Vehicle Detection Project

The goals / steps of this project are the following:

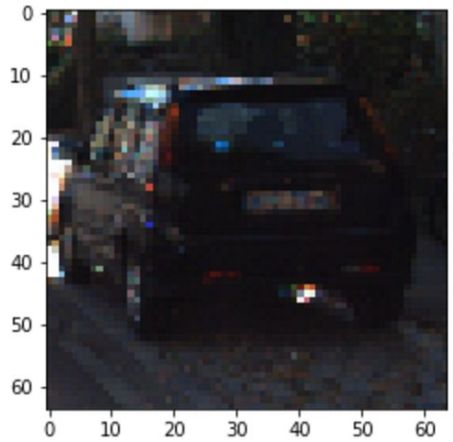
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: 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.

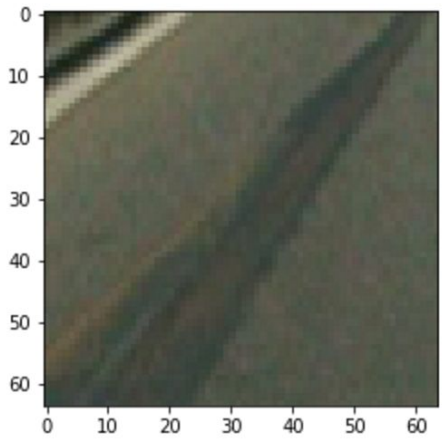
Histogram of Oriented Gradients (HOG)

1. **Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.**

See the code in IPython Notebook from [1] to [7], and from [12] to [30].

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:

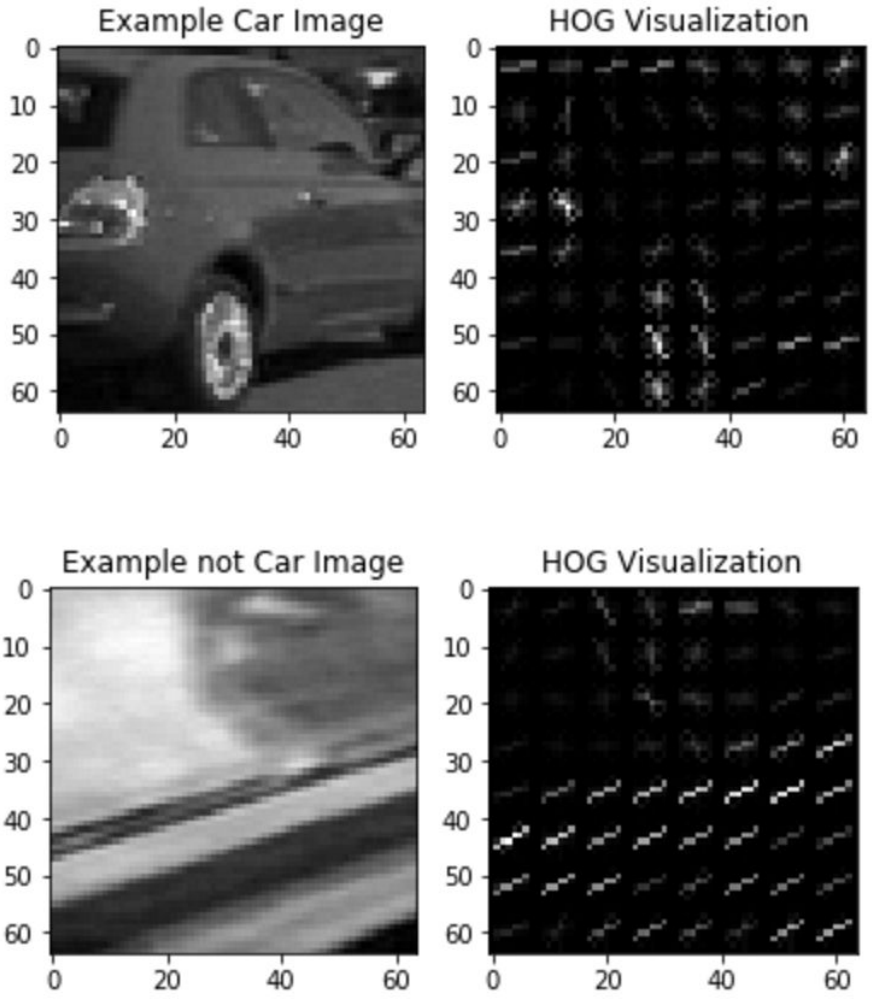




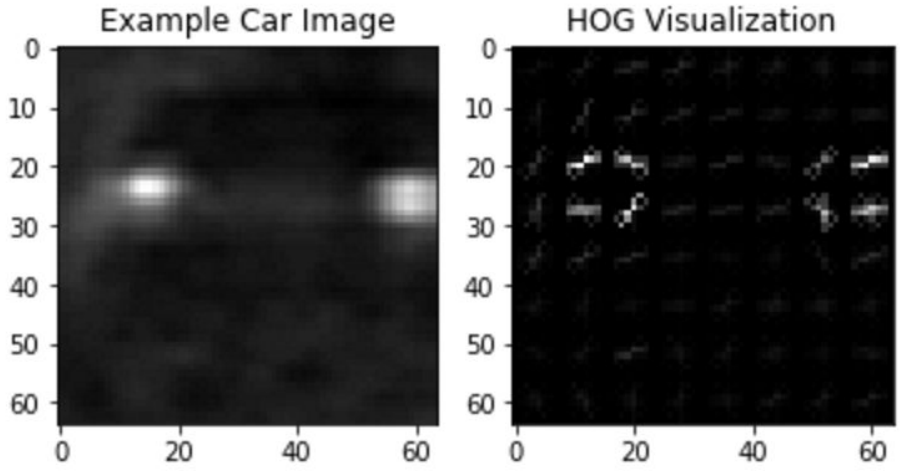
I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like.

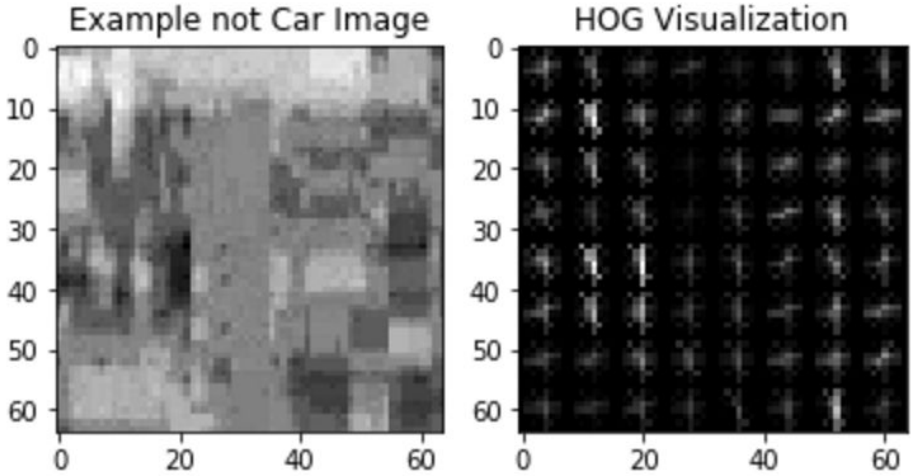
Here is an example using the YCrCb color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):

YCrCb Channel 0:

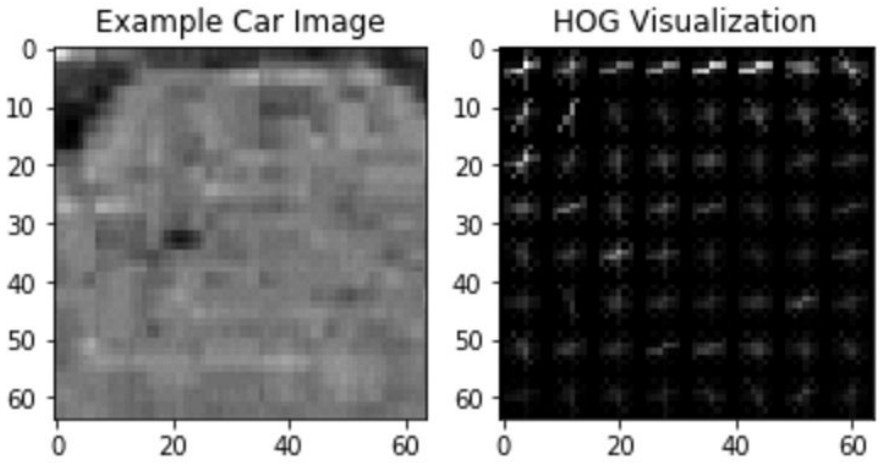


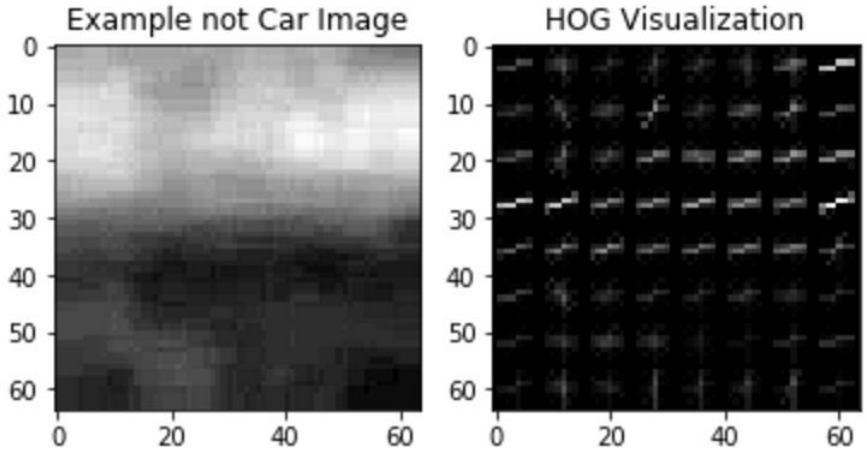
YCrCb Channel 1:





YCrCb Channel 2:





From my intuitive thoughts, more orientations and more channels will add on the accuracy of detection. I also did a lot of experiments before finalizing the appropriate parameters for HOG features.

In the model, I applied all hog channels in color space YCrCb with HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2). Plus I made use of spatial features and histogram features in order to enhance the accuracy of the classifier.

2. **Describe how (and identify where in your code) you trained a classifier using your selected HOG features and color features.**

In [31] of the IPython Notebook.

I used LinearSVM to build the classifier with HOG features and color features.

Parameters table

```
colorspace = 'YCrCb'
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = "ALL"
spatial_size=(32, 32)
hist_bins=32
```

I used StandardScaler() to normalize features, and then random split training and testing data to 9: 1.

125.42 Seconds to extract HOG features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 8460
8.28 Seconds to train SVC...
Test Accuracy of SVC = 0.9899
My SVC predicts: [1. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
For these 10 labels: [1. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
0.00685 Seconds to predict 10 labels with SVC

Sliding Window Search

3. **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?**

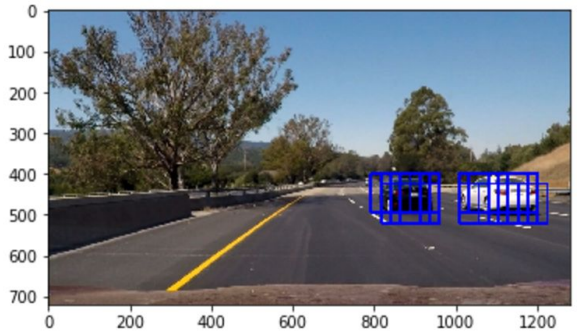
In [34] in IPython Notebook. I defined a single function that can extract features using hog sub-sampling and make predictions.

In search_window, I set ystart = 400, ystop = 656 so that the search will skip searching sky. I combine scale 1.5 and 1 together to the heatmap. I set cells_per_step = 2.

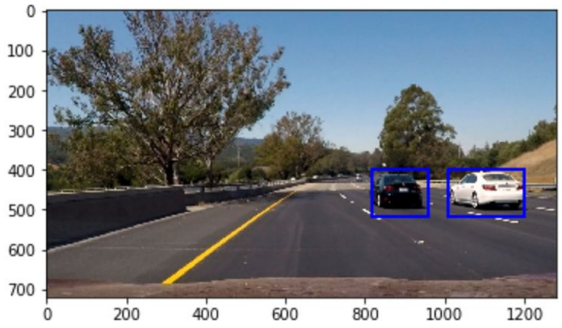
I did several experiments before finalizing the scales. Here are some examples when I player around with scaling factors.

Example:

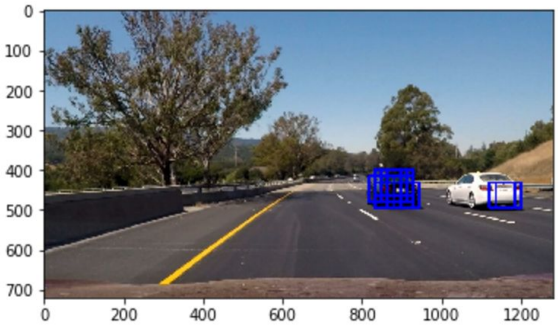
Scale = 1.5 **After apply heatmap threshold:**

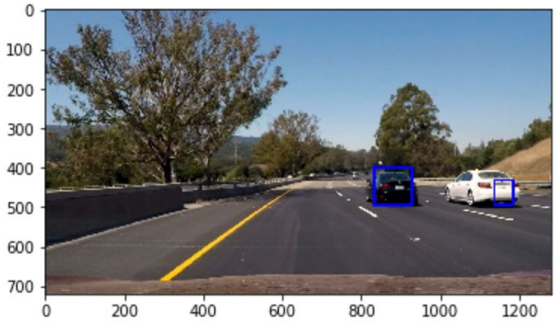


Scale = 1.0



After apply heatmap threshold:



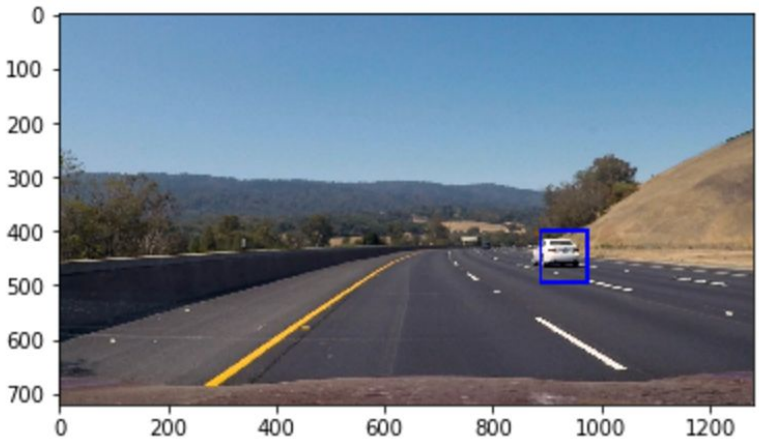


4. **Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?**

Ultimately I searched on two scales (1, 1.5) 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:

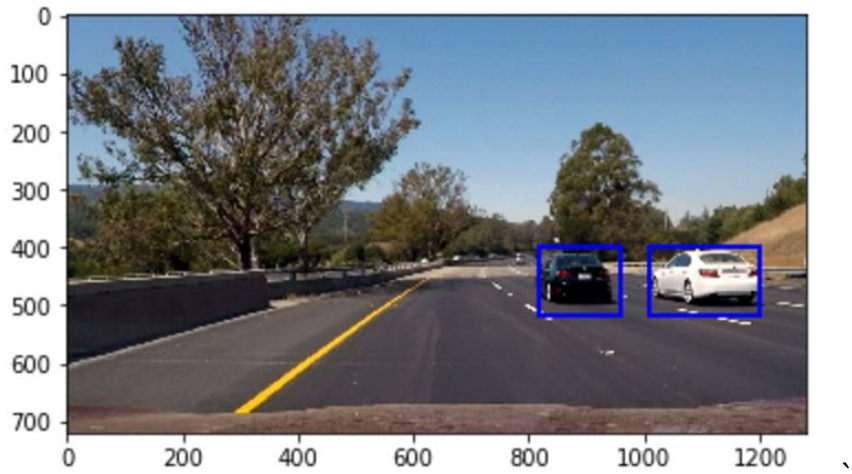
Example 1:

Example 2:



Example 3:

Example 4:



Video Implementation

5. **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.)**

In [48] of IPython Notebook

6. **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.**

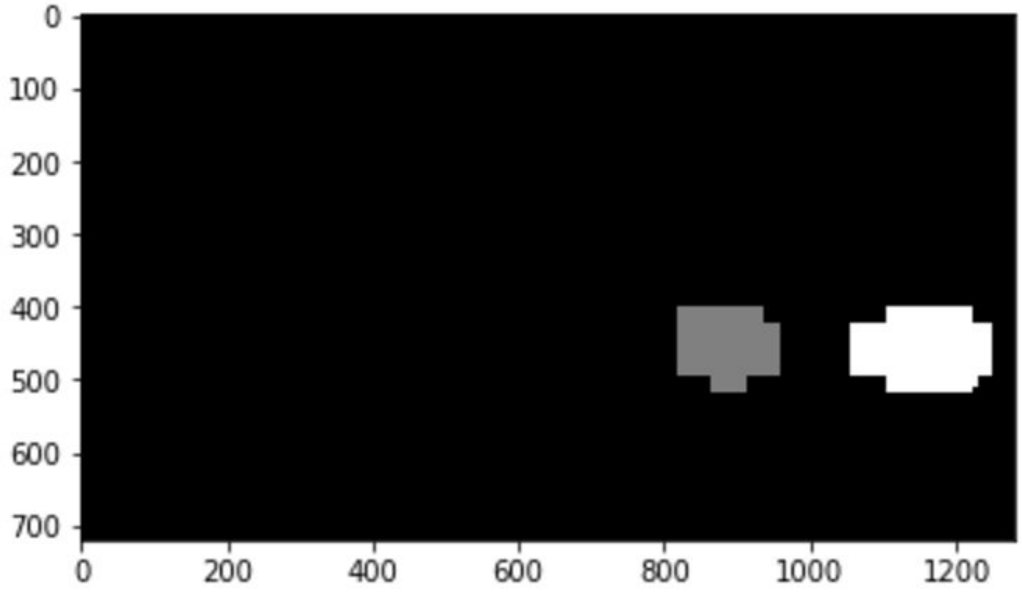
In [44] of IPython Notebook

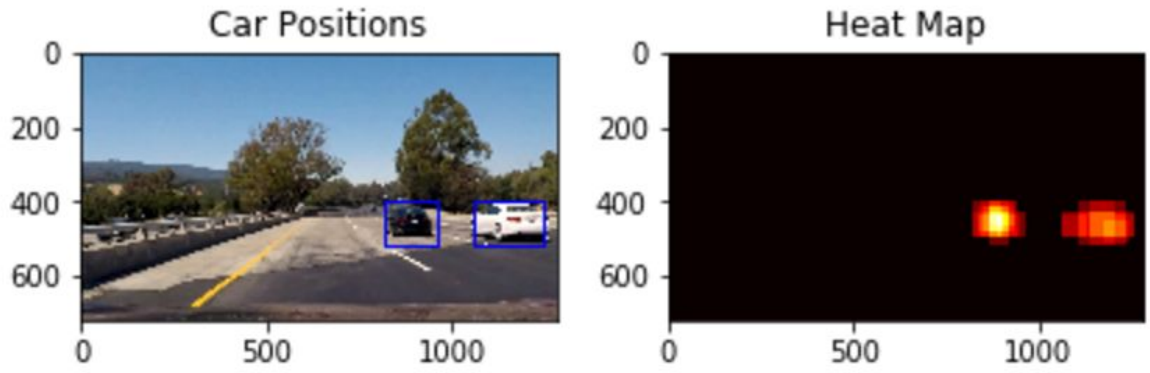
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. For the first 5 frames, my threshold for the heatmap is 1. After 5 frames, for each new frame, I calculated the heatmap based on the sum of the latest 5 frames, and apply a threshold = 2 to the heatmap.

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.

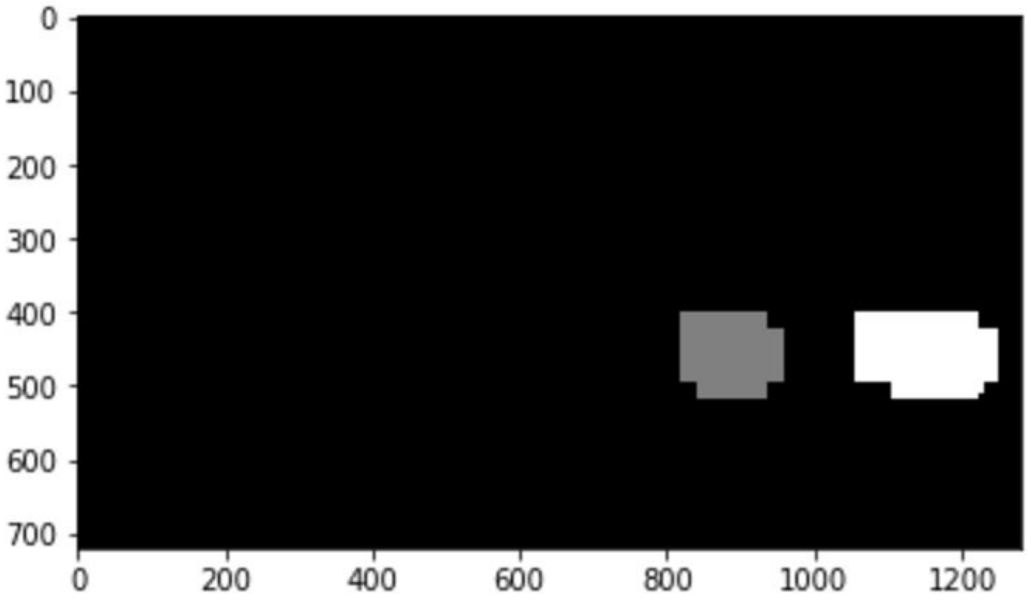
Here's an example result showing the heatmap from a series of frames of video in test video, the result of scipy.ndimage.measurements.label() and the bounding boxes then overlaid on the last frame of the test video.

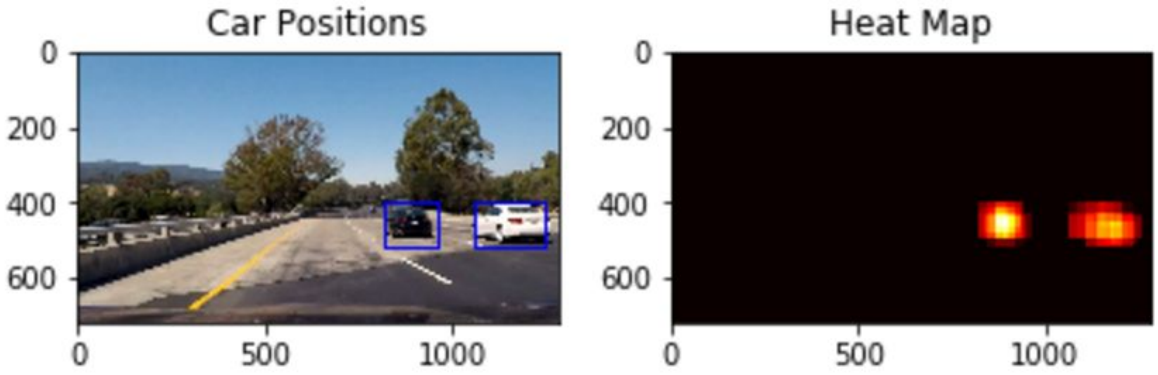
36th



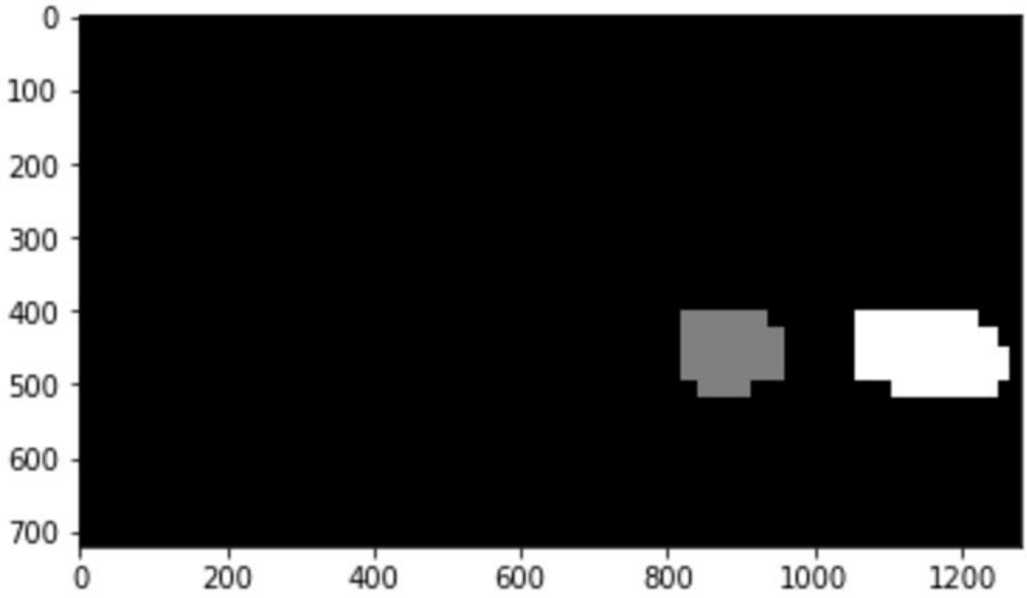


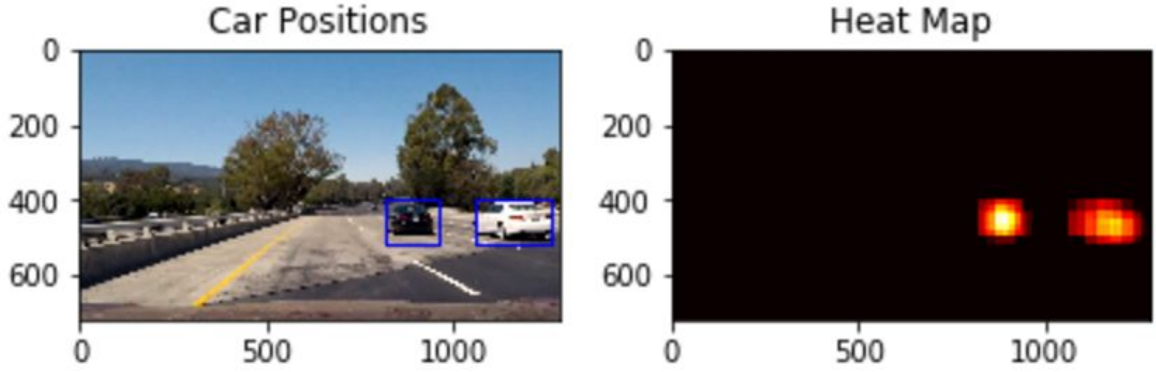
37th


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

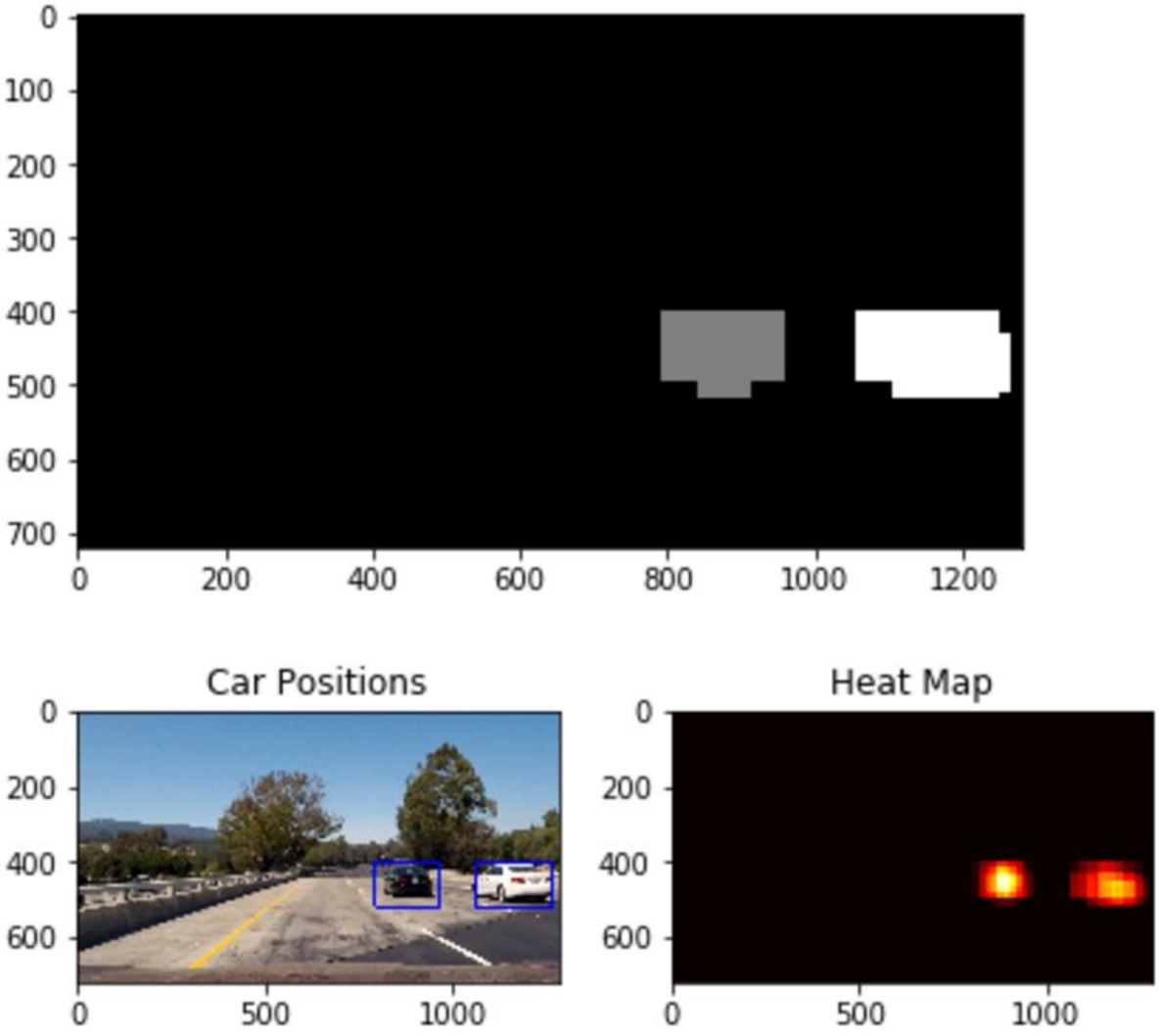

Car Positions
0
200
400
600
0
500
1000
Heat Map
0
200
400
600
0
500
1000

38th


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


Car Positions
Heat Map
0
200
400
600
0
500
1000
0
200
400
600
0
500
1000

39th



Discussion

To make my pipeline more robust, I should apply a class to keep track of the information.

```
class Vehicle():
    def __init__(self):
        self.detected =  False # was the vehicle detected in last iteration?
        self.n_detections = 0 # number of time this vehicle has been detected
        self.n_nondetections = 0 # number of consecutive times this car has not been detected
        self.xpixels = None
        self.ypixels = None
        self.recent_xfitted = []
        self.bestx = None
        self.recent_yfitted = []
        self.besty = None
        self.recentwfitted = []
        self.bestw = None
        self.recenthfitted = []
        self.besth = None
```

And when doing HOG subsampling, I should combine more scale factors together, or may be do a scale range search. So that smaller vehicles far away may be caught. The negative part about this is the processing time for the video will be extremely long. Thus in this project I made a trade off by just applying 2 scale factors.