the code for processing a single image defined in `process_img` using methods described above.

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?

The problem I faced during this project were mainly concerned with the detection accuracy. With different color_space and `extract_feature` parameters, the accuracy of the classifier is usually always higher than 95%. But the classifiers performance after applied sliding window search varies a lot. The sliding window parameters also play an significant role in preventing false positives.

The pipeline is probably most likely to fail in cases where vehicles (or the HOG features thereof) don't resemble those in the training dataset, but lighting and environmental conditions might also play a role. As we can see in the final result of image 5, some area of shadows were also detected as cars.

I believe that if time permits, more time need to be spent on combining a high accuracy classifier with robust search windows.