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.

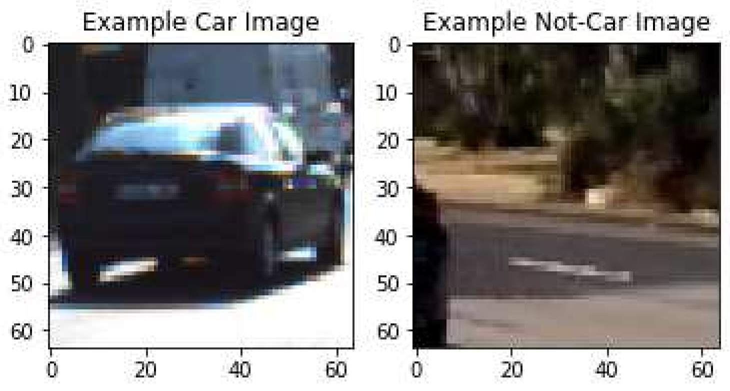
Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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 first code cell of the IPython notebook.

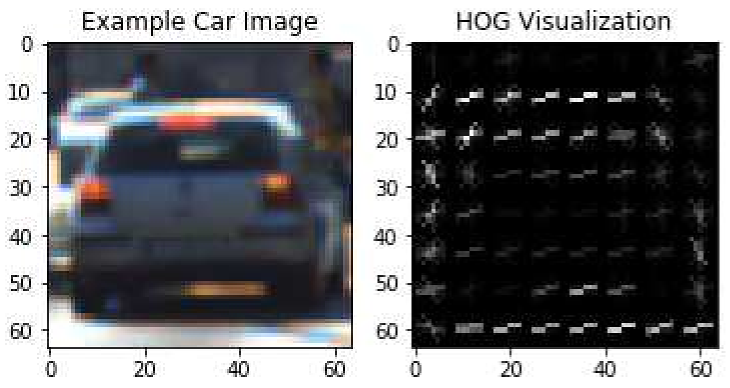
I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:



Note: for the "car" images, I decided to use the "KITTI_extracted" data set which I minimally augmented by adding around 20 images of cars extracted from the project video's frames / images. I'm not sure whether in the end it improved my SVC classifier, but I thought it's worth trying.

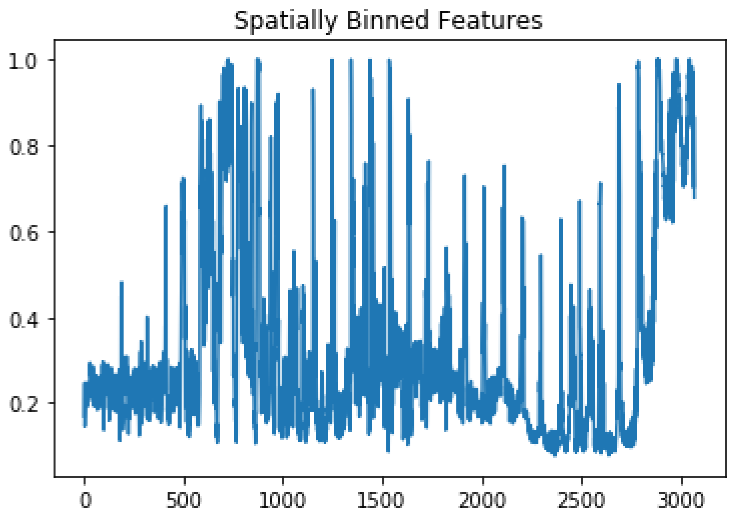
I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like.

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

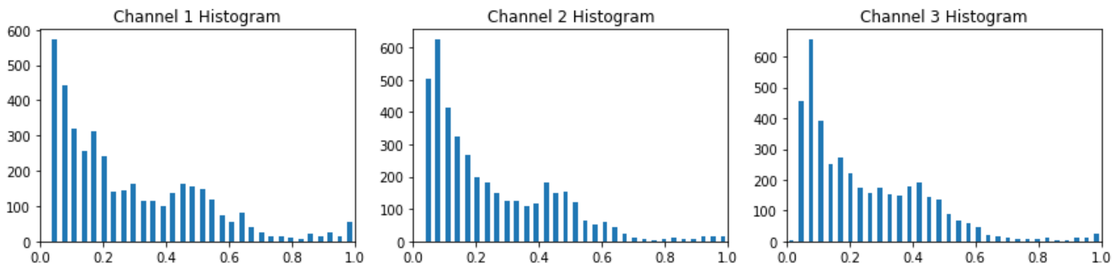


I also defined functions for binning color features and extracting color histograms.

Example of binned color features:



Example of color histogram features for each image channel:



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

I tried various combinations of parameters and finally settled for the following setup which yielded the highest SVM accuracy (99.75%):

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (8, 8) # Spatial binning dimensions
hist_bins = 8    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
```

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 where the above parameters were tweaked and SVM was trained is contained in the code cell #11 of the IPython notebook.

Here are the final SVM training results:

```
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 5508
59.64 Seconds to train SVC...
Test Accuracy of SVC =  0.9975
```

Sliding Window Search

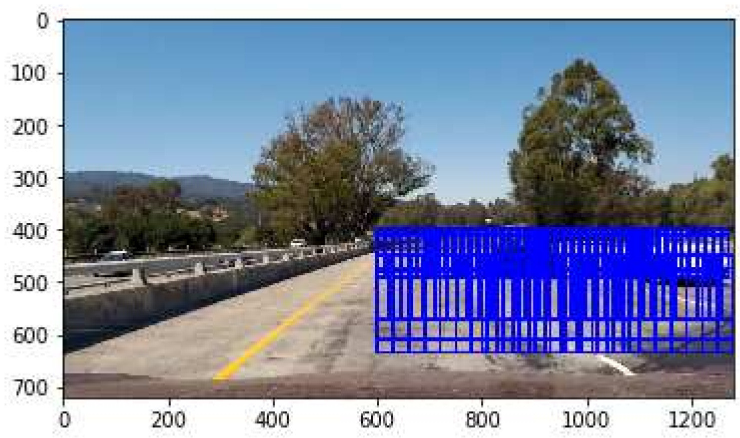
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The sliding window search is implemented in the functions slide_window(), draw_boxes() , single_img_features() and search_windows().

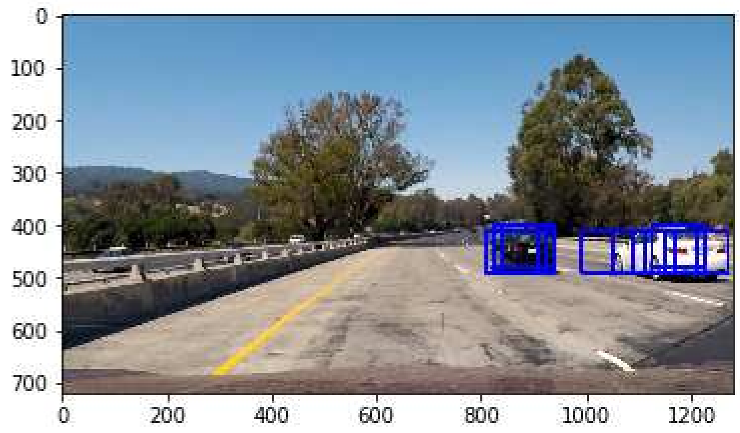
I settled for the following approach:
- Search only the bottom right "quarter" of the image by setting y_start_stop = [450, 656] and x_start_stop = [600, 1280] (region of interest).
- Search the top part of the region of interest with small windows (100x80 px)
- Search the middle part of the region of interest with mid-size windows (150x120 px)
- Search the bottom part of the region of interest with large windows (200x160 px)

Sample results can be seen below:

```
Test image - showing all windows that were searched:
```

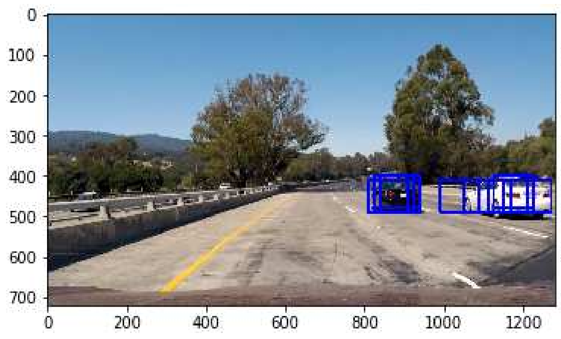


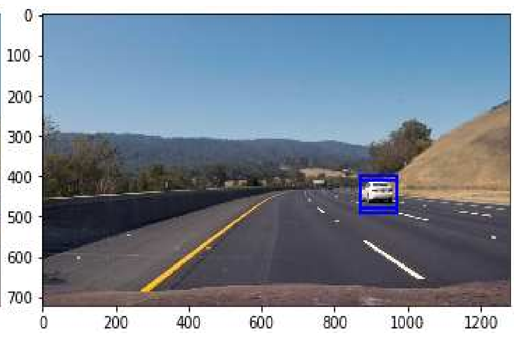
...and the result:

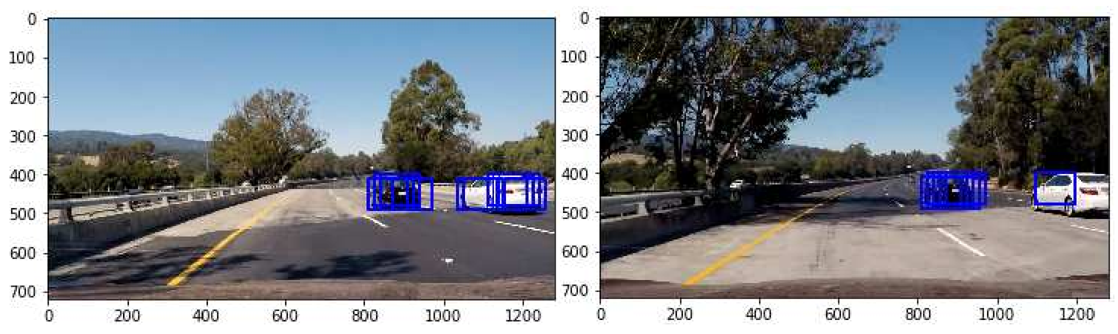


2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

As mentioned above, ultimately I searched on three scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:







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.)

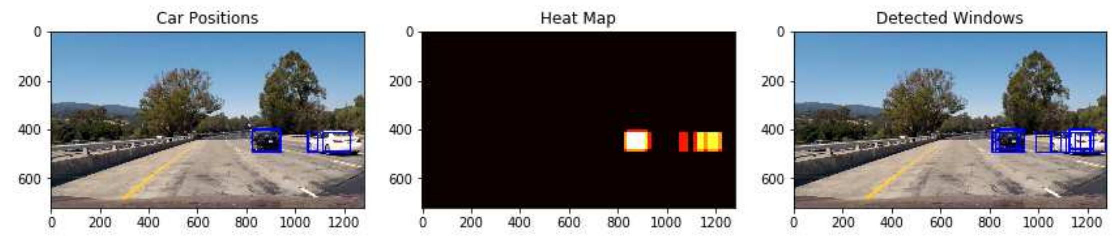
The video is attached to the submission (project_output_video.mp4)

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.

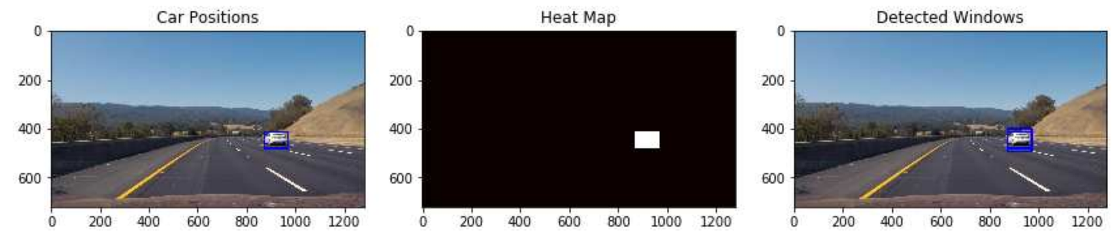
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. I then assumed each blob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

Here's an example result showing the heatmap:

```
****************
Heat map test result:
```



```
****************
Heat map test result:
```



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?

My pipeline detects the black car quite reliably, but it's weak point is the detection of the white car. The bounding boxes around the white car are a bit instable. In order to improve on this issue, I would need to investigate to which the "problematic" detection of the white car can be attributed to:
a) Training data (too few white car pictures in the training set)?
b) My parameters for training SVC?
c) Could different scaling and overlapping of sliding windows yield an improvement?

Since testing all these possibilities would be very time-consuming, I'm leaving this investigation for a later point in time.