Project 5 Vehicle Detection

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.

## [Rubric](https://review.udacity.com/#!/rubrics/513/view) Points

###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](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

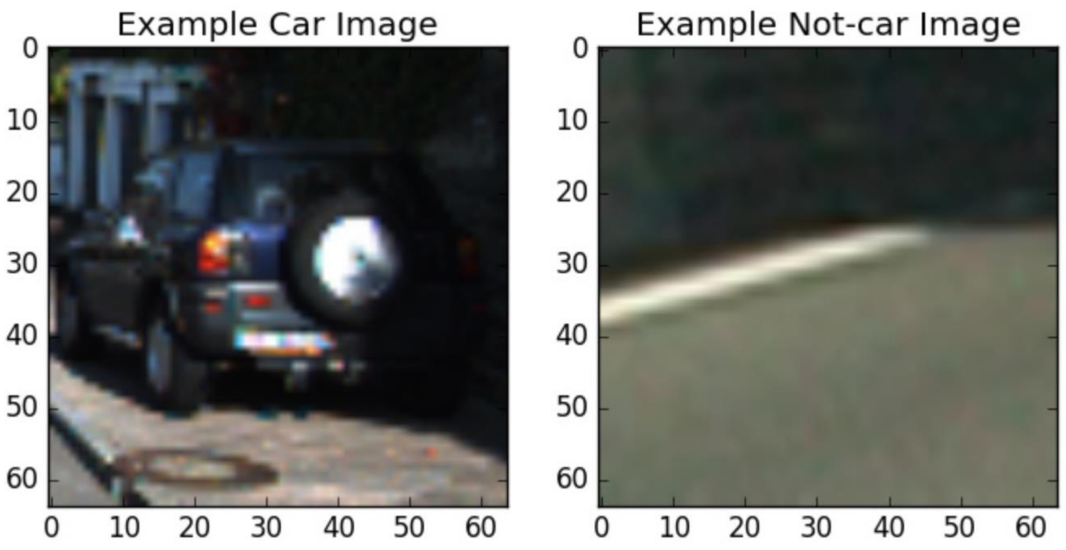
You're reading it!

###Histogram of Oriented Gradients (HOG)

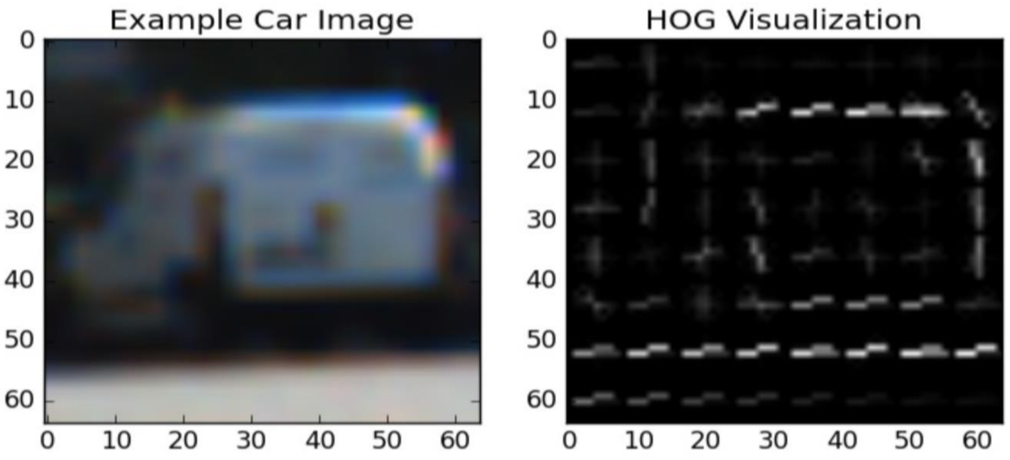
####1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the first code cell of the IPython notebook (or in lines # through # of the file called `some_file.py`).

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 randomly chose an image and plotting hog image, see below:



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

I tried various combinations of parameters. Specially, I found that hog channel is much more important for the training. The second import parameters is “orient” and I increased it to 10. After 10, it shows little influence. Both parameters “pix_per_cell” and “cell_per_block” are not very sensitive for the classifier results.

```
orient = 10  # 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 = (16, 16) # Spatial binning dimensions
```

####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 both HOG features and color features. “YUV” is chosen as the color feature. The training codes are from line 341 to 395.

StandardScaler is used to normalize the data first.

20% of data was randomly chosen as validation data.

The validation accuracy finally reached to 99.03 %.

###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?

The sliding windows search includes two steps:

a. **Create windows list (function “slide_window” from line 137 to 176)**
b. **Search through windows to identify if car exist (function “search_windows” from line 244 to 271)**

For overlap windows, I used default 50%.

For windows size, I following suggestion from Udacity:

a. **Do not put window on the top of images (sky)**
b. **Use larger window size in the bottom and smaller window size in the middle**

Besides, I did not put any window on a small part at the image bottom, where partial underhood of the driver’s car is shown in the images. Otherwise, this area could be a distraction for vehicle detection.

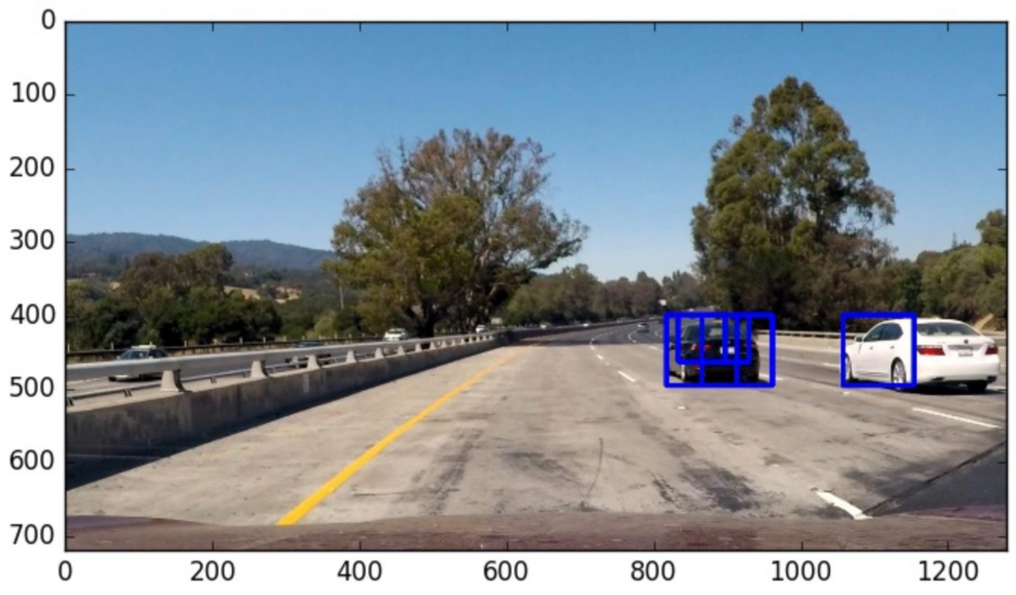
####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

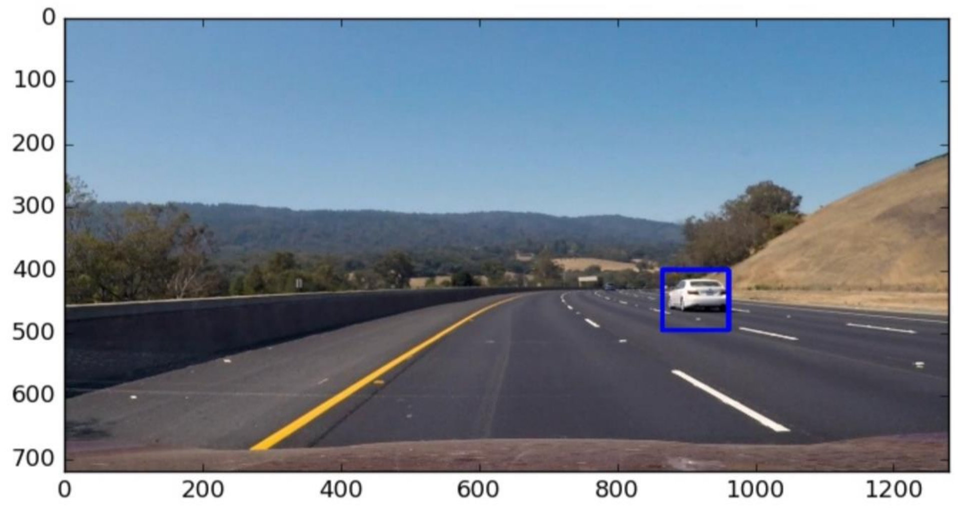
The first step is doing training for classifier. The accuracy should be over 98%.

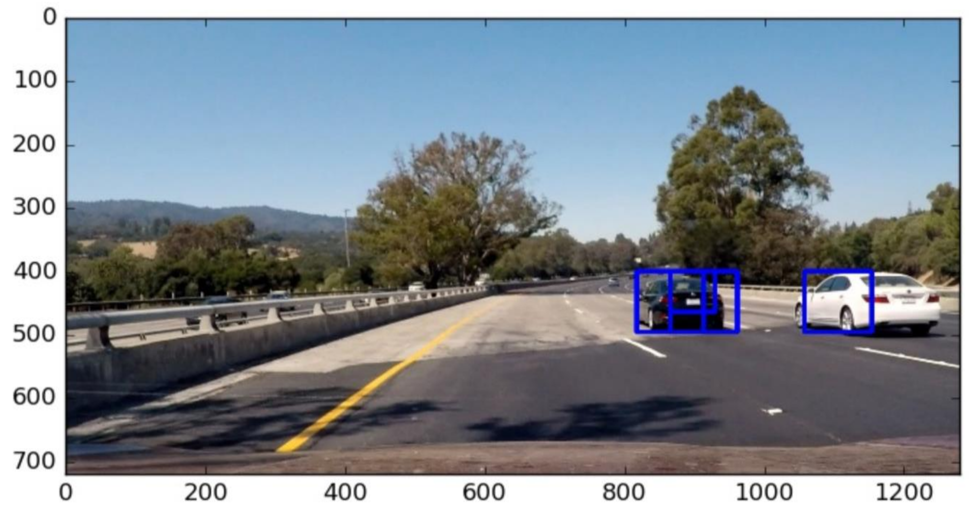
The 2nd step is defining areas and window size for windows search.

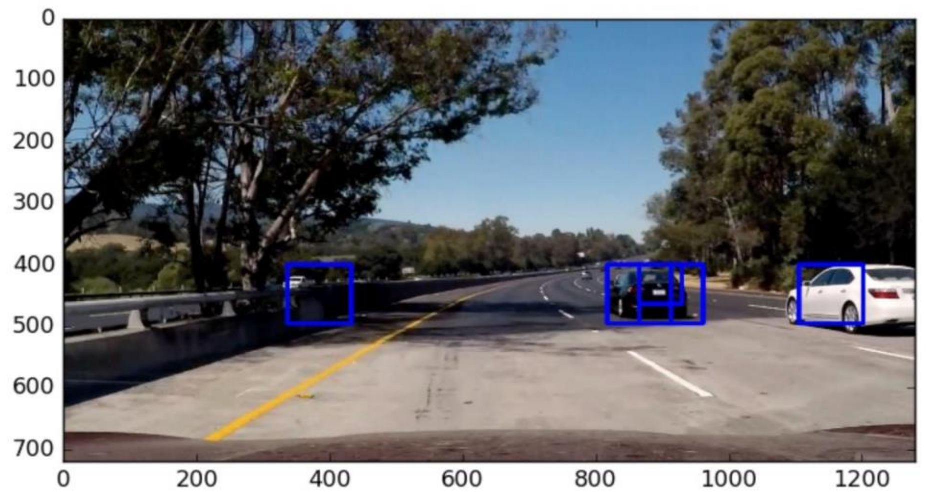
The trick part is choice of color channel and HOG features.

Ultimately I searched on two scales using **YUV all-channel HOG features plus spatially binned color and histograms of color** in the feature vector, which provided the best result. Here are some 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


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

### 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.)

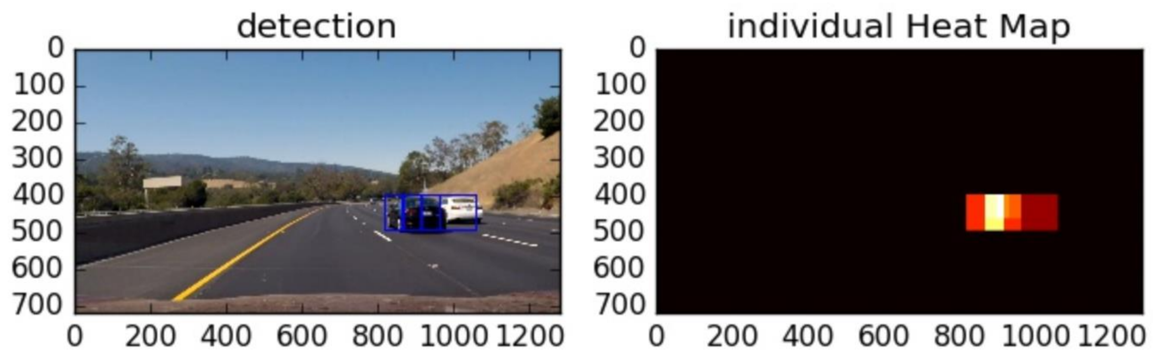
video.mp4

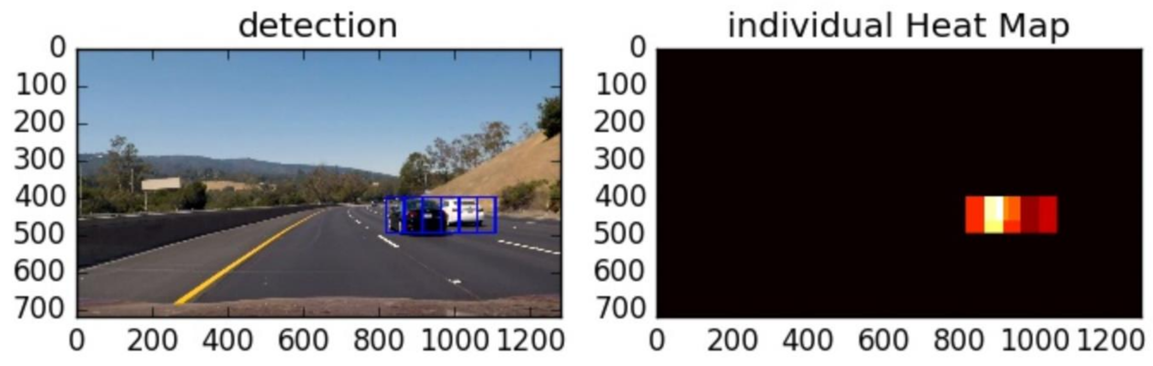
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

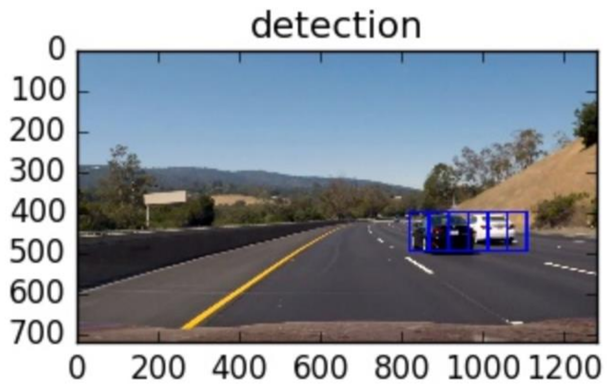
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob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

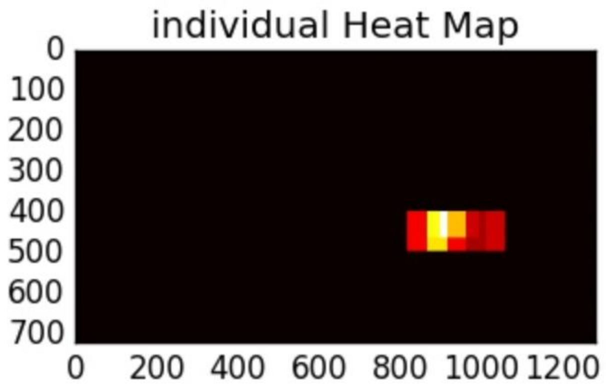
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

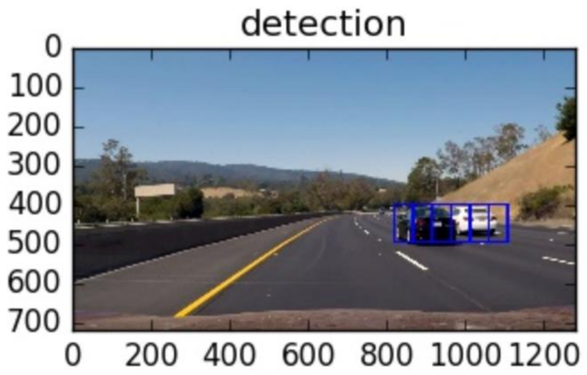
### Here are six frames and their corresponding heatmaps:

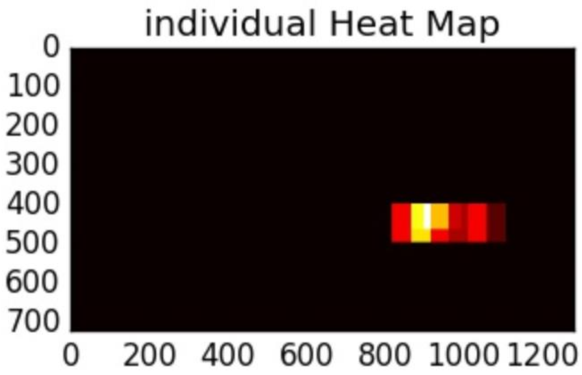


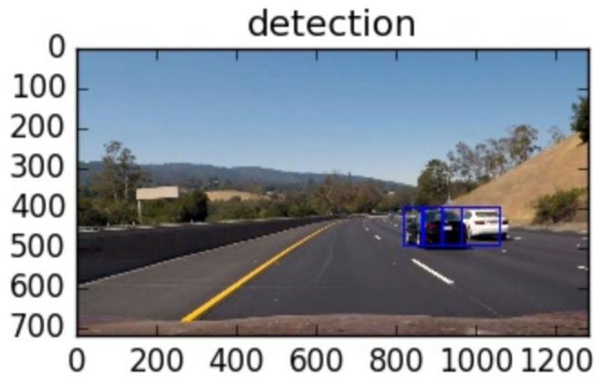


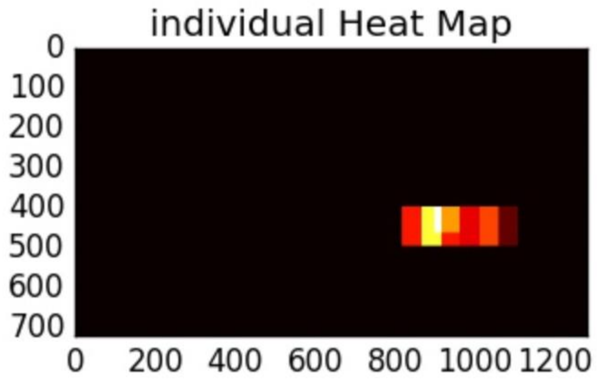

detection
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

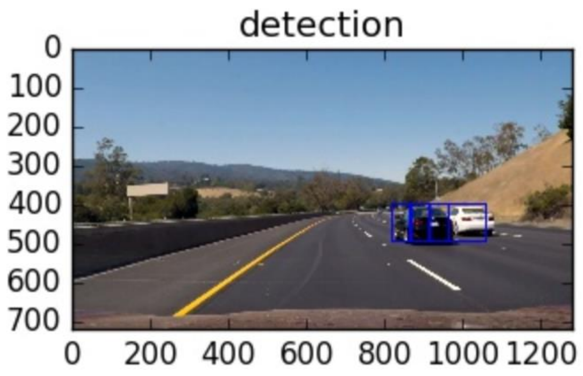

individual Heat Map
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

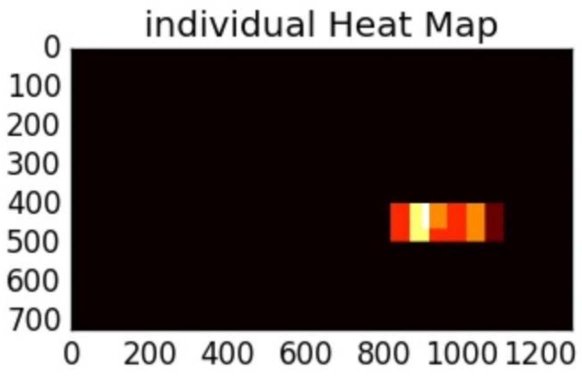

detection
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


individual Heat Map
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

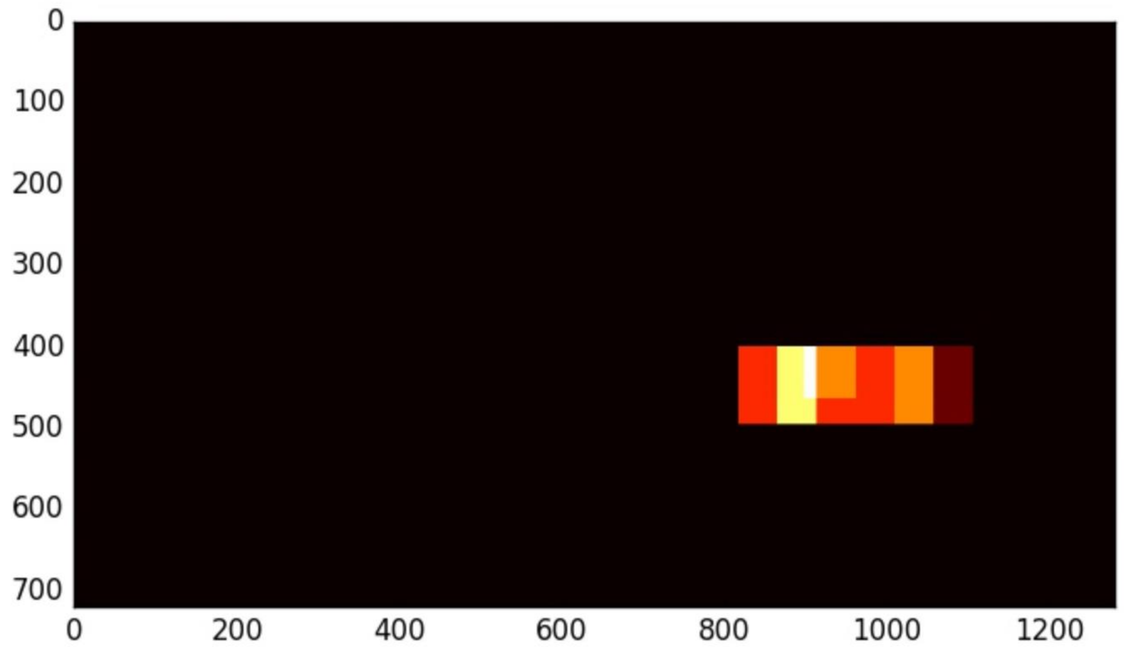

detection
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


individual Heat Map
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

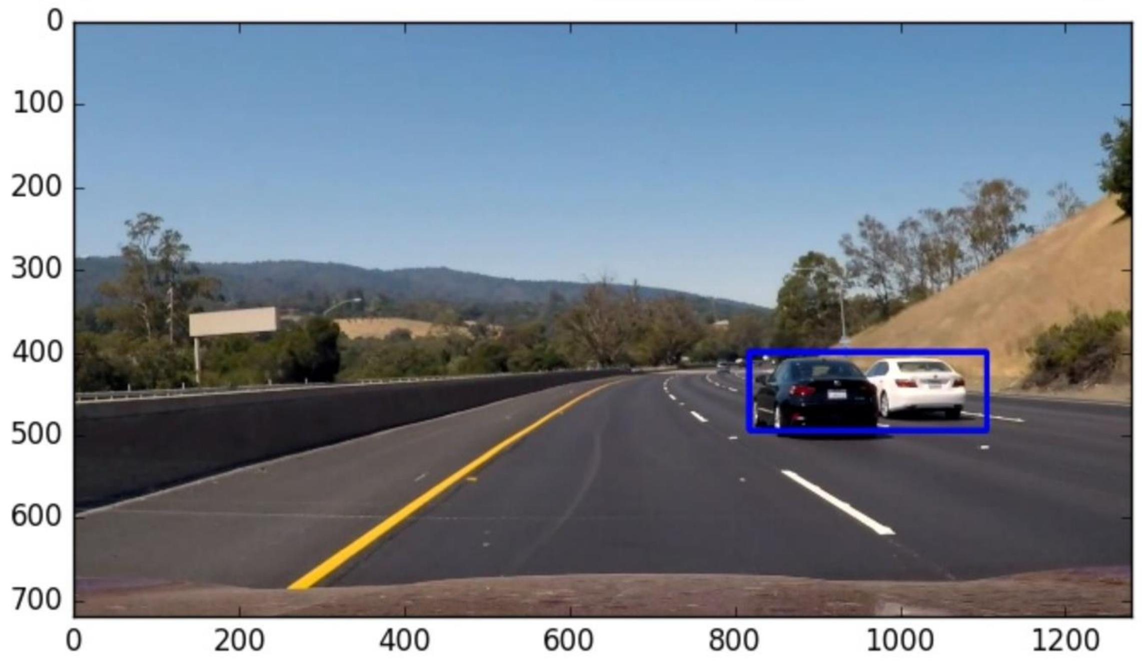

detection
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


individual Heat Map
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

### Here is the output of the integrated heatmap from all six frames:



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



###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?

I am not sure how many frames should be chosen to collect windows info for generating heatmaps. I chose 6. We may need to further optimize this number.

Due to time constraints, I have not included lane tracking to this project. If time allows, I will update it later.