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.

Project Rubric Points As below

###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

###Writeup / README

####1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

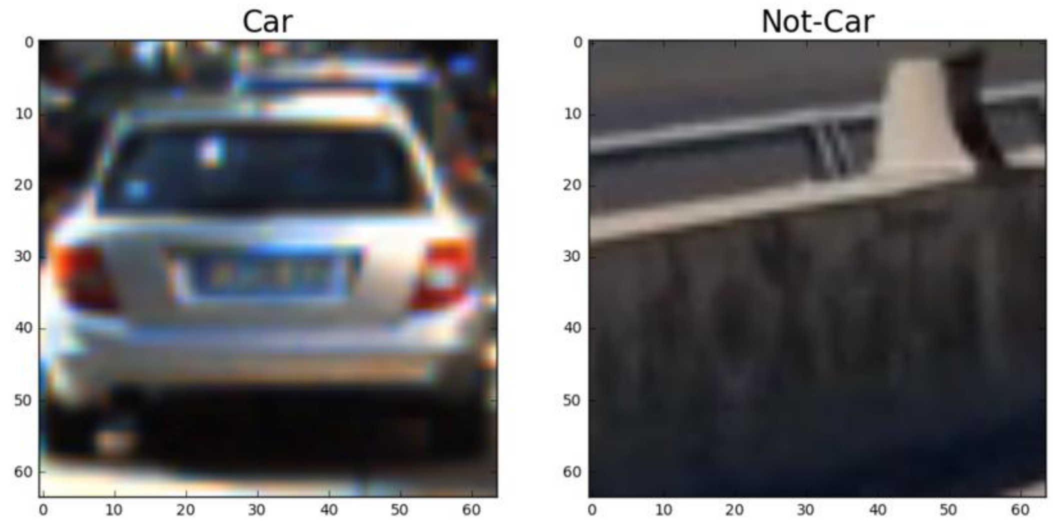
You're reading it! And uploaded same at GitHub repo

###Histogram of Oriented Gradients (HOG)

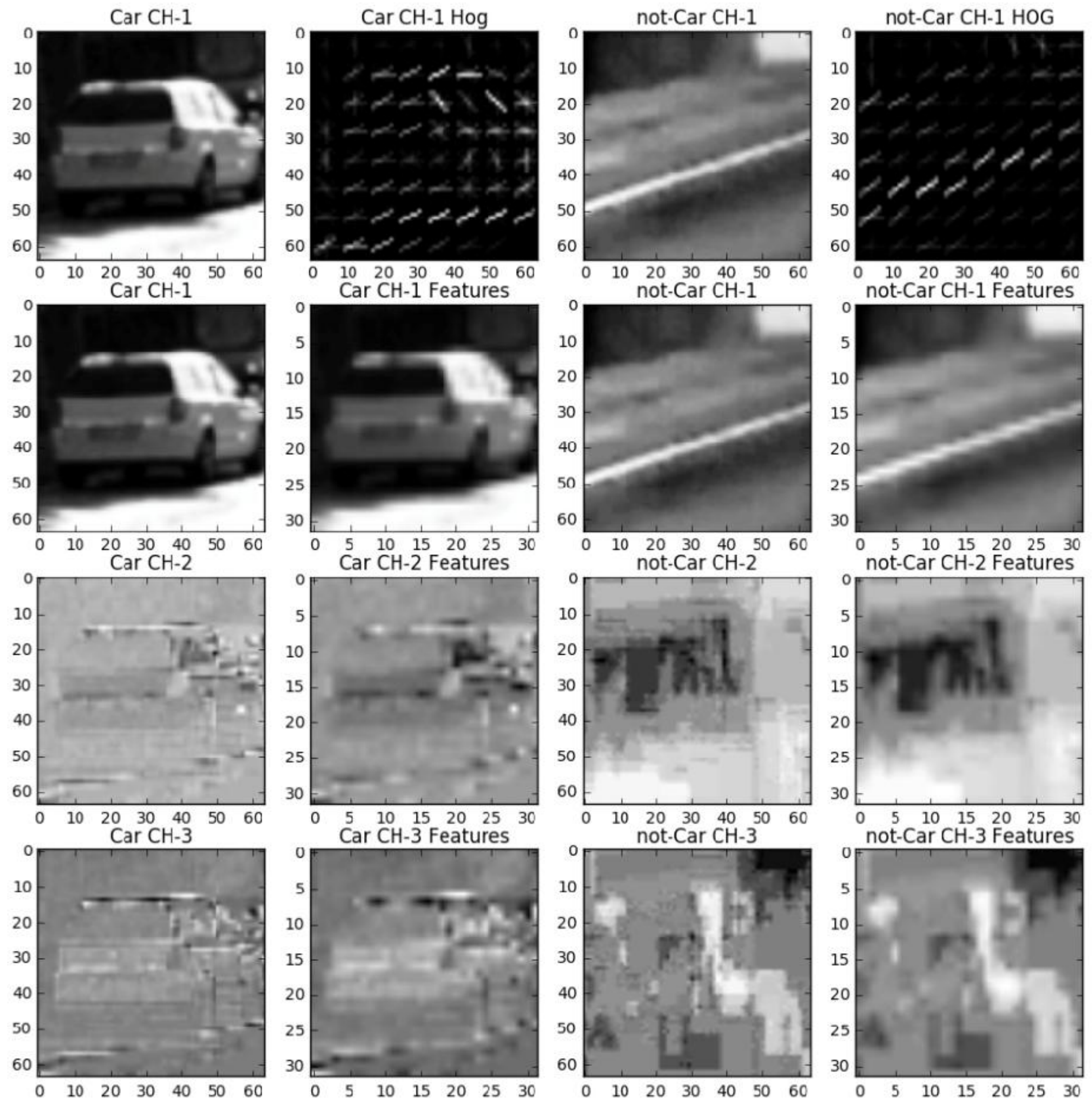
####1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the third code cell of the Project_Vechicle_Detection.ipynb .classifier setting are loaded using pickle file.

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=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



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

The following were chosen as the final values of the HOG parameters:

orientations: 16. This is the number of orientation bins. I have experimented with larger values of this parameter upto 35, but it doesn't improve the accuracy of the classifier much, so I settled with 16 which also allows the training and predictions to happen significantly faster.

pixels_per_cell: 7. I experimented with this value to find this optimum value. This was mostly done by trial and error by looking at the corresponding HOG image output produced.

cells_per_block: (2, 2)

feature_vector: True. Since I want the feature vector returned which will be useful for training, I always set this parameter to be true

####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 `single_img_features` method in cell `[38]` is used to extract all features from a list of images including the HOG features, the spatially binned features and the histogram of colors features.
- In cell `in [3]` the features are fed to classifier using `sklearn.preprocessing.StandardScaler`.
  Below are the different HOG features applied to classifier

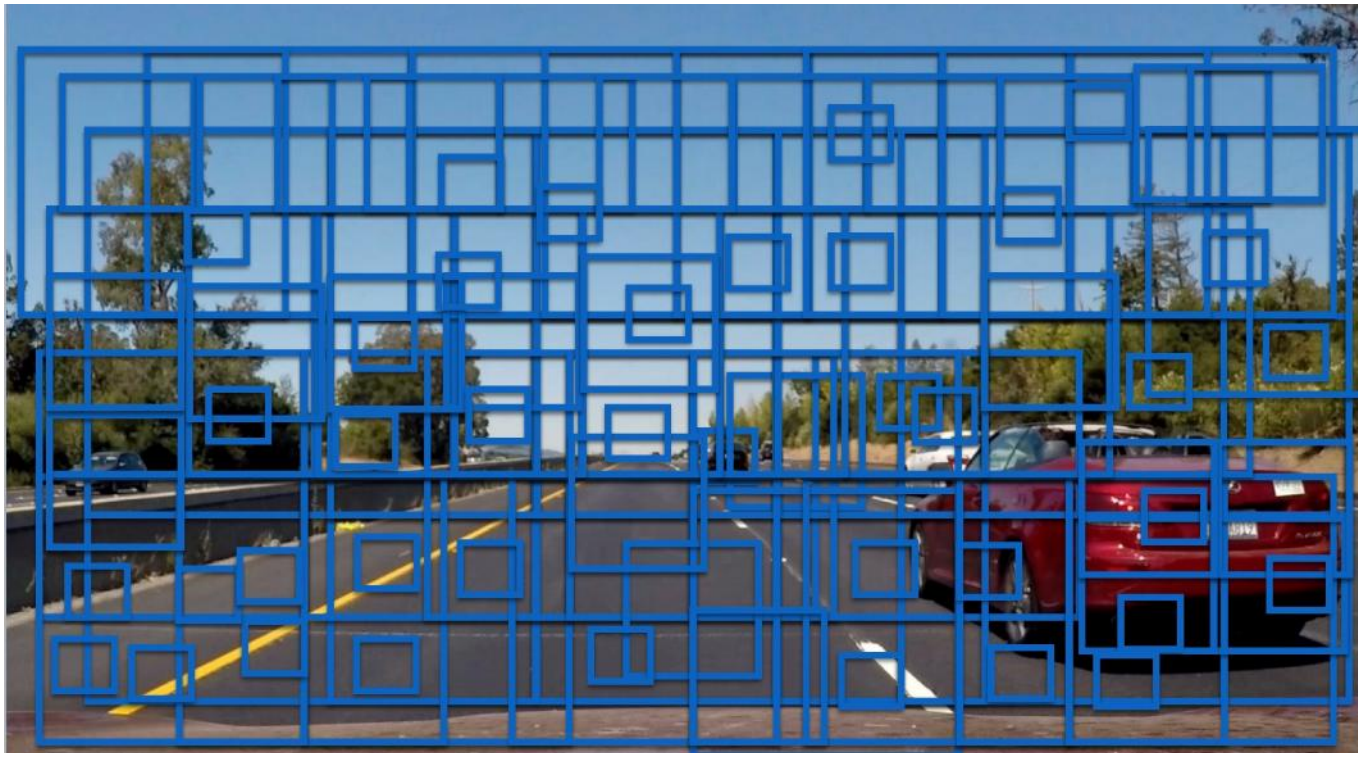
  ```
  scaler : <class 'sklearn.preprocessing.data.StandardScaler'>
  settings_classifier :
    {'hog_channel': 'ALL', 'hist_bins': 16, 'hist_feat': True, 'color_space': 'RGB
     ', 'pix_per_cell': 7, 'hog_feat': True, 'orient': 8, 'spatial_feat': False, '
       spatial_size': (16, 16), 'cell_per_block': 2}
  ```

  single_img_features function returns concatenated array of features

###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 decided to search random window positions at random scales all over the image and came up with this



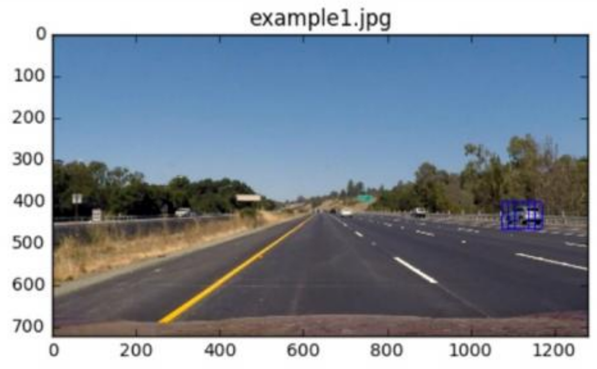
I choose sliding windows search because Sliding windows play an integral role in object classification, as they allow us to localize exactly *“where”* in an image an object resides.

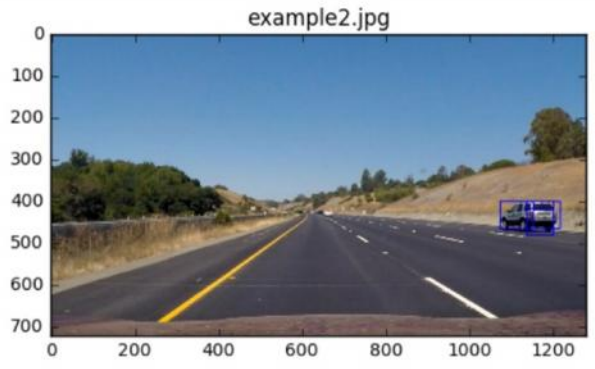
Utilizing both a sliding window and an image pyramid we are able to detect objects in images at various scales and locations.

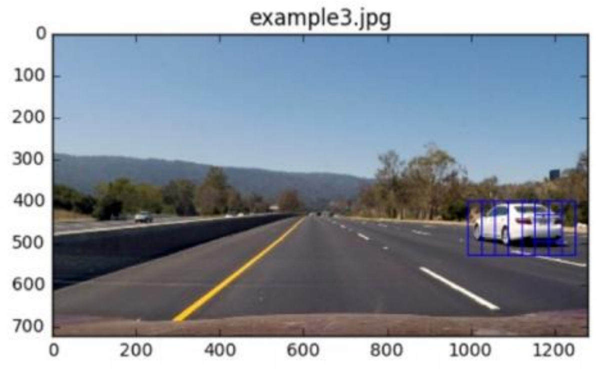
####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

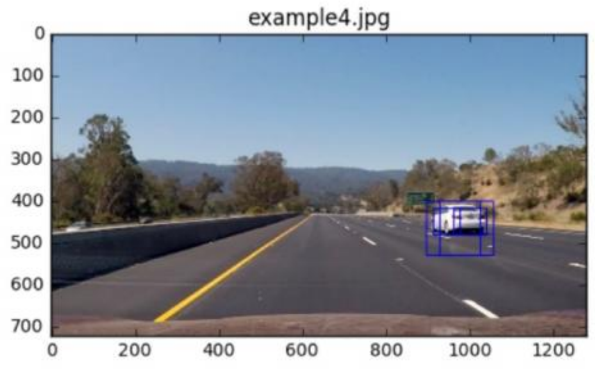
Ultimately, I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.

Here are some example images:









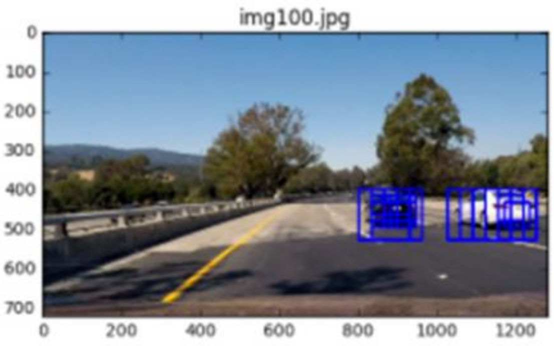
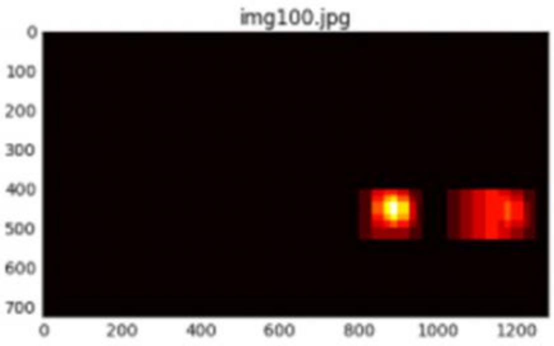
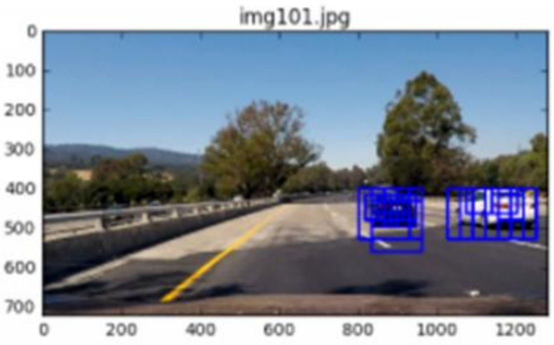
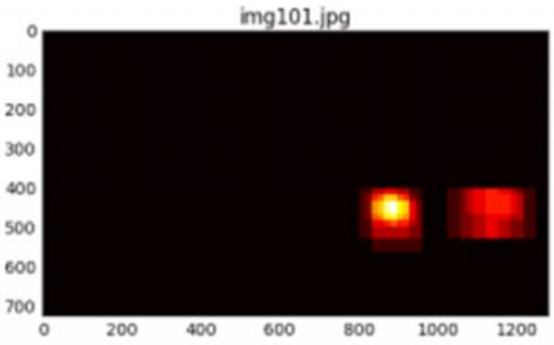
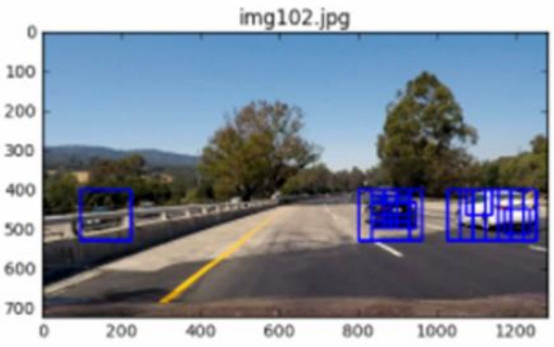
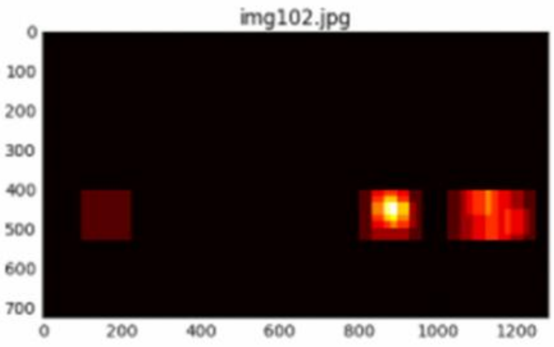
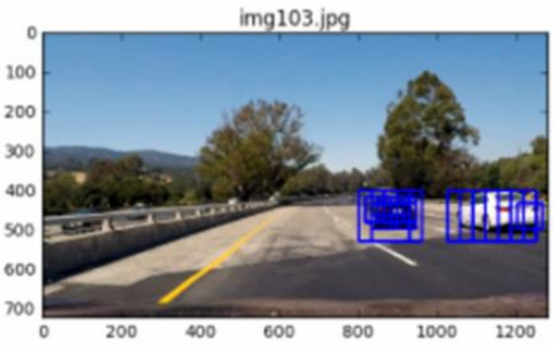
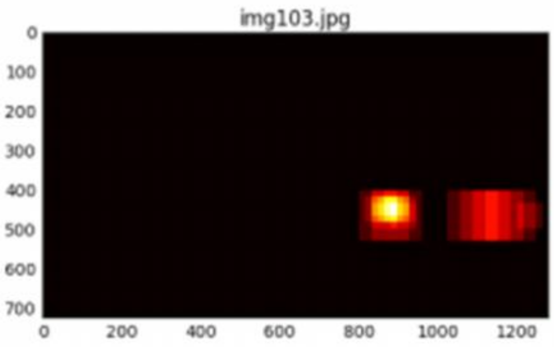
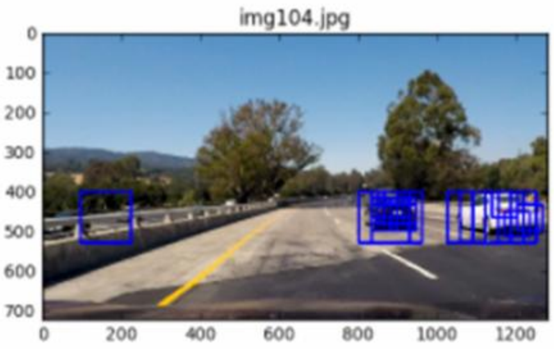
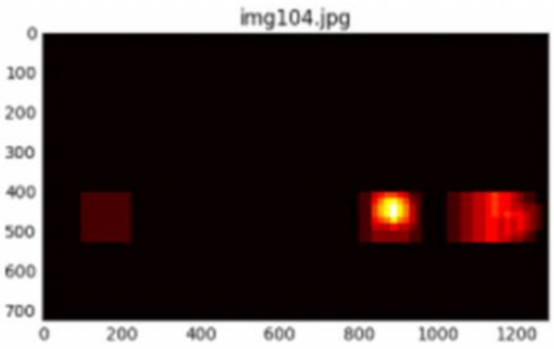
Video Implementation

####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

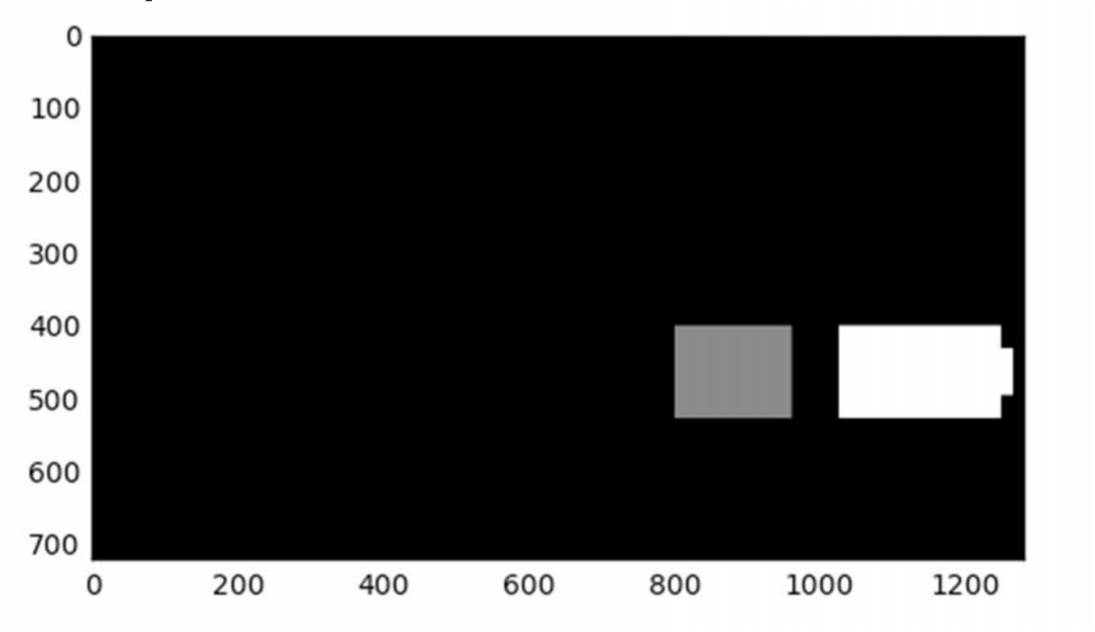
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

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

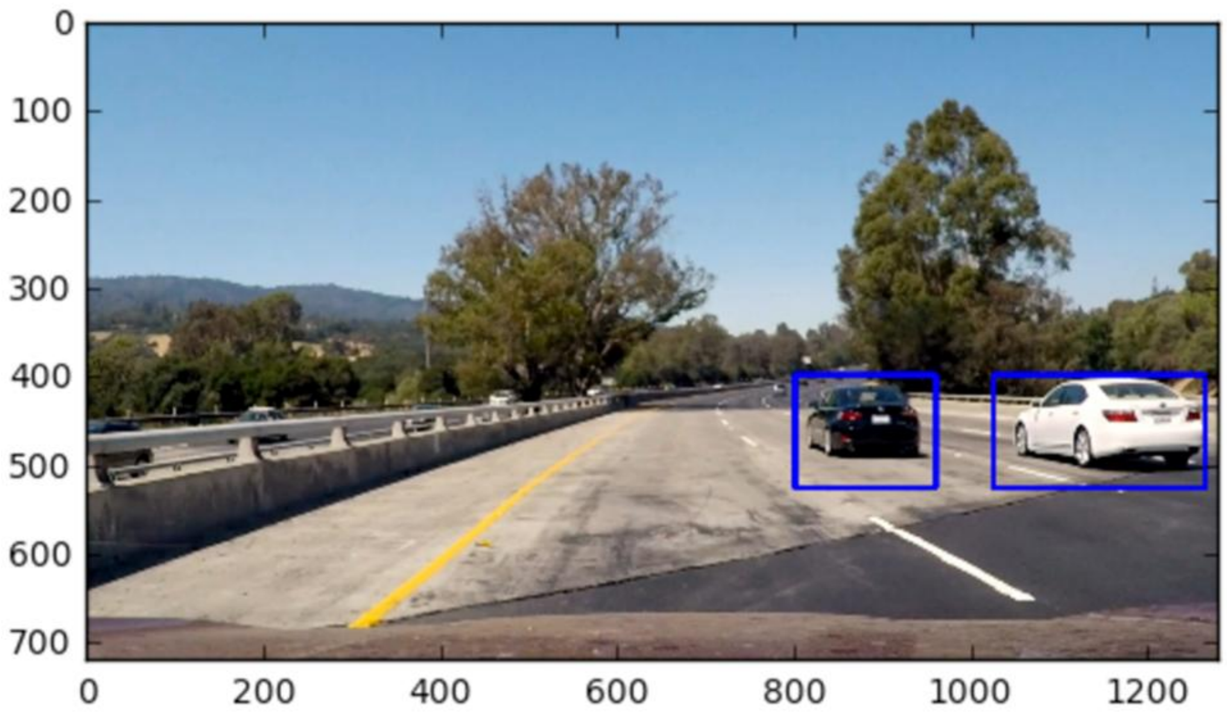
heatmaps:


img100.jpg

img100.jpg

img101.jpg

img101.jpg

img102.jpg

img102.jpg

img103.jpg

img103.jpg

img104.jpg

img104.jpg

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 includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

My pipeline may fail as trucks, motorbikes, cyclists and pedestrians would not be detected as they are not in the training data. I need to train model with those additional images.

Detecting cars with SVM in sliding windows is interesting method but it has a number of disadvantages. While trying to make my classifier more quick I faced with problem that it triggers not only on cars but on other parts of an image that is far from car look like. So, it doesn't generalizes well and produces lot of false positives in some situations. To struggle this I used bigger amount of non-car images for SVM training. Also, sliding windows slows computation as it requires many classifier tries per image. Again, for computational reduction not whole area of input image is scanned. So, when road has another placement in the image like in strong curved turns or camera movements sliding windows may fail to detect cars.

Also we need to give more time to complete this project.