Vehicle Detection and Tracking

Udacity Self-Driving Car Nanodegree Program

Sulabh Matele - May 14, 2017

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, I could also apply a color transform and append binned color features, as well as histograms of color, to my HOG feature vector.
- Implement a sliding-window technique and use my trained classifier to search for vehicles in images.
- Run the pipeline on a video stream 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.

Submission contains following different files and folders:
- **VehicleDetectionAndTracking.py** - This file contains the main implementation of project pipeline and also contains all the data structure and methods used.
- **project_result.mp4** - This is the final output and processed video of the detected vehicles. **visualize_stages.pynb** - This is a python notebook with different generated images.
- **P5_Writeup.pdf** - This document
- **Processed_Images** - This folder contains sample processed images at different stages of project pipeline.

Rubric Points

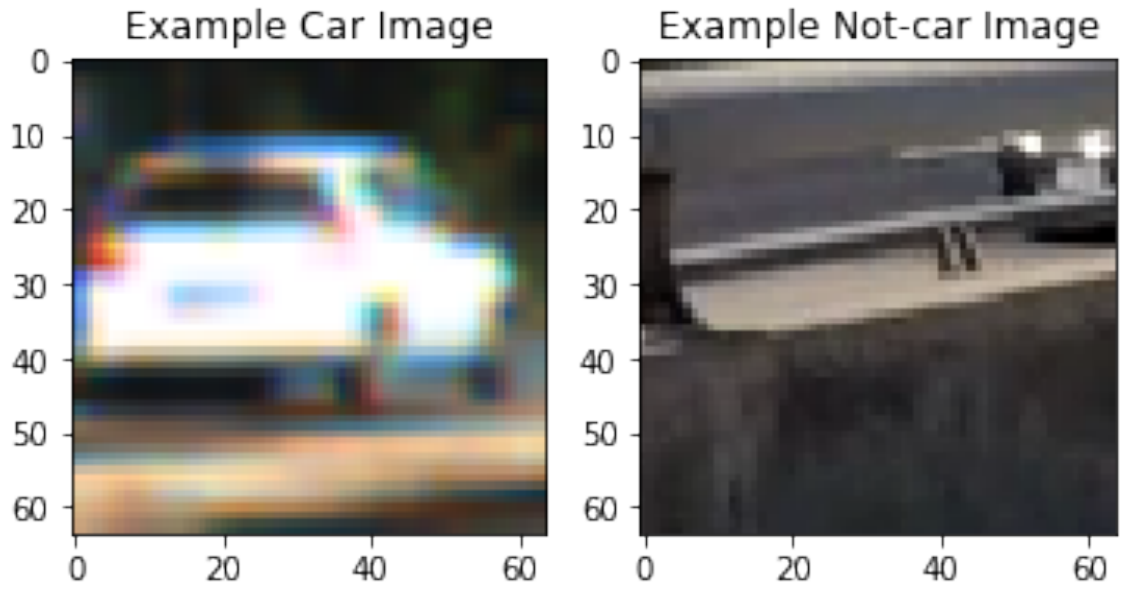
Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Histogram of Oriented Gradients (HOG)

The code for this step is contained in file "**VehicleDetectionAndTracking.py**" at line number 65 method "get_hog_features()".

I started by creating my own required collection of images from the provided set of images of GTI and KITTI, I also learnt that the created set of both the classes should be balanced for the classifier to learn sufficient differences between CAR and NonCar objects. Imbalance in either class could easily cause the over- fitting. I used about 5968 images for each class for CAR and NonCar.

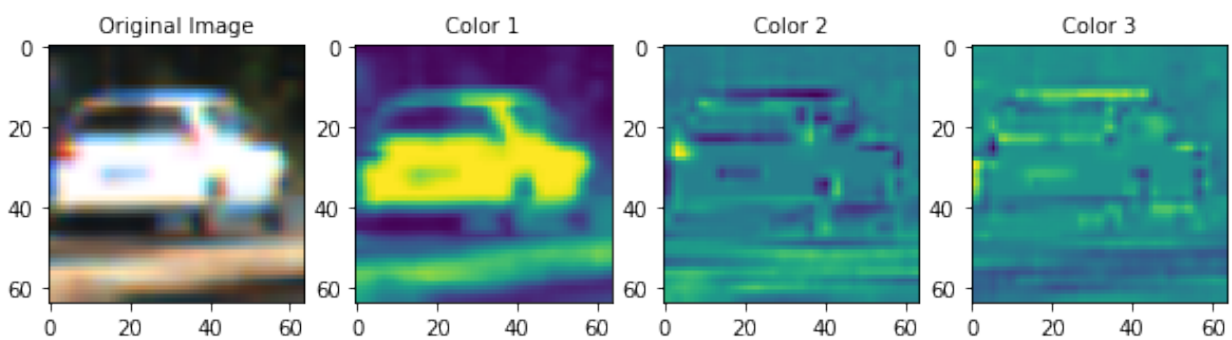
Here is the example image extracted from the same set.



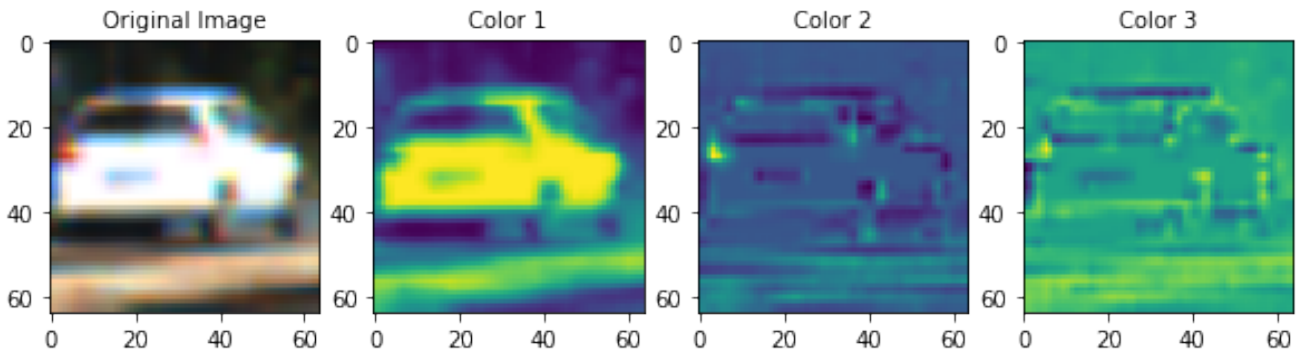
As we know that, HOG - histogram of oriented gradient feature creates an oriented map of color histograms and that map could be useful for us as the trained model could generalize the how a car and non car looks by matching its gradient's orientations.

But the gradient might not give nice results when we work on single color image and hence I tried converting images to different color schemes and then taking HOG for each color channel:

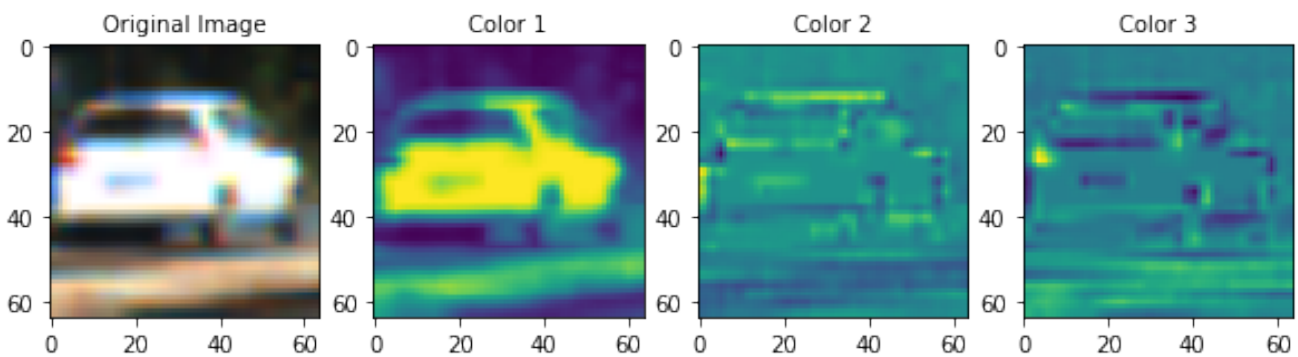
YUV Color space:



LUV Color space:



YCrCb Color space:

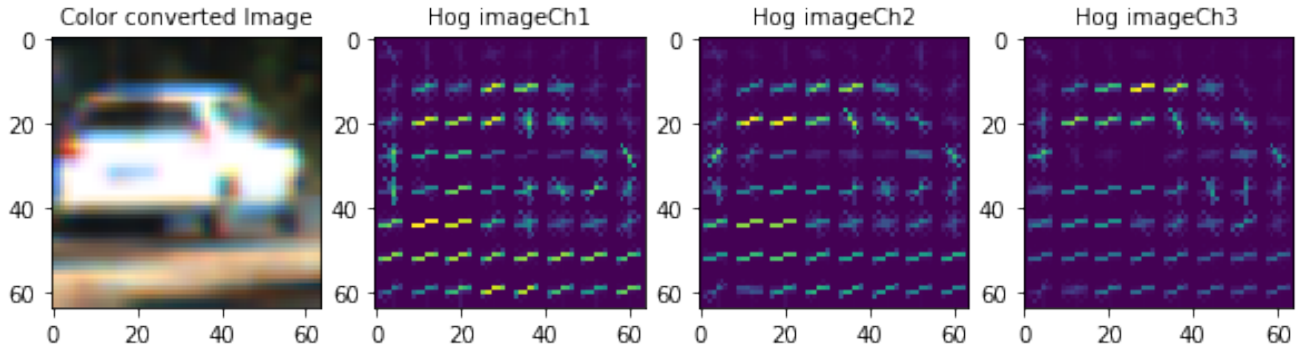


Then converting them to HOG transformed images using “skimage.feature.hog” and setting the combination of following parameters to get the best results for vehicle detection:

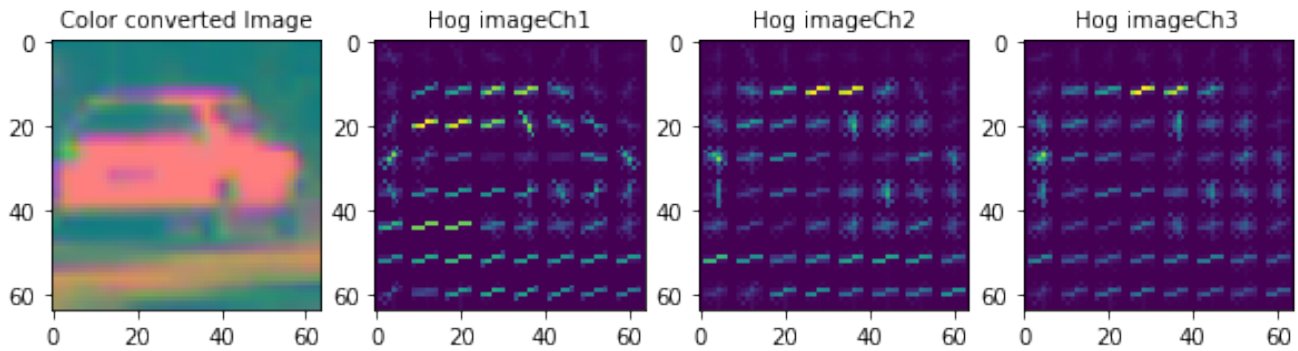
orientation:
pixels_per_cell:
cells_per_block:

Following are some examples of different iterations tried to find the best match and combinations:

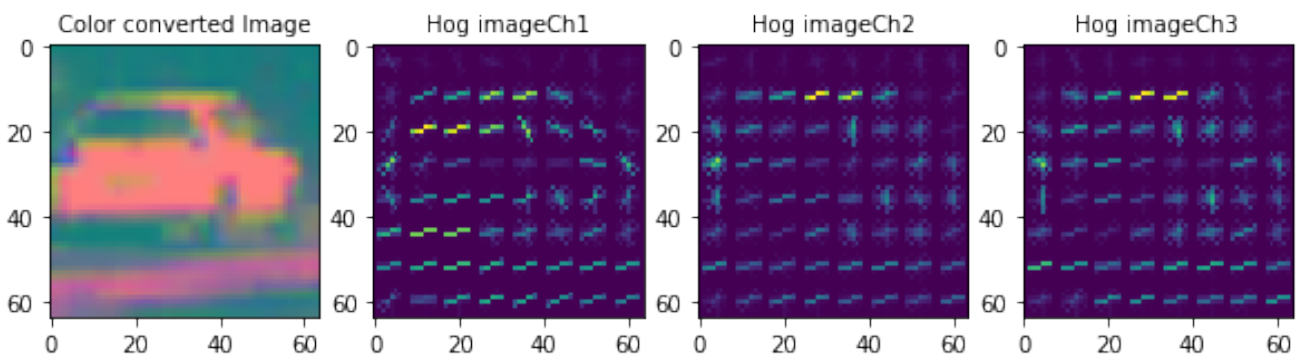
HOG on RGB image:



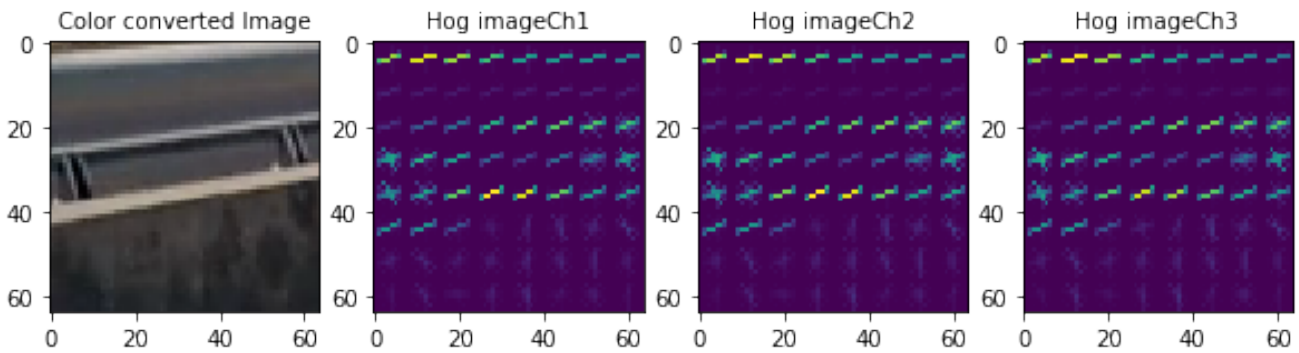
HOG on YUV image:



HOG on YCrCb:



Where as if we see HOG on non car image on simple RGB, it looks like:



Explain how you settled on your final choice of HOG parameters

By observation its very clear that why we took HOG as feature and also, since its based on orientations, it makes it generalized to different sizes and also light conditions for detecting cars and non cars.

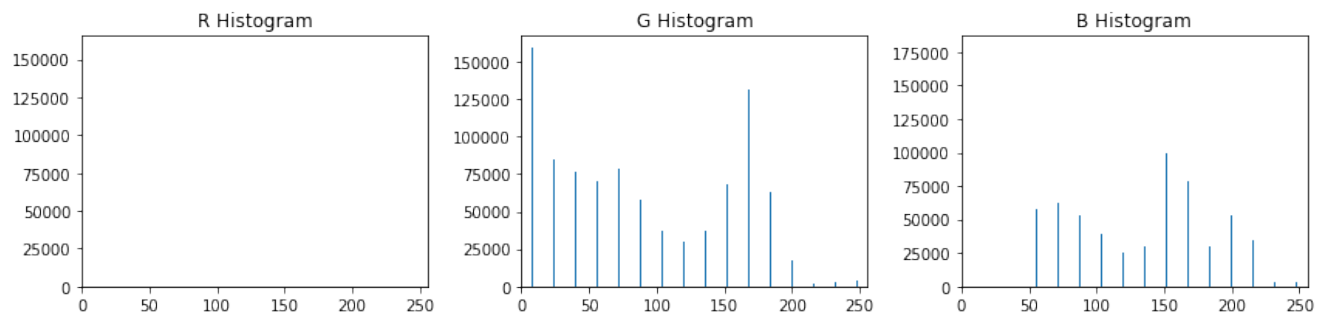
After different trials and based on best detection, I found following combination of parameter values worked for me:

orientation: 9
pixels_per_cell: (8, 8)
cells_per_block: 1

Describe how you trained a classifier

[Code: "VehicleDetectionAndTracking.py", line: 200]
Since I want to make my model more robust then may be just relying on the one type of feature set to predict the Car might not work in all cases, so I have explored more and used histogram of colors as one more feature vector and to more generalize I also resized the images to 32, 32 to make it fast.

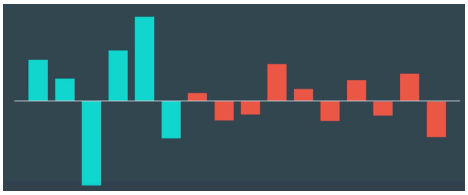
Following is the example image and histogram, just to get idea about how it could work as feature vector for us:



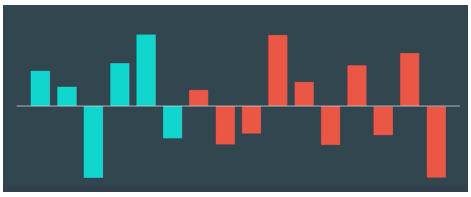
The combination of HOG and histogram creates a combination which we could rely for detection of Cars. But also I used the "sklearn.preprocessing.StandardScaler" to transform the feature vector to generalized range to avoid the one type of feature set dominating the other by values. [Code: "VehicleDetectionAndTracking.py", line: 177]

Following is the effect of Standard scalar on the feature vector:

Before:



After:

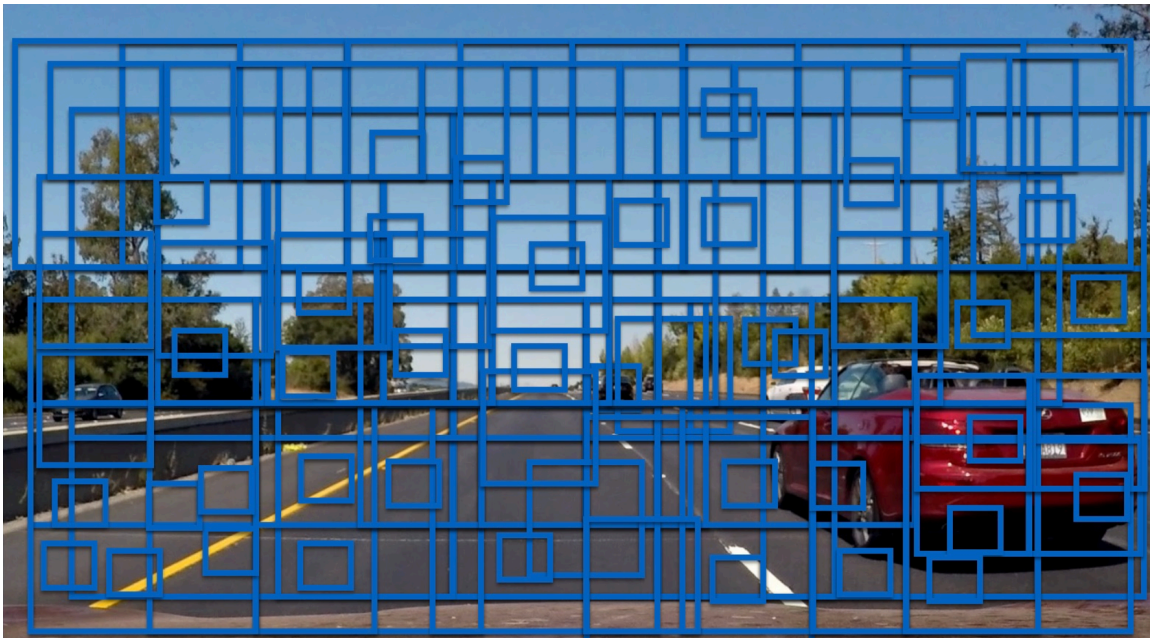


Then I used Linear SVM to train the Classifier, using “sklearn.svm.LinearSVC”, I also split the feature set to training and test set to test the accuracy of the my trained classifier and found the accuracy was about **99** %.

Sliding Window Search

Code: “VehicleDetectionAndTracking.py”, line: 276, Method: “find_cars()”
On a given image to find the car I used the sliding window method which processes the whole image by dividing it into multiple small windows and then process the window and predicts it that window meets the learned pattern of a car.

Typically if we draw the all windows of random size on the given image it looks like:



Show some examples of test images

Now since we know that, the car could be found only at ground and also when we consider the camera position is fixed and we are taking only center camera in consideration for this project then we can just use the region of interest as lower part of the image typically from y value : **400 to 656**, which I used to limit the search.

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. Following are some images from detection:

The overlapping rectangles shows the multiple windows which are predicted by classifier as Car.

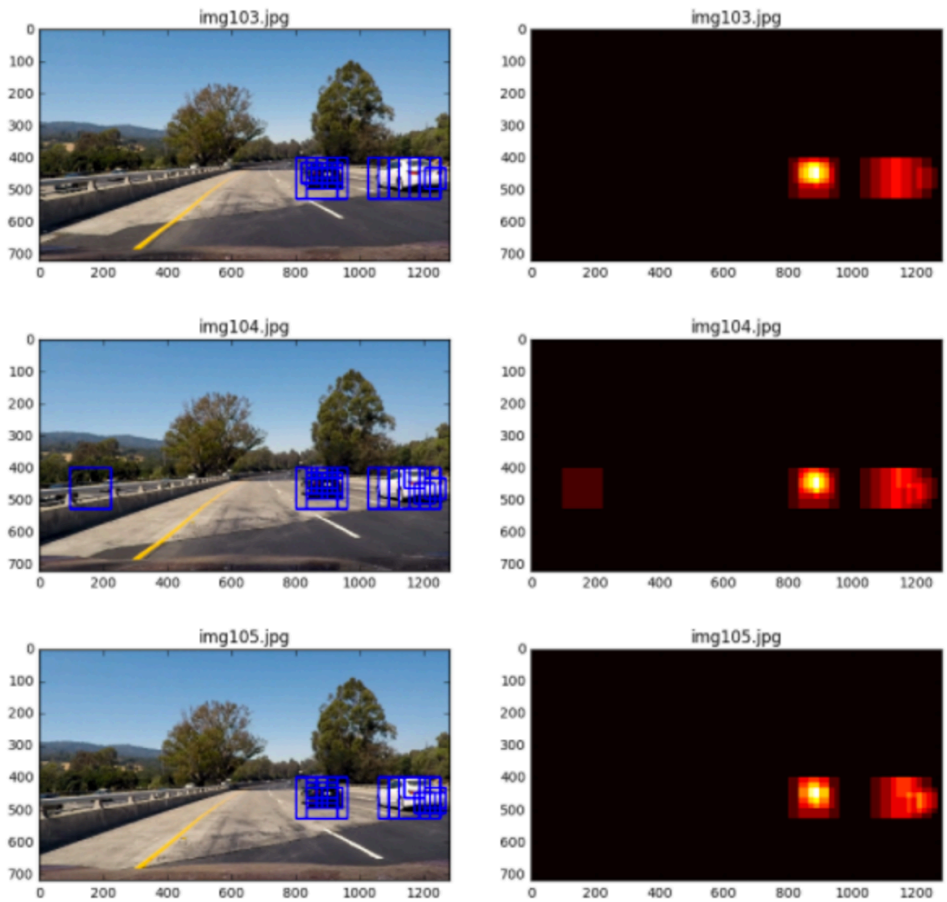
Video Implementation:

Code: "VehicleDetectionAndTracking.py", line: 376, Method "detectCars()"
The project also require me to detect car on a stream of images which are captured on a video and then process the image to construct its feature matrix and then predict it to be a Car or nonCar using the classifier that I trained.

I chose to restrict the search on the lower portion of the image and scaling the image windows to 1.5 to generate similar kind of feature set which was used to train the classifier.

After generating the feature matrix for the test video frame, I recorded the positions of positive detections in each frame of the video using sliding window.

From the positive detection points I created a heatmap and then thresholded that map to 5 to identify vehicle positions. Then I used scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. Then I assumed each blob corresponded to a vehicle. I

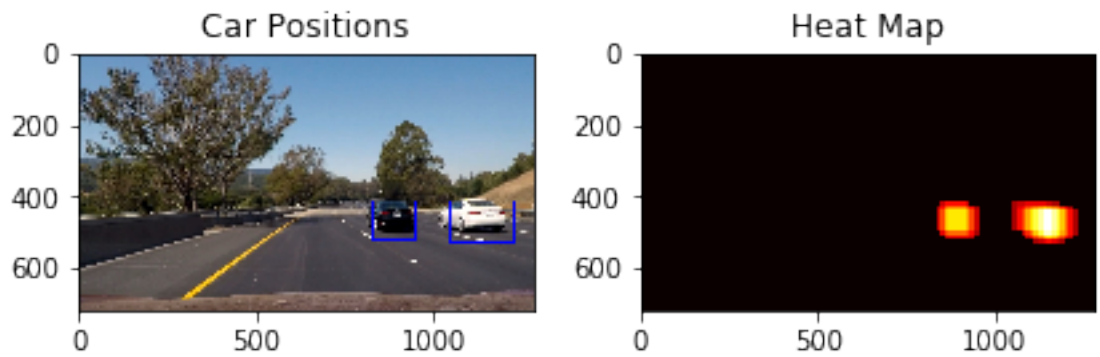


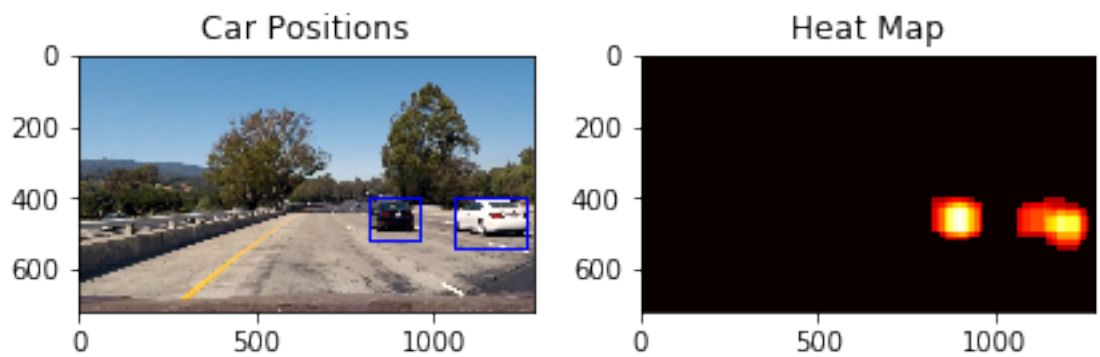
constructed bounding boxes to cover the area of each blob detected.

Above images are 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.

Code: "VehicleDetectionAndTracking.py", line: 350, Method: "addHeatFeature()"

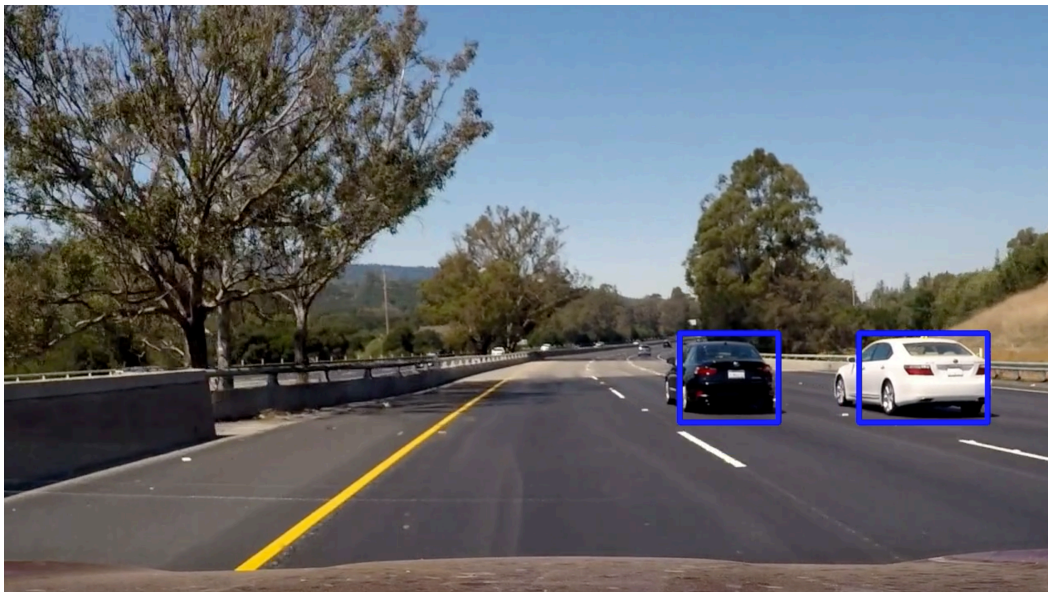
Here is the output image after heat map and label, and drawing the corresponding rectangles:





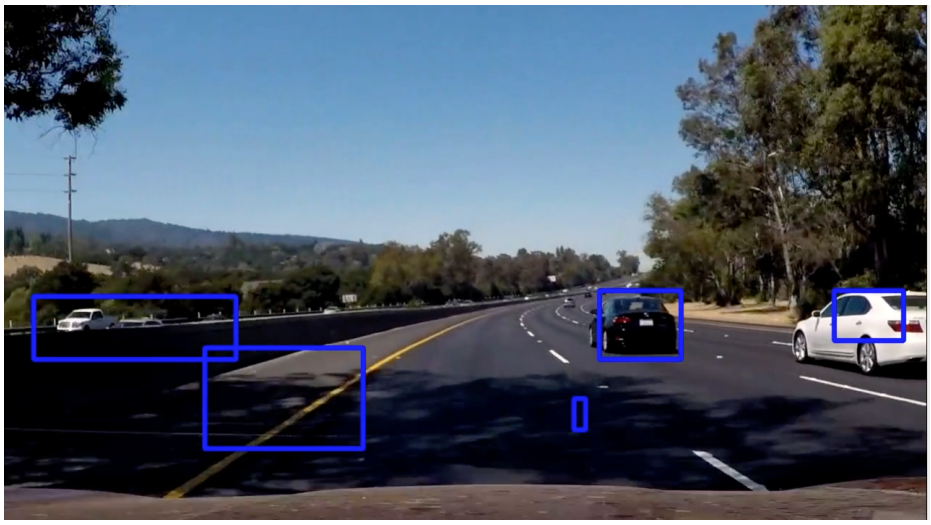
Pipelined video

Pipelined output video is attached as submission. Following is a screenshot from the video:



Discussion

This was challenging project in terms of different tuning problems and also optimizing the classifier for running fast. I mainly faced the false detection problem on the image where it was finding shadows and hence classifying that as Car:



The above problem was tricky to solve since there could be multiple reasons. I solved this issue by trying different color spaces and then choosing the YCrCb with different optimized parameters, also increasing my scaling factor from 1 to 1.5 gradually, which allowed me to remove false detections.

The current classifier has potential to work with very limited type of road conditions, given time to work to improve the same I would like to work on improving its generalization ability to precisely detect Cars. Also I may want to try to train different classifier for different type of environment conditions and design algorithm in such a way that it can choose which classifier to use.

Combining Advanced Lane detection and Vehicle Tracking:

I also tried to combine the project "Advanced Lane detection" with this project to find lanes and cars together, the video is submitted with both the pipelines combined. Following are some screen shots for reference:

References:

— Udacity class room pseudo code helped a lot in giving directions to complete this project in the given time frame.

— Udacity template images, which were very helpful in explaining the results at different stages. (Few images used are from Udacity material for better understanding)