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
- In addition extract binned color features, as well as histograms of color, to the 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 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 is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

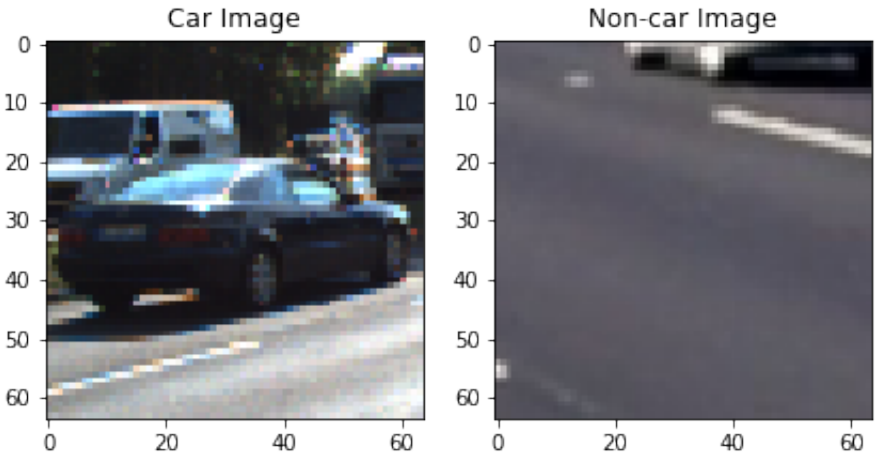
Histogram of Oriented Gradients (HOG)

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 second code cell of the IPython notebook.

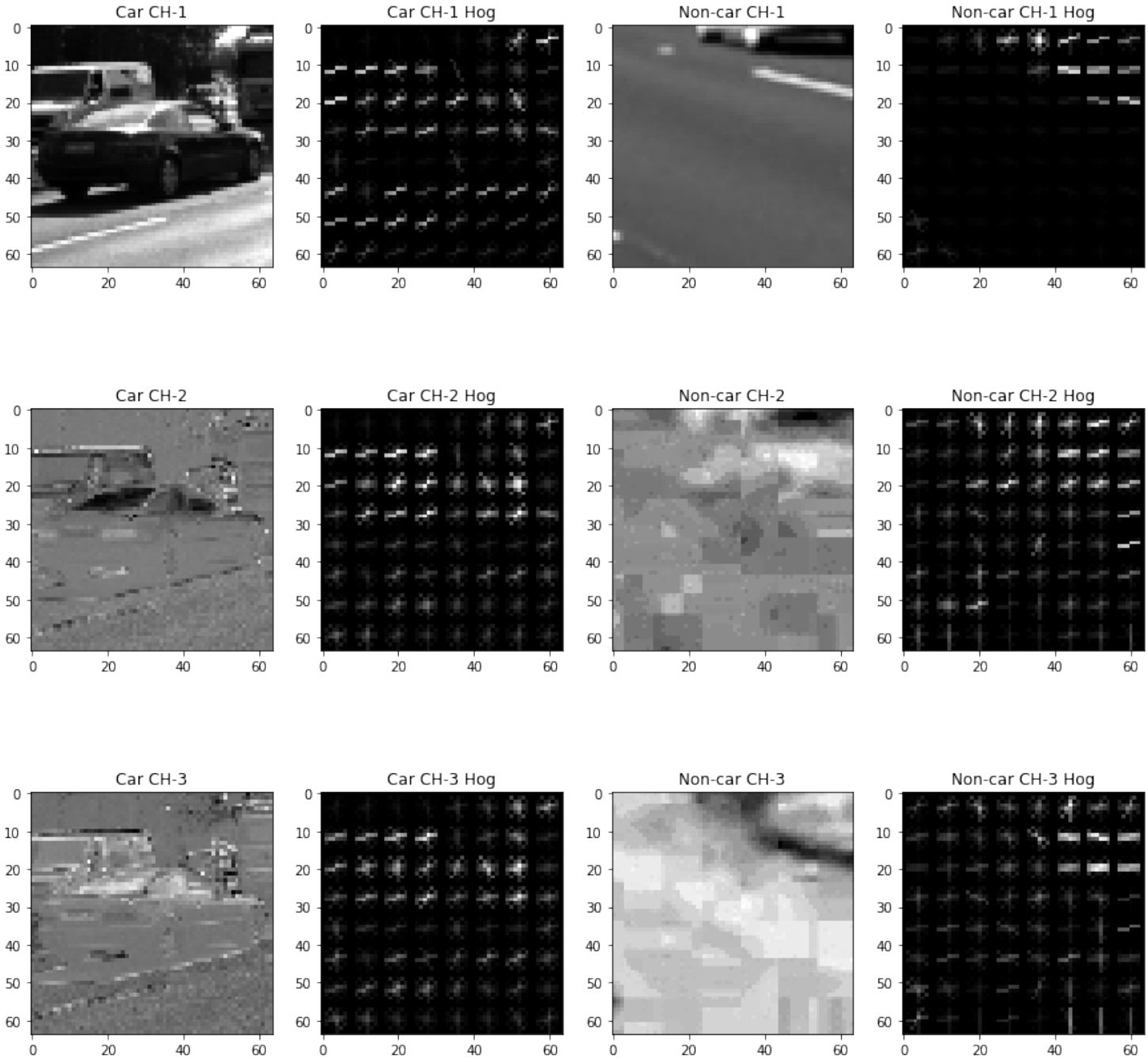
The data is loaded in the first code the of the notebook. Here is an example of one of each of the `vehicle`

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 the 3rd 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.

I tried various combinations of parameters and decided to the parameters suggested in the lecture, i.e. the parameters described aforementioned. The HOG features from all the three channels are utilized to capture more difference between vehicles and non-vehicles.

In addition, the spatial features and color histograms are also extracted for each image in order to train the classifier. I tested to remove the spatial features and the classification accuracy decreases. Therefore, I used the three types of features finally.

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).

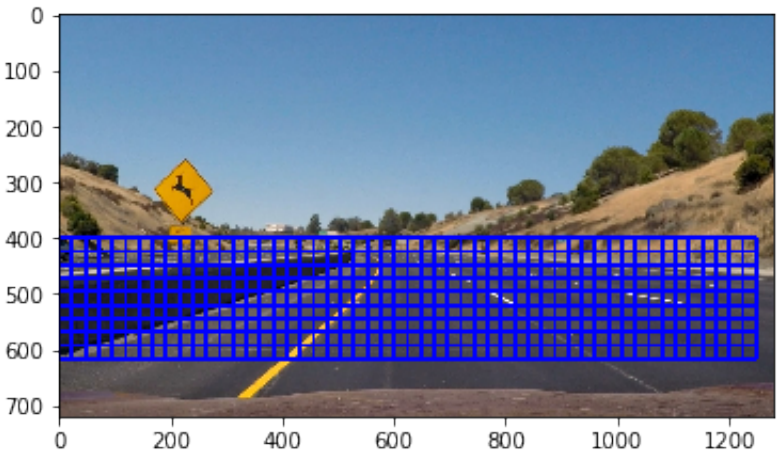
After scaling the data and splitting it into test and train datasets in the 3rd code cell, the GridSerarchCV is utilized to find the best parameters for SVM. The code for finding the best parameters is in the 4th code cell.

The best parameters set are {'gamma': 0.0001, 'kernel': 'rbf', 'C': 10} in terms of both precision and recall accuracy. Using these parameters, the SVM classifier is trained in the 5th code cell.

Sliding Window Search

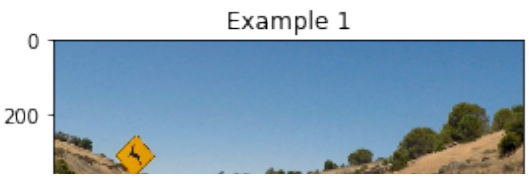
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

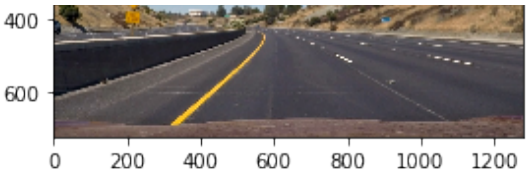
The sliding window search method is defined in the 6th in the function find_cars. The windows are only searched in the lower part of the image and a 8x8 cells are created. The searched windows are listed in the following image.

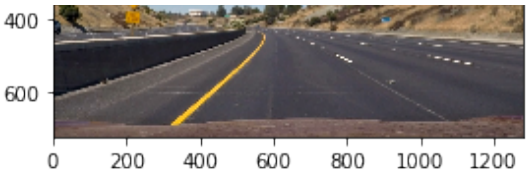


2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on just one scale 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 results of the test images:

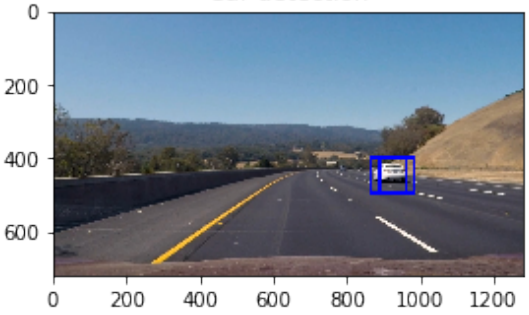






Example 2

Car detection

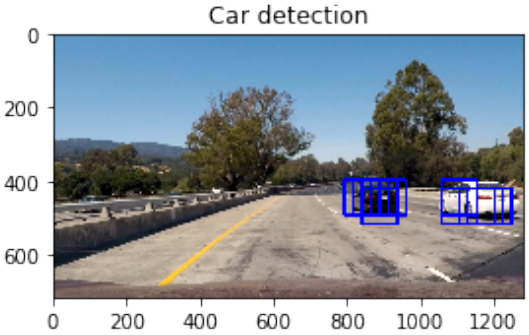


Example 3

Car detection

Example 4

Car detection

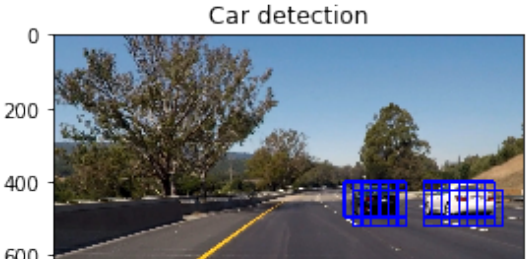


Example 5

Car detection

Example 6

Car detection



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.)

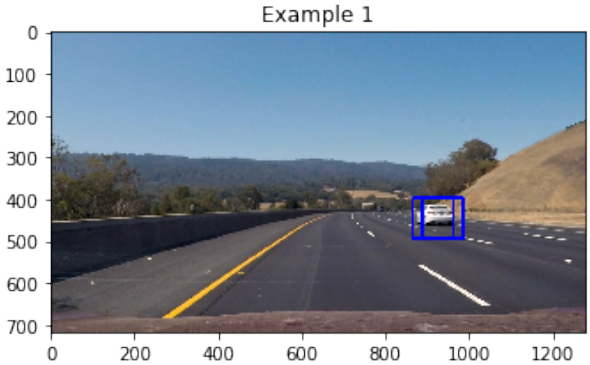
Here's a link to my video result

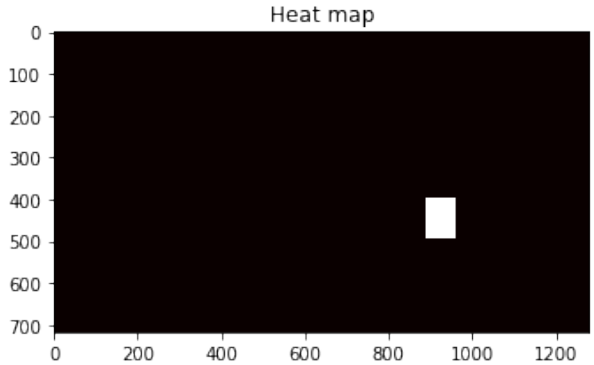
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

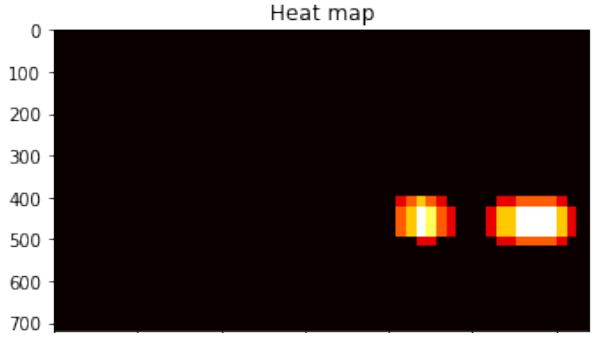
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob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

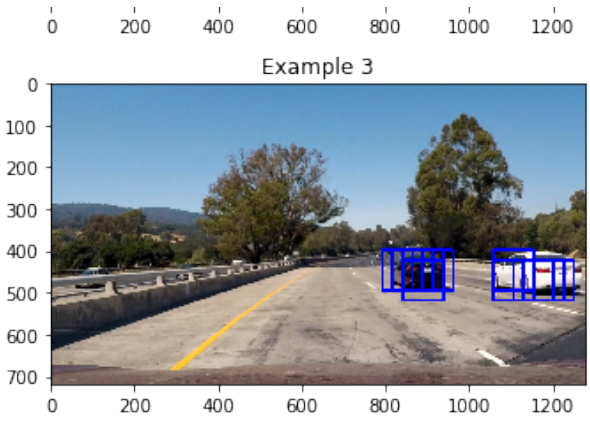
Here are example results showing the heatmap for the five frames that containing vehicles and the finala results with the bounding boxes overlaid on the five images:

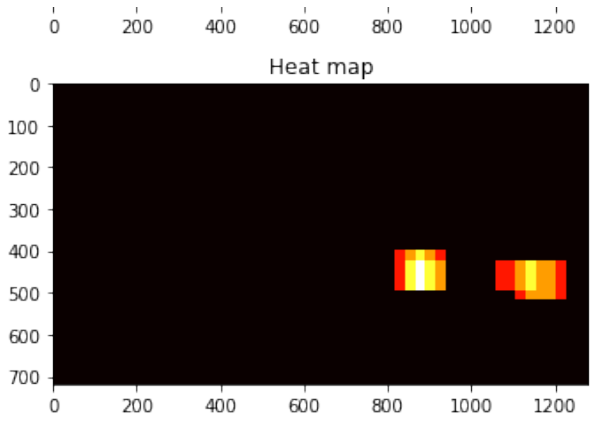
Here are five test frames and their corresponding heatmaps:

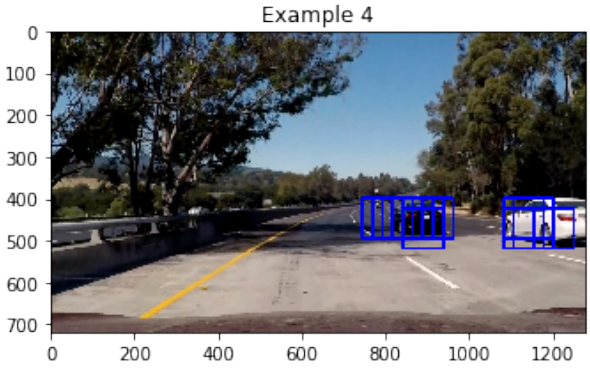


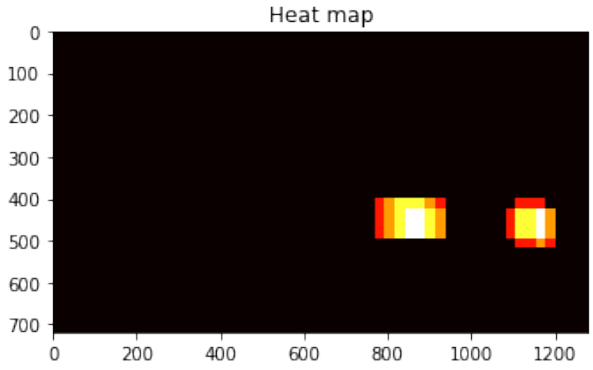


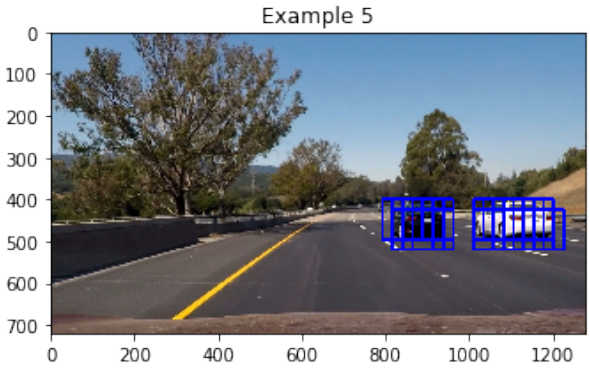


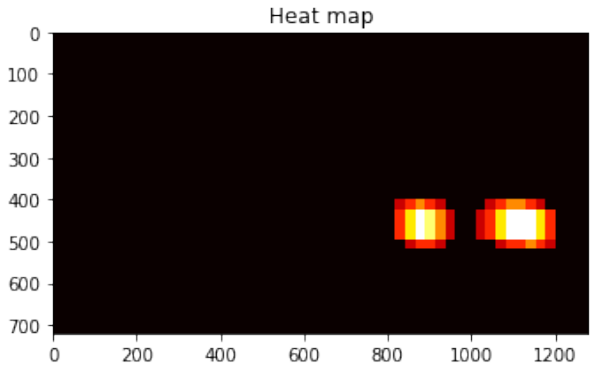










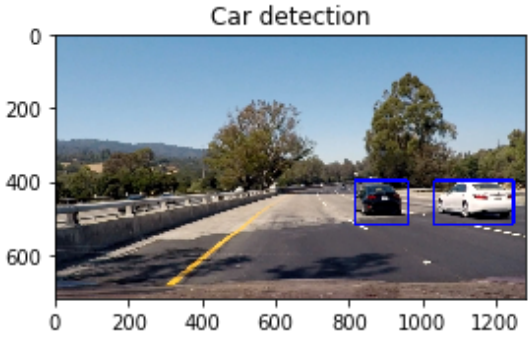


Here are five test frames and their final vehicles detection results:

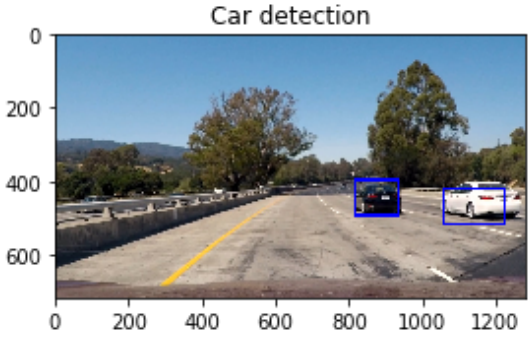

Example 2
0
200
400
600
0
200
400
600
800
1000
1200


Car detection
0
200
400
600
0
200
400
600
800
1000
1200


Example 3
0
200
400
600
0
200
400
600
800
1000
1200

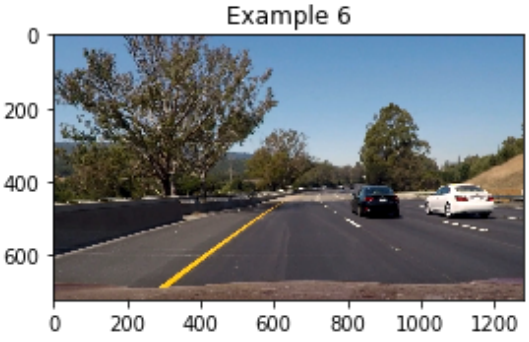

Car detection
0
200
400
600
0
200
400
600
800
1000
1200

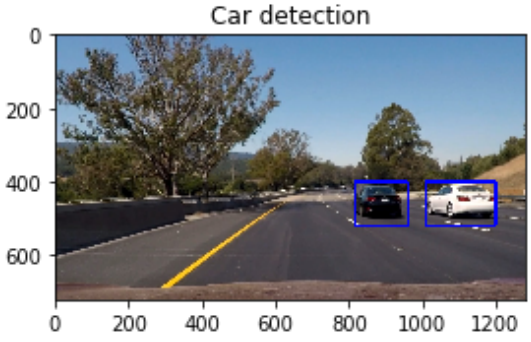

Example 4
0
200
400
600
0
200
400
600
800
1000
1200


Car detection
0
200
400
600
0
200
400
600
800
1000
1200


Example 5
0
200
400
600
0
200
400
600
800
1000
1200


Car detection
0
200
400
600
0
200
400
600
800
1000
1200


Example 6
0
200
400
600
0
200
400
600
800
1000
1200


Car detection
0
200
400
600
0
200
400
600
800
1000
1200

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 current implementation works on the test video. By observation on the video results, the current pipeline MIGHT fail when (1) a vehicle goes partially into the image, (2) the vechiles is outside of the window searching the region (the higher part of the images), (3) the road is white and the vechile is white too. The first one is mainly due to the reason that the classifier trained for featurs extracted for a whole car instead of a partial car, which can be improved if the training images containing vehicles partially in images. The second one can be improved by extending the searching region, which will lead to more computational cost. The last one only happens for few images, which can be eliminated by using a tracking strategy, that using the vehicle's previous detection results infers the current posistion if it is not found in the image.

However, the current method has a very high computational cost, about 4.5 seconds per frame, even thought only the lower part of the images are utlized for searching windows. In order to improve the efficiency, future work will explore how to utilize less features and probably utilzie linear SVM which is fast than SVM with an RBF kernel.