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.

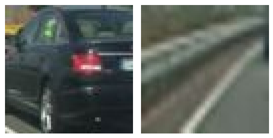
Writeup / README

Histogram of Oriented Gradients (HOG)

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

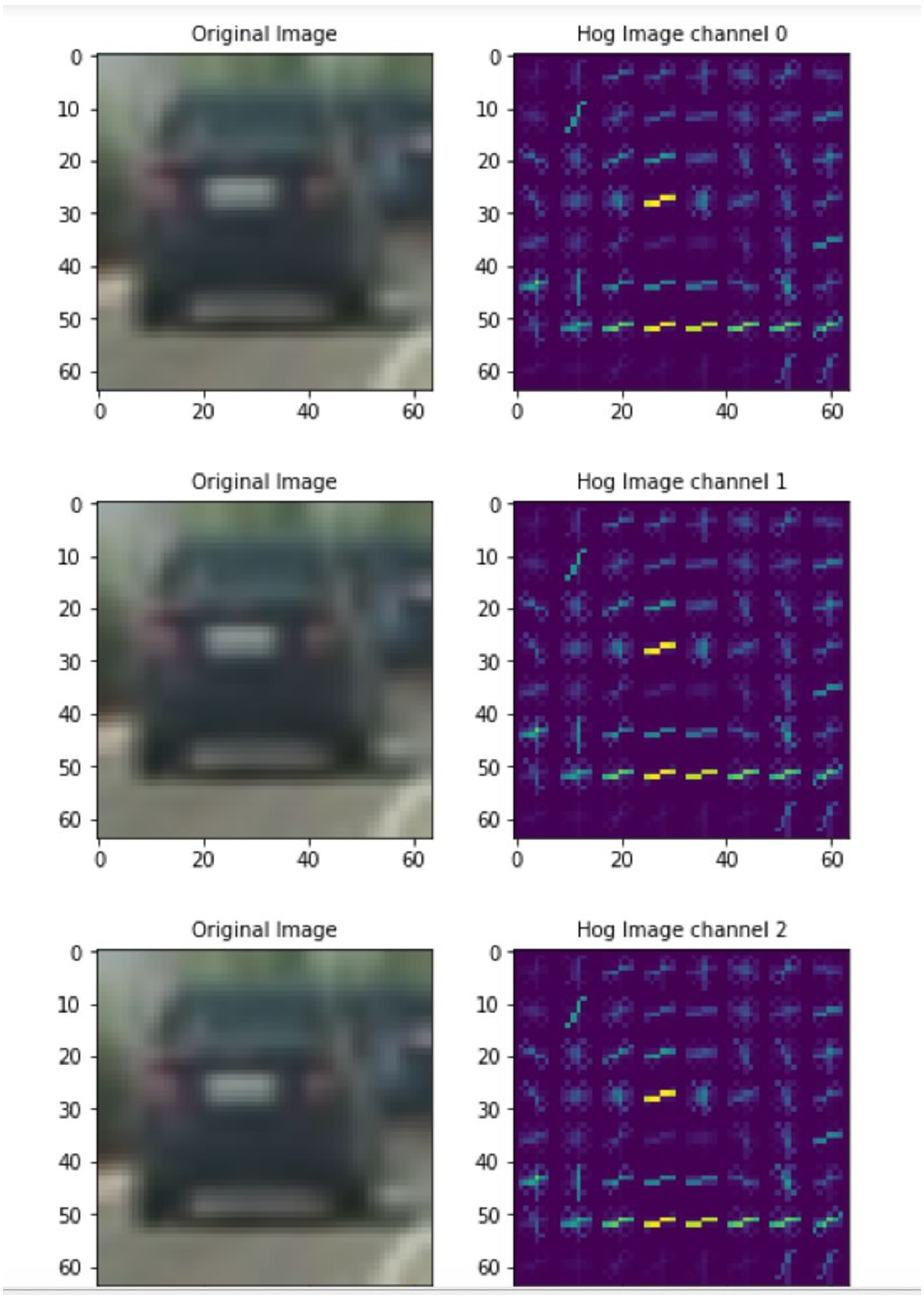
The code for this step is contained in the first code cell of the IPython notebook (cell titled **"hog images"**).

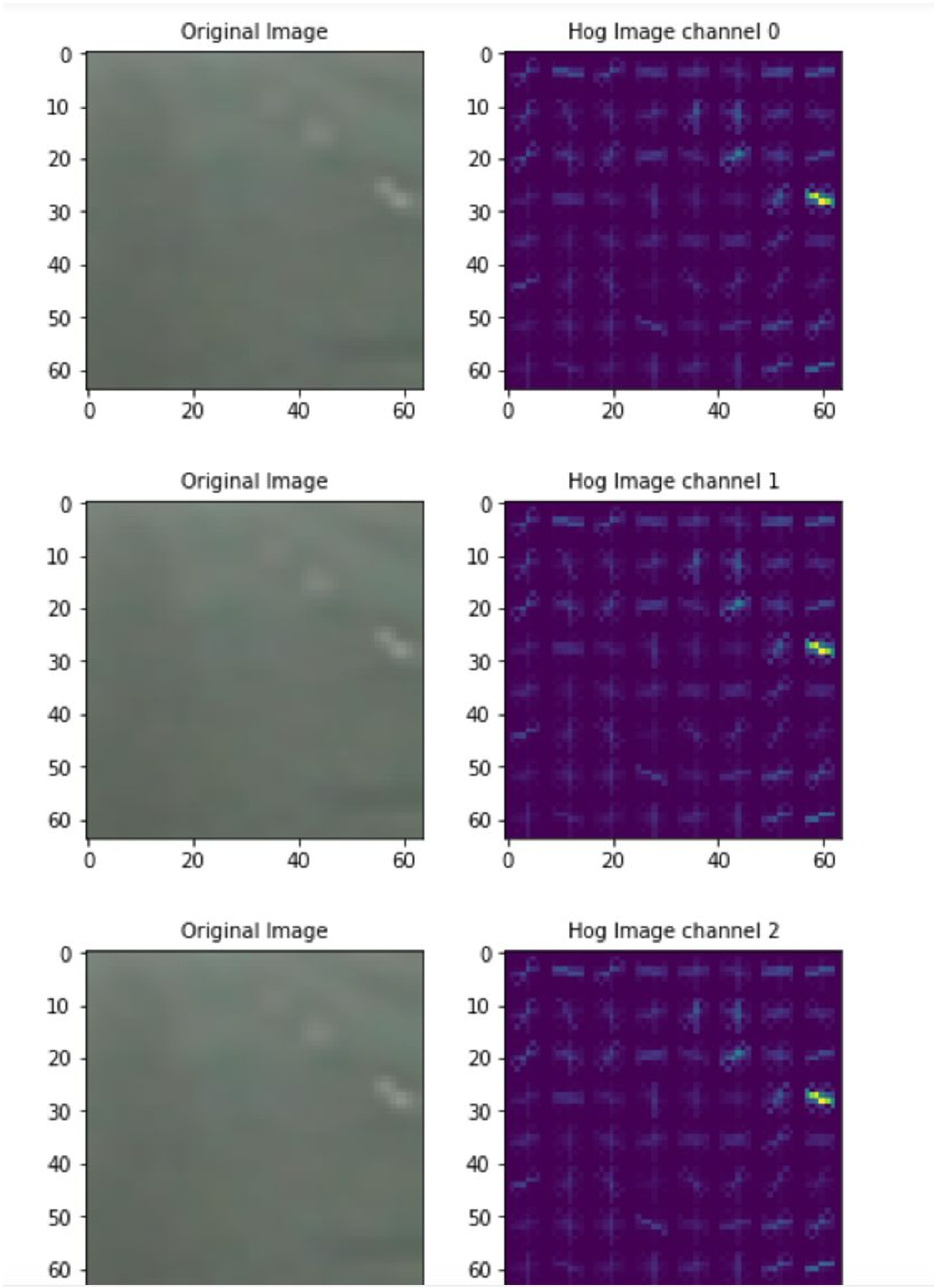
I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



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. Code is present in cell titled **"hog images".**

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)`, for a car a non-car image:


Original Image
Hog Image channel 0
Original Image
Hog Image channel 1
Original Image
Hog Image channel 2
0
10
20
30
40
50
60
0
20
40
60


Original Image
Hog Image channel 0
Original Image
Hog Image channel 1
Original Image
Hog Image channel 2

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

I tried various combinations of parameters, mostly color space and number of HOG channels. COnsidering all HOG channels performed better than just individual ones, so I went with the ‘ALL’ option.

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 following parameter setup:

```
color_space = 'HSV' # Can be RGB, HSV, LUV, HLS, YUV, 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 = (32, 32) # (16, 16) Spatial binning dimensions
hist_bins = 32    # 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
```

Code is present in cell titled **“train model**”, along with helper functions for hog feature detection, color histogram feature detection, and spatial detection (image resizing), in first cell titled **helper functions**. I achieved a 0.985 accuracy:

```
Using spatial binning of: 32 and 32 histogram bins
Feature vector length: 8460
17.99 Seconds to train SVC...
Test Accuracy of SVC =  0.985
My SVC predicts:  [ 1.  0.  1.  1.  0.  0.  0.  0.  1.  1.]
For these 10 labels:  [ 1.  0.  1.  1.  0.  0.  0.  0.  1.  1.]
0.00508 Seconds to predict 10 labels with SVC
```

Note that was accomplished with an HSV color space, and that was the space that achieved the highest accuracy for me. However, down the line after performing window search, as it was not working well for me, I saw that the YCrCb color space provided better results, so I retrained the classified with this new color space. Accuracy in this case was 0.9724

```
Using spatial binning of: 32 and 32 histogram bins
Feature vector length: 8460
16.3 Seconds to train SVC...
Test Accuracy of SVC =  0.9876
My SVC predicts:  [ 0.  1.  1.  1.  0.  1.  1.  1.  0.  0.]
For these 10 labels:  [ 0.  1.  1.  1.  0.  1.  1.  1.  0.  0.]
0.02292 Seconds to predict 10 labels with SVC
```

(and after retraining with full non-vehicle images, accracy improved to 0.987)

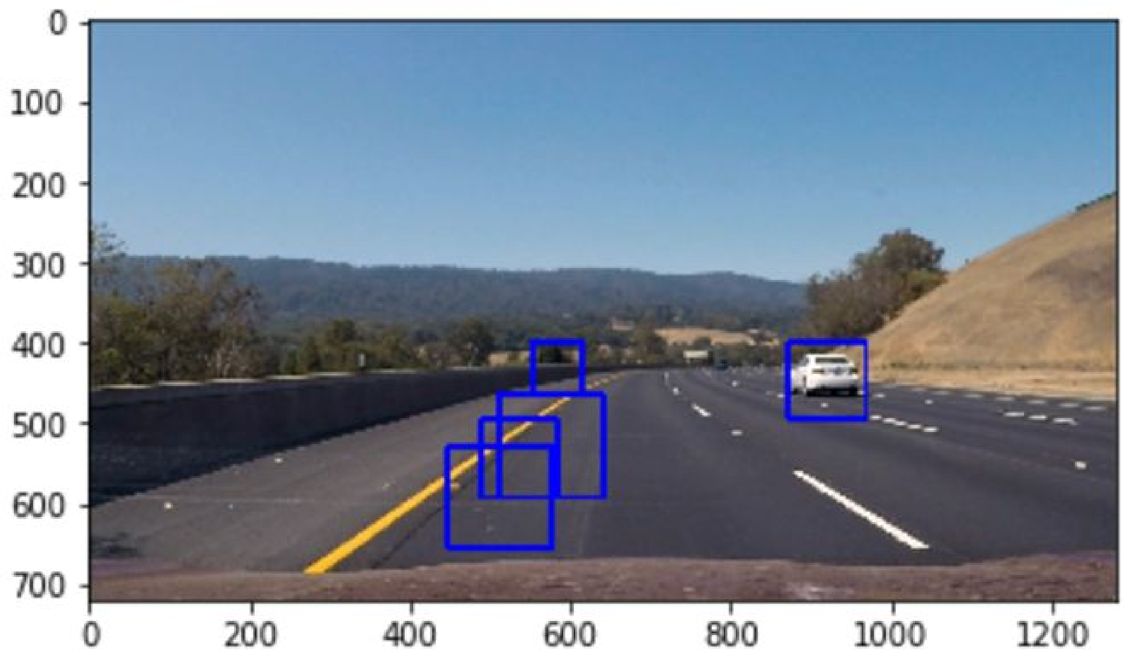
Sliding Window Search

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 decided to search at three scales, with larger windows for closer vehicles, and smaller windows for cars located further away. I only consider the area where cars could be located (y range between 400 and 720 pixels), and I further adjusted the y range of each scale, e.g., cars far way will be closer to the horizon and thus they will not be located close the bottom of the images. These are the param used:

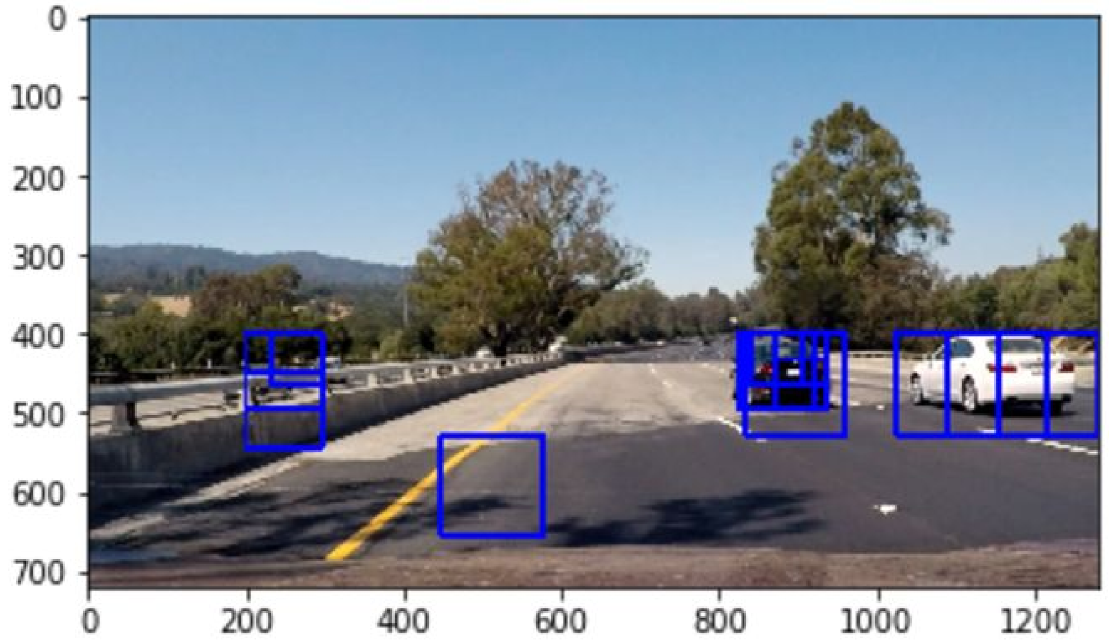
```
x_start_stop = [(0, 1280),(200, 1080),(200, 1080)]
y_start_stop = [(400,656), (400,600), (400,530)]
xy_window = [(128,128), (96, 96), (64,64)]
```

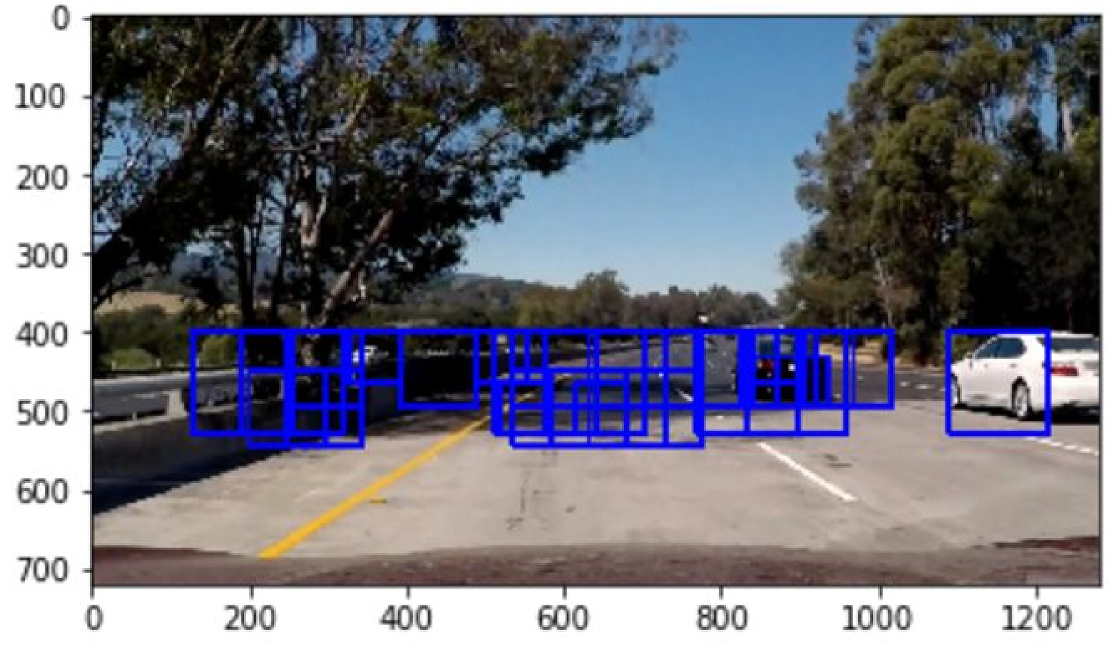
Code is defined in cell **“Search cars in images using specified list of windows”**, and here is an example image:

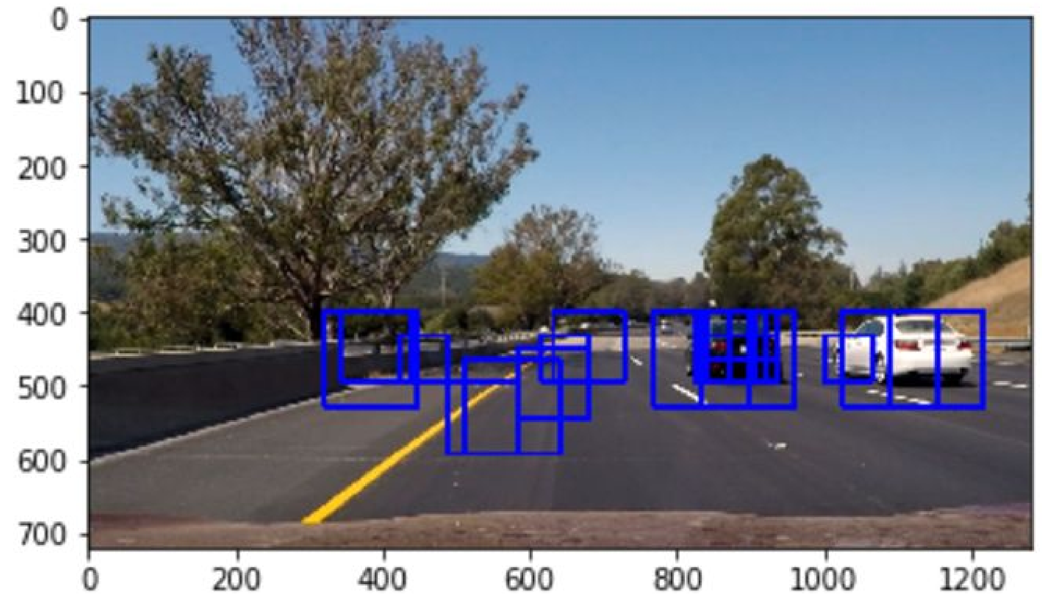


2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

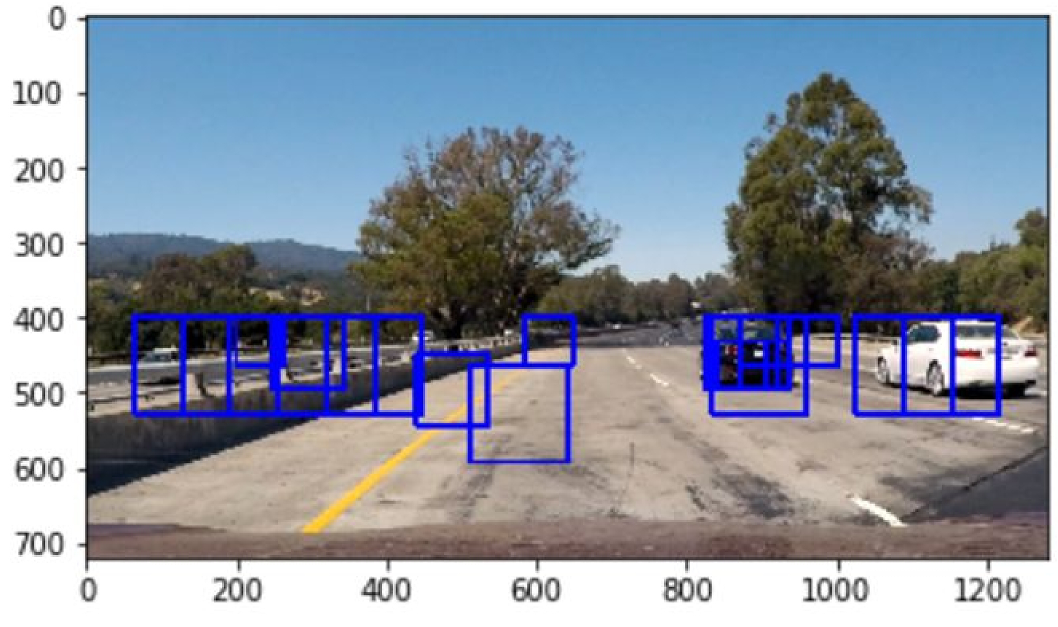
Ultimately I searched on three scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some additional example images:

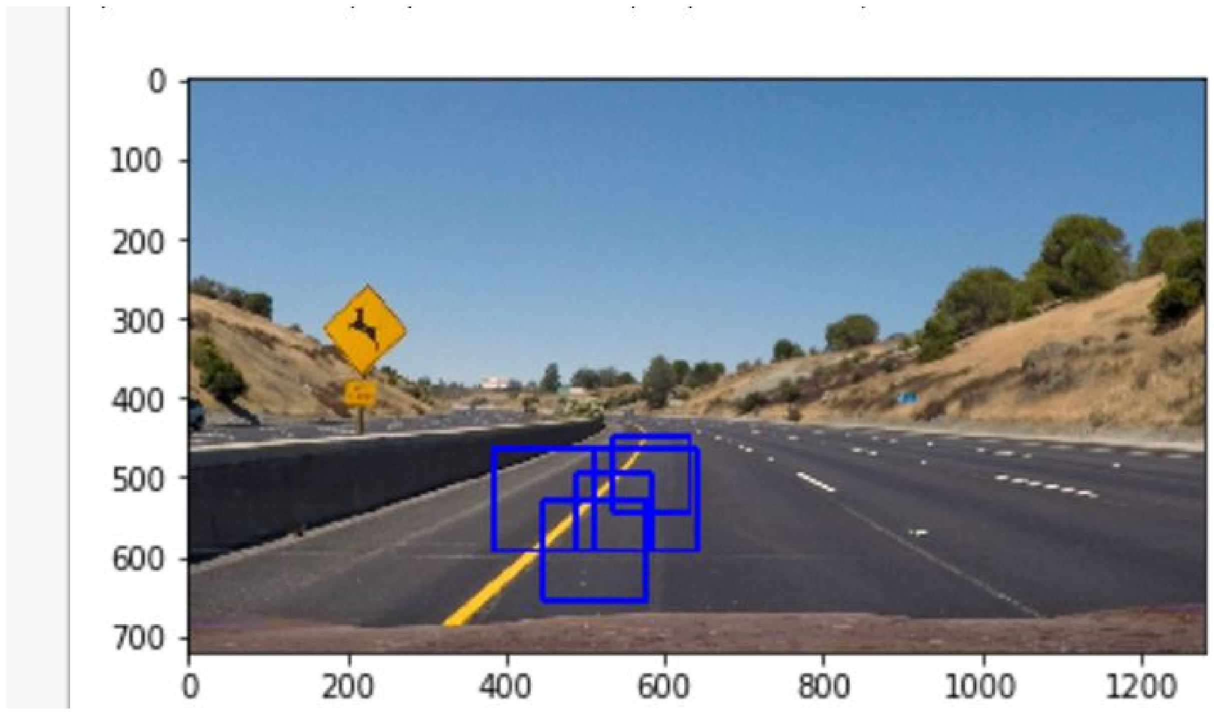

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


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



```
[illegible]
```





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.)

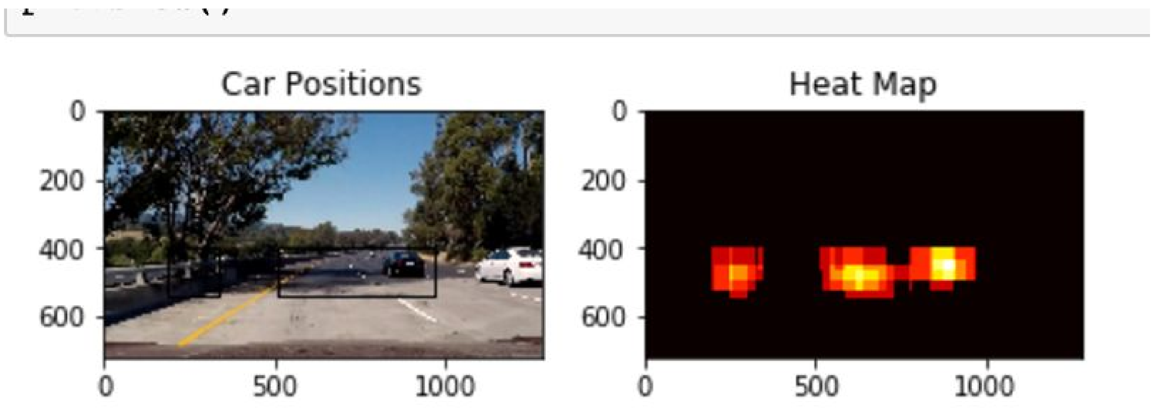
Video available at: https://youtu.be/b0OY084fcvc

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.

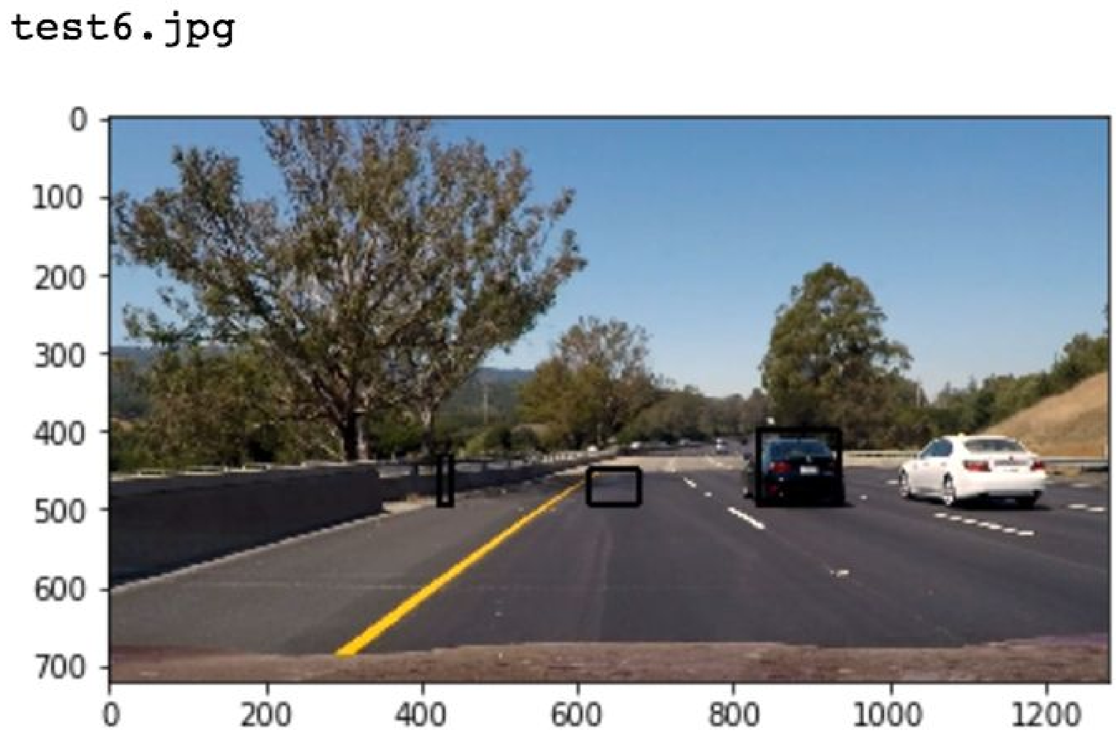
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob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

I also built a variation of the heatmap, where I recorded windows for N consecutive frames, and applied a higher threshold over the N windows that the threshold applied to a single frame across the three scales I searched on. The class that implements the recording of windows across multiple frames is “Car_Windows”, defined in the corresponding cell.

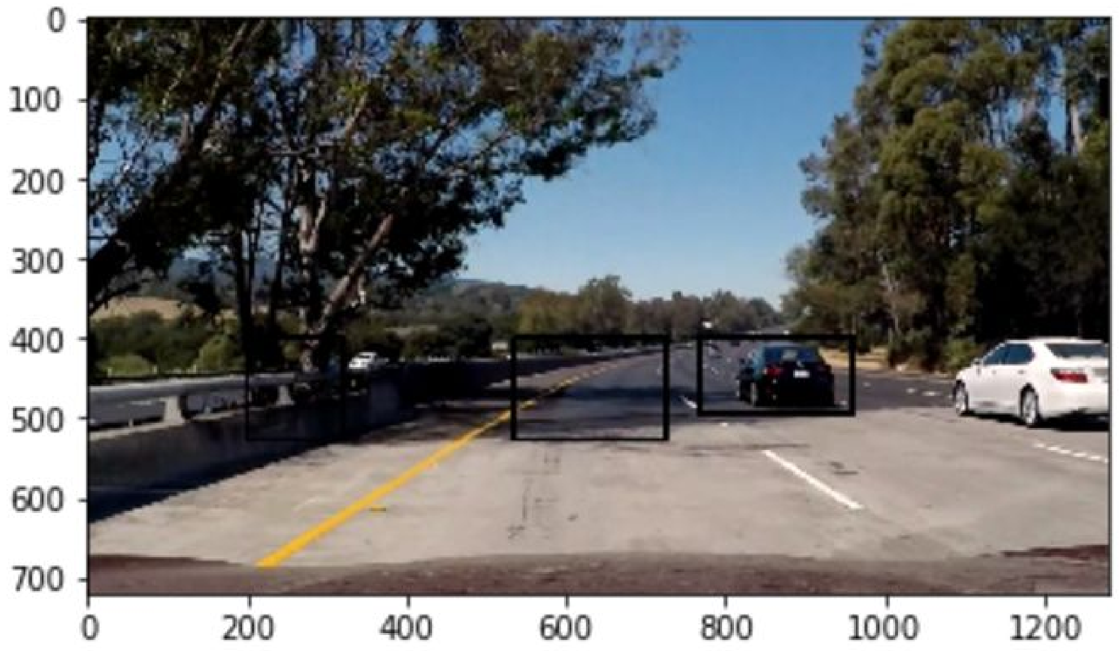
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:



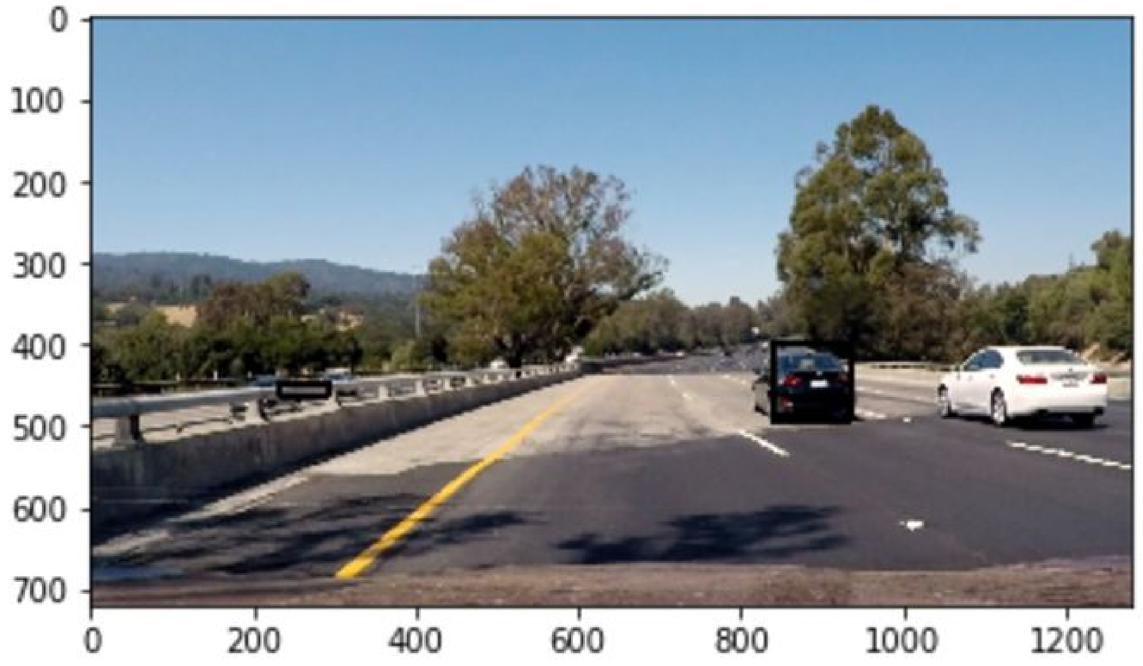
Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames:



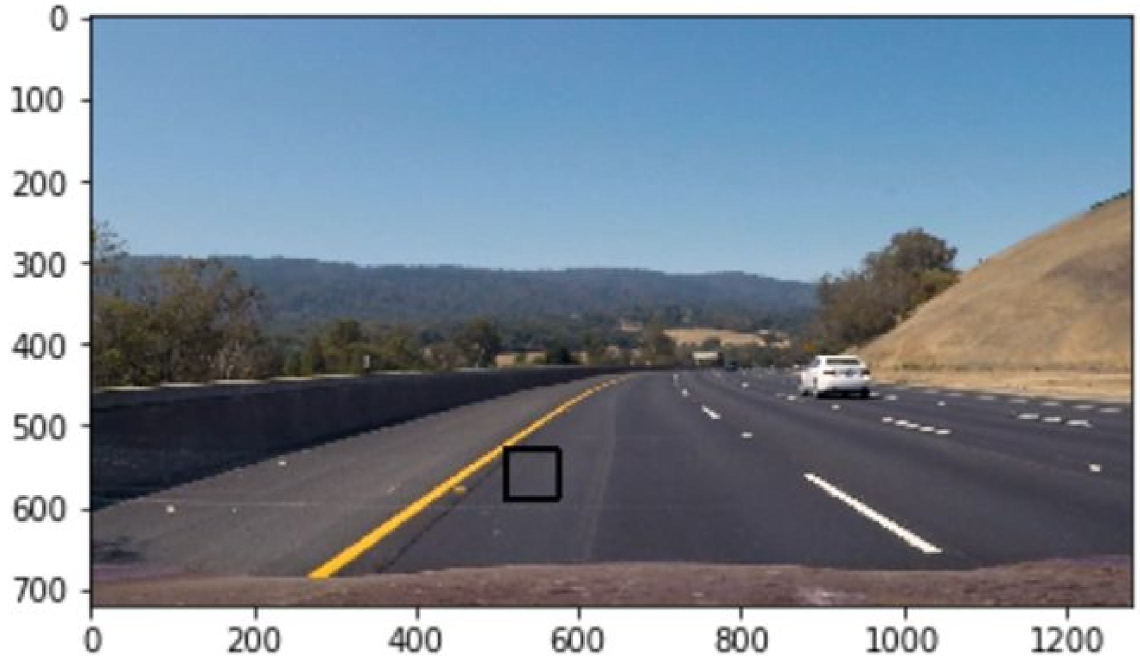
test5.jpg



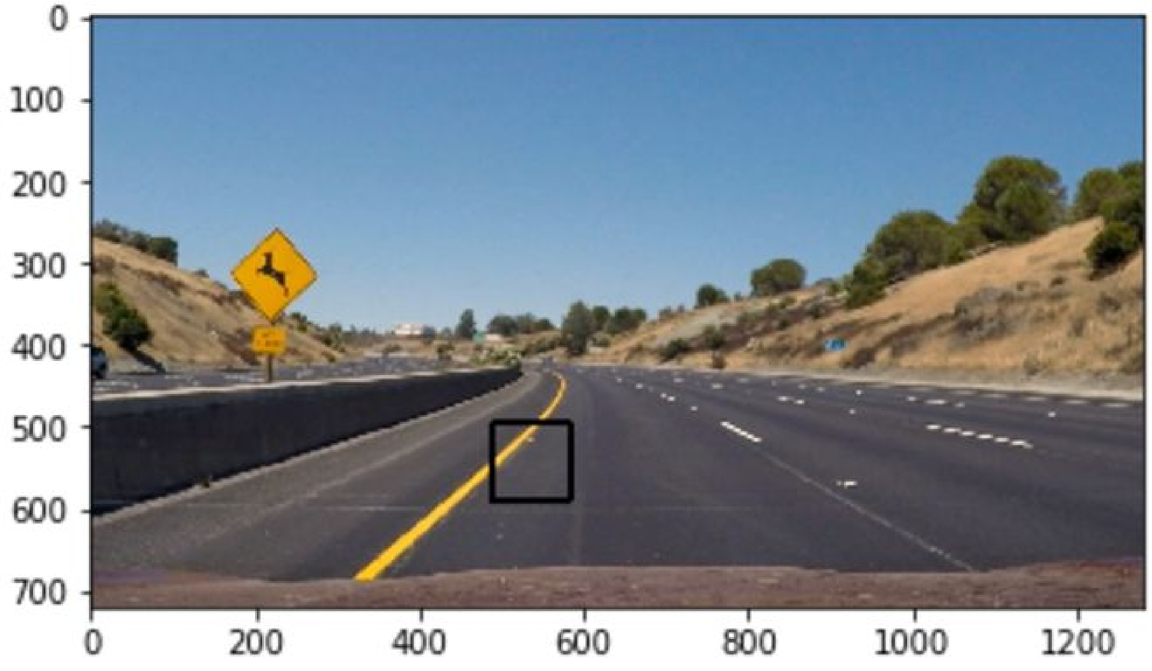
test4.jpg



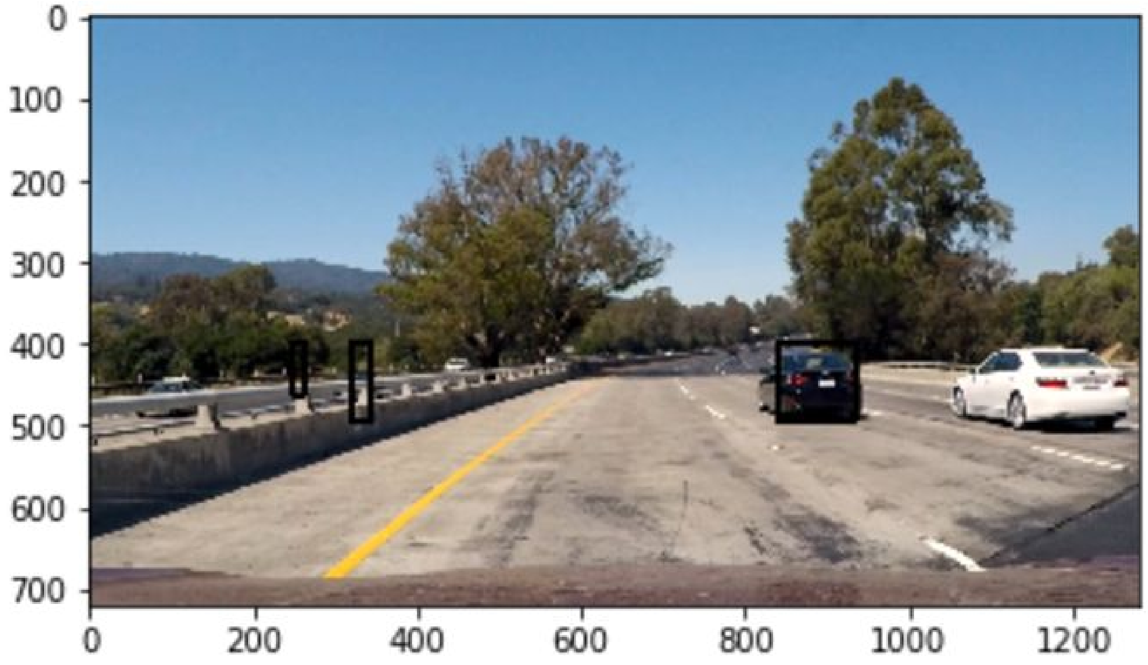
test3.jpg



test2.jpg



test1.jpg



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/ I missed this comment:

```
# Uncomment the following line if you extracted training
# data from .png images (scaled 0 to 1 by mpimg) and the
# image you are searching is a .jpg (scaled 0 to 255)
image = image.astype(np.float32)/255
```

And it took me some debugging until I realised how to work with jpg images after training with png images. E.g., the pipeline needs to work with the scaled image, but the image (and video) that gets displayed needs to use the unscaled version to be visible.

2/ I have not been able to detect the white car reliably. I thought it was related to the color space I was using, but I tried with all of them and none was successful.

Two issues were affecting this situation:
a/ I was not training with the full set of non-vehicle images, as the first reviewer pointed out to me. I was missing the ~5k coming from the Extra folder, and that was leading to too false positives in road-only windows.

(test accuracy improved from 0.972 to 0.987. One reason I probably missed earlier that I was training with a subset of the data is that accuracy wa already pretty high)

b/ I considered windows from past frames *before* searching windows, not after. Therefore, even though I was thinking I was applying a heatmap considering hot windows from the past, I was instead only considering the hot windows from the past frame . The first reviewer suggested an implementation of the inter-frame heatmap, and when I went to apply it I realised I was applying in the wrong position in the pipeline. I have modified the Car_Windows class with the implementation suggested by the reviewer, and invoked in the right place

After both changes, results look better: https://youtu.be/b0OY084fcvc