Vehicle Detection Project

By Ahmad Chatha

Introduction: 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.

Histogram of Oriented Gradients (HOG): The hog features were extracted following the steps mentioned in the lecture. The code is in *helpers.py* more specifically the function get_hog_features() [l-72] which uses the scikit-image hog() function takes in a single color channel or grayscaled image as input, as well as various parameters. These parameters include orientations, pixels_per_cell and cells_per_block. Following Hog parameters were used:

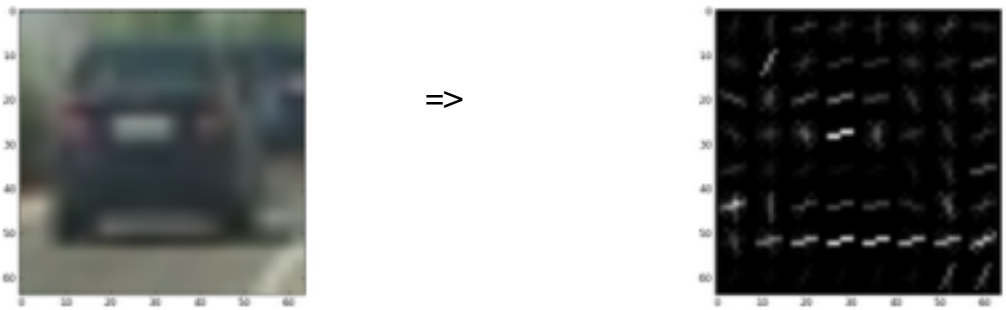
- color_space = 'YCrCb'
- orient = 9
- pix_per_cell = 8
- cell_per_block = 2
- hog_channel = 'ALL'

These were selected after trying many different configurations of parameters. These seem to be giving the best test accuracy for the classifier.

Here is the result for hog feature extraction:

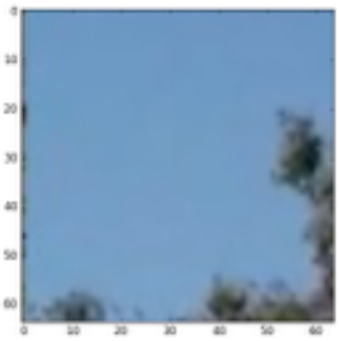
Original Car

Car HOG

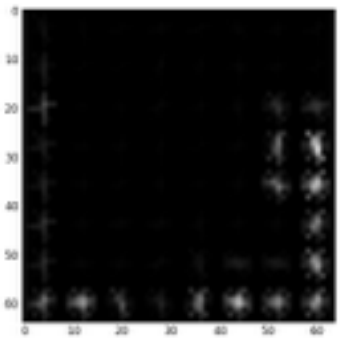


Original Non Car

HOG Non Car



=>



These features were extracted for 5000 images for both car and non-car. They were used to train a SVM classifier. The code is in main.py from line 120-191. Training result are shown below:

```
Total cars: 8792
Total noncars: 8968
63.45 Seconds to extract HOG features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 8460
22.3 Seconds to train SVC...
Test Accuracy of SVC =  0.9907
My SVC predicts:  [ 1.  0.  1.  0.  1.  1.  0.  1.  1.  0.]
For these 10 labels:  [ 1.  0.  1.  0.  1.  1.  0.  1.  1.  0.]
0.0015 Seconds to predict 10 labels with SVC
```

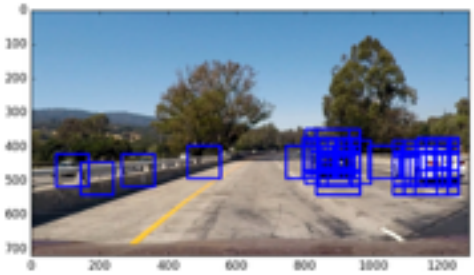
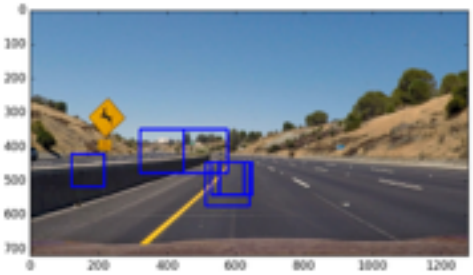
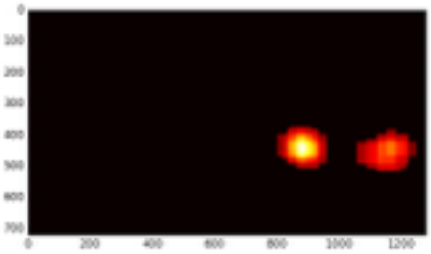
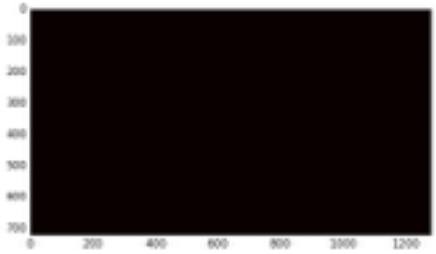
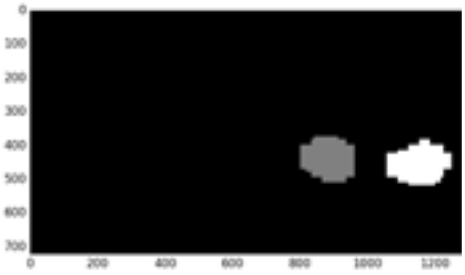
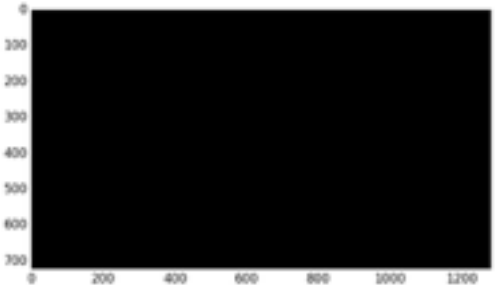
As you can see the accuracy came out to be 99% witch the parameters mentioned above.

Sliding Window Search:

The sliding window search was performed following the steps mentioned in the lectures. The code is in *main.py* in the function find_cars() on line 32 which extracts hog features once and then sub-samples to get all of its overlaying windows. Each window is defined by a scaling factor. The same method performs the predictions and returns the bounding boxes. Three scales were used (1.0,1.5 and 2.0) to collect boxes of different sizes.

The returned bounding boxes actually contain many false positives in addition to the correct boxes containing cars. So to reduce the number of false positives we generate the heat map as suggested in the lecture and perform thresholding. The code can be found in *helpers.py* in the add_heat() and apply_threshold() functions on line 118-132.

Following is an example of both car and non car images:

	Car	Non Car
Original		
All Boxes		
Heat Map		
Labels		

Final

Video Implementation:
The video is available at https://www.youtube.com/watch?v=gLQ1kC1729o or at https://github.com/ahmadchatha/CarND-Vehicle-Detection

Just like the test images examples shown above, the same process was repeated for the video frames except that we only used 1 scale (1.5) only for performance reasons. The performance is pretty good as false positives were eliminated.

In addition to that we also keep the heat maps for subsequent frames inside a global array and continuously trim the array to a size of 35. The thresholding for the video frames was set to a higher value of 45 as a result. The code for processing the video frames is in main.py in the function process_image() on line 99. This helped with reducing the number of false positives.

Discussion/Conclusion:
The main problem in this project was to achieve a balance between extracting more features vs. processing speed. A lot of time is consumed performing feature extraction and doing the window search. As a result only one scale was used for processing video because even that takes around 5 minutes to process the given video. The classifier accuracy was pretty high and may signal over-fitting.

The pipeline kind of fails when we have vehicle overlap and only works for the vehicles we had in the training data. It is unable to deal with the variety of vehicles available on the road and only works for detecting the vehicles from the back side.

In order to improve the algorithm we can possible look into some other features besides hog to improve the real time performance. Also we could use neural networks instead of generating the hog features manually. It seems like we humans don't have to do this much amount of work to detect cars.