Vehicle Detection Project

The goals / steps of this project are the following:

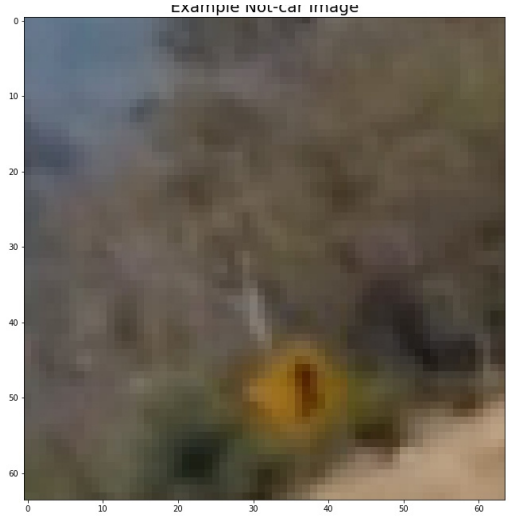
- Perform a Histogram of Oriented Gradients (HOG) feature extraction (also apply a color transform and append binned color features, as well as histograms of color) on a labeled training set of images and train a classifier Linear SVM classifier (Normalize features and randomize a selection for training 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.

Rubric Points

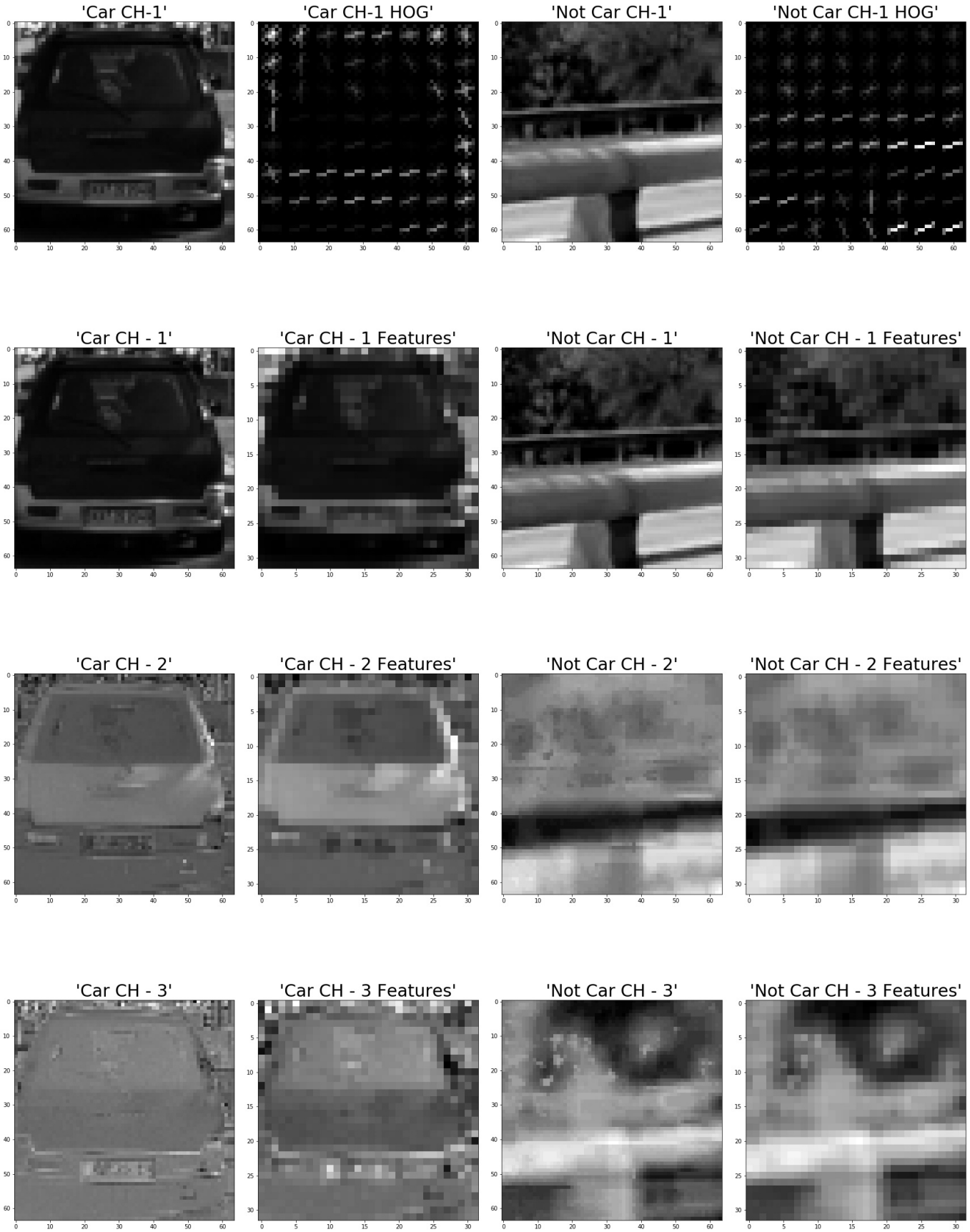
Image Features: Histogram of Oriented Gradients (HOG), Spatial and Color Histogram

1. For this project the training image data has been read into two lists containing image file names.

The cars and notcar image file lists have been created by reading in the 'vehicle' and 'non-vehicle' images. These are RGB .png images of size (64,64,3) and come from a combination of the GTI vehicle image database, the KITTI vision benchmark suite. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



For extracting the features of the images I have used HOG, Spatial binning and Color histogram features of the images. The code extracting the image features are defined in a function in the IPython notebook P5.ipynb in the second cell (from lines 124 to 174). 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 `YUV` color space and HOG parameters of `orientations=11`, `pixels_per_cell=(8,8)` and `cells_per_block=(2,2)`:



2. Settling on final choice of HOG parameters.

I tried various combinations of parameters, pixels_per_cell, cells_per_block and orientations. The parameter selection is relevant to the training data image sizes (64x64), so picking larger values ex: pixel_per_cell = (16,16) made the features more uniform and not able to distinguish the image features. So choosing pixel_per_cell = (8,8) showed distinct features of the images. Similarly the number of orientation bins of 12, proved better option to distinguish the features. Finally I settled on the following parameters for extracting HOG features pixels_per_cell = (8,8), cells_per_block = (2,2) and orientations = 12. The block normalization was set to default. Since I used YUV color space, transform_sqrt=True could not be used for Normalization as image contained negative values. I used HOG features of all color channels making using the color gradient values.

3. Training the Classifier

I trained a linear SVM using the image features that had combined features of HOG from all color channels (Y,U,V) and also spatial binning features where the image is resized to (32,32) and used the binned values for feature vector. The color channel histogram features are also added to feature vector with bin size of 32. Before using the data for training the classifier the feature vector data has been Normalized using the sklearn.preprocessing.StandardScaler() lib. Also the image database has been split into train data (80%) and test data (20%). The data split has been randomized to make sure to avoid over fitting of the data. The code for training the SVM classifier is in the 6th cell of the IPython notebook P5.ipynb

Sliding Window Search

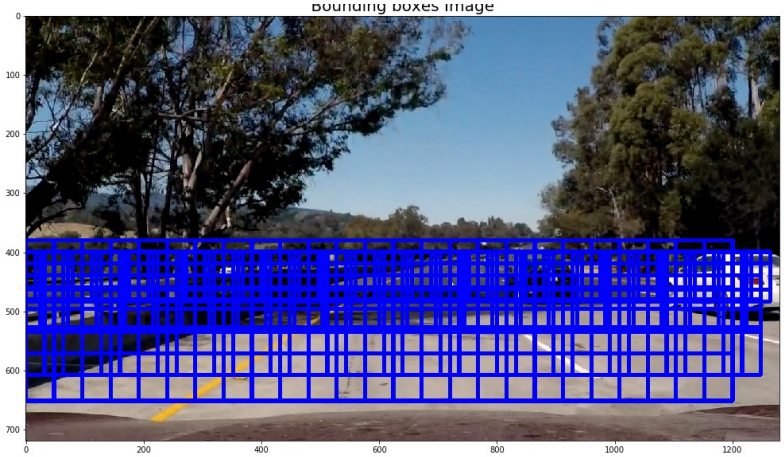
1. Identifying the Scales and Window sizes

Looking at the images of the video, first the relevant area to search for vehicle has been decided, in the vertical direction the search area is limited to y = 380, 720. And as the vehicles at the horizon would appear smaller compared to closer to the camera, the window size (scale) has been chose smaller and the scale size was chosen to be big for closer views. And some intermediate scales have been chose with the assumption of image scaling. The scales and window sizes chosen are as shown below.

The function find_cars (second cell of P5.ipynb) extraction of hog features was done only once and then used sub-sampled to get all of its overlaying windows. Each window is defined by a scaling factor where a scale of 1 would result in a window that's 8 x 8 cells then the overlap of each window is in terms of the cell distance. Tusing this approach it was possible to run this same function multiple times for different scale values to generate multiple-scaled search windows.

Scale	Window size in y-direction
1.0x	400, 480
1.0x	420, 500
1.5x	400, 520
1.5x	440, 560
2.0x	400, 560
2.0x	480, 640
3.0x	380, 620
3.0x	460, 700

Here is the image of search windows generated by scales and window sizes shown above:



2. Optimizing the Classifier and Choosing Color space and Features

I have tried using two color spaces YCrCb and YUV and also tried different combination of HOG channel features (HOG for only Y channel and also all the 3 channels). Normalization feature transform_sqrt=True option was not allowed for YUV because of negative image values. But YUV produced optimal detection and less False positives. Also using both HOG and spatial/color histogram features help reduce the false positives and identify the Car images of different colors and also using the HOG features for color channels helped with the same.

Here are some example images:

Video Implementation

1. Using the pipeline developed for the images, I have ran the implementation on the project_video.mp4.

Here's a link to my video result

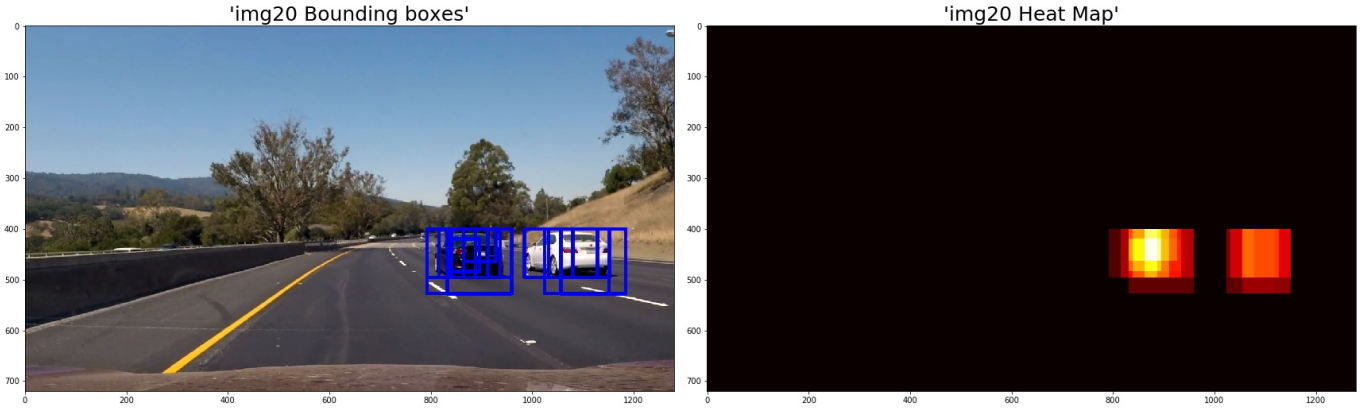
2. Identify False Positives and creating Bounding box around vehicle detection

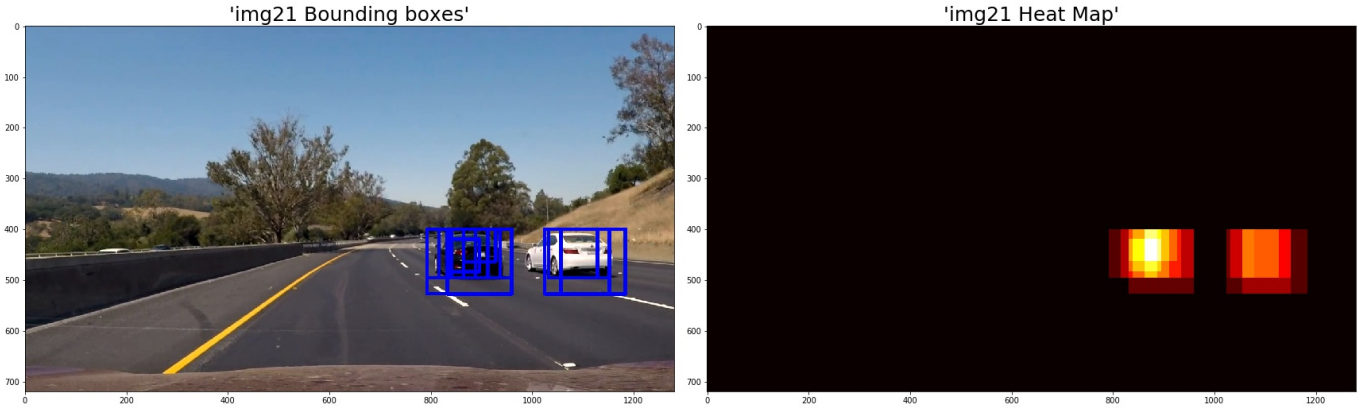
The code for detecting the false positives and bounding boxes is implemented using heatmap and is second cell of the IPython notebook P5.ipynb. I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded the heatmap 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 creating the heatmap in the video I have used the information from the previous six frames that was stored using

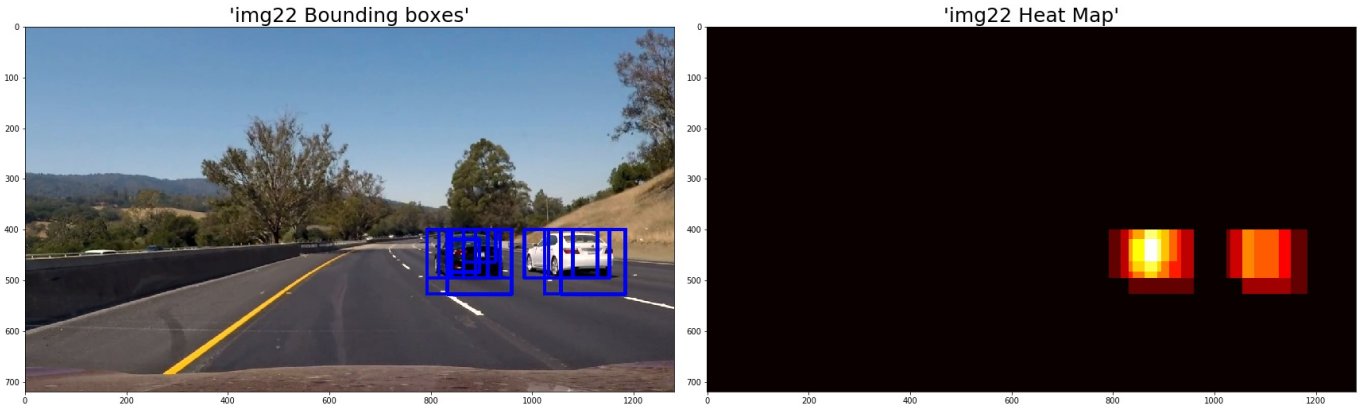
a class (class Car_Detect) as implemented in the last cell of the P5.ipynb. When there were no detections the history of the detections (on_windows) have been deleted and the accumulation of the detected bounding boxes and heatmap is started again to make sure that data from previous frames is not used. The threshold value for detecting the vehicles is taken as average of the last few frames to remove any false positives.

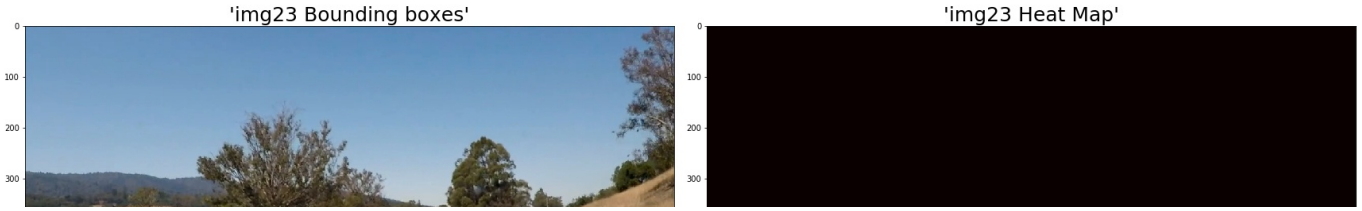
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

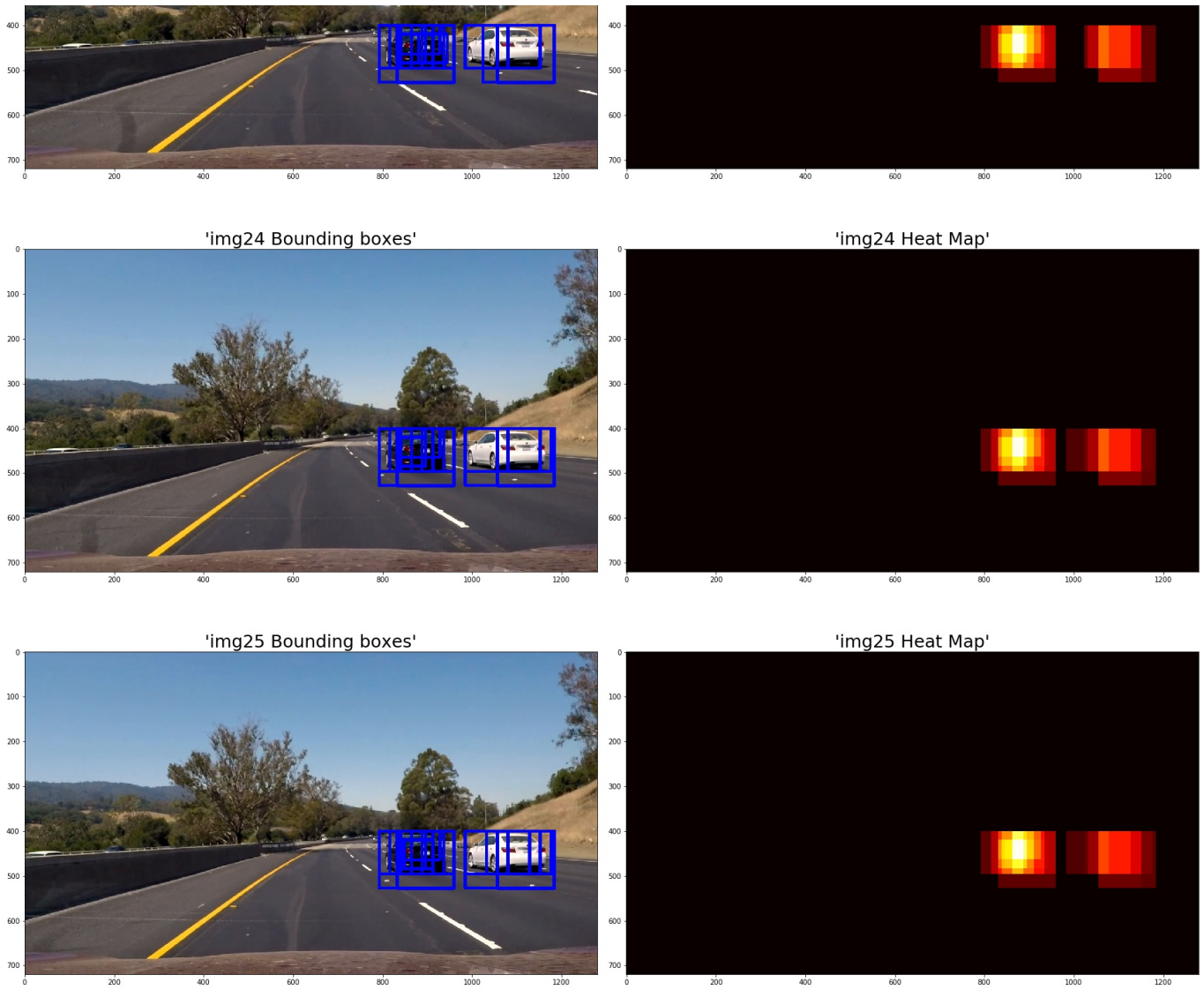
Here are six frames and their corresponding heatmaps:



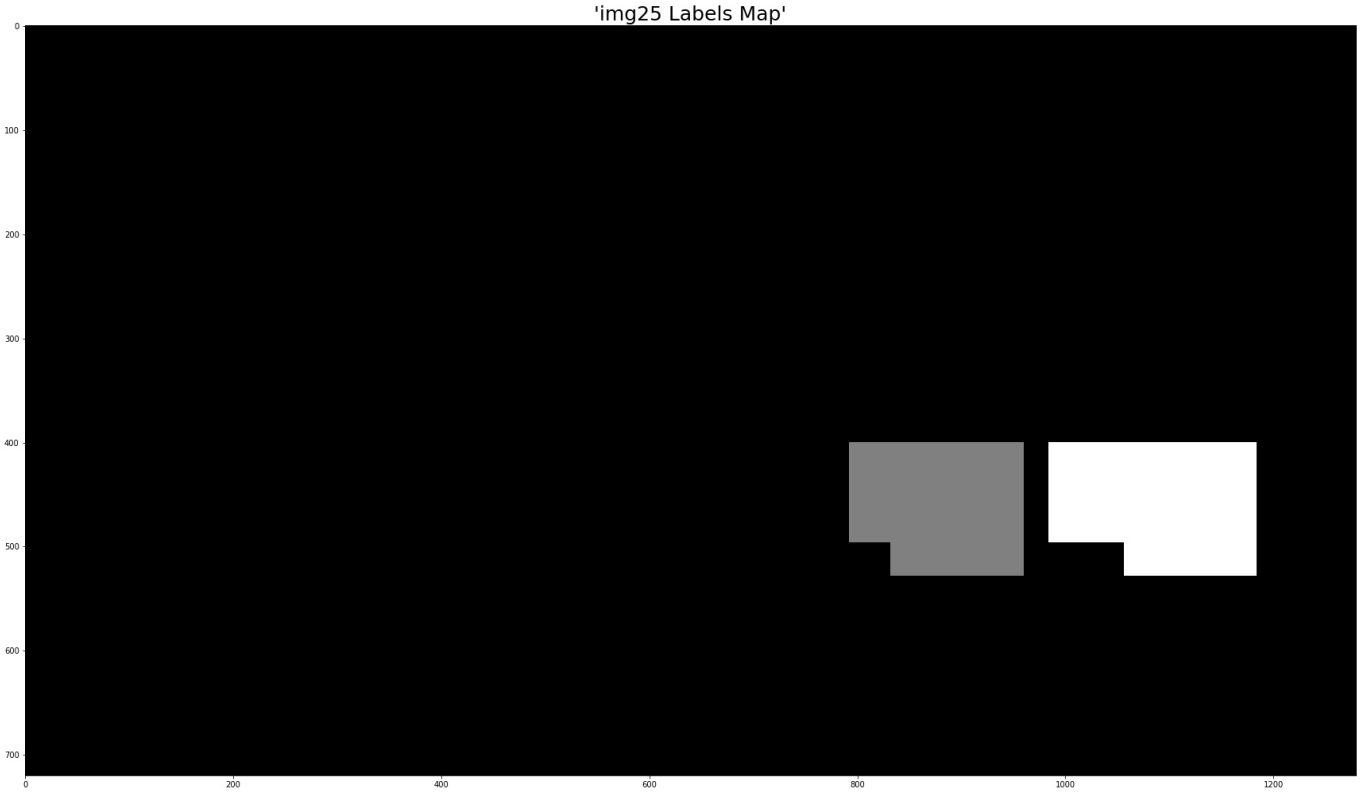




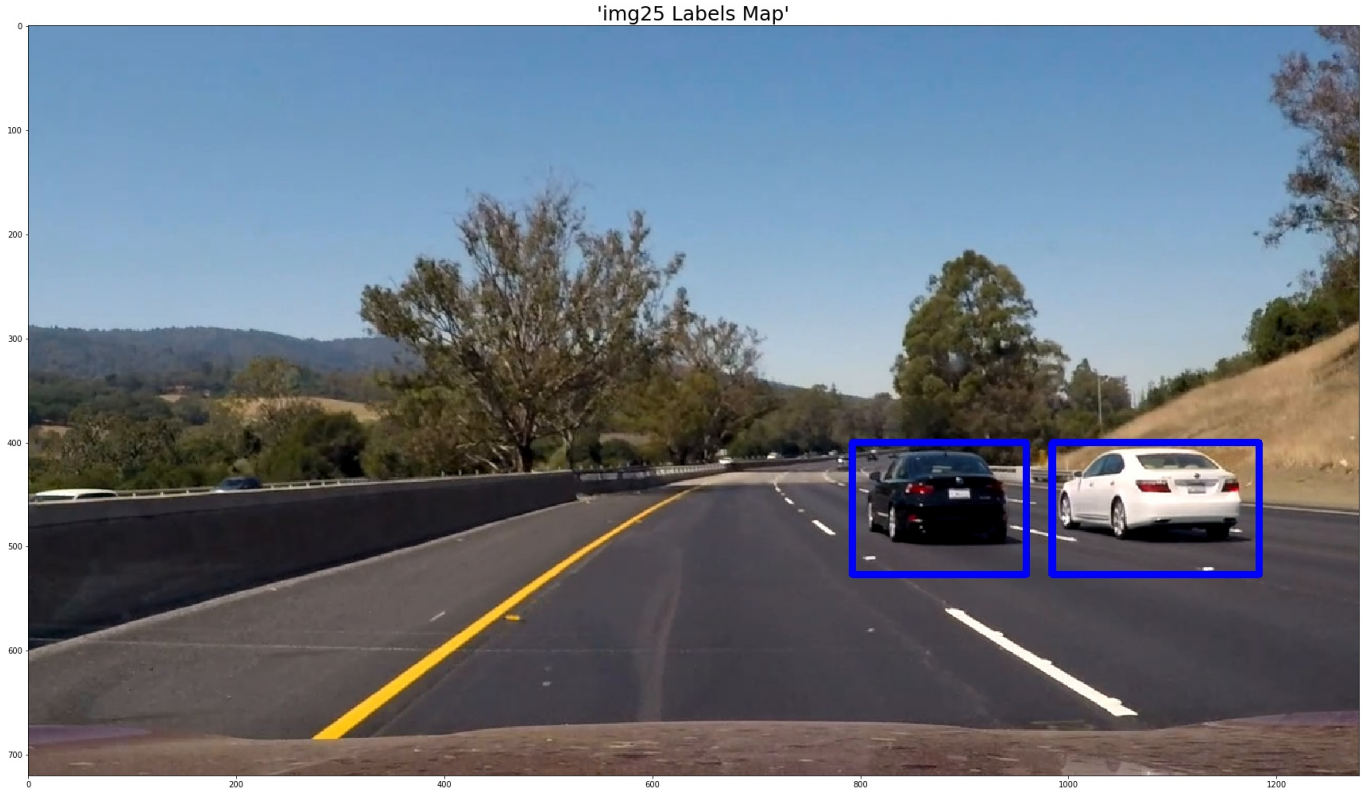




Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames:



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



Discussion

1. Problems / issues you faced in your implementation of this project.

This approach used in the project is based on the training data collected from some standard database and is limited to Cars. Also if there are vehicles with different shapes (not present in training data) are present then this approach might fail. Also for varying conditions like bad light, rain or fog this might fail. These could be improved by using a better training data set and also since this is a binary classification problem (vehicle or not), there are better approached to solve this issue (1x1 convolution approaches. Probably there are other issues that could arise with limitation f training data set and image conditions (brightness, real-world scenarios of shadows, occlusion, etc.)