# Udacity Project 5: 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.

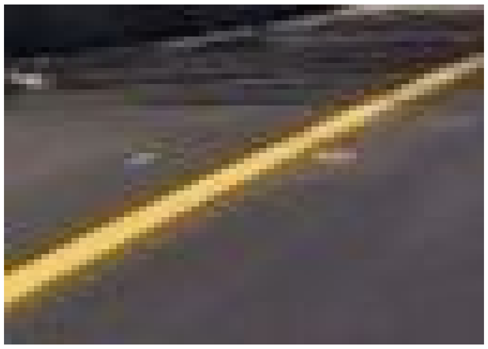
## Histogram of Oriented Gradients (HOG)

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

I extract the HOG features in a function named `get_hog_features.` I extracted the features using the in-built function `hog` that takes as input the parameters `orient, pix_per_cell, cell_per_block.`

The code is in the first cell of the Vehicle-Detection.ipynb jupyter 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:

| Non-car | Car |
|---|---|
|  | |

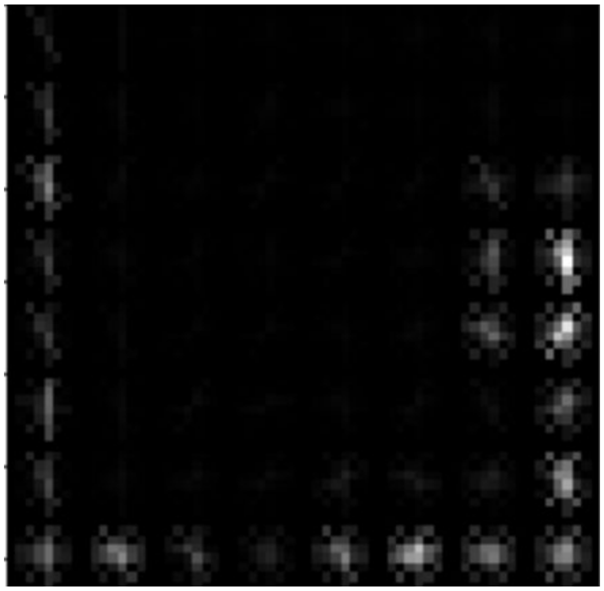
I initially started with both color and histogram features. However based on the suggestions from the udacity lessons, I then started to experiment with only the HOG features. It became pretty obvious soon then that the SVM was performing pretty well with just the HOG features, Below I list a small set of parameters that I experimented with.

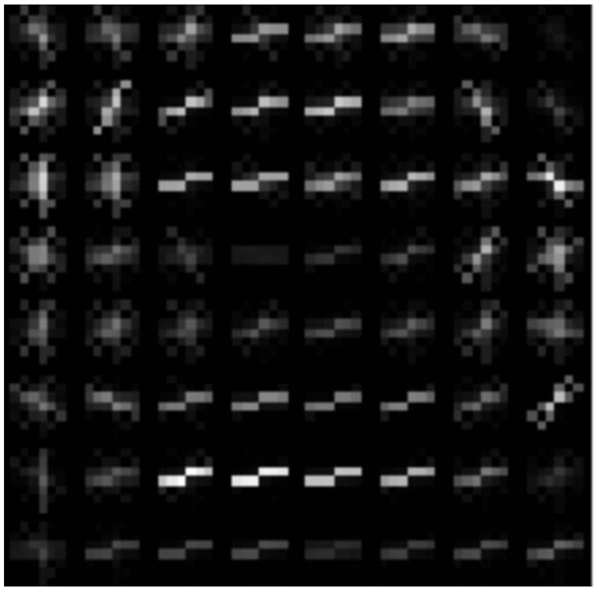
| Parameters | Test set accuracy |
|---|---|
| color_space = 'YUV'<br>orient = 11<br>pix_per_cell = 16<br>cell_per_block = 2<br>hog_channel = 0<br>spatial_size = (16, 16)<br>hist_bins = 16<br>spatial_feat = False<br>hist_feat = False<br>hog_feat = True<br>y_start_stop = [None, None] | 0.17 Seconds to train SVC<br>Test Accuracy of SVC = **0.9376** |
| color_space = 'RGB'<br>orient = 9<br>pix_per_cell = 8<br>cell_per_block = 2<br>hog_channel = 0<br>spatial_size = (16, 16)<br>hist_bins = 16<br>spatial_feat = True<br>hist_feat = True<br>hog_feat = True<br>y_start_stop = [None, None] | 0.38 Seconds to train SVC<br>Test Accuracy of SVC = **0.9806** |

| | |
|---|---|
| color_space = 'RGB'<br>orient = 9<br>pix_per_cell = 16<br>cell_per_block = 2<br>hog_channel = 0<br>spatial_size = (16, 16)<br>hist_bins = 16<br>spatial_feat = True<br>hist_feat = True<br>hog_feat = True<br>y_start_stop = [None, None] | 0.22 Seconds to train SVC.<br>Test Accuracy of SVC = **0.9806** |
| color_space = 'RGB'<br>orient = 9<br>pix_per_cell = 16<br>cell_per_block = 2<br>hog_channel = 0<br>spatial_size = (16, 16)<br>hist_bins = 16<br>spatial_feat = True<br>hist_feat = True<br>hog_feat = True<br>y_start_stop = [None, None] | 0.22 Seconds to train SVC.<br>Test Accuracy of SVC = **0.9806** |
| color_space = 'RGB'<br>orient = 9<br>pix_per_cell = 16<br>cell_per_block = 2hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"<br>spatial_size = (16, 16) # Spatial binning dimensions<br>hist_bins = 16 # Number of histogram bins<br>spatial_feat = False # Spatial features on or off<br>hist_feat = False # Histogram features on or off<br>hog_feat = True # HOG features on or off<br>y_start_stop = [None, None] # Min and max in y to search in slide_window() | 0.54 Seconds to train SVC.<br>Test Accuracy of SVC = **0.9677** |
| color_space = 'YUV'<br>orient = 9<br>pix_per_cell = 16<br>cell_per_block = 2<br>hog_channel = "ALL"<br>spatial_size = (16, 16)<br>hist_bins = 16 | 0.06 Seconds to train SVC.<br>Test Accuracy of SVC = **0.9935** |

| | |
|---|---|
| spatial_feat = False<br>hist_feat = False<br>hog_feat = True<br>y_start_stop = [None, None] | |

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

| **Original Image** | **HOG Image** |
|---|---|
| |  |

| **Original Image** | **HOG Image** |
|---|---|
| |  |

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

My final choice for these parameters were based on a param sweep on the test accuracy of the classifier. The best test accuracy were obtained with the following
The final values I used are as follows:

```
orient = 9
pix_per_cell = 16
cell_per_block = 2
hog_channel = 'ALL'
```

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).

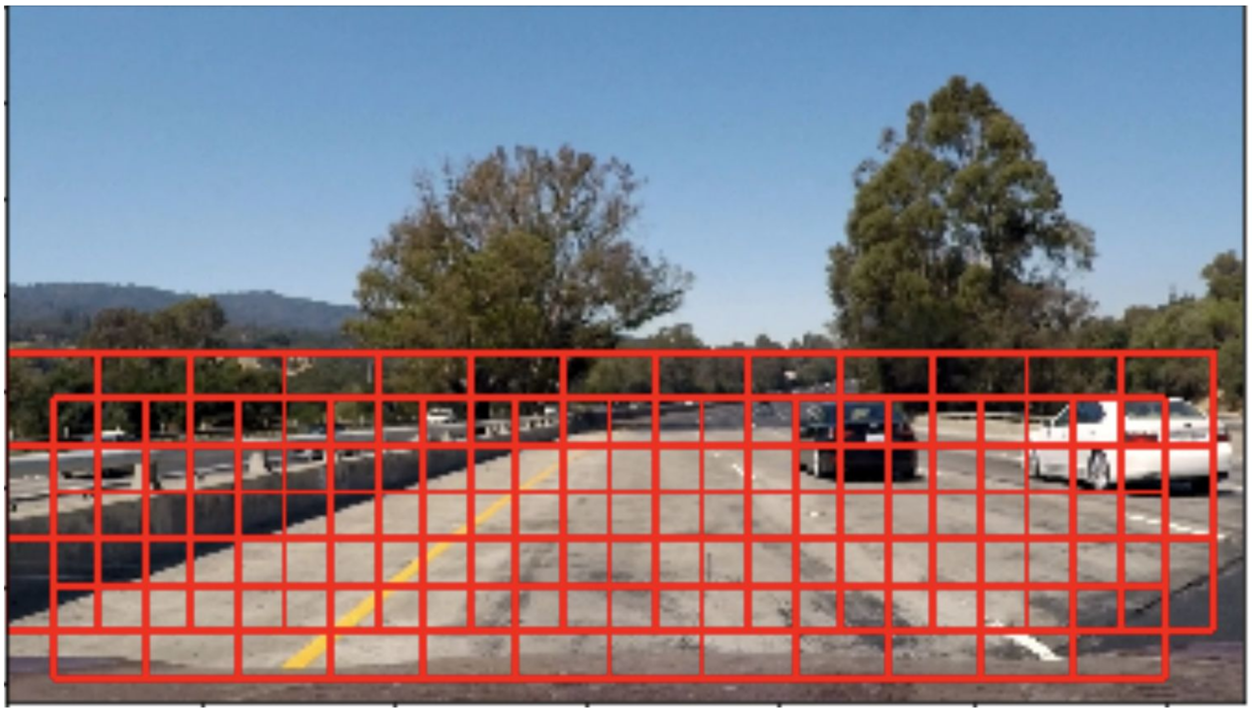
I trained a linear SVM with the default loss function of squared_hinge. The model was trained using the default sklearn framework.
I train the model in the second cell in the function train_classifier.
The model was trained using just the HOG features which did a consistently better job of identifying the white car albeit with higher number of false positives, which I handled with heat maps.

## 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?

I started with the default given size of 96 X 96 window size with 0.5 overlap. Initially I modified the window size to a smaller value to see if the model performs better with less noise mixed with car images. That is I hoped a better resolution to find a better bounding box. However this increased the processing time with any obvious performance difference in test images. I then plotted every alternate window on the image to see what the captured windows look like.

I then decided that this was in fact a good resolution and hence stuck with these values.



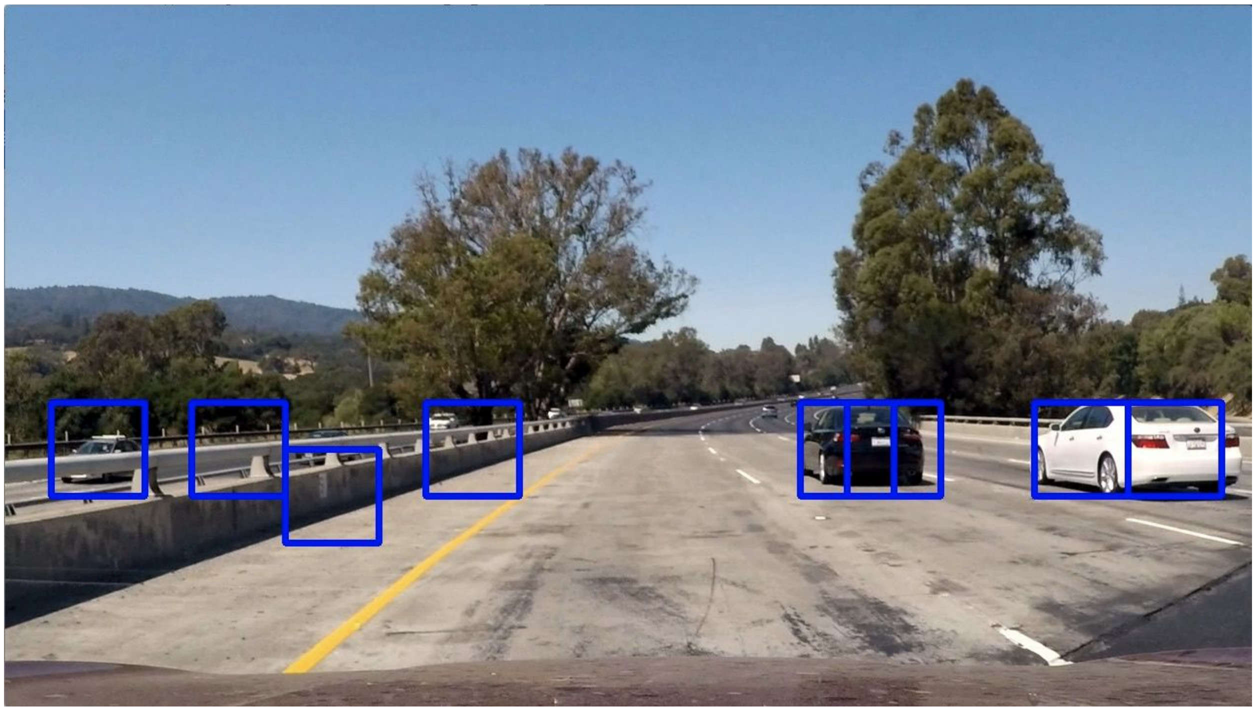
2. Show some examples of test images to demonstrate how your pipeline is working.  What did you do to optimize the performance of your classifier?

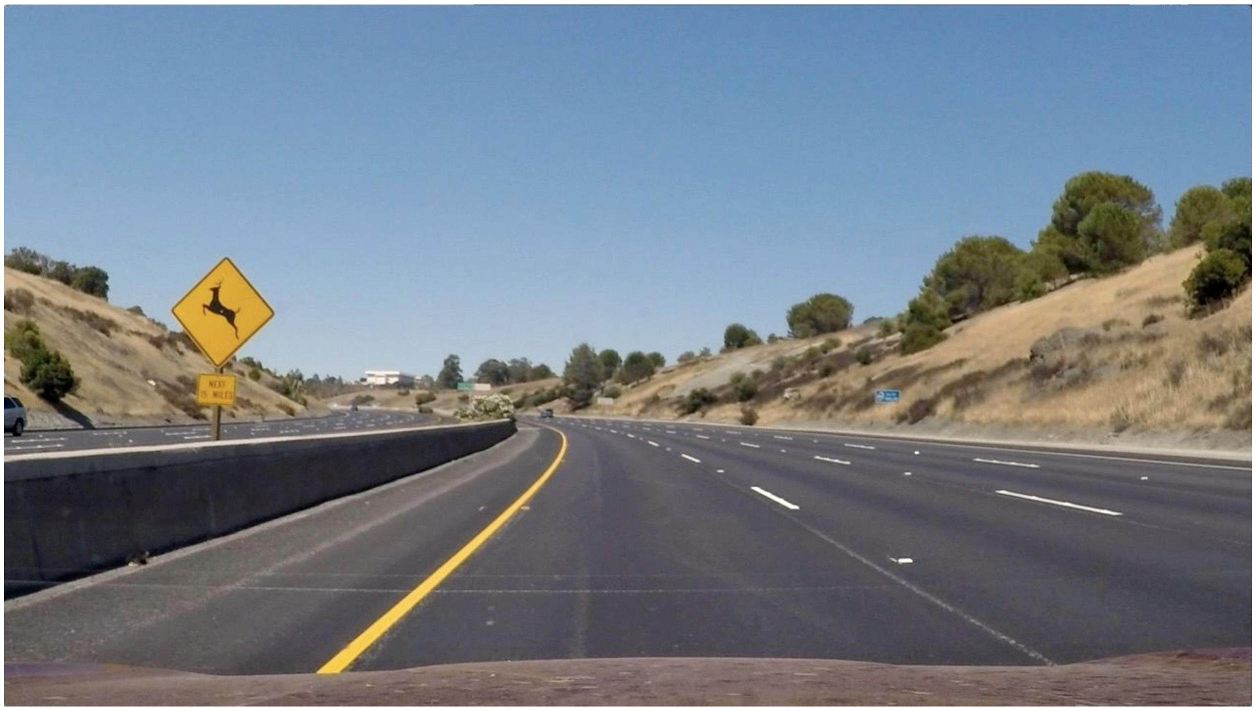
After a playing around with the parameters I decided on the following values for the hyper parameters.

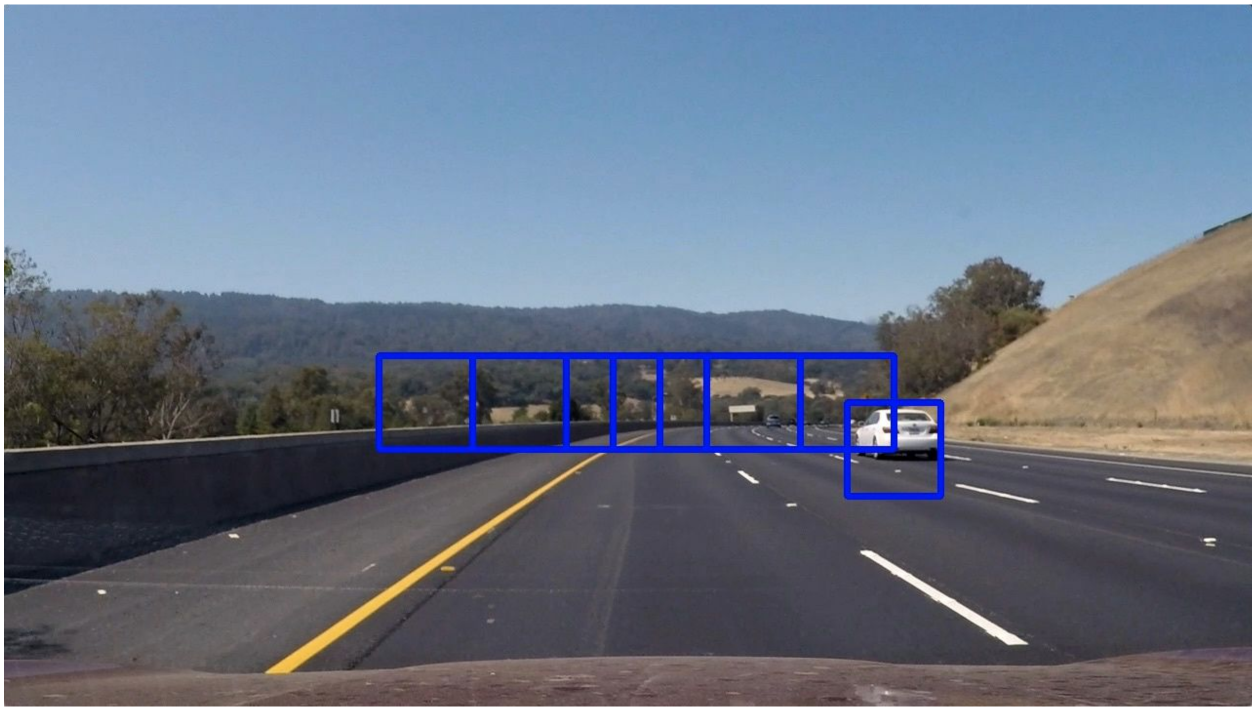
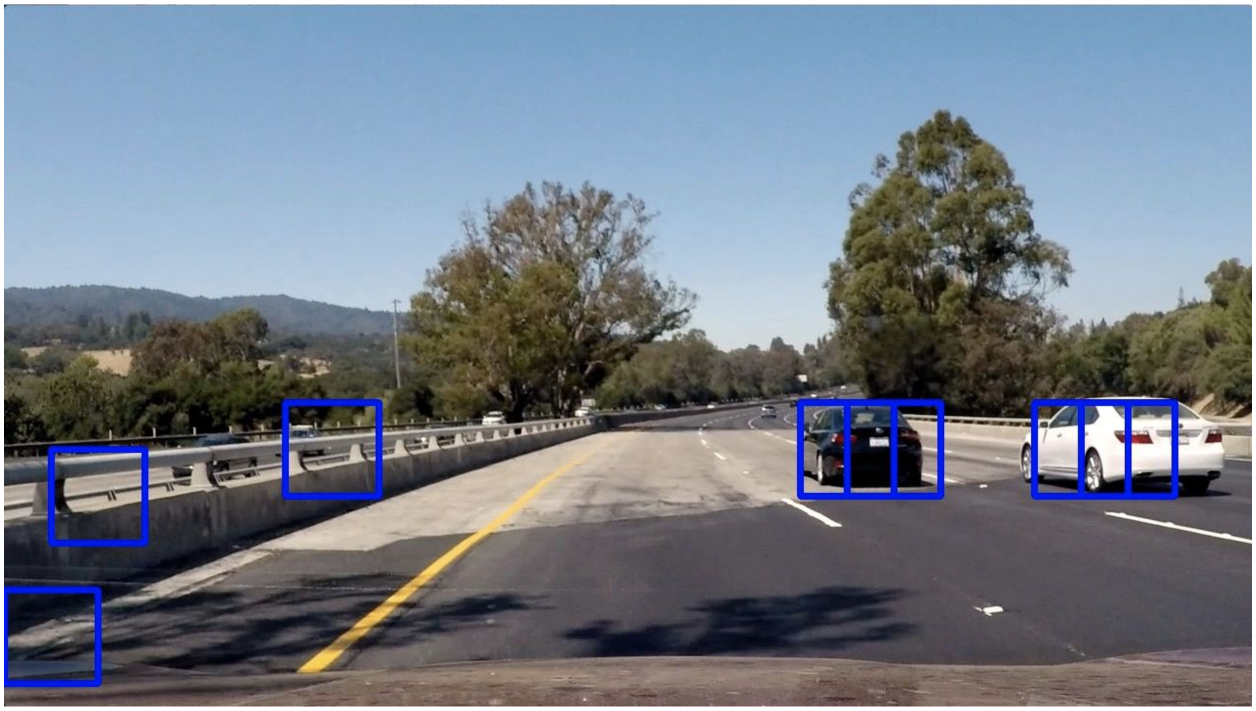
This classifier has a high accuracy and consistently identifies all car images while producing more false positives than most other classifiers I experimented with.

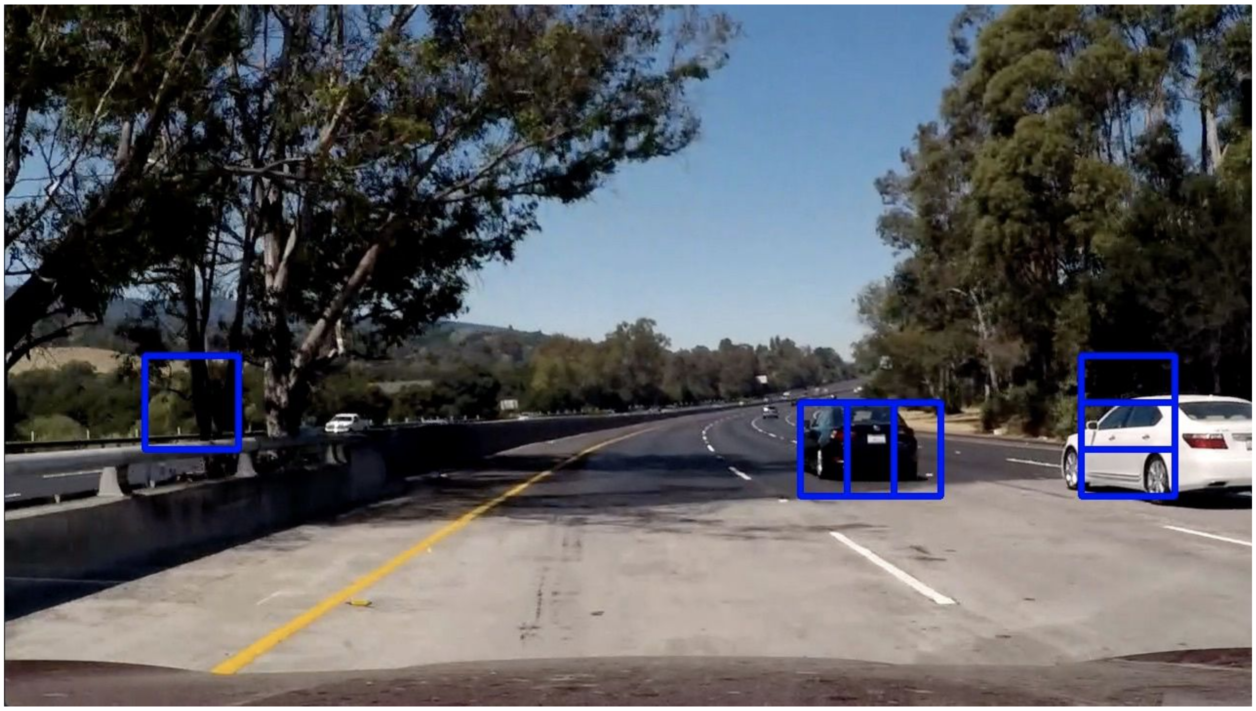
Therefore I made a tradeoff between high recall and high precision by picking high recall as I could improve precision by using heatmaps.

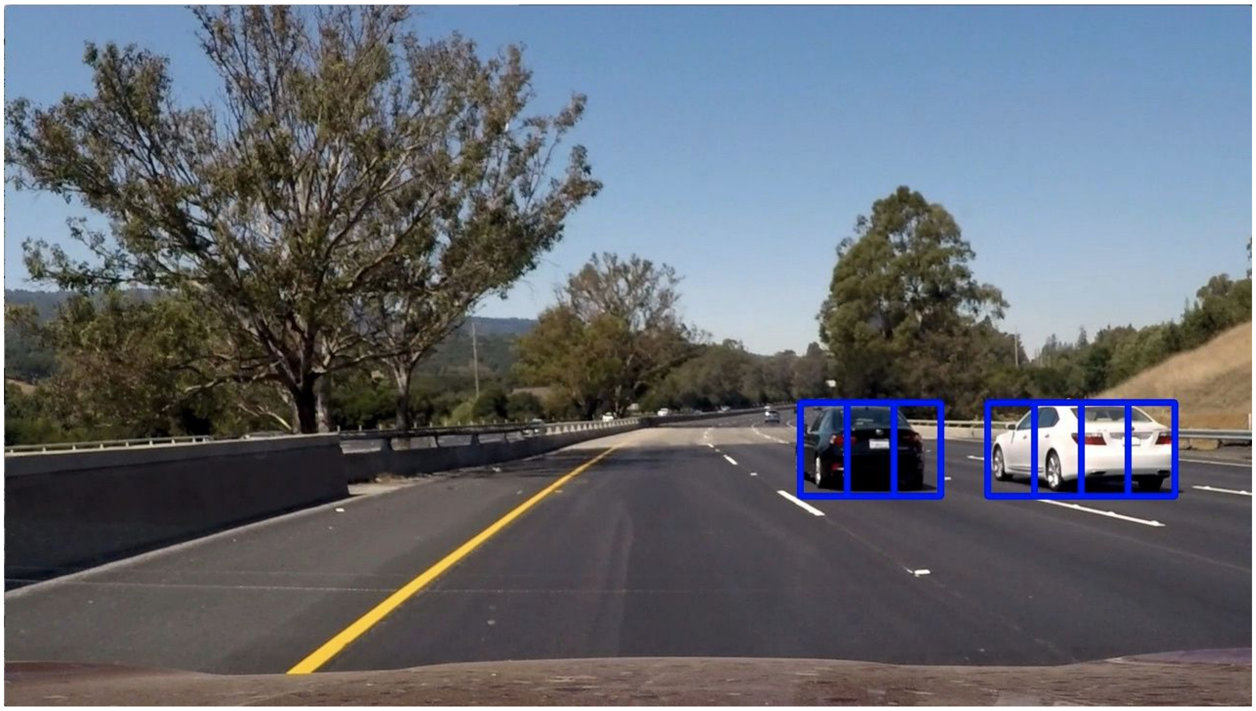
Here are some of the images the classifier produced.




NEXT
15 MILES





## 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.)

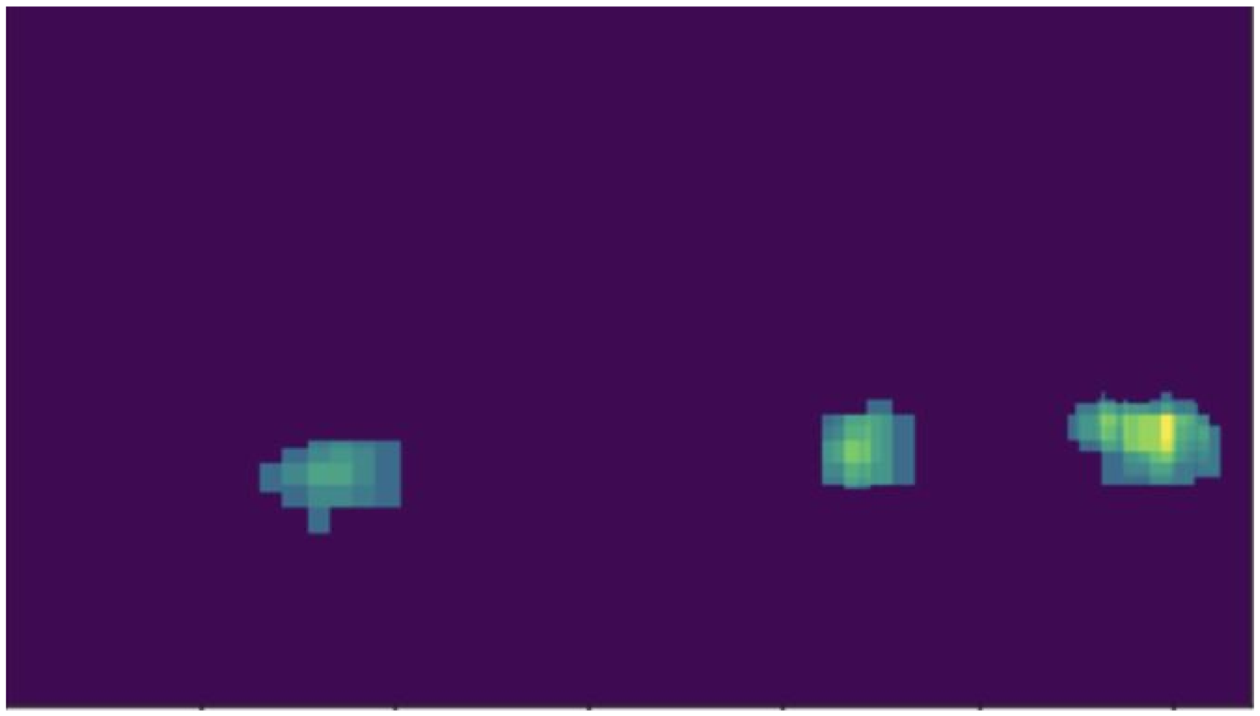
The final output is in project_video_output.mp4.

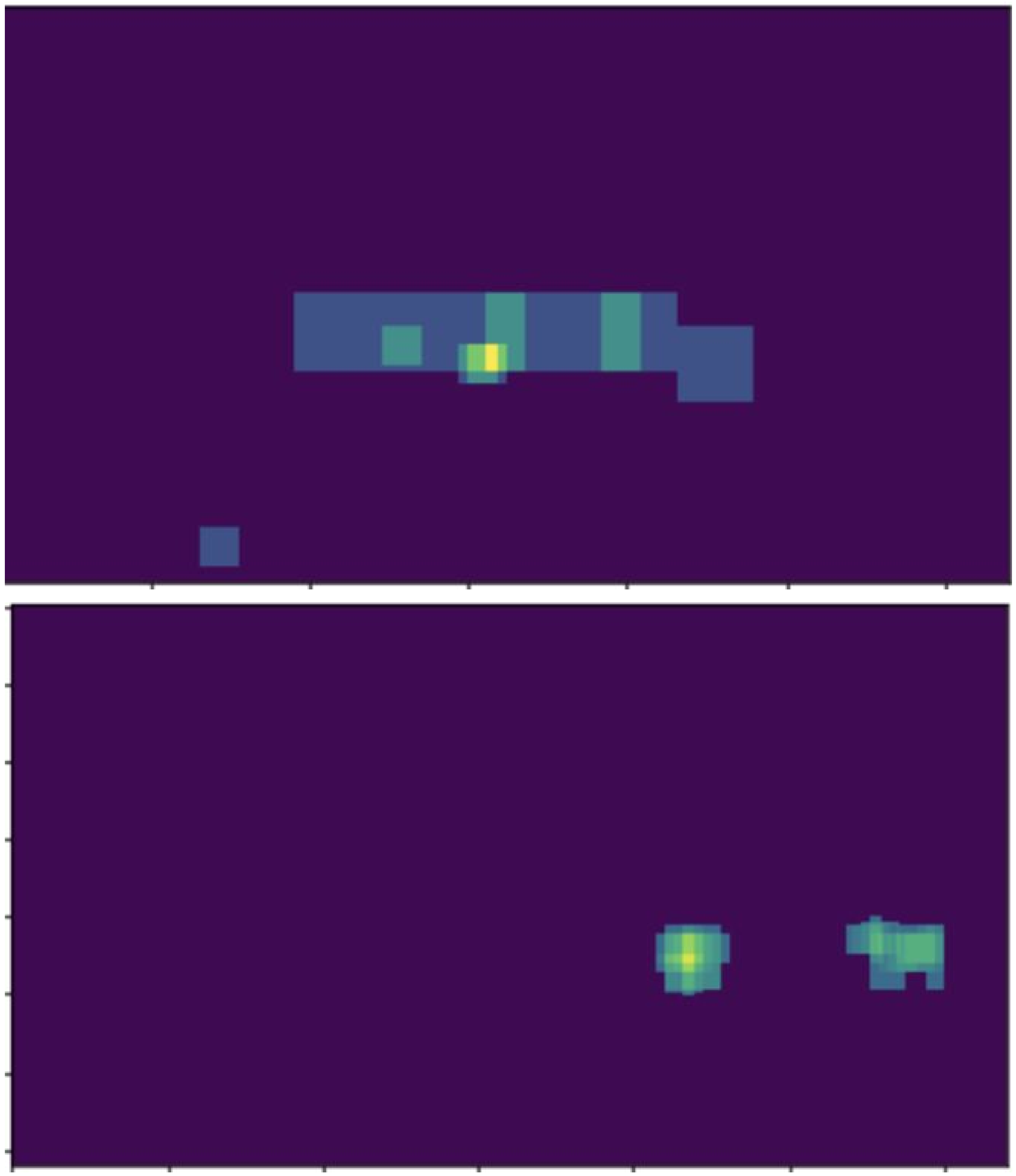
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

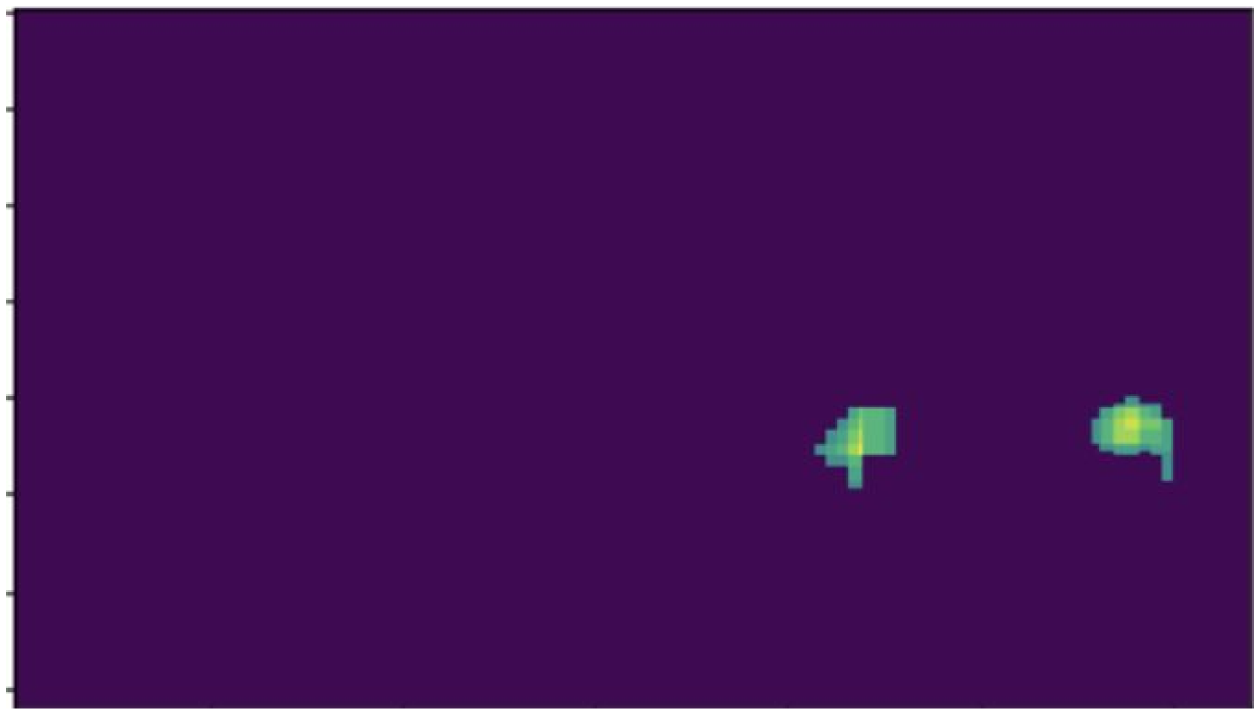
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob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

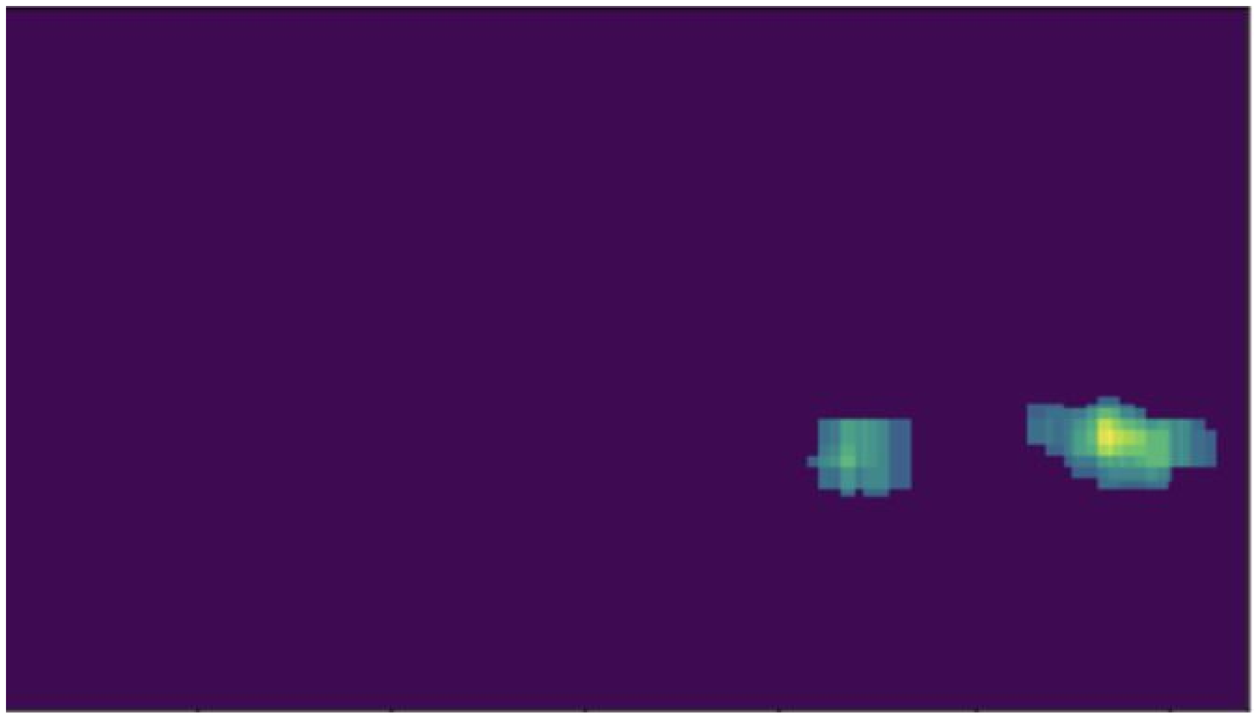
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

Here are six frames and their corresponding heatmaps:

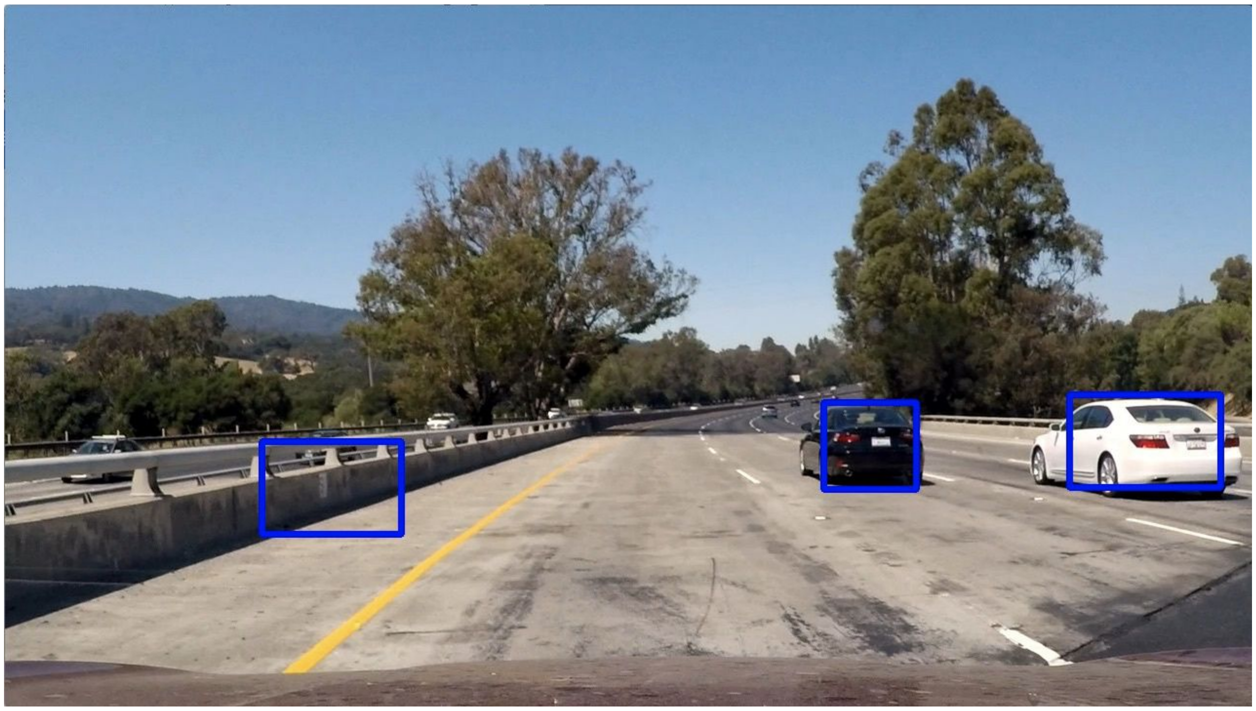


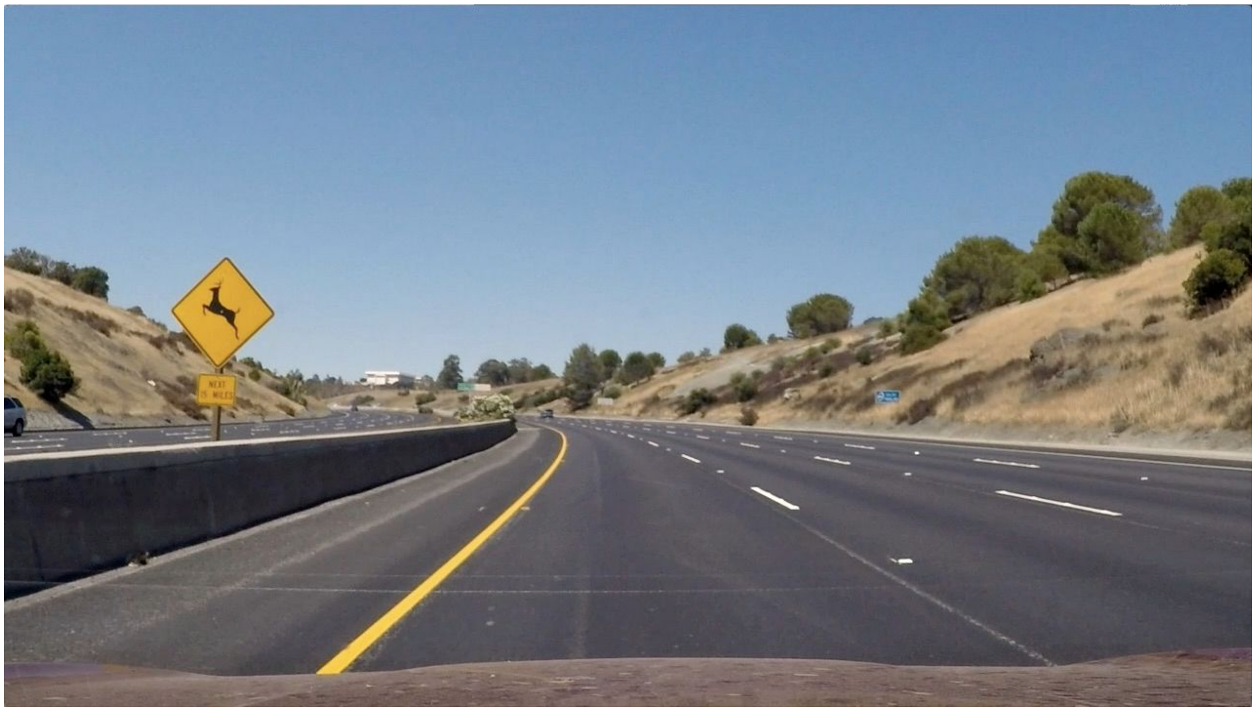


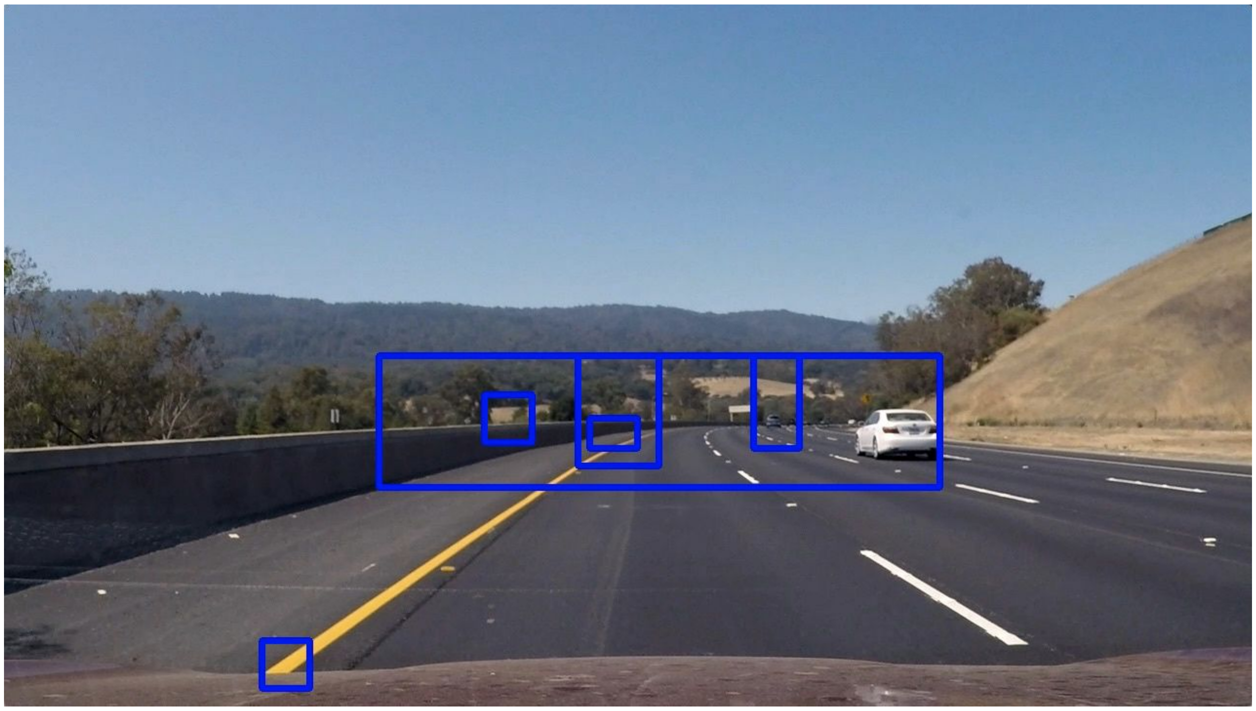
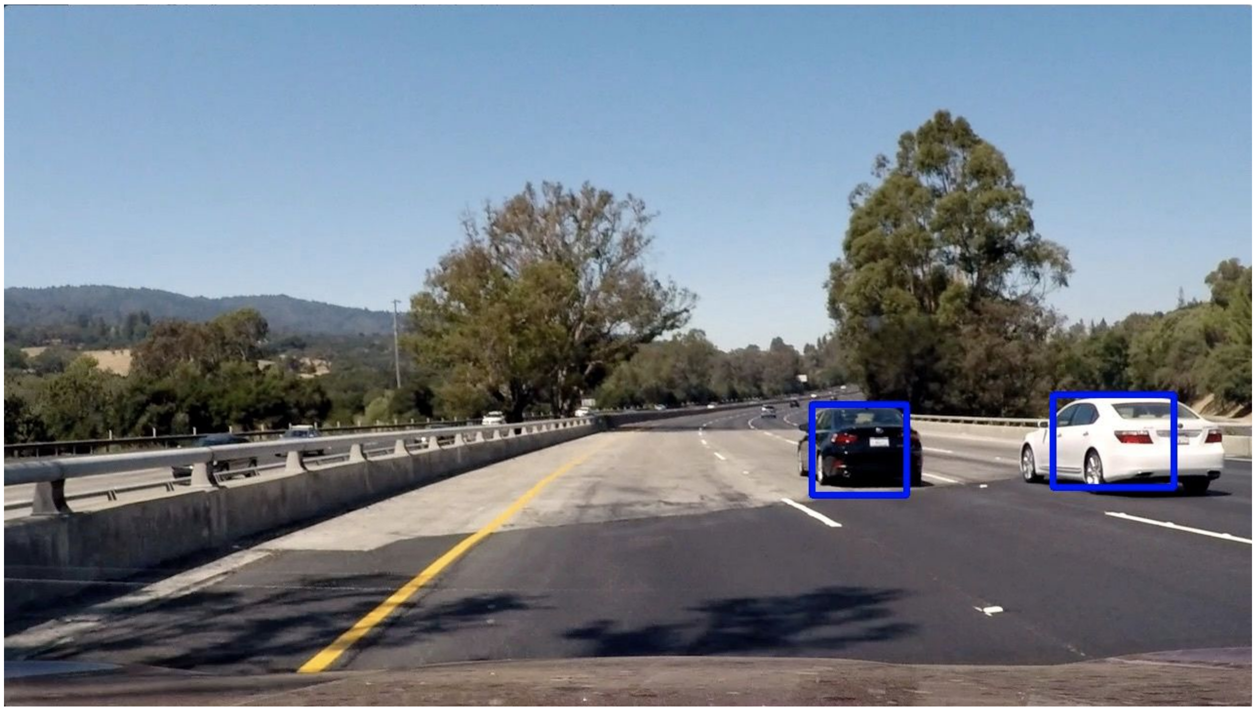


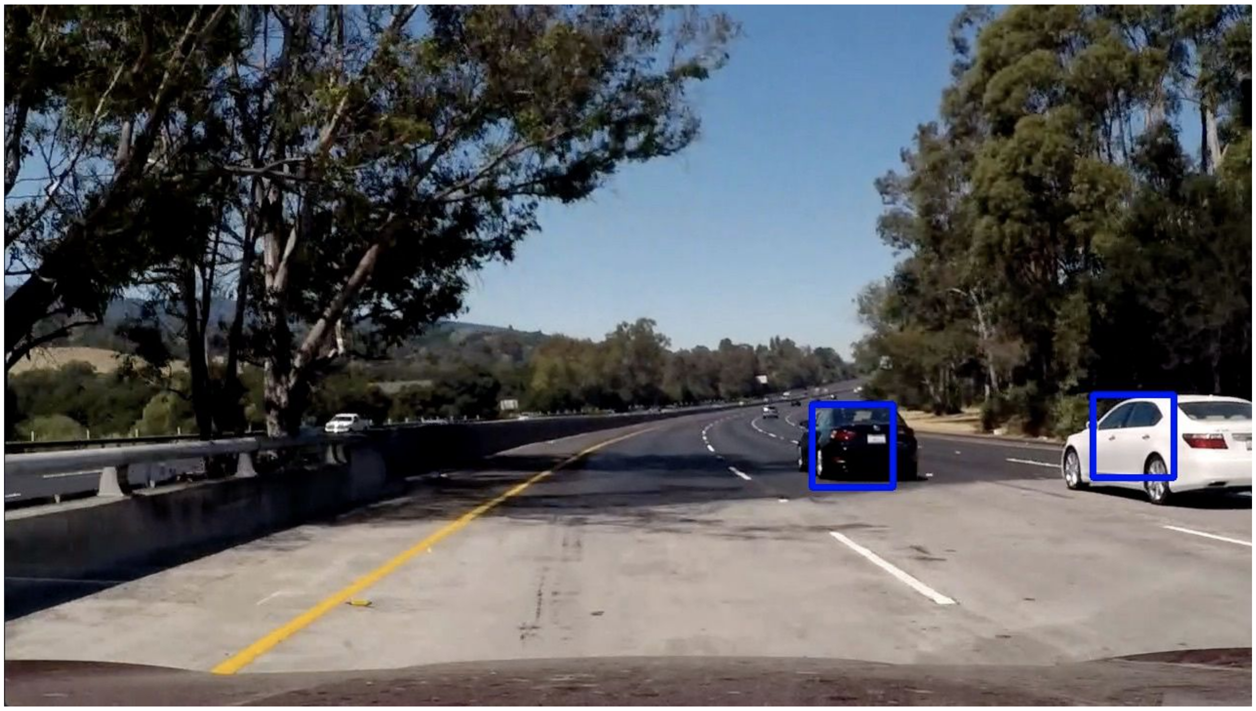


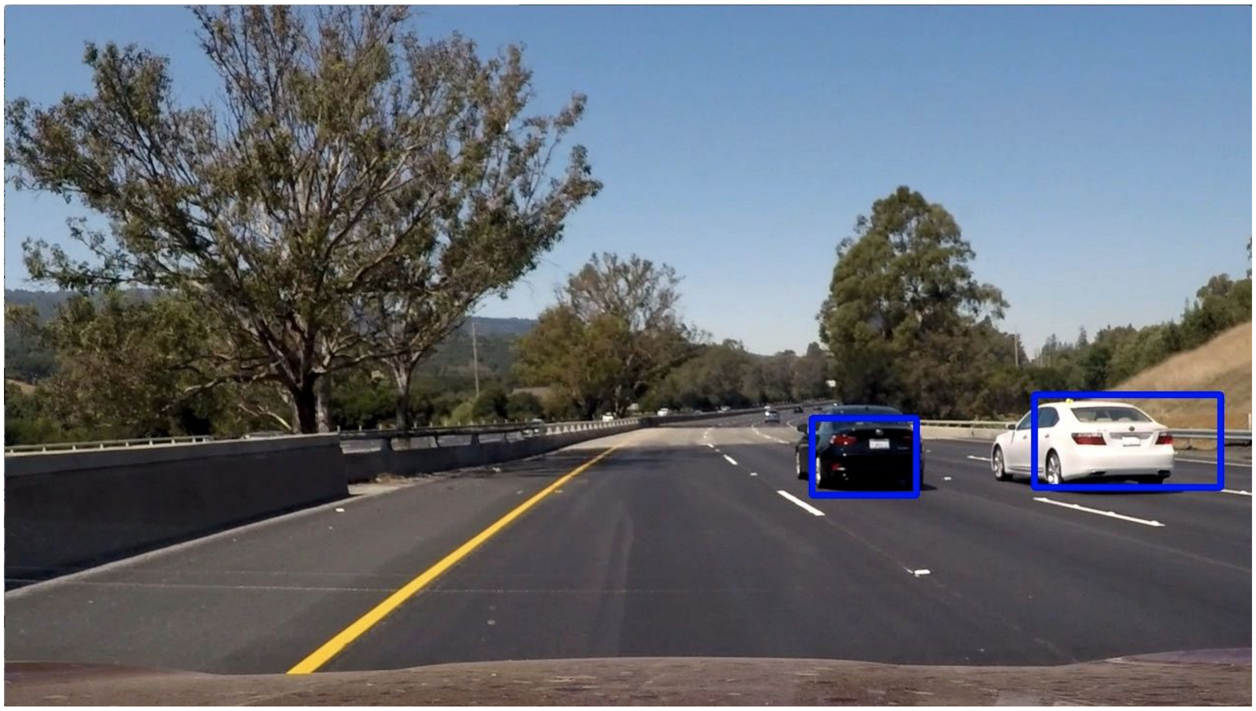
Here the resulting bounding boxes are drawn onto the last frame in the series:




NEXT
15 MILES





## 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?

There are a couple of issues with the pipeline.

1. Computational efficiency
2. Precision
3. Recall

One of the major issues I faced was how to balance speed vs accuracy. While it was possible to get better accuracy by doing a more exhaustive sliding window search, it is not a good idea for online code.
My code in its current form is unfit for running using online. However a number of optimization techniques like pruning and caching can be used to improve performance here.

As for precision and recall go, it was important to favour recall as precision can be improved by observing actors over multiple cycles, however failing identify an actor is far worse than false positives now and then.