Vehicle Detection

Mohit Pandey

2018
January

Chapter 1

Introduction

Vehicle Detection Project
The goals / steps of this project are the following:
1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier.
2. Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
3. Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
4. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
5. 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.
6. Estimate a bounding box for vehicles detected.

Chapter 2

Writeup/Readme

2.1 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!

Chapter 3

Histogram of Oriented Gradients (HOG)

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

The dataset contains vehicles and non-vehicles images. Each image is an RGB image of size 64x64.

- Vehicle Count: 8792
- Non Vehicle Count: 8968

Figure 3.1: Sample Vehicle Image in Dataset

Figure 3.2: Sample Non-Vehicle Image in Dataset

The dataset is loaded in Cell#2 of Vehicle Detection Project.ipynb.
After loading the data we extract the HOG features from each image. We also calculate spatial and histogram (color based) features in addition to HOG features. This is done in Cell#4 in *extract_features* method. We concatenate all the three kinds of features to return a consolidated feature vector.
The following parameters where choosen after careful experimentations and checking SVM accuracy with each combination of chosen parameters. We were also helped by Udacity forum articles where people discussed the optimal parameters that worked the best for their cases.

- Orientation : 8
- Colorspace : YCrCb
- Pix_per_cell : 8
- Cell_per_block : 2
- Hog Channel : ALL
- Spatial Size : (16,16)
- Histogram Bins : 32
- Histogram Range : (0,256)

It took us 119.29 Seconds to extract the concatednated feature matrix.
Our feature vector length was 5568.

3.2 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 aforementioned feature matrix was first normalized and scaled to zero mean and unit variance. Thereafter we split our dataset containing positive and negative (vehicle and non-vehicle) examples into train and test set of 80% and 20% ratio respectively. All this is achieved in Cell#4 of the jupyter notebook.
In Cell#5, we train an SVM classifier. It took 14.74 seconds to train it. We got an accuracy of 99.41% on the test set.

Chapter 4

Sliding Window Search

4.1 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?

In Cell#6 we define slide window function. This fuction slides a 64x64 (by default) window over the given image and returns a list of all the windows thus possible. We chose an overlap size of (0.5,0.5). We experimented with other overlap and this seems to give the best result [fig]. We make sure to calculate boxes only in lower half of the image since it does not make sense to look for cars in the sky (above the horizon)
In Cell#7, once we have all there prospective car boxes, we feed them to the *search_windows* method along with the test image. We calculate the feature vector of each of the window in test image. If our SVM classifier predicts it as a positive car label, we draw a box on this window using *draw_boxes* method and store these windows as hot windows.

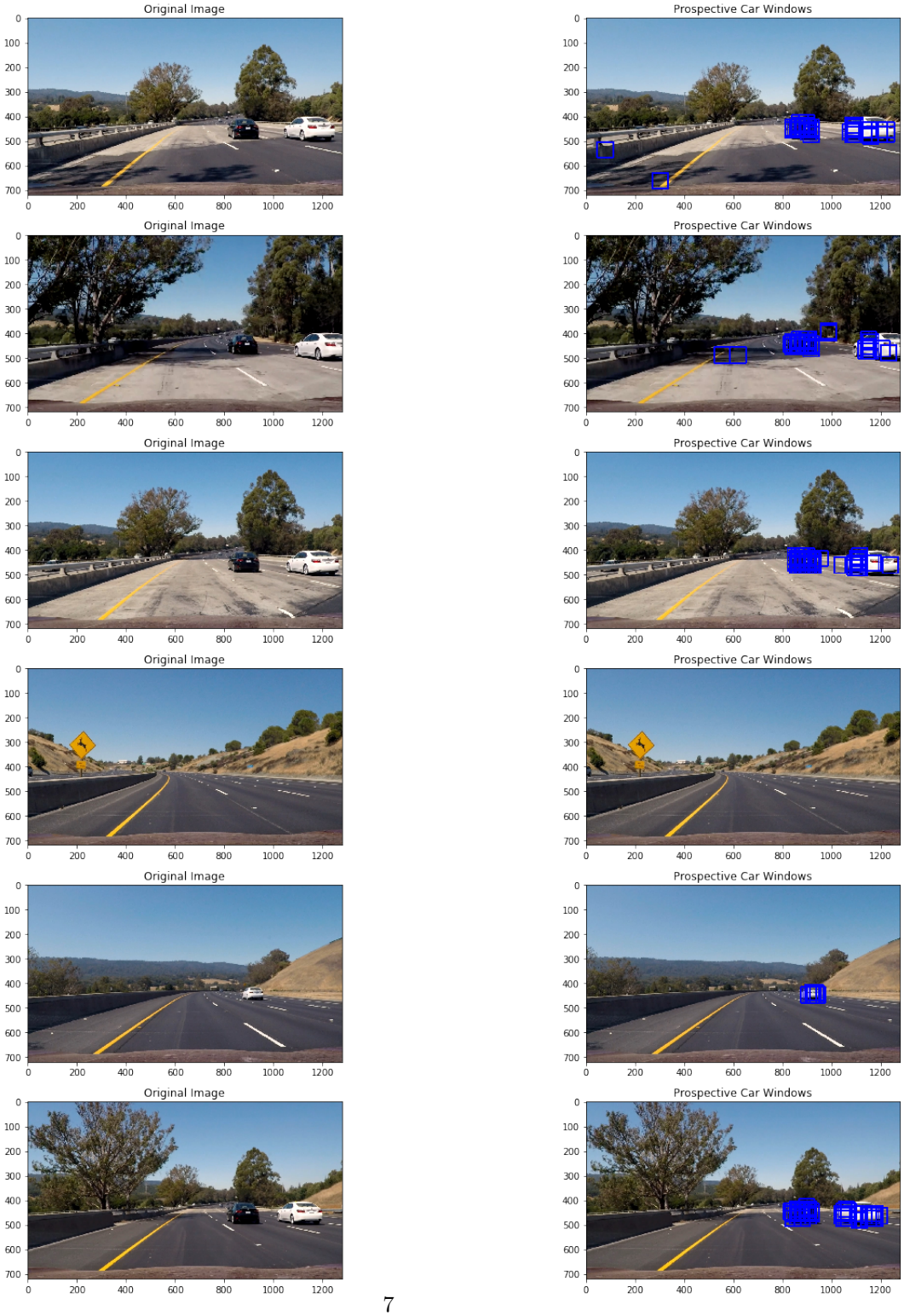

Figure 4.1: Search windows after classification on test images

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

We see in *fig* that the output of search windows after classification performs decently on car images but there is a serious problem of false positives. Such false positives can lead the self driving car to take inappropriate actions and must be removed.
This is achieved in Cell#10 using heatmaps. We used *scipy.ndimage.measurements.label()* to locate the cars using thresholding heatmaps based on presence of cars.

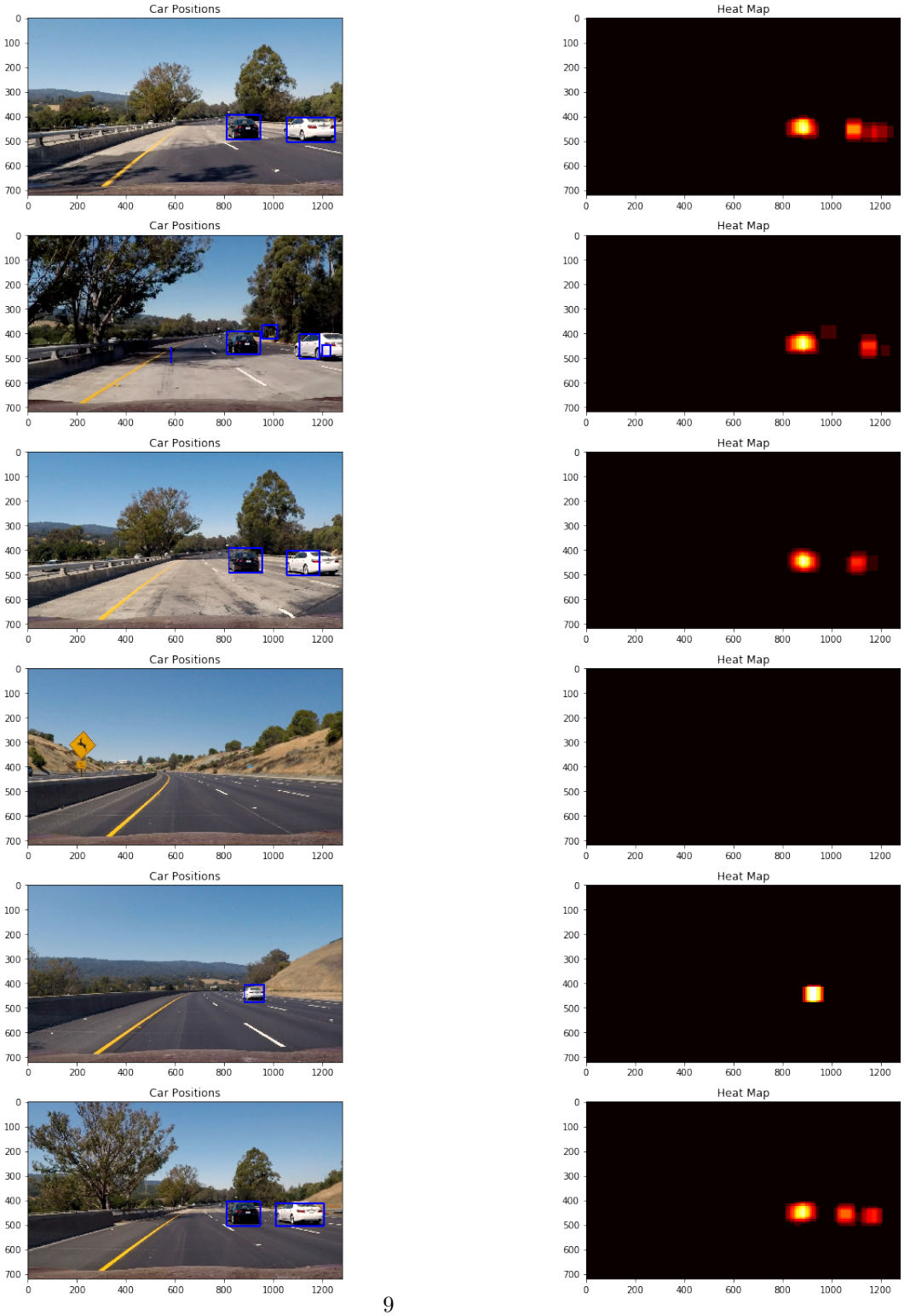

Figure 4.2: Removal of false positives and multiple detection using Heat-Maps

This seems to work well now. However there was one problem.
This method of calculating HOG was too slow to be meaningfully process a video in realtime. Hence we used fast HOG calculation method from Udacity lectures. This is done in Cell#12 in *find_cars* method.
When we combined the windows from *find_cars* with heatmaps, we got very good looking results for car detection and we were ready to proceed to working with video.

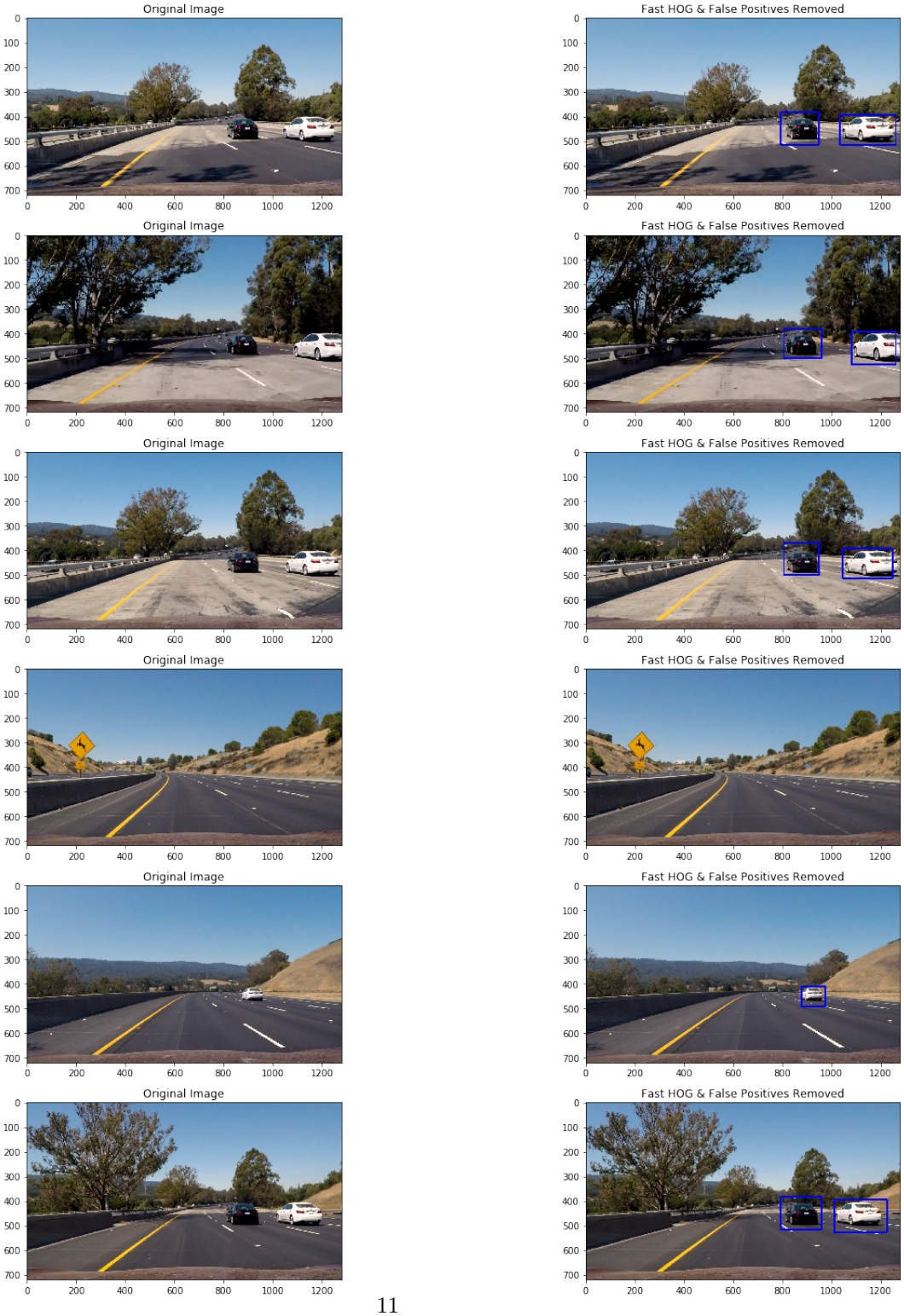

Figure 4.3: Fast HOG and heatmaps together in action

Chapter 5

Video Implementation

5.1 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.)

Link to video:
www.github.com/diamondspark/Vehicle-Detection/blob/master/result_video.mp4

5.2 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.

This was done in two steps.

- Used heatmaps to remove false positives and combine multiple detection windows. This is discussed in detail in section 4.2
- We averaged 3 consecutive frames of the video before making decisions about presence of car. This helps to ensure that we draw boxes for 64x64 windows if they are detected as car for 3 consecutive frames. This helps in reducing false positives that might have escaped in above step. This is done in *pipeline()* in Cell#17.

Chapter 6

Discussion

6.1 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?

- One of the significant hardship faced was wrt accuracy of the classifier. It was easy to achieve an accuracy of about 98% but even that didn't perform very well with loads of false positives. One problem that I observed was if we read in the image using matplotlib, the accuracy hovered around mid 98 but reading the image using opencv2 and then convering BGR to RGB and using the rest of pipeline as it is, yielded accuracy around 99.5%.
- The sliding window step could have been avoided by checking the ROC curve after training SVM. We would straight away see there were too many false positives and we need some measures to tackle it.
- We feel the pipeline gets confused at shadows especially at the junction of shadow and sunny region. This would mean the pipeline would fail at pedestrian (Zebra) crossings.
- The pipeline might also fail if there are vehicles coming in opposite direction since their HOG features wouldn't be the same as in our feature vector.
- Using a convolutional neural network should help us achieve higher accuracy and thus better classification of cars.