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 Points

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

The project is implemented using IPython Jupyter notebook and the contents are below,

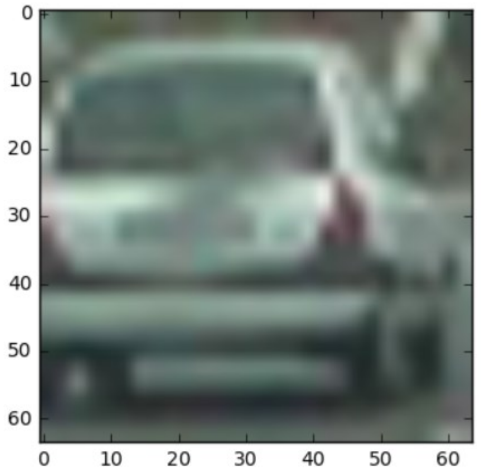
File Name	**Description**
Project5-solution.ipynb	IPython notebook containing solution code
Project5-solution.html	HTML snapshot of the final solution code
project_video_out.mp4	Solution Output video
Project5-solution-writeup.pdf	Project Write up

Histogram of Oriented Gradients (HOG)

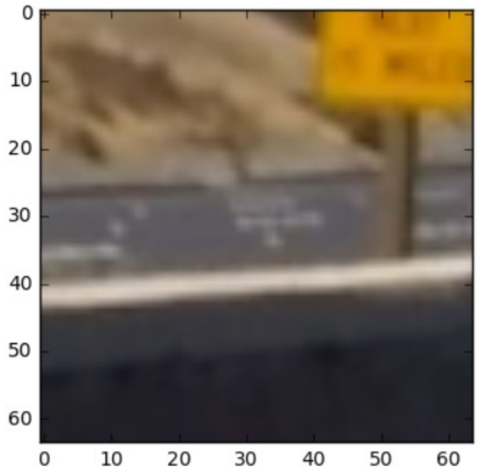
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

In the iPython notebook, the vehicle and non-vehicle images are read from the provided dataset in cells 2 and 3. Each image is RGB and of resolution 64x64 as shown below.

Vehicle



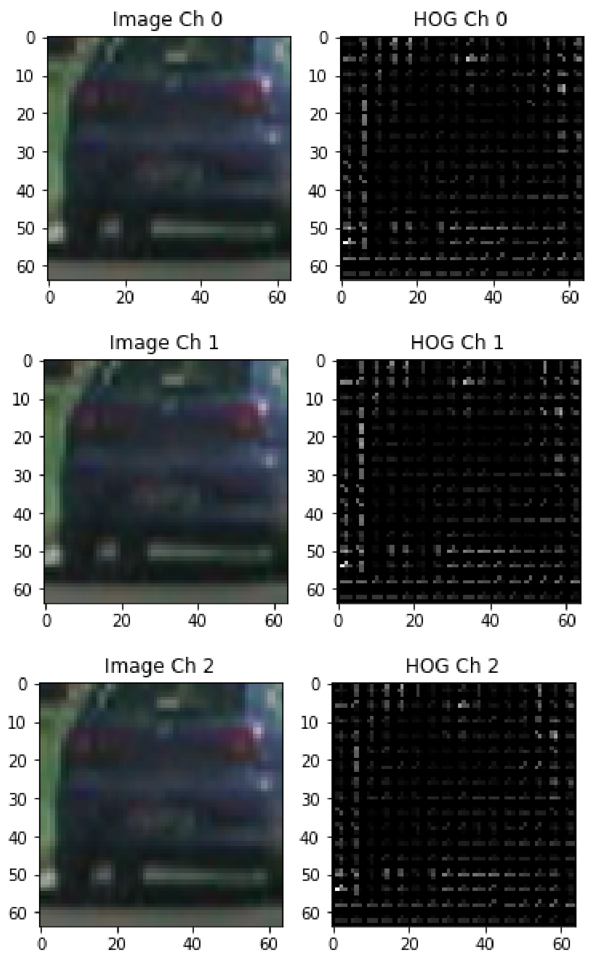
Non-Vehicle



Cells 8 and 12 does the extracting of HOG features from the given training set. From `skimage.feature` I imported **`hog`** class and used it for computing HOG. Parameters are as below

Parameters	**Value**
Color space	YCrCb
Orientation	6 bins
Pixel per cell	4x4
Cells per block	2x2
Num channels	All 3

Hence for 64x64 image, the number of HOG features per channel is (15 x 15) x (2 x 2) x 6 = 5400 /ch.
For all 3 channels it adds up to (5400 x 3) = 16200 features / image



Additionally, spatial binning features and color histogram features were appended to HOG features to make the feature vector more robust. Total feature vector size is, 16,200 + (32 x 32 x 3) + (32 x 3) = 19,368

Parameters	**Value**
Color space	YCrCb
Spatial Size	32x32
Histogram Bins	32

Once the features are constructed they are normalized using **`sklearn.preprocessing`**, **`StandardScalar()`** before feeding to the classifier

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

HOG parameters along with other spatial and histogram features were chosen after multiple trials. All these parameter knobs helped increase classifier accuracy and also reduce false-positives in the test video. Some of the observations are listed below,

- Increasing feature vector size provides more accuracy but takes more time to train and has a tendency to over-fit.
- Decreasing pixels-per-cell from 8 to 4 and increasing number of cells per image makes the feature vector more robust but takes more time to compute.
- Keeping cells-per-block as 2 instead of 4 helps speed up feature vector computation time both while training the classifier and testing on the video.
- Number of bins was reduced from 9 to 6 after reducing cell size from 8x8 to 4x4 just to keep the total feature vector size in check
- Spatial binning of images was kept at a resolution of 32x32 from a source of 64x64 image
- Histogram binning was increased from 16 to 32

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).

A linear SVM classifier was trained to classify vehicles from non-vehicles. The implementation can be found in cell 13 in the iPython notebook. Details of training are provided as below,

- Vehicles were given a class label of “1” and Non-Vehicles were given a class label of “0”.
- The input data set comprised of a total of 8792 Vehicles and 8968 Non-Vehicles of 64x64x3 RGB images
- Using **`sklearn.model_selection`**, the dataset is randomly shuffled and split into 90% training and 10% test set
- This is fed to extract features function (cells 8 and 12) to compute and retrieve a list of HOG and other color features as explained above.

After several iterations, the classifier reported an accuracy of 99% which was used for vehicle detection.

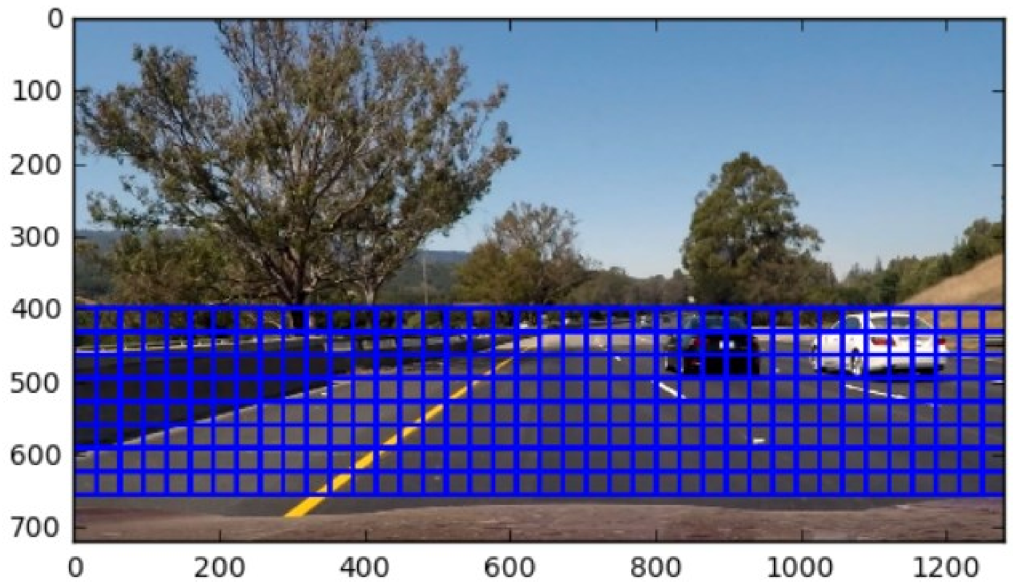
Sliding Window Search

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?

Please refer to cells 16 and 17 in the iPython notebook.

A sliding window mechanism was employed to search for vehicles in the given input frame.

- Since the HOG features used for training requires a 64x64 patch of image, the window sliding across the image is of size 64x64.
- An ROI was set to identify vehicles on the road and also to reduce false positives occurring on trees, skies etc. To identify vehicles at different depth an image pyramid or ROIs was computed with different startY and endYs as shown in the below table. Cars nearer to the camera will be larger in size and farther will be smaller in size, so there is no point in searching for 64x64 sized cars nearer to the camera. This also helps reduces false positives significantly



- In the base image, searching for a patch of 64x64 will detect cars at a far distance. To search cars nearby we need to search in a higher scales. In this experiment, 4 scales were used [1.0, 1.25, 1.5 and 1.75]

ROI Num	**Start Y**	**End Y**	**Scale**
ROI 1 (Base Image)	400	464	1.0
ROI 2	400	528	1.25
ROI 3	400	592	1.5
ROI 4	400	656	1.75

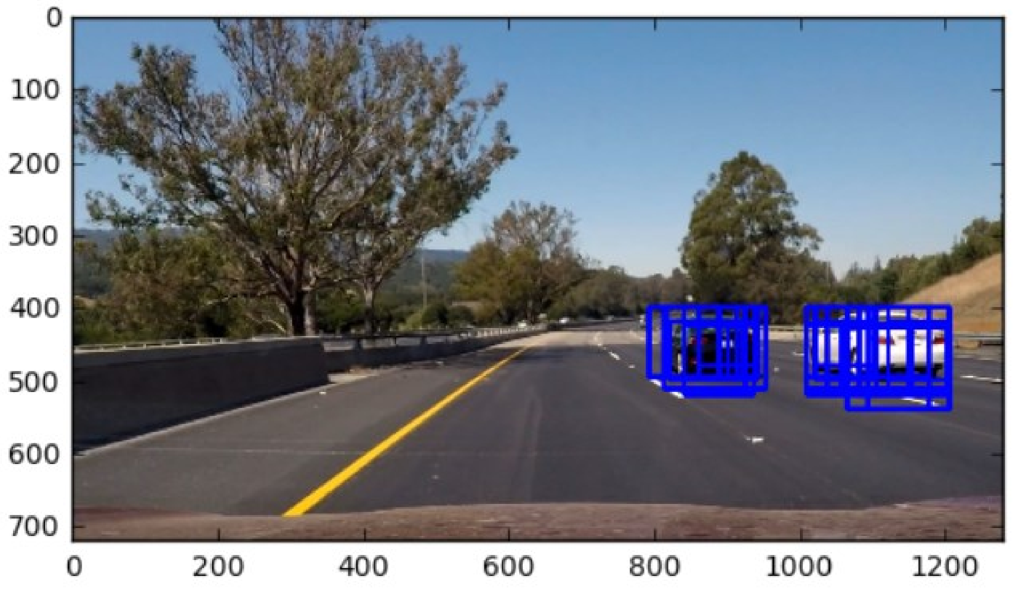
- Once the scales are formed, HOG features were computed and kept in 2D fashion (not flattened). This helps is reusing HOG features as we slide across the image and save significant compute time.

  Eg. HOG feature of **(15 x 15)** x (2 x 2) x 6 was slid by 1 point in feature space to move to the window by 4 pixels in image space.

- Different slide amount (horizontal and vertical) was experimented and finally kept at steps of 4 pixels. The window was traversed horizontally first and then vertically
- At each location, the features are cherry-picked from the ROI and fed to SVM classifier which reports either “1” for detected vehicles or “0” for Non-Vehicles.
- When the classifier reports a “1” the (X, Y) location and size of Window is captured and scaled appropriately to the base scale for rendering.
- Multiple detections can be found for Vehicles and sometimes “Non-Vehicles” as well as we are searching every 4th pixel position and also repeating the same using multiple scales.

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?

Below are some images which clearly detects vehicles on the road and also does a good job of "Not detecting" non-vehicles in the scene. The trained classifier was tried on the below test images first and depending on the number of false detections, HOG parameters, SVM parameters, ROI selection, sliding window amount etc was tuned.



Video Implementation

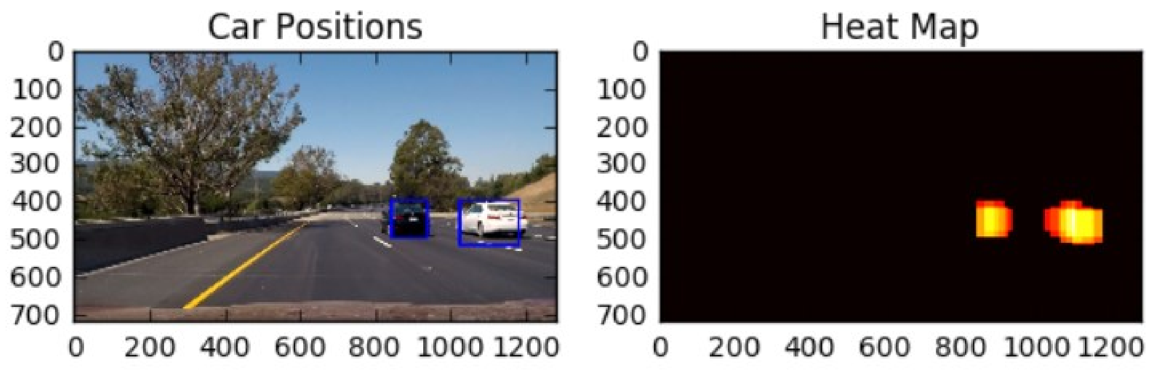
1. ***Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)***

Please refer to project_video_out.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

To merge overlapping detection to one single detection tightly encompassing the car, a heat map was used.
Please refer to cells 15 and 18 in the iPython notebook.

The heat-map computes a histogram of overlapped regions and fits a sort of bounding box to locations were the overlap is maximum as shown below,



Around vehicles, the detections are usually more as we are sliding spatially by 4 pixels in the horizontal and vertical directions and also across multiple scales. This is represented by a strong heat signature as shown in the image. For places where false positives occur, the heat signature is low and can be removed by thresholding. This is a simple and effective method of removing false positives.

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?

Issues

- Eliminating false positives is the biggest challenge. Despite classifier reporting very high accuracy, on actual deployment the performance degraded.
- Since this implementation relied on strong heat-signatures on vehicles and weak heat-signatures on non-vehicles, maximizing the classifier detections around vehicles was important.
- To get more detections on vehicles, the sliding window positions were first made as fine search with the window moving one pixel at a time. This pretty much covered the entire vehicle but also increased the number of false positives on the road. Thresholding the false positives in heat-map helps but does not entirely eliminate them.
- Constructing a robust feature vector was also a challenge. Keeping pixels-per-cell as 4x4 and cells-per-block also as 4x4 increased the feature vector size by 4 times! This took a lot of time to compute features, train classifier and also to perform sliding window on input stream of video. It is important to keep the algorithm complexity under check when deploying on real-time sequences.
- ROI selection was most critical to eliminate false positives. For a model size of 64x64, a vehicle will get detected only at a distance far from camera. Vehicles closer to camera will be larger in size and will only get detected in higher scales (1.75x or 2.0x of base image). Hence reducing the search only near the horizon for detecting vehicles a distance helps reduce false detections to a minimum.

Areas of Improvement

- The detections are too jumpy and the rendered size of the bounding boxes is not consistent between frames. The heat-map approach is a good simple technique to eliminate false positives but not sufficient to keep the detections consistent between frames. Also the bounding box formed is only across maximum intensity region (overlap) which makes the size of the window smaller than the actual size of the car especially when the car is close by.
- What is more effective is usage of centroid finding algorithms such as meanShift which takes a bunch of points and shifts the center towards the centroid of the cluster.
- To solve jumpiness KalmanFilter can be used by tracking (X, Y and window width) locations. This will not only help eliminate window wobble across frames but also help eliminate false positives.
- It is important to separate and match tracks (or vehicles) across frames especially when they are close by (overlapping) by using NCC (normalized cross-correlation) techniques.
- Dynamic selection of ROI's and smart search strategy will help improve detection time while running on real-time sequences and also false positives.