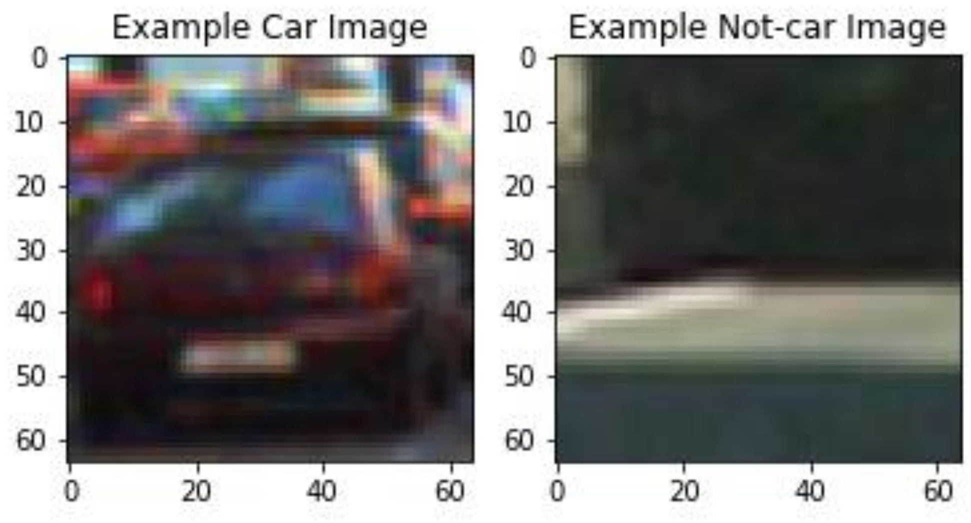
Vehicle Detection

###Histogram of Oriented Gradients (HOG)

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

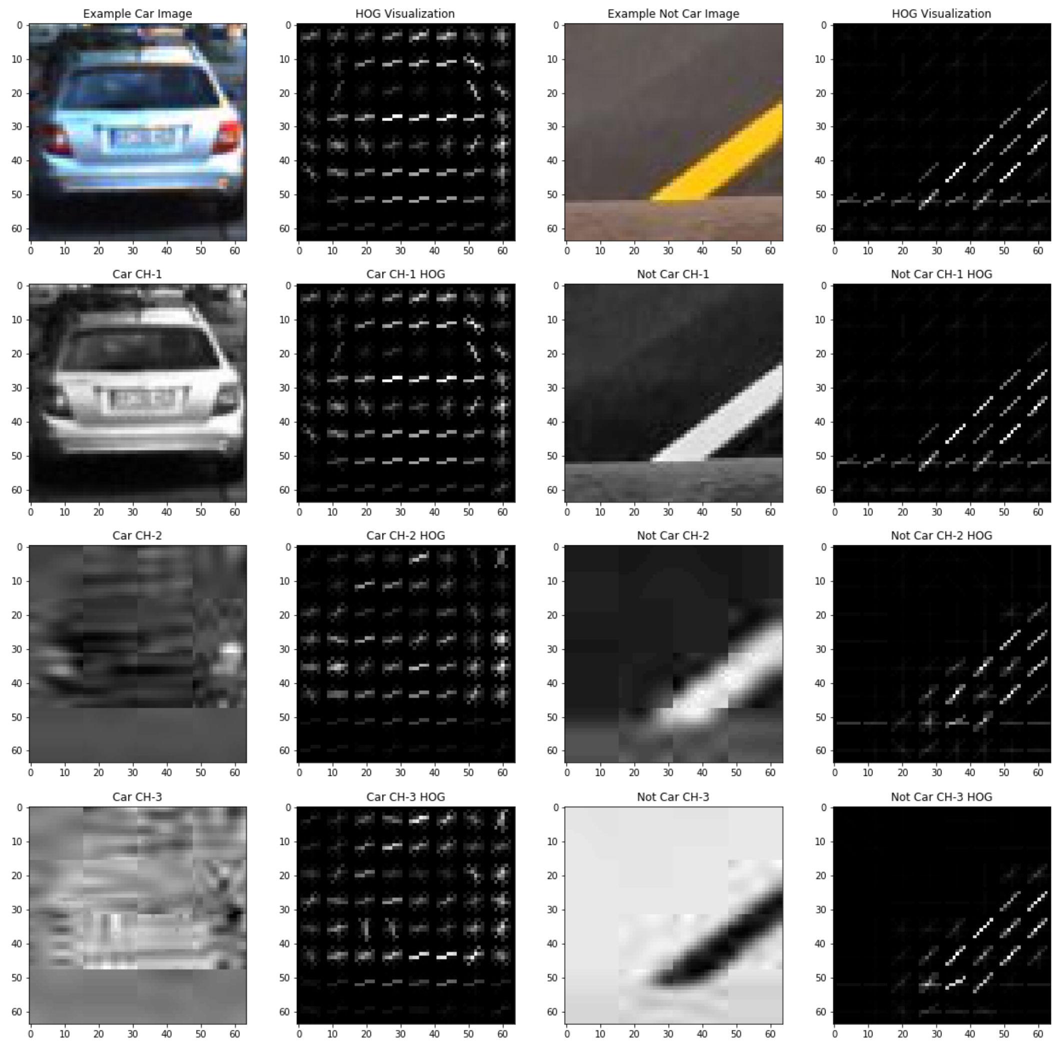
The code for this step is contained in the first code cell of the IPython notebook located in "Vehicle Detection.ipynb" .

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):



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

I tried various combinations of parameters and I decided HOG parameters as following.

Color_space	YCrCb
HOG orientations	9
HOG pixels per cell	8
HOG cells per block	2
Hog channel	ALL
Spatial binning dimensions	(32, 32)
Number of histogram bins	32
Spatial features	ON
Histogram features	ON
HOG features	ON

####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 for this step is contained in the 3rd code cell of the IPython notebook located in "Vehicle Detection.ipynb" .
I trained a linear SVM using 9 orientations 8 pixels per cell and 2 cells per block.
Feature vector length is 8460

###Sliding Window Search

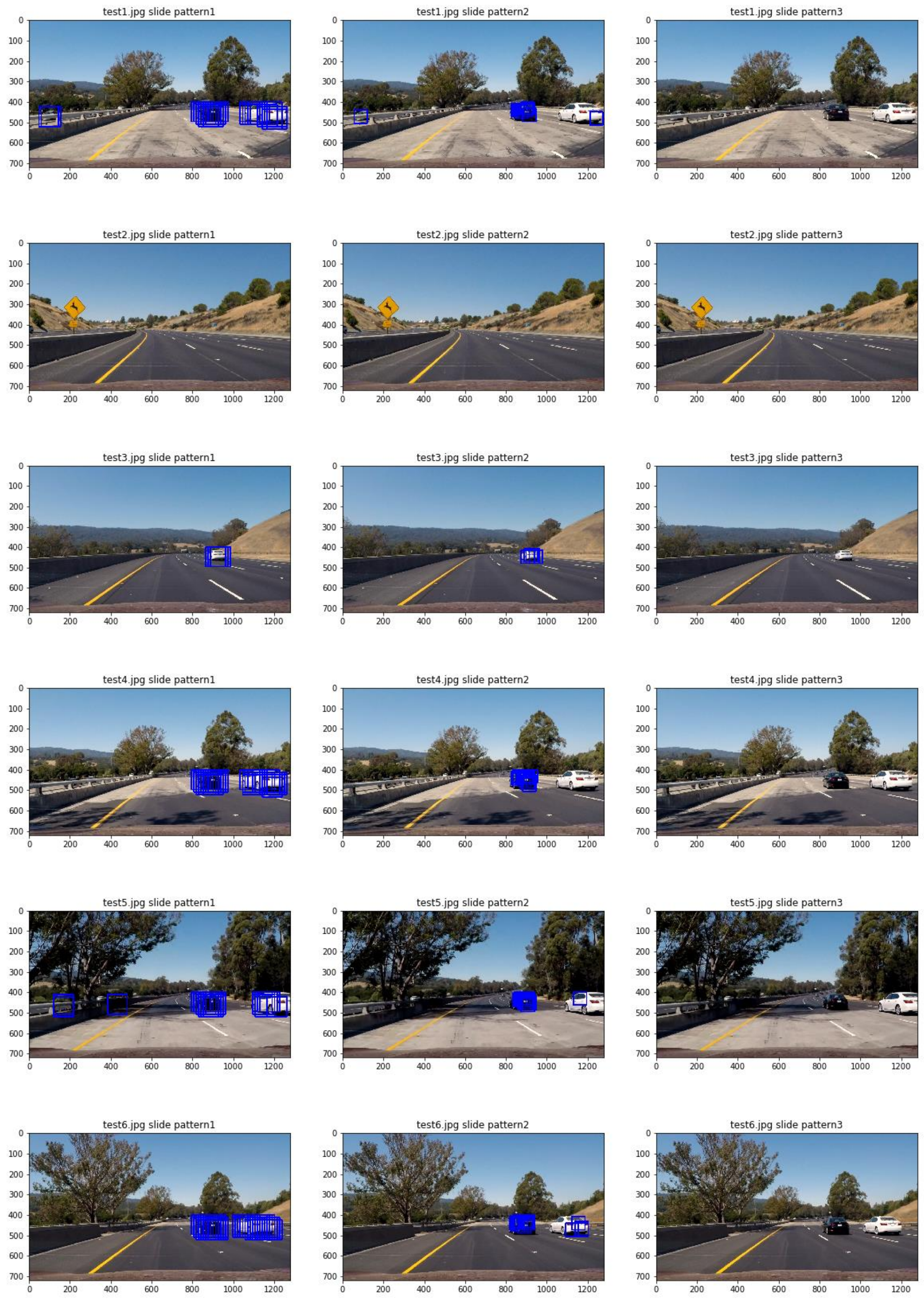
####1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The code for this step is contained in the 4th code cell of the IPython notebook located in "Vehicle Detection.ipynb" .
For improving recognition performance, I decided to combine three patterns of parameters of Sliding Window Search as following.

	pattern1	pattern2	pattern3
Sampling rate	64	64	128
Cells per step	1	1	1
Start search position	400	400	400
Stop search position	656	656	656
Scale	1.5	1	2

####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 two 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.)

Here's a link to my video result.
"vehicle_detection_result.avi"

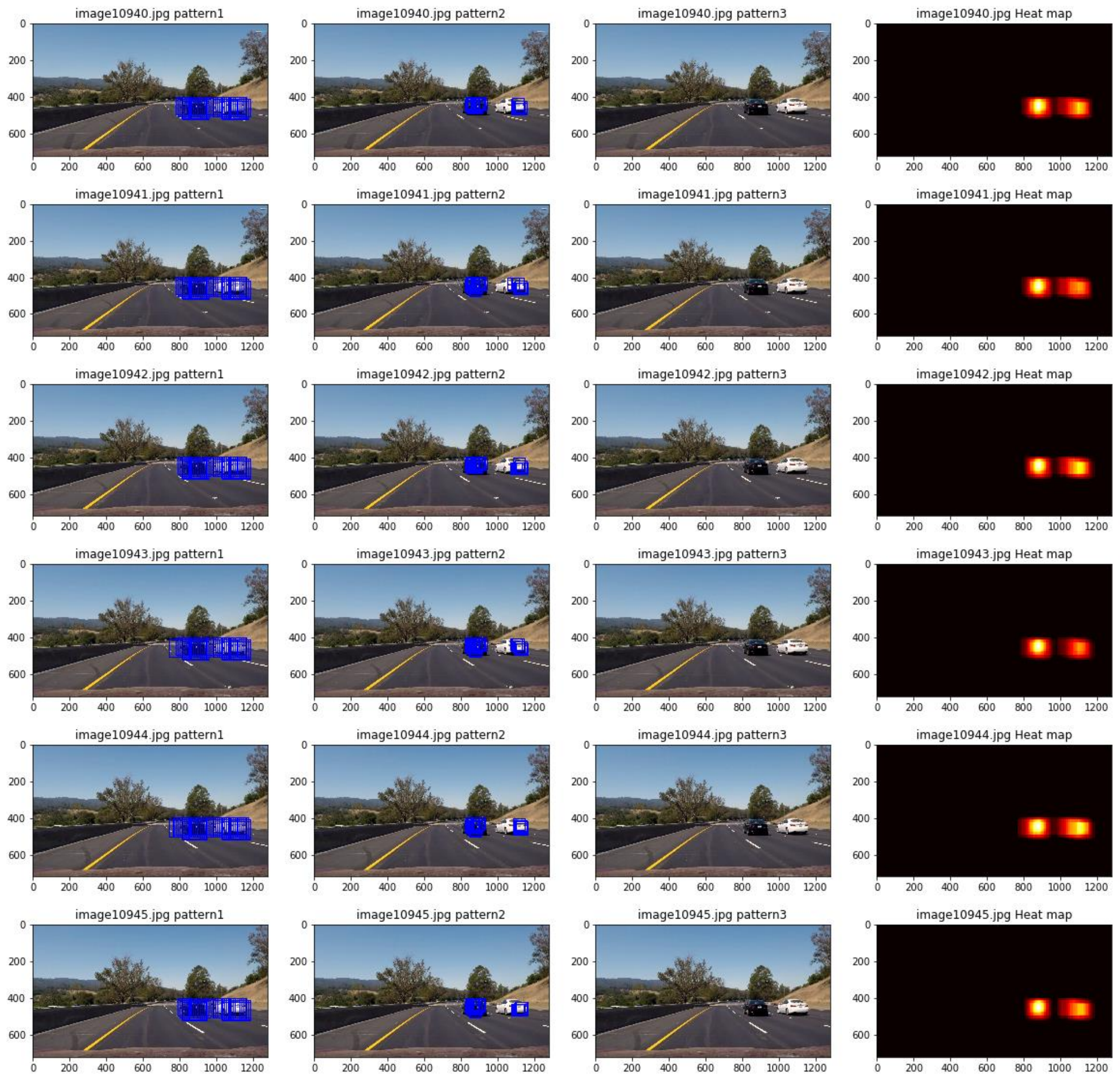
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

The code for this step is contained in the 6th code cell of the IPython notebook located in "Vehicle Detection.ipynb" .
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.

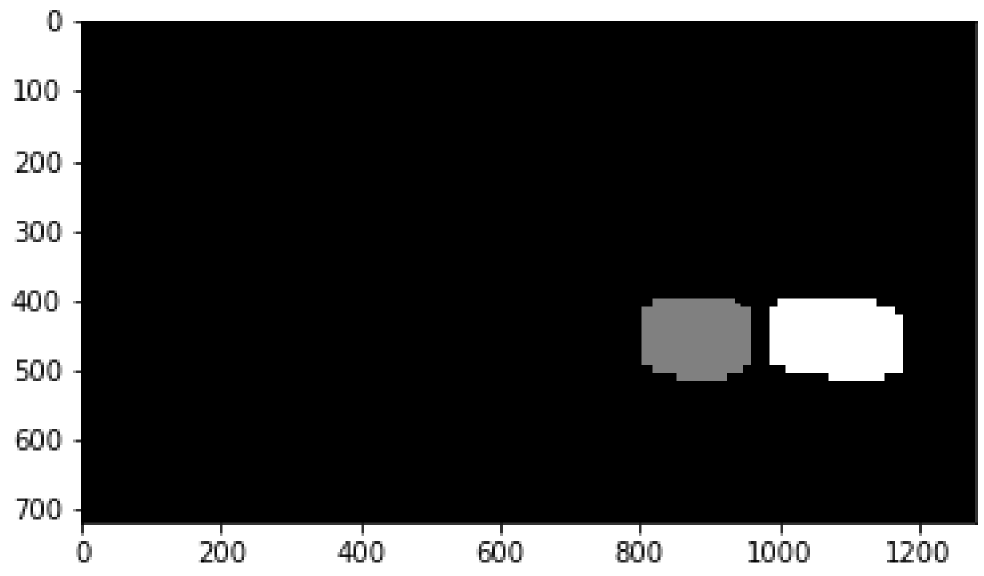
For reducing false positive, I combined detection results of past five frames and made heatmap.

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:

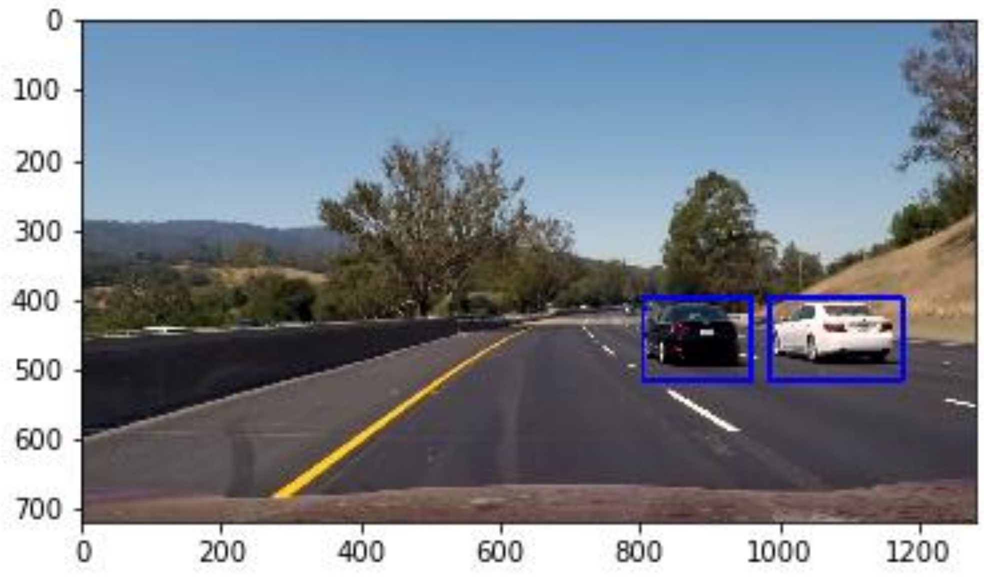
Here are 6 frames and their corresponding heatmaps:



Here is the output of scipy.ndimage.measurements.label() on the integrated heatmap from all six frames:



Here the resulting bounding boxes are drawn onto the last frame in the series:



###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?

1. At first I used small data set of vehicles and non vehicles for training SVM. However recognition performance was not good. Therefore I used all data set including GTI and KITTI. Then the performance became better.

2. For reducing false positive, I combined detection results of past five frames. It worked well for the moment. However it might not work when the road geometry or the distance between ego vehicle and target vehicle change in a short time.

3. I set constant values as the start and stop of sliding window search position. However it might not work on a sloping road.