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 (https://review.udacity.com/#!/rubrics/513/view) 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 (https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) 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. Extracting HOG features from the training images.

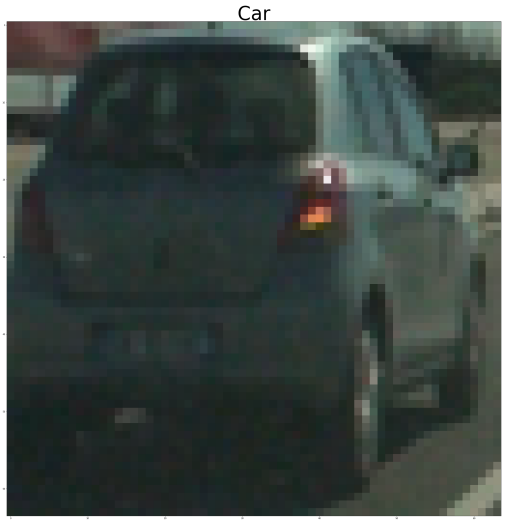
The extract_features() code consists of applying color conversion to choice of color space use, applying bin spatial features, generating color histogram features, and getting HOG features base on various parameters like number of orientations, pixel per cell, cell per block etc. The

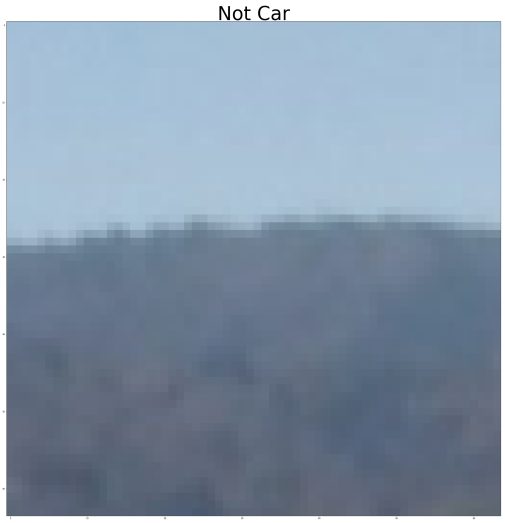
extract_features() concatenated all three features from above to create a comprehensive feature base for the training dataset.

I balanced the dataset a bit by including flip images of the vehicles so that the ratio between vehicles and non-vehicles images are close to 50/50 to avoid bias or overfitting. I split the dataset to 80/20 for training and testing set as shown below.

```
car_features len: 4646
notcar_features len: 4404
Feature vector length:scaled_X 9050
Feature vector length:y 9050
Feature vector length:X_train 7240
Feature vector length:y_train 7240
Feature vector length:X_test 1810
Feature vector length:y_test 1810
```

Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



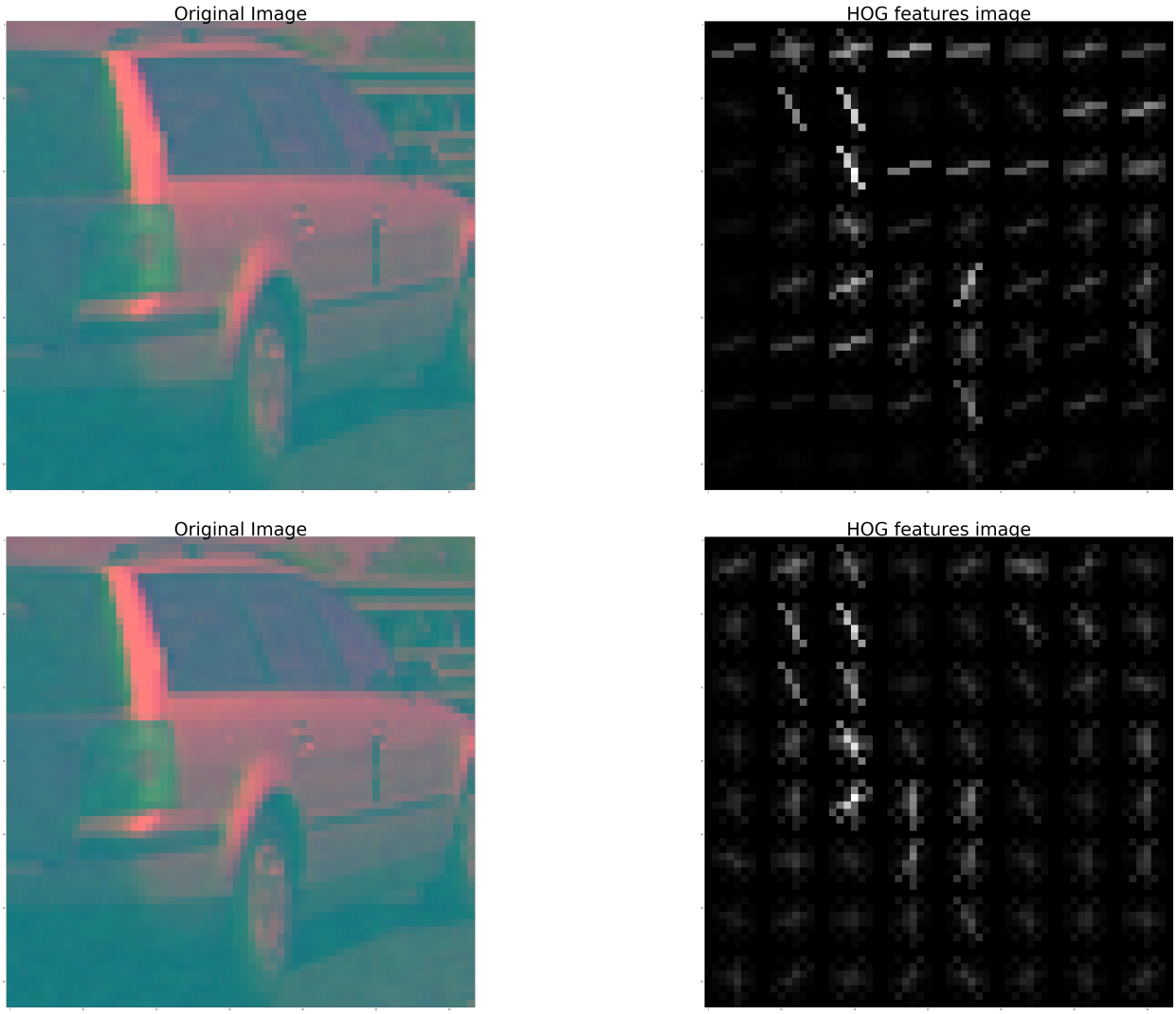


2. Choice of HOG parameters.

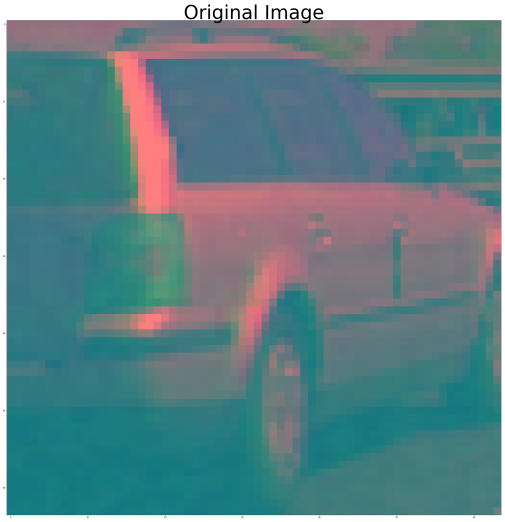
I tried various combinations of parameters and the best results seem to be with the following settings:

```
# parameters for color classify
colorspace = 'YCrCb' # Can be RGB, HSV, LUV, HLS (good), YUV (better), YC
rCb
spatial_size=32
hist_bins=32
# parameters for HOG classify
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_feat = True
hist_feat = True
hog_feat = True
heat_thresh = 3
```

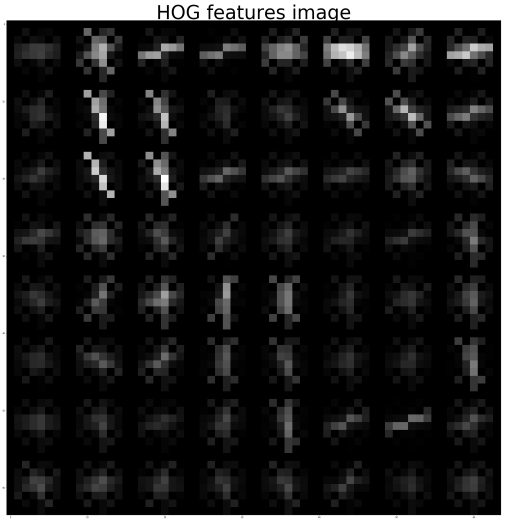
Here is an example using the YUV color space and HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



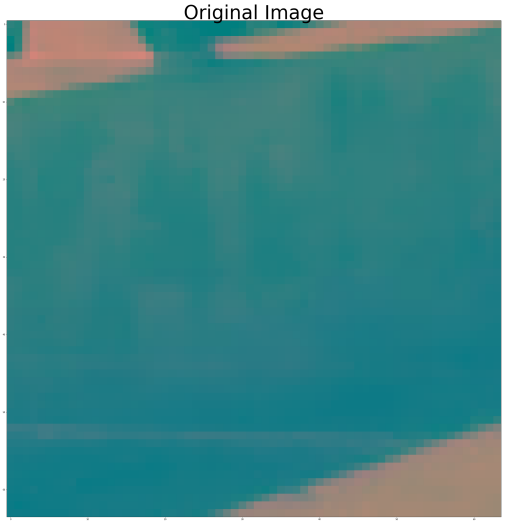
Original Image



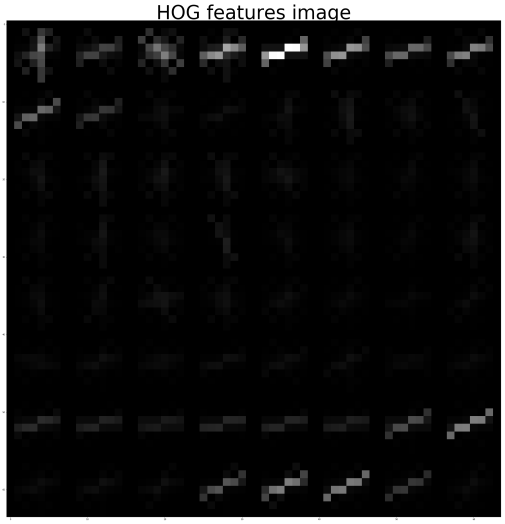
HOG features image



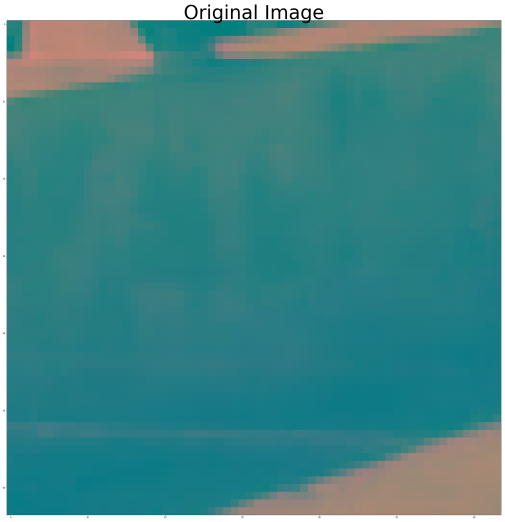
Original Image



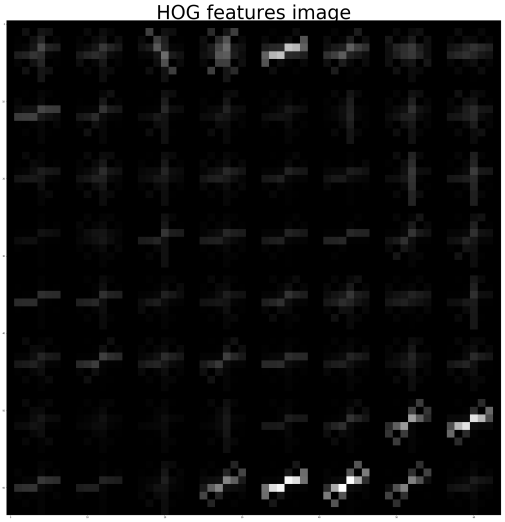
HOG features image

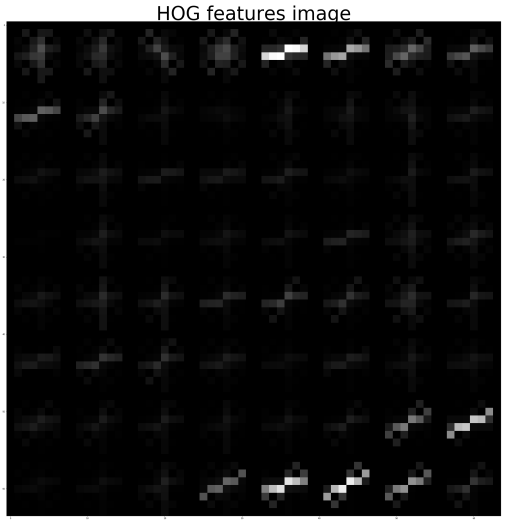


Original Image



HOG features image





3. Trained a classifier using selected HOG features and color features

Before reading in all the vehicles and non-vehicles images, I used extract_features() to extracted color bins, histogram bins, and HOG features from each images.

```
    # extract features like color, spatial bin, historgram, and HOG for b
oth cars and notcars images
    car_features = extract_features(cars[0], color_space=colorspace,
                                    spatial_size=(spatial_size, spatial_s
ize),
                                    hist_bins=hist_bins, hist_range=(0, 2
56),
                                    orient=orient, pix_per_cell=pix_per_c
ell, cell_per_block=cell_per_block,
                                    hog_channel=hog_channel,
                                    spatial_feat=spatial_feat, hist_feat=
hist_feat, hog_feat=hog_feat, inc_flip=True)
    notcar_features = extract_features(notcars[0], color_space=colorspace
,
                                       spatial_size=(spatial_size, spatia
l_size),
                                       hist_bins=hist_bins, hist_range=(0
, 256),
                                       orient=orient, pix_per_cell=pix_pe
r_cell, cell_per_block=cell_per_block,
                                       hog_channel=hog_channel,
                                       spatial_feat=spatial_feat, hist_fe
at=hist_feat, hog_feat=hog_feat)
```

I also normalized the dataset by fitting the dataset into StandardScaler() and did scaler transform().

```
    X_scaler = StandardScaler().fit(X)
    scaled_X = X_scaler.transform(X)
```

As for training model, I used sklearn's support vector machine (SVM) LinearSVC. The accurracy from the trained model reached a very high percentage at 99.83% from 1000 testing dataset as shown above.

```
5.47 Seconds to train SVC...
Test Accuracy of SVC =  0.9983
0.0 Seconds to predict 1000 labels with SVC
```

Sliding Window Search

1. Sliding window implementation

Taking advise from the lecture "Hog Sub-sampling Window Search", I defined a region that I wanted to search the car images between y=350 to y=656 to narrow down my search window, and stepping through the search window with sliding window of 64, and each step of 2 cells in size to begin with. This had no overlapping on blocks when sliding the window.

I started by trying various scale starting from 1.0 to 2.5 with an increament of 0.25 to get an idea how the search window worked. Then I found that not every scale worked on the test images specially on the white car near the low right corner. I decided to narrow down to scales that produce the most successful car search, plus reducing each step to 1 instead of 2 cells. This created an overlapping of every other block by 1 cell. The search improved as I fine tuned the scale and steps to [1.25, 1.75, 2.1, 2.25]

scale 1.25

scale 1.75

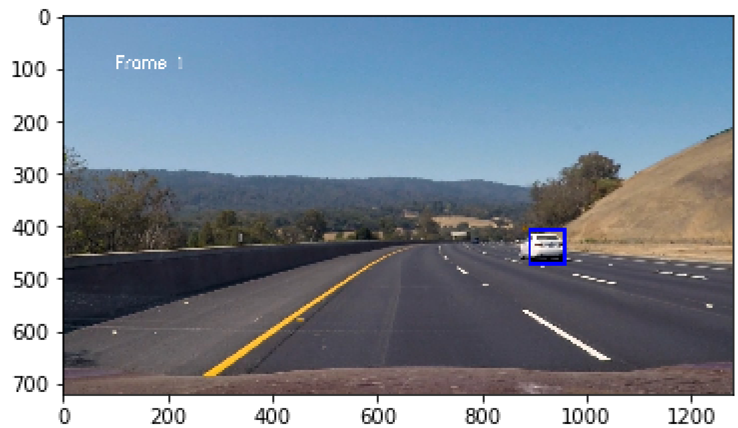
scale 2.1

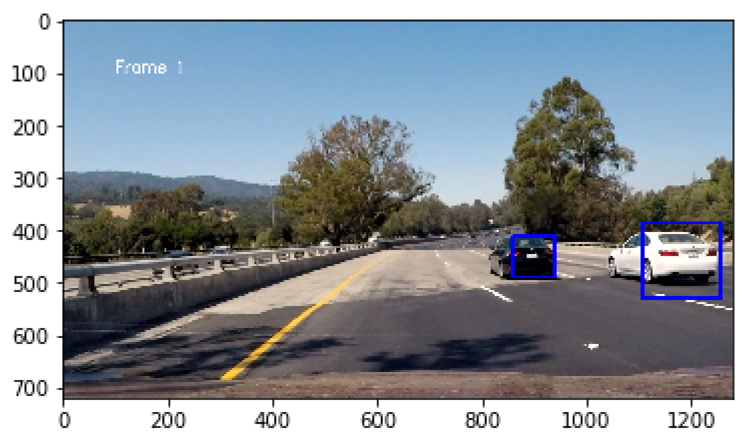
scale 2.25

2. Test images results

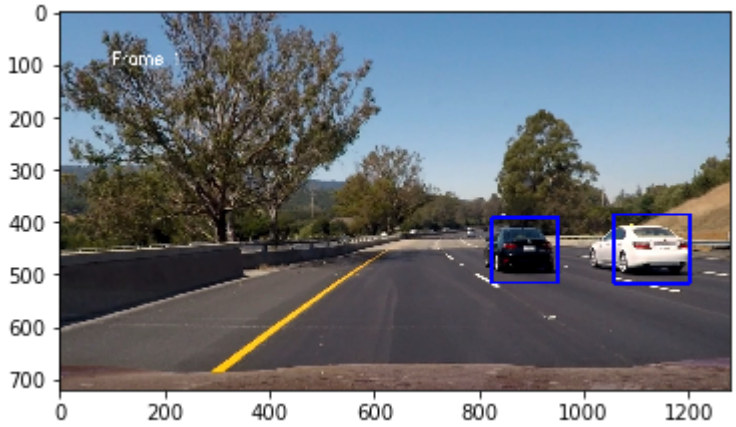
I started searching on 4 scales using YUV 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided not the greatest but a reasonable result. But then I updated to use YCrCb color space, which provided a better result than YUV. Here are some example images:


Frame 1
0
100
200
300
400
500
600
700
0 200 400 600 800 1000 1200


Frame 1
0
100
200
300
400
500
600
700
0 200 400 600 800 1000 1200


Frame 1
0
100
200
300
400
500
600
700
0 200 400 600 800 1000 1200


Frame
0
100
200
300
400
500
600
700
0 200 400 600 800 1000 1200



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.)

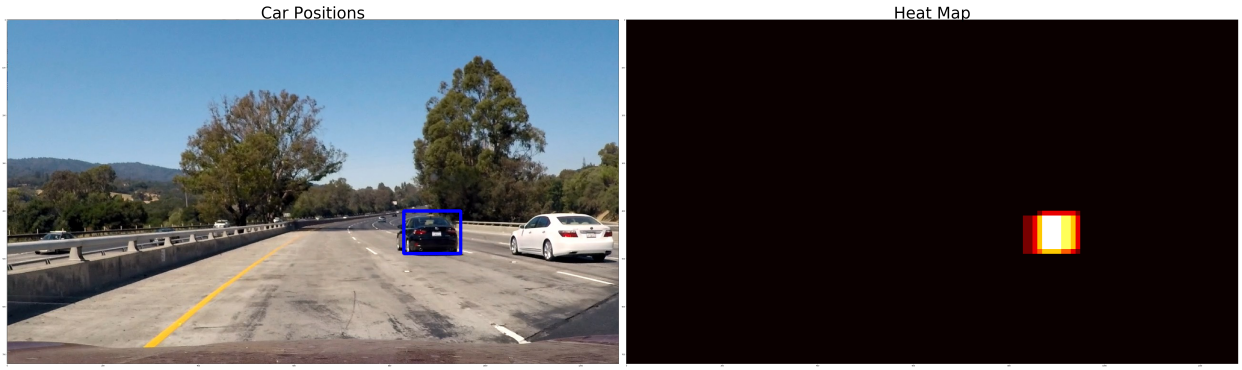
Here's a link to my video result (./test_video_outputs/project_video1.mp4)

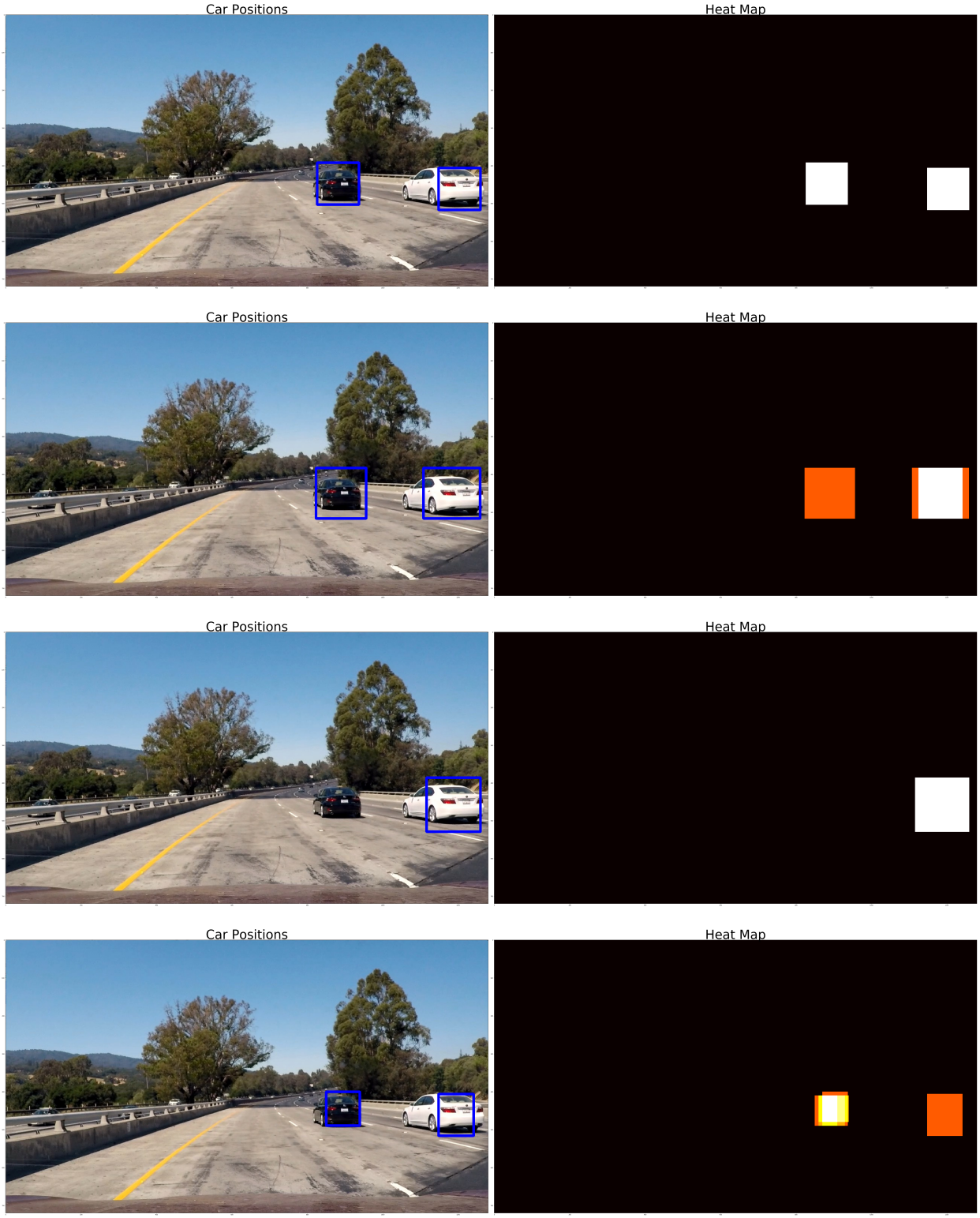
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

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 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:





![alt text][image3_6]

Discussion

1. Brief discussion

The difficulty of this project is to pick the scale and setting that works best for all video frame given that cars in vehicle is consistently changing. I first thought YUV color space will be the better choice with 11 orient, 8 pix_per_cell, 2 cell_per_block, 32 spartial_bins, 32 hist_bins, 1 step per cell. There

was still a few frames not picking up the white car, even on one of the test images specailly when the road color changed from dark to lighter color. Then after a few more explorations, YCrCb seemed to be able to pick up the white car during the road color transistion. So I settled with YCrCb, 9 orient as my final choice while keeping all other setting the same as before. There are still improvement that I can do, such as expanding the sliding windows to cover the last column of pixels in larger scale, fine tune a set of scale to have better coverage across video frame to cater different sizes of car images.

In [ ]: