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.

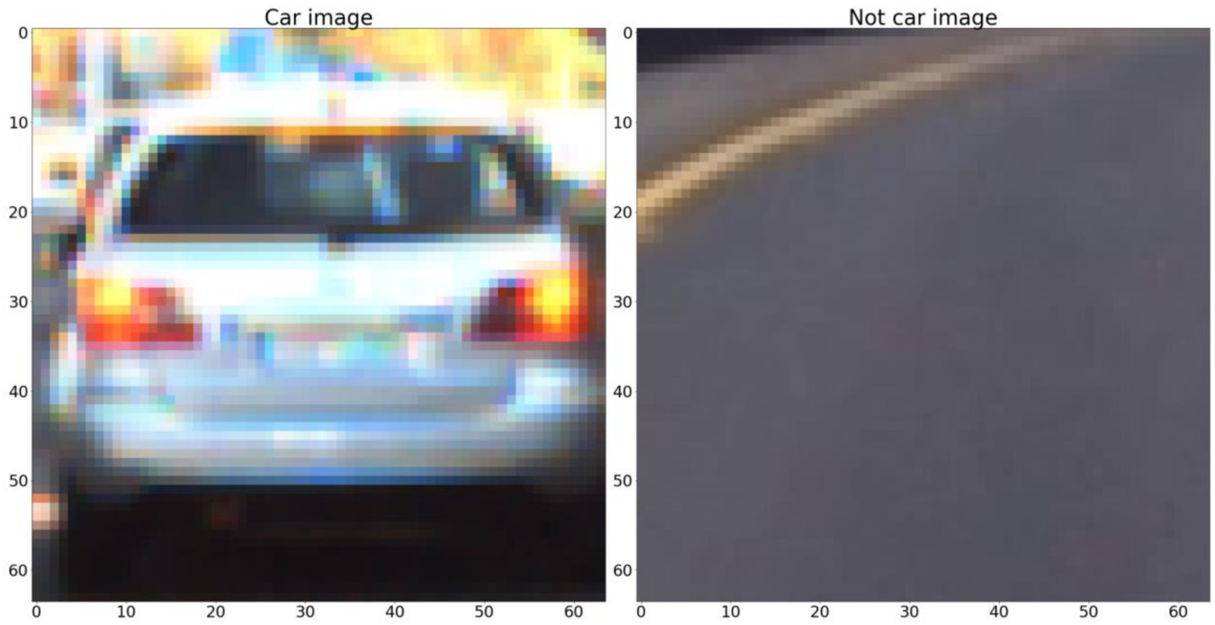
You're reading it!

###Histogram of Oriented Gradients (HOG)

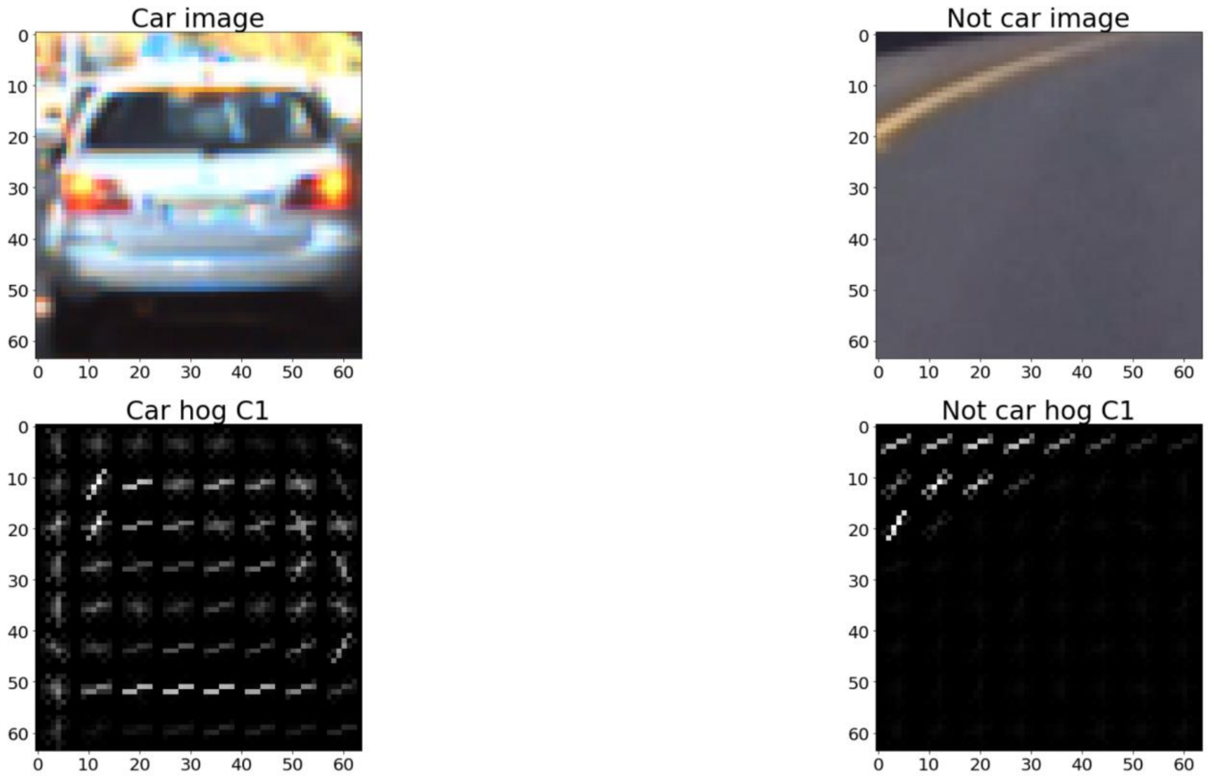
####1. Explain how (and identify where in your code) you extracted HOG features from the training images.

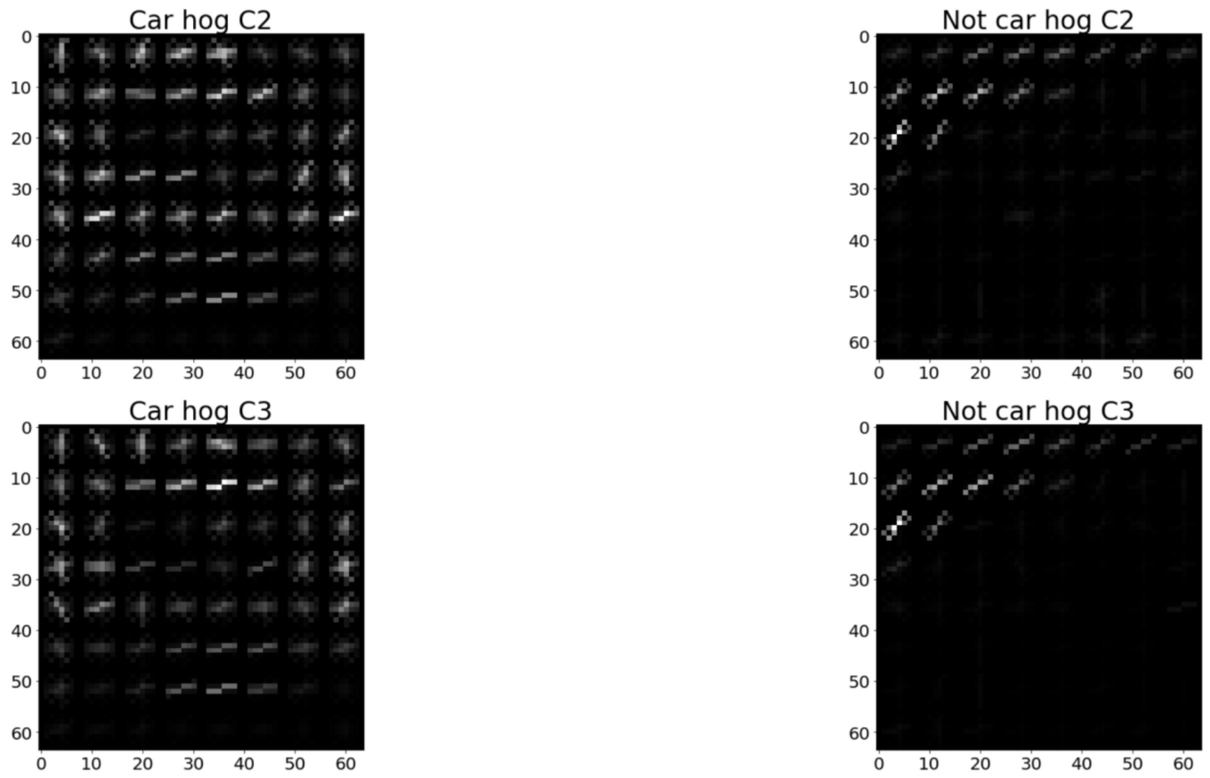
The code for this step is contained in the second code cell of the IPython notebook. The function is defined as:
get_hog_features(img, orient, pix_per_cell, cell_per_block, vis=False, feature_vec=True)

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:



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=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:





Car hog C2

Not car hog C2

Car hog C3

Not car hog C3

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

I tried various combinations of parameters and then chose the ones that gave the best accuracy :

```
color_space = '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 = (16, 16) # Spatial binning dimensions

hist_bins = 16    # 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
```

Which lead to the Test Accuracy of SVC = 0.9925

####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).

I trained a linear SVM using the HOG features, the histogram features and the spatial features.

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

I split up the data into randomized training and test sets using the train_test_split() function, then I created a linear SVC using the LinearSVC() function and trained it using svc.fit(X_train, y_train).

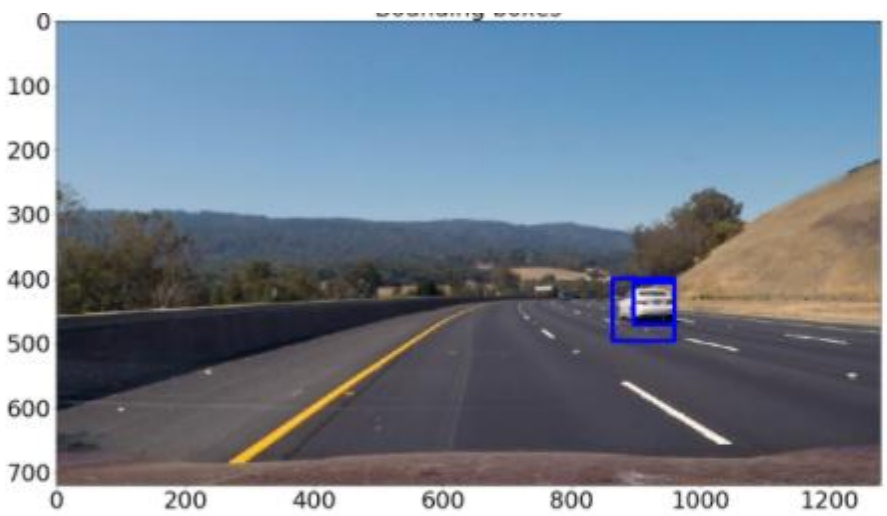
###Sliding Window Search

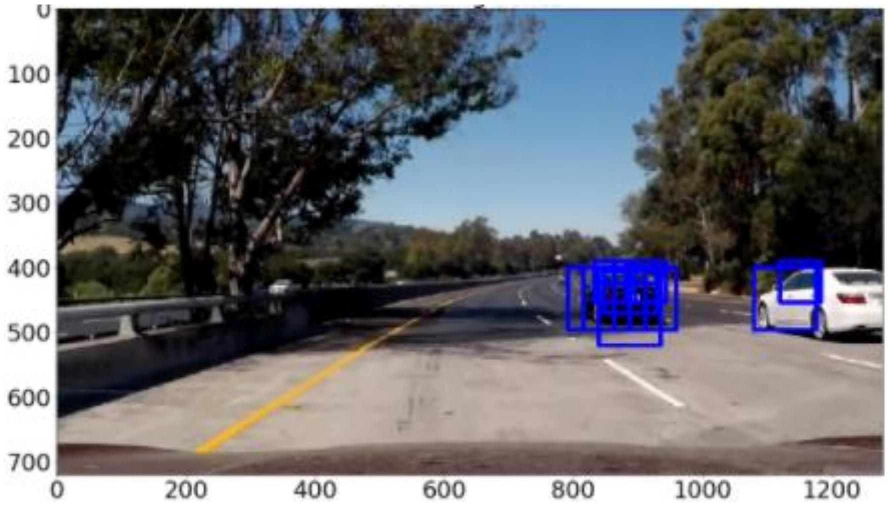
####1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

At first I used the sliding window search and converted each window to hog features, but it turned out that this method takes too much processing time. Hence, I used the subsampling method with different scales.

####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?

Ultimately I searched on three scales (1, 1.5 and 2) 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.)

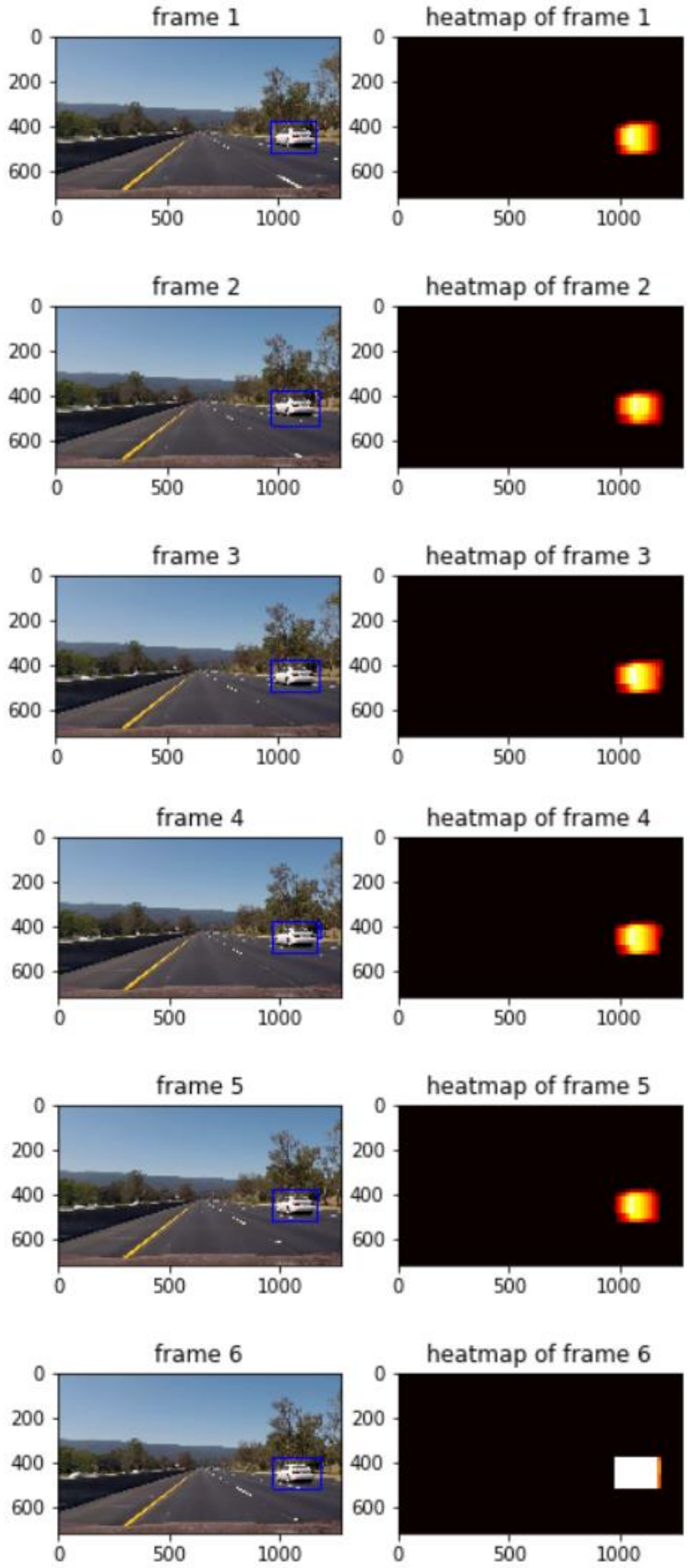
The video is attached in the main folder directory.

####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.
The code is in the 8th cell of the notebook (last part of the image pipeline).

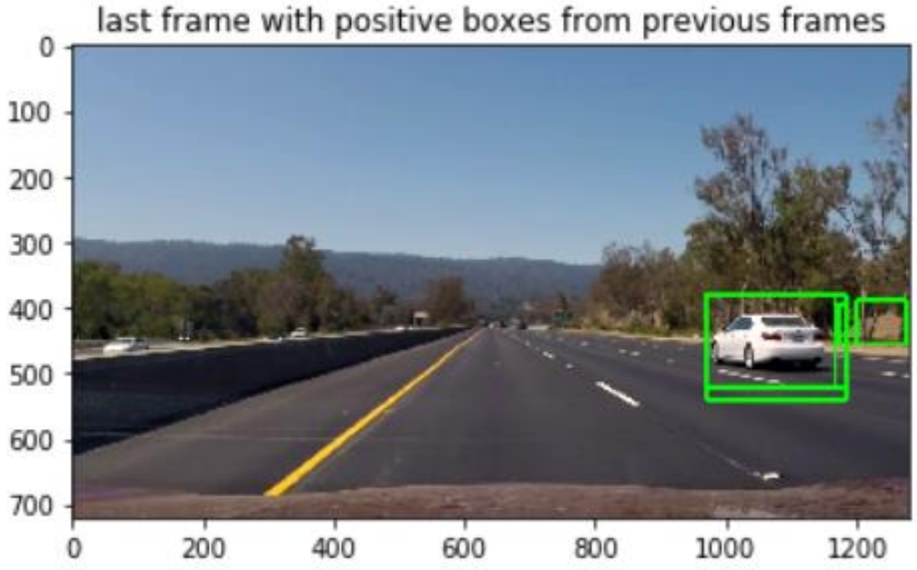
Here's an example result showing the heatmap from a series of frames of video, the result of scipy.ndimage.measurements.label() and the bounding boxes then overlaid on the last frame of video:

Here are six frames and their corresponding heatmaps:



frame 1
heatmap of frame 1
frame 2
heatmap of frame 2
frame 3
heatmap of frame 3
frame 4
heatmap of frame 4
frame 5
heatmap of frame 5
frame 6
heatmap of frame 6
0
200
400
600
0
500
1000

Here is the last frame with all the bounding boxes from previous frames:



###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 first problem I faced was the processing time with the sliding window search. This method takes a lot of processing time. Hence, I used the HOG subsampling method which decreased the processing time considerably, but it still does not meet the requirements for real time system.

In my opinion, I should use a deep neural network for image classification to achieve better performances.

Another issue I faced was with the classifier. It shows a high accuracy after the training, but I get some false positives. So I used `CalibratedClassifierCV()` and `predict_proba()` functions to obtain the probability from the classifier, then I draw only boxes with a probability higher than 70%.