Self-Driving Car Engineer Nanodegree Program

Vehicle Detection and Tracking

John Reilly

Due date July 2018

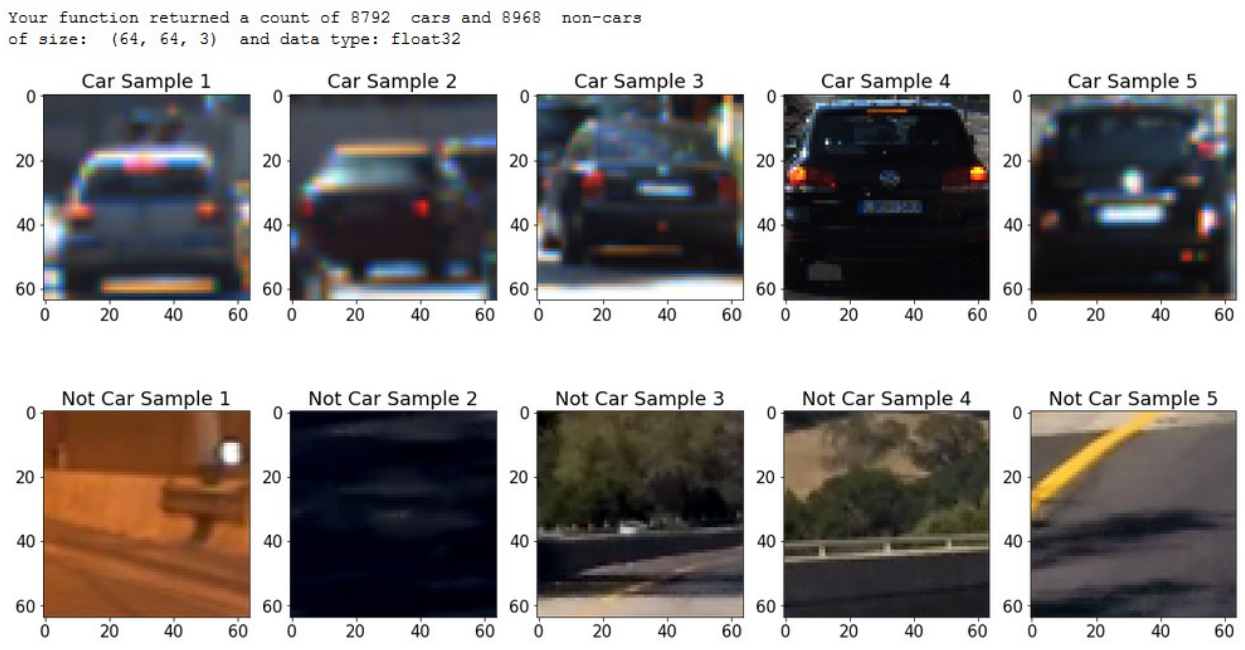
<u>Table of Contents:</u>

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labelled 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.

Data Set Overview:

The Data Set provided by Udacity was examined to get an overview. Many random samples were viewed and a selection of samples are presented. The data is of two general types pictures of Cars and Pictures of things that are not cars and this data is then fed into a Linear SVM Classifier. Pictures are 64*64 Pixels with 8792 examples of cars and 8968 of non-cars. Random samples generated from code Cell #1 below.

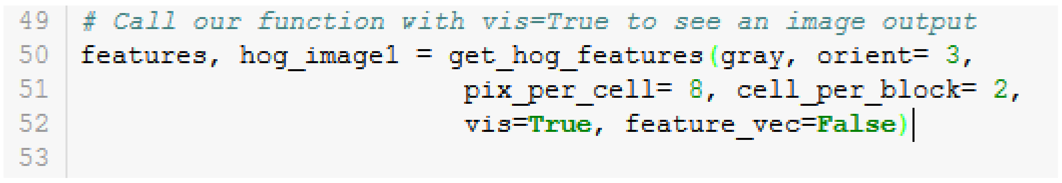


```
Your function returned a count of 8792  cars and 8968  non-cars
of size:  (64, 64, 3)  and data type: float32
```

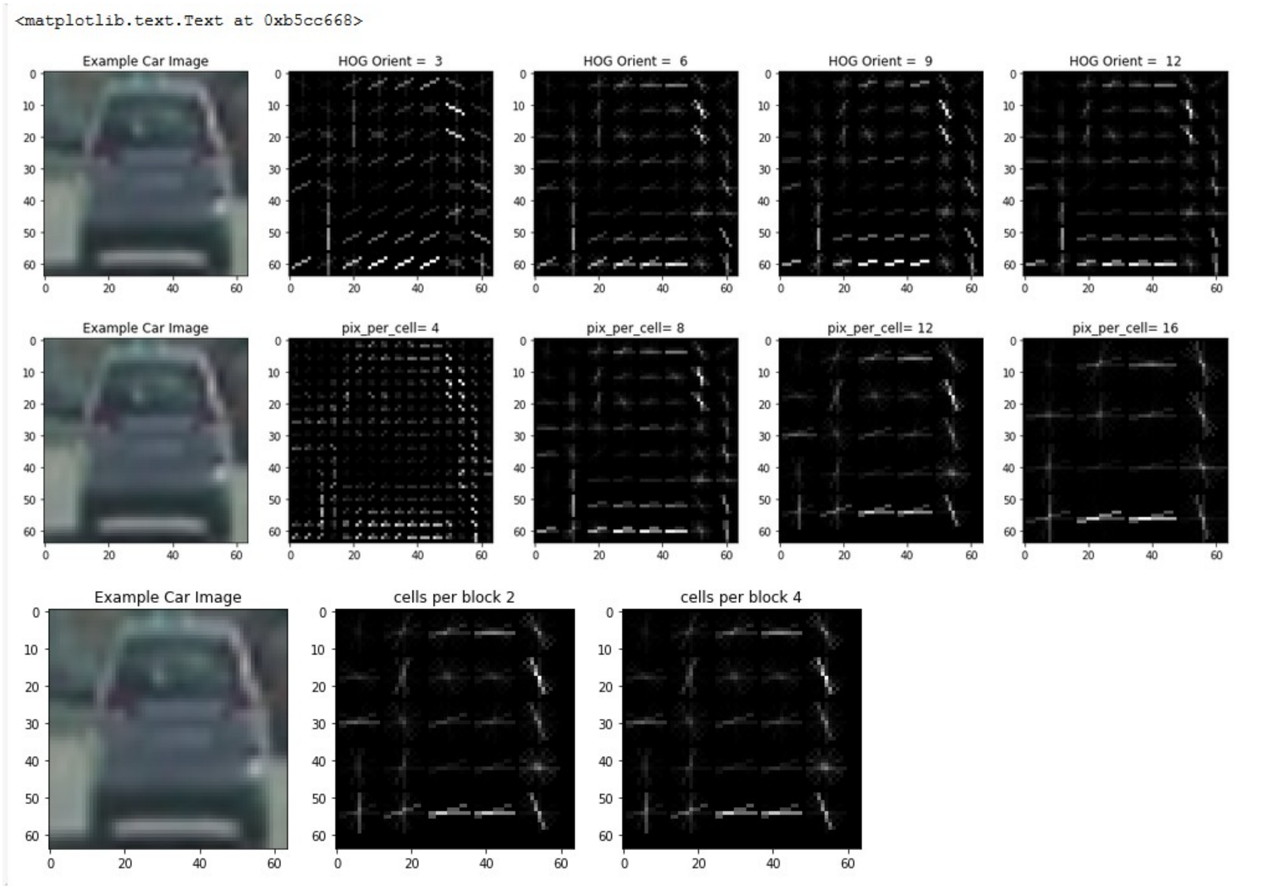
Histogram of Oriented Gradients (HOG)

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

The function for processing HOG features was based on the Udacity supplied function “get_HOG_features”. This function was the basis for experimentation. Orientation, pixels per cell and Cells per block were all varied to see results using the parameters as shown below. Output from Cell #2 below.



```python
# Call our function with vis=True to see an image output
features, hog_image1 = get_hog_features(gray, orient= 3,
                        pix_per_cell= 8, cell_per_block= 2,
                        vis=True, feature_vec=False)

```



```
<matplotlib.text.Text at 0xb5cc668>
```

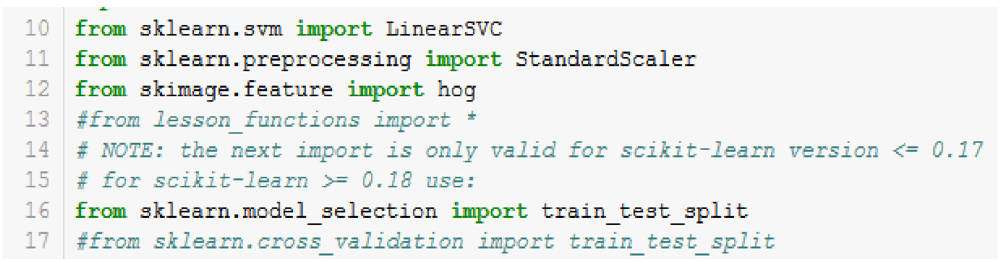
Sample outputs from the get_HOG_features function are shown below. Subjectively to the human eye Orientation of 12 is best but this is contradiction to a suggestion by a teacher in the Q+A video. He said that typically above 9 there is not much improvement and quoted a research paper. Note how the pixels at the lower part of the image in HOG Orient 12 form the most straight line of the samples for the corresponding straight line of the bottom of the car between the wheels. The question is which worked the best which is explored later. Similarly 16 and 12 seem better than 4 and 8 but this is subjective and the results may not correspond.

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

Having experimented with the various options I personally wanted to go with Orientation 12 and 16 pixels per cell and 2 cells per block but I did extensive experimentation much of it on the full video to see if what appealed to my judgements also worked well in reality and also to be thorough in my work. Higher Orientations led to longer processing times which was a major reason to go with as low as level orientation as was effective. Comparisons of results presented below.

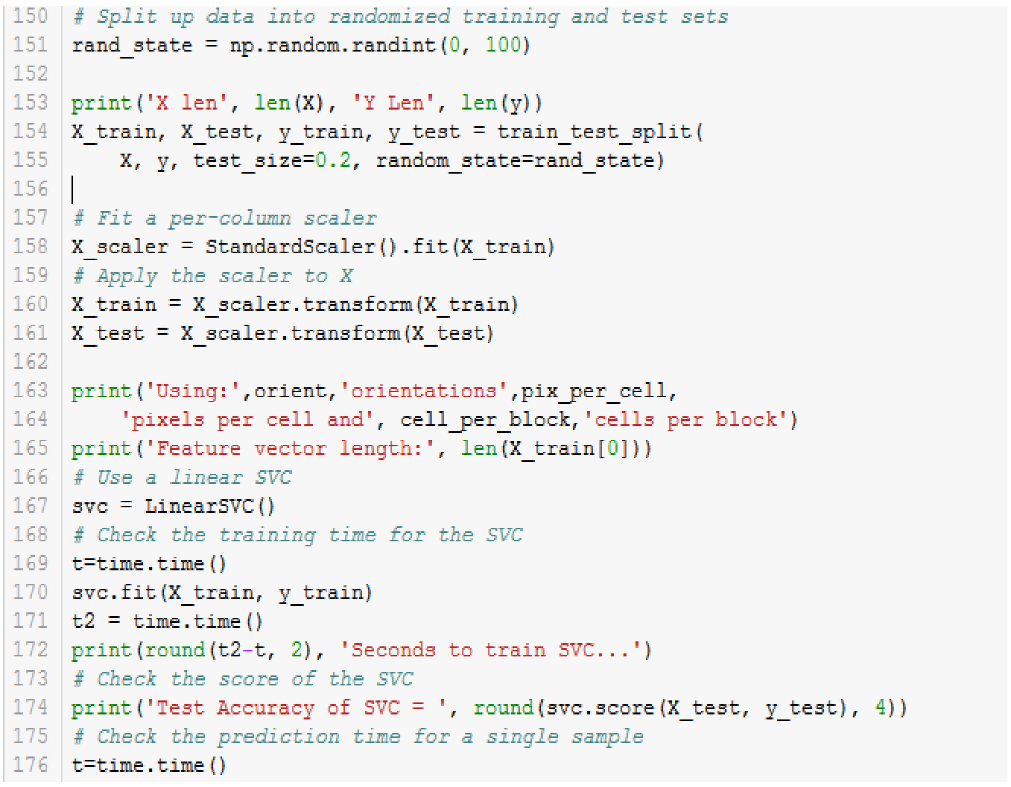
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVM using... *functions bin_spatial , color_hist and extract_features* were functions used supplied by Udacity also used was a Linear SVC from Sklearn and a train test split function also from SK learn library.



```python
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
#from lesson_functions import *
# NOTE: the next import is only valid for scikit-learn version <= 0.17
# for scikit-learn >= 0.18 use:
from sklearn.model_selection import train_test_split
#from sklearn.cross_validation import train_test_split
```

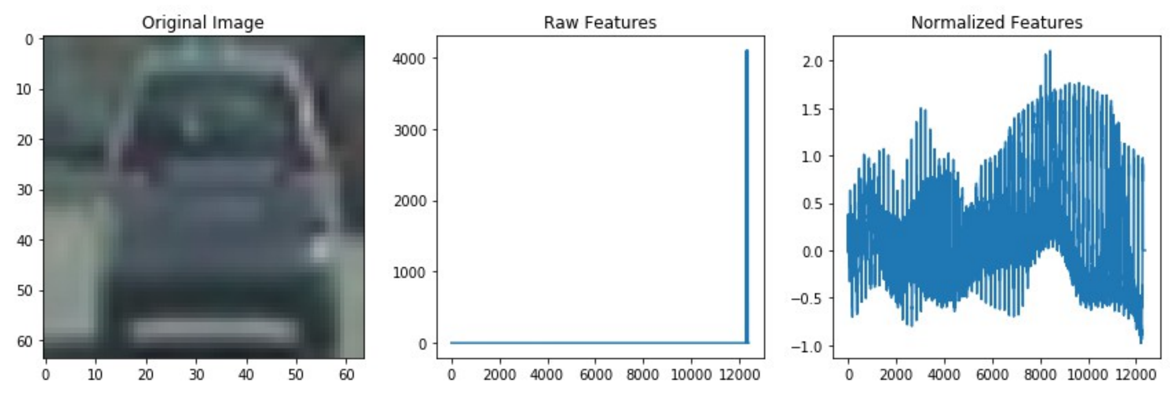
First the data sets were split into train and test subsets and then the Linear SVC was trained. A timer was applied and the results printed to the output. Code from Udacity



```python
# Split up data into randomized training and test sets
rand_state = np.random.randint(0, 100)

print('X len', len(X), 'Y Len', len(y))
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=rand_state)

# Fit a per-column scaler
X_scaler = StandardScaler().fit(X_train)
# Apply the scaler to X
X_train = X_scaler.transform(X_train)
X_test = X_scaler.transform(X_test)

print('Using:',orient,'orientations',pix_per_cell,
    'pixels per cell and', cell_per_block,'cells per block')
print('Feature vector length:', len(X_train[0]))
# Use a linear SVC
svc = LinearSVC()
# Check the training time for the SVC
t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
# Check the prediction time for a single sample
t=time.time()
```

Colour features were used and explored in Cell # 3 with output below

```
cars len 8700
```

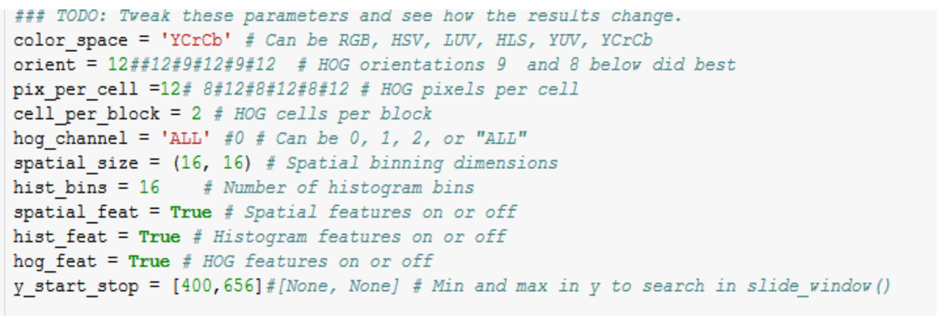


Sliding Window Search

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 used the Udacity provided code for the function "search_windows" and "single_image_features". I experimented with the below parameters to generate test results. I used the "Test Accuracy of SVC" percentage generated by the classifier to assess the results and also and in fact more importantly I visually assessed the results to decided which worked the best.

The parameters that made the biggest difference were the color_space with 'YCrCb' being the best on visual inspection. Orientation of 12 was best but slower and not significantly better on visual inspection. There was a little difference between pix_per_cell of 8 and 12 but speed was slowed down. Hog_Channel seemed to work better on 'All'. Spatial_size and hist_bins were left at 16 as they didn't seem to make much difference. Spatial_feat, hist_feat and hog_feat were all set at True ans they were all better than not using them but most so Hog_feat. y_start and y_stop were important , these set the range that the sliding windows checked. Often false positives were found in the sky and trees etc, by limiting the search to the area that cars actually were this greatly reduced false positives and greatly improved the speed of the process. In this way ignoring half the image reduced the speed by half.



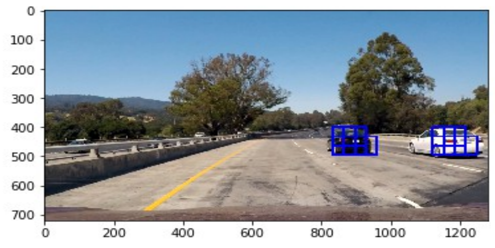
```
### TODO: Tweak these parameters and see how the results change.
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 12##12#9#12#9#12  # HOG orientations 9  and 8 below did best
pix_per_cell =12# 8#12#8#12#8#12 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' #0 # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 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
y_start_stop = [400,656]#[None, None] # Min and max in y to search in slide_window()
```

Outputs such as the below were used to determine the best way forward. This shows 12

Orientations 12 pixels per cell and 2 cells per block. These setting were used on the final videos and seemed to produce the best results but were very slow to process. Output from Cell #5 Below

```
cars len 8700 notcars len 8700
X len 17400 Y Len 17400
Using: 12 orientations 12 pixels per cell and 2 cells per block
Feature vector length: 3120
15.9 Seconds to train SVC...
Test Accuracy of SVC =  0.9888
```

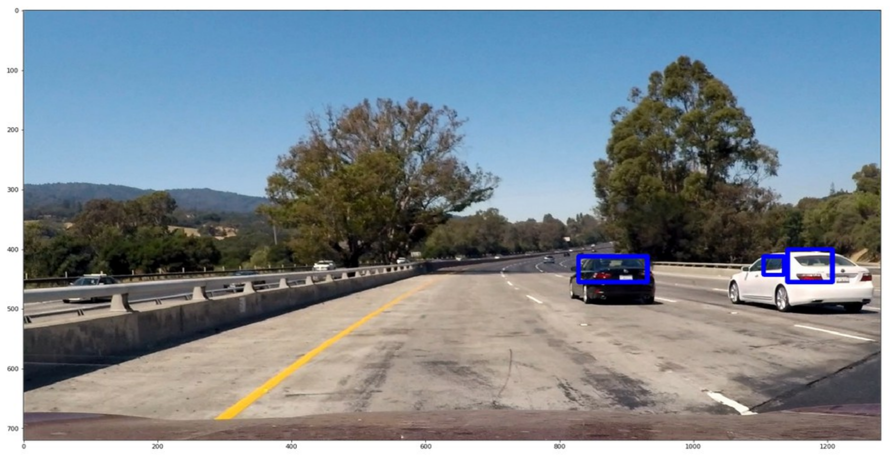
```
Out[8]: <matplotlib.image.AxesImage at 0xfa290f0>
```



After experimenting with the above I used the Udacity provided code to sample the entire image and do HOG on the entire image and then subsample it rather than the previous way of breaking up the image and then running HOG on each window. This was achieved with the function *find_cars* which took as a parameter the value “scale”. A variety of scales were used to generate windows of varying sizes. A scale of 0.5 did give some good results on smaller cars in the distance but the smaller windows took a lot of time to process and generated too many false positives despite many attempts to address this. Scales of 1.0 gave good results for most of the image and cars and scales of 1.5 worked best for large cars close to the camera. I tired to get best results by using 3 scales of 0.5 and 1.0 and 1.5 but this generated to many false positives so I used just 1.0 and 1.5 for most of the experiments.

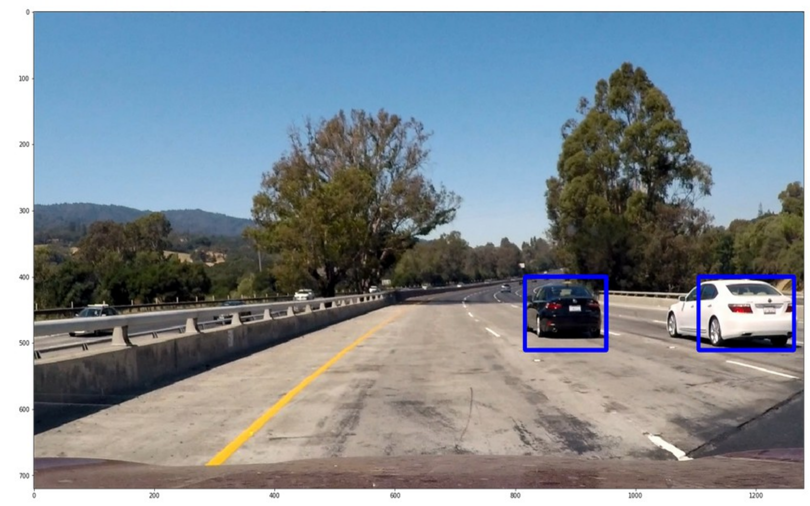
Below output shows scale 0.5, note the boxes are small with more than one box per car.

```
Out[32]: <matplotlib.image.AxesImage at 0x31943048>
```



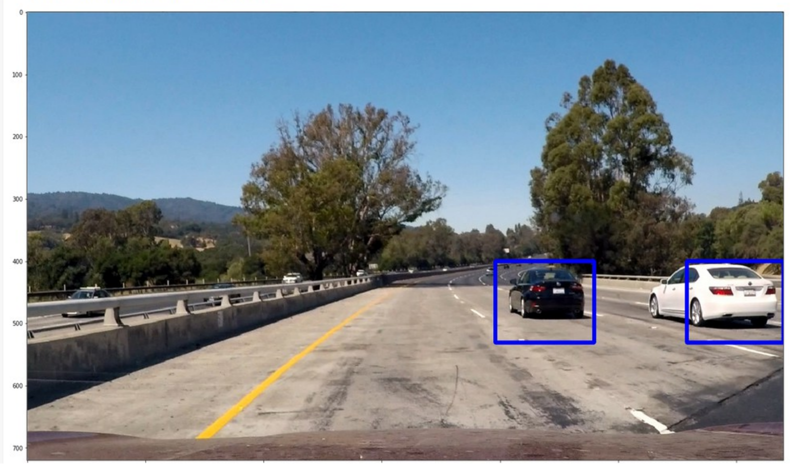
Below output shows scale 1.0 note the boxes are tight around the cars.

```
Out[30]: <matplotlib.image.AxesImage at 0x2fad2cf8>
```



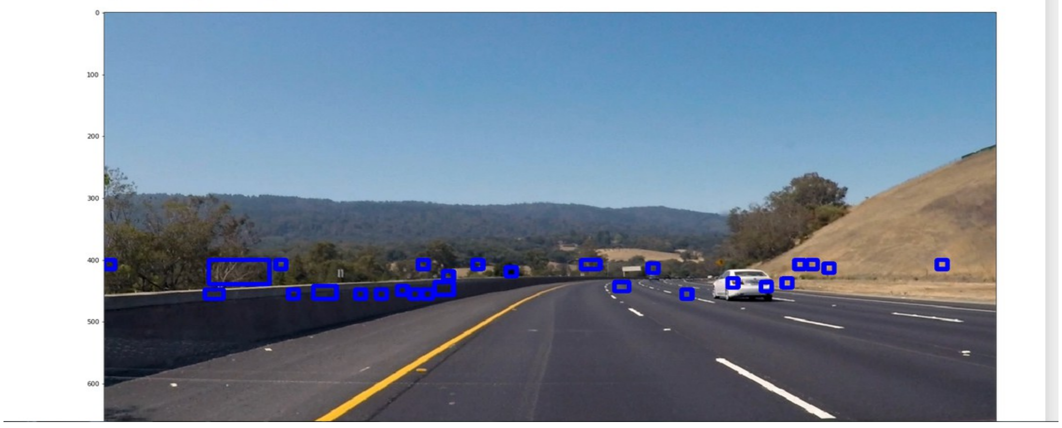
Below image shows scale 1.5 note the boxes around the cars are larger preceding image.

```
Out[31]: <matplotlib.image.AxesImage at 0x318cfe80>
```



Some images did not work for scales 0.5 or 1.0 or 1.5 such as “test_image3”. A scale of 0.25 did detect the white car when the other scales failed to but there was too many false positives to be useful, as shown below.

```
Out[35]: <matplotlib.image.AxesImage at 0x37b374e0>
```



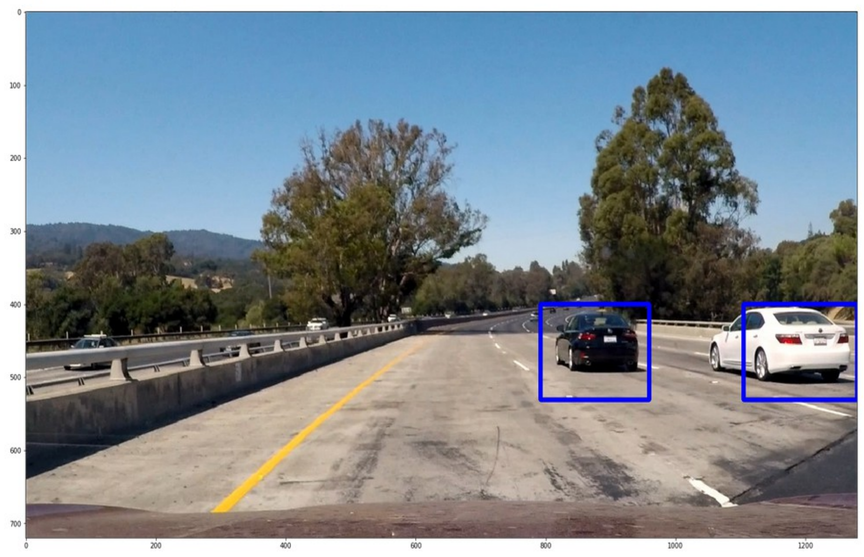
I concluded to use Scales of 1.0 and 1.5 and had a third scale in the pipeline which I experimented with usually at scale 0.5 but I never got good results and the large number of false positives threw off the heat map system which is discussed below.

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 1.0 and 1.5 using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector.

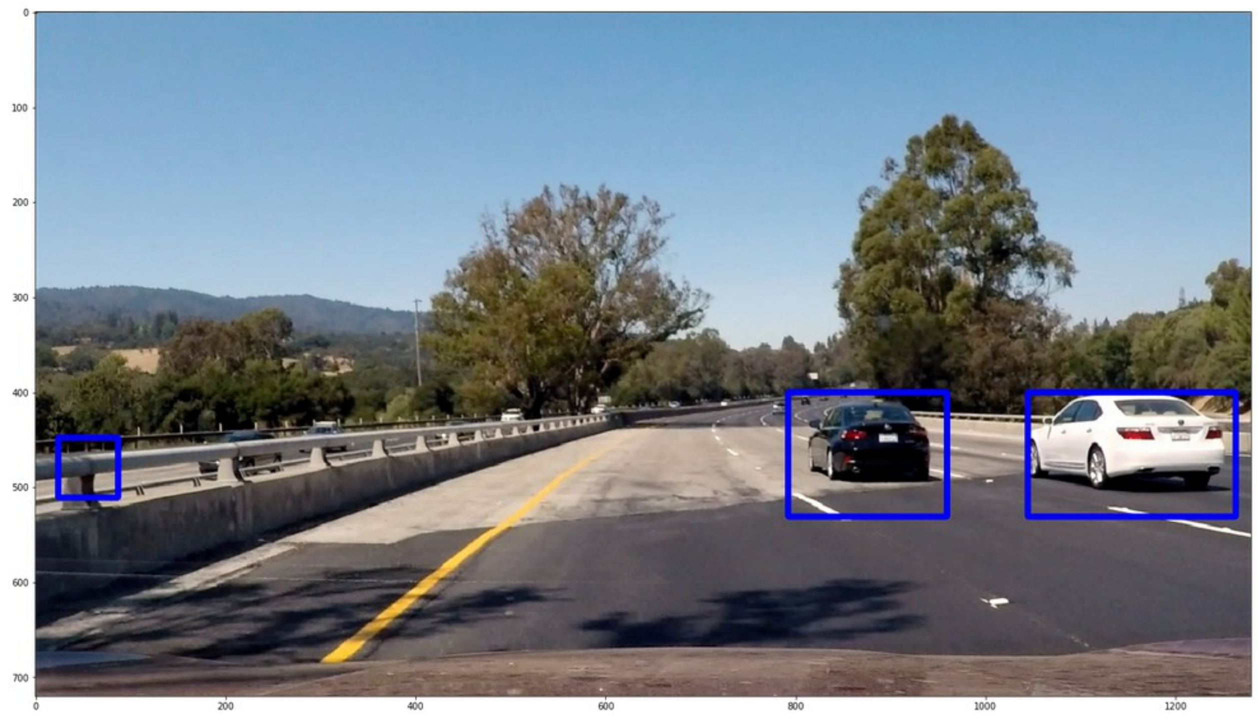
Here is a sample of test 1 image using both scale 1.0 and 1.5

```
Out[45]: <matplotlib.image.AxesImage at 0x3b5391d0>
```



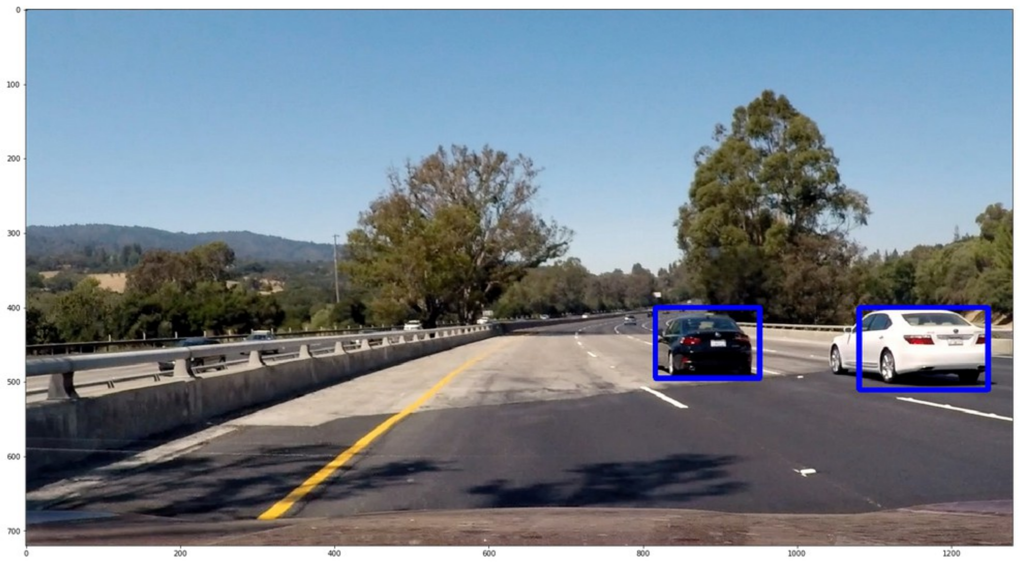
Here is a sample of test image 4 note a false positive on left side.

```
<matplotlib.image.AxesImage at 0x3c440710>
```



Here is the same image again with Scale 1.0 and 1.5 but with a threshold of 4 applied. Note the bounding boxes on the cars are smaller and the false positive has been removed.

```
Out[50]: <matplotlib.image.AxesImage at 0x3c4a8f60>
```

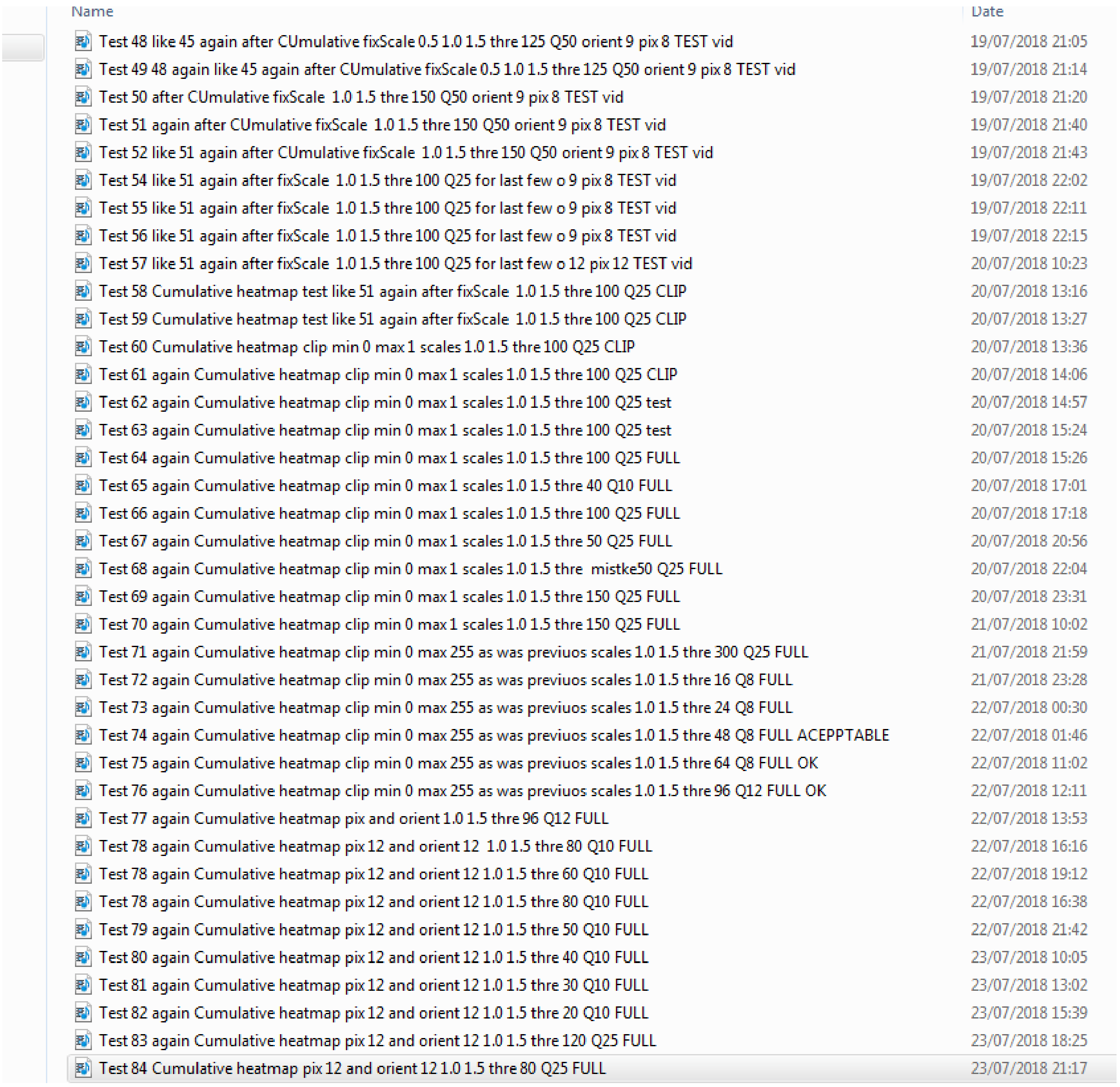


On the basis of this result I used a threshold of 4 per frame as a general rule so if I was looking at 10 frames in a heat map I would begin with a threshold of 40 and then experiment above and below that.

Video Implementation

1. Please find my output video in the ZIP file with Final_Output

Also I just want to show how many attempts I did....none of them perfect....last count over 80 and I will try one or two more....note I put the number and some information into each file name so I can track what I did.



Name	Date
Test 48 like 45 again after CUmulative fixScale 0.5 1.0 1.5 thre 125 Q50 orient 9 pix 8 TEST vid	19/07/2018 21:05
Test 49 48 again like 45 again after CUmulative fixScale 0.5 1.0 1.5 thre 125 Q50 orient 9 pix 8 TEST vid	19/07/2018 21:14
Test 50 after CUmulative fixScale 1.0 1.5 thre 150 Q50 orient 9 pix 8 TEST vid	19/07/2018 21:20
Test 51 again after CUmulative fixScale 1.0 1.5 thre 150 Q50 orient 9 pix 8 TEST vid	19/07/2018 21:40
Test 52 like 51 again after CUmulative fixScale 1.0 1.5 thre 150 Q50 orient 9 pix 8 TEST vid	19/07/2018 21:43
Test 54 like 51 again after fixScale 1.0 1.5 thre 100 Q25 for last few o 9 pix 8 TEST vid	19/07/2018 22:02
Test 55 like 51 again after fixScale 1.0 1.5 thre 100 Q25 for last few o 9 pix 8 TEST vid	19/07/2018 22:11
Test 56 like 51 again after fixScale 1.0 1.5 thre 100 Q25 for last few o 9 pix 8 TEST vid	19/07/2018 22:15
Test 57 like 51 again after fixScale 1.0 1.5 thre 100 Q25 for last few o 12 pix 12 TEST vid	20/07/2018 10:23
Test 58 Cumulative heatmap test like 51 again after fixScale 1.0 1.5 thre 100 Q25 CLIP	20/07/2018 13:16
Test 59 Cumulative heatmap test like 51 again after fixScale 1.0 1.5 thre 100 Q25 CLIP	20/07/2018 13:27
Test 60 Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 100 Q25 CLIP	20/07/2018 13:36
Test 61 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 100 Q25 CLIP	20/07/2018 14:06
Test 62 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 100 Q25 test	20/07/2018 14:57
Test 63 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 100 Q25 test	20/07/2018 15:24
Test 64 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 100 Q25 FULL	20/07/2018 15:26
Test 65 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 40 Q10 FULL	20/07/2018 17:01
Test 66 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 100 Q25 FULL	20/07/2018 17:18
Test 67 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 50 Q25 FULL	20/07/2018 20:56
Test 68 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre mistke50 Q25 FULL	20/07/2018 22:04
Test 69 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 150 Q25 FULL	20/07/2018 23:31
Test 70 again Cumulative heatmap clip min 0 max 1 scales 1.0 1.5 thre 150 Q25 FULL	21/07/2018 10:02
Test 71 again Cumulative heatmap clip min 0 max 255 as was previuos scales 1.0 1.5 thre 300 Q25 FULL	21/07/2018 21:59
Test 72 again Cumulative heatmap clip min 0 max 255 as was previuos scales 1.0 1.5 thre 16 Q8 FULL	21/07/2018 23:28
Test 73 again Cumulative heatmap clip min 0 max 255 as was previuos scales 1.0 1.5 thre 24 Q8 FULL	22/07/2018 00:30
Test 74 again Cumulative heatmap clip min 0 max 255 as was previuos scales 1.0 1.5 thre 48 Q8 FULL ACEPPTABLE	22/07/2018 01:46
Test 75 again Cumulative heatmap clip min 0 max 255 as was previuos scales 1.0 1.5 thre 64 Q8 FULL OK	22/07/2018 11:02
Test 76 again Cumulative heatmap clip min 0 max 255 as was previuos scales 1.0 1.5 thre 96 Q12 FULL OK	22/07/2018 12:11
Test 77 again Cumulative heatmap pix and orient 1.0 1.5 thre 96 Q12 FULL	22/07/2018 13:53
Test 78 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 80 Q10 FULL	22/07/2018 16:16
Test 78 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 60 Q10 FULL	22/07/2018 19:12
Test 78 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 80 Q10 FULL	22/07/2018 16:38
Test 79 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 50 Q10 FULL	22/07/2018 21:42
Test 80 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 40 Q10 FULL	23/07/2018 10:05
Test 81 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 30 Q10 FULL	23/07/2018 13:02
Test 82 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 20 Q10 FULL	23/07/2018 15:39
Test 83 again Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 120 Q25 FULL	23/07/2018 18:25
Test 84 Cumulative heatmap pix 12 and orient 12 1.0 1.5 thre 80 Q25 FULL	23/07/2018 21:17

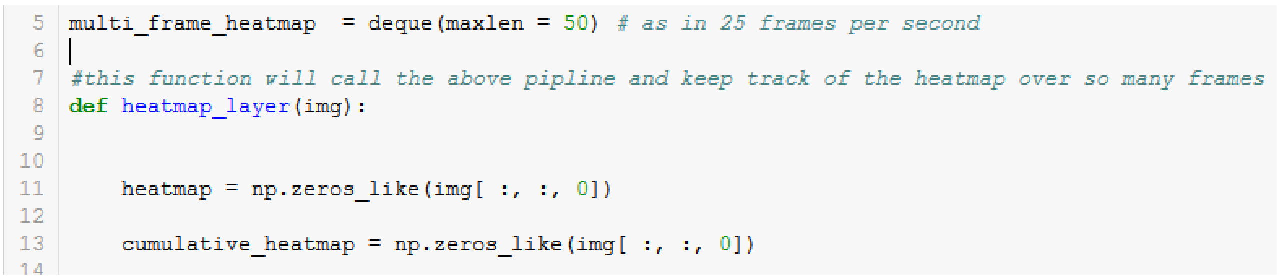
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

Filtering false positives was a major issue and became the main focus of the project. The concept suggested by Udacity tutors was to address false positives was to use a heat map and then make lables using *scipy labels* library. This entails recording where the true and false positives are found in an image and accumulating that information over several images. For example the video is 25 frames per second. 12 frames would be half a second. Recording all the indicated matches both true and false over 12 frames would give many true and false positives. However assuming that false positives are randomly distributed and that true positives are more consistent in location we can use a threshold to filter out false positives. This assumption proved reasonably true. For most of the provided video false positives could indeed be filtered out effectively using scales of 1.0 and 1.5. Once the heat maps were generated and accumulated over a number of frames labels were created to generate bounding boxes.

The code below shows a deque *"multi_frame_heatmap"* which is used to keep track of the recent heat maps. The amount of retained heat maps varied typically 10 or 25 were used this shows and experimental 50 for 2 seconds of frames.

The *heatmap* is a *numpy* array same dimensions to the image and is for a single heat map

Cumulative_heatmap is a heatmap that all the currently cached heatmaps in multi_frame_heatmap are added up and stored into. This Cumulative_Heatmap has a threshold applied to it for the labels to be generated.



```
multi_frame_heatmap  = deque(maxlen = 50) # as in 25 frames per second

#this function will call the above pipline and keep track of the heatmap over so many frames
def heatmap_layer(img):


    heatmap = np.zeros_like(img[ :, :, 0])

    cumulative_heatmap = np.zeros_like(img[ :, :, 0])
```

There is a section in the video that *"test3.jpg"* is taken from shown below. This particular section of video was particularly troublesome. When the car shown was detected invariably there were false positives as well. When the threshold levels were adjusted to filter out the false positives then the bounding boxes on the car was lost. This issue summarised the the whole exploration of the project.

To address the issues of reducing false positives while maintaining true positives particularly in the areas shown in test3.jp the following parameters were experimented with.

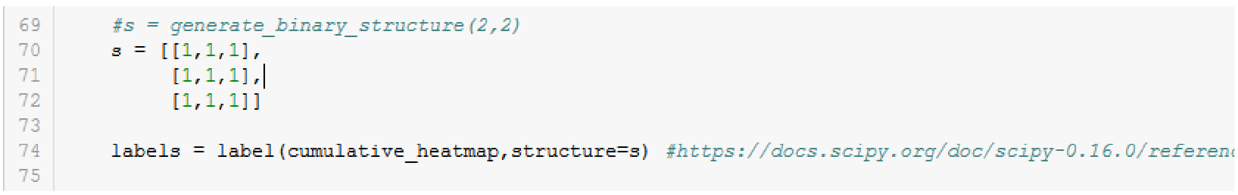
1) The number of frames (images) that a heatmap would keep track of. A "deque" was used with length of 8, 12, 25 and 50 experimented with.
2) The threshold set to filter out false positives. This was 0 to 8 for single frames and that number usually around 4 multiplied by the number in the deque. Eg 4 by 25 for threshold of

100. Values of 30,40,60, 80, 100, 150 and 200 were used for various deque lengths.

3) The number of scales used in the pipeline up to 3 were used at a time and thresholds were adjusted up or down with less or more scales. 0.5,1.0 and 1.5 were the most commonly explored but I did uses as low as 0.25 and as high as 3. I settled on 1.0 and 1.5 as the best.

4) Pixels per cell and HOG Orientation were also experimented with to confirm earlier experimentations on static images. HOG Orientations of 12 were marginally better subjectively than 9 and 12 pixels per cell was marginally better than 8 but this was at a price of approximately doubling the processing time. HOG Orientations of 9 and pixels per cell of 8 gave the best compromise as suggested by single frame experiments.

5) Using a variation on the labels it was possible to get adjacent bounding boxes to be treated as one. This made a small improvement code below from scipy.org



```
    #s = generate_binary_structure(2,2)
    s = [[1,1,1],
         [1,1,1],
         [1,1,1]]

    labels = label(cumulative_heatmap,structure=s) #https://docs.scipy.org/doc/scipy-0.16.0/referen
```

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?

The most obvious issue is the compromise between eliminating false positives using heatmaps and threshold and maintaining track of real cars. No mater what I did I never go a perfect zero false positives with perfect tracking of cars. I assume that I reached the limit of what can be achieved using heatmaps and thresholds. To address this I suggest the following.

Once a car is identified some way of tracking it separate to heatmaps would be useful. For example when the white car in test3.jpg moves away from the camera at a certain point it usually looses its bounding box. While this happens it appearance remains almost exactly the same. This suggests that once a car is identified with high confidence it could be added as a recognisable pattern so that is can be tracked even if the classifier no longer detects it. This approach suggests two classifiers one that is trained in advance and another that is “real time” that can cache some images of cars to improve performance. Obviously that would have implications for performance of the process in terms of speed but it may be a solution.

Vehicles that are unlike those in the sample images are likely to cause a problem and the very wide variety of vehicles available will present a challenge. Having different data sets for USA and Europe would improve detection rates in both areas because the cars look quite different. I guess you could call that localisation of data. Certainly a motorbike in Vietnam and a pickup truck in Texas and a Bus in Europe all look extremely different and that would need to be addressed in the datasets to improve performance.

Similar to the advanced lane finding project variations in light and shadow in the image are significant and I believe this is an issue in the section of video represent by test3.jpg. Perhaps brightness and contract levels could be normalises to some standard level to make the process easier for the classifier but this would be at the expense of performance.

It is possible that a neural network could be used to deliver better results especially if it was able to accept corrections from the user , eg when I see that the white car is not being track if I could manually select it and then have the system add the feedback to it learning that would be a good way to evolve over time. A neural net would be a nice follow on project to try.

On a practical level the processing time varied form under 3 seconds to over 11 per frame depending on how many scales and length of deque I used. Clearly this is no where near good enough for real time use. Using C++ or software dedicated in embedded hardware would run the same ideas much faster. Similarly optimising the program to use pipelines in graphic GPUS should greatly improve processing times. Right now I can't do C++ optimised for GPU's but I can see that I will need to learn that eventually.