Vehicle Detection and Tracking

Udacity Self Driving Engineer –Term 1 Project 5

Overview

The goals of this project are the following:

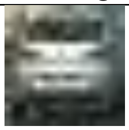
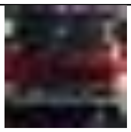
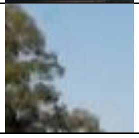
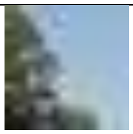
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier.
- As required, apply a color transform and append binned color features, as well as histograms of color, to HOG feature vector.
- Implement a sliding-window technique and use 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.

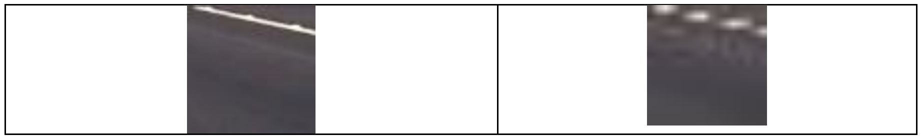
The entire project code with relevant comments are implemented in vehicle_detection.py file.

Preparing and Processing Feature Vectors:

The input images – vehicles and non-vehicles are extracted and feature vectors are prepared to train the classifier. I chose the feature vector containing HOG, spatial bin, color histogram and gradient magnitude features.

Spatial Binning: The input images of 64x64 are resized to 16x16 and taken as part of the feature vectors. The value of 16x16 is chosen on trial and error with satisfactory compromise between the size and the quality of the results. Below figure shows the example images from vehicle and non-vehicle classes and the output feature.

Input image	Spatial binned image (enlarged)
	
	
	



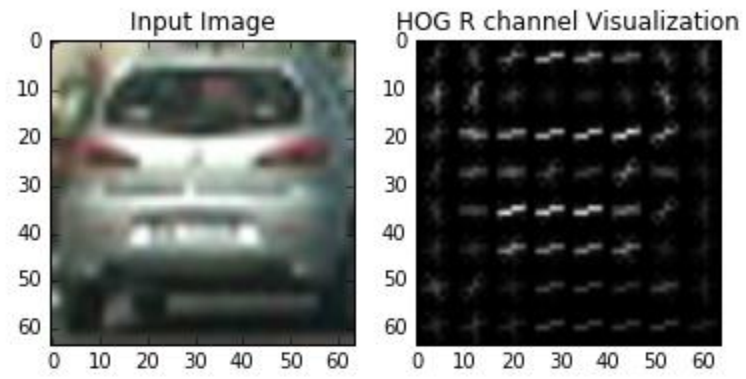
Histogram of Oriented Gradients (HOG): HOG feature with below parameters are extracted from the input images:

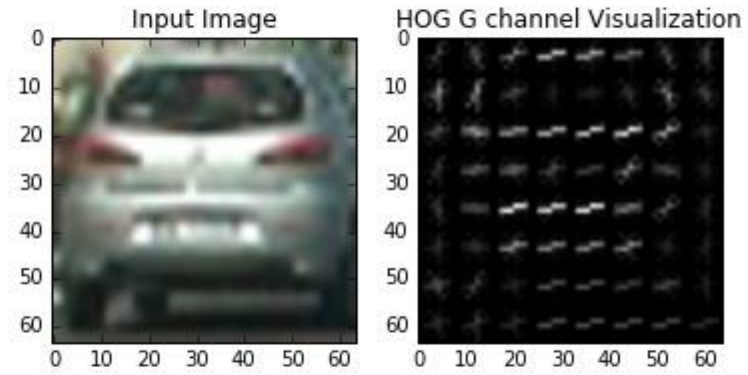
orient_value = 9

pix_per_cell_value = 8

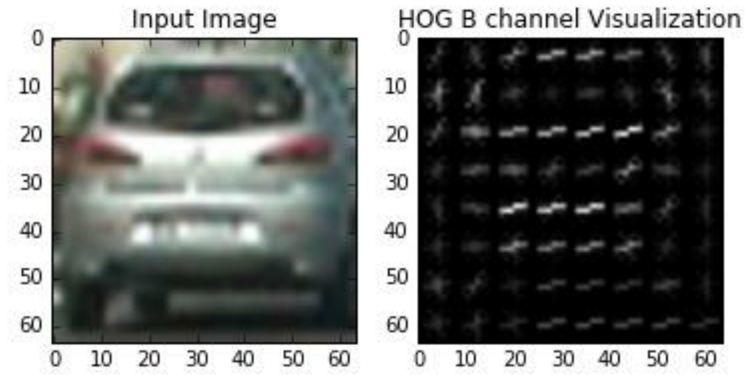
cell_per_block_value = 2

The above values were arrived after trying several different combinations of the values based on quality of the results. Below figure shows the example images from vehicle and non-vehicle classes and the output feature using above derived values.

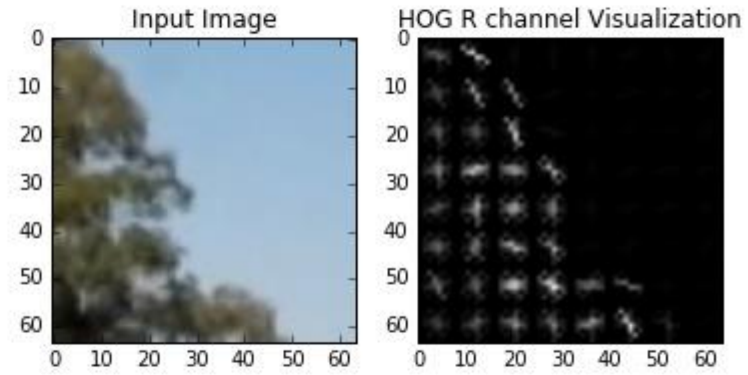




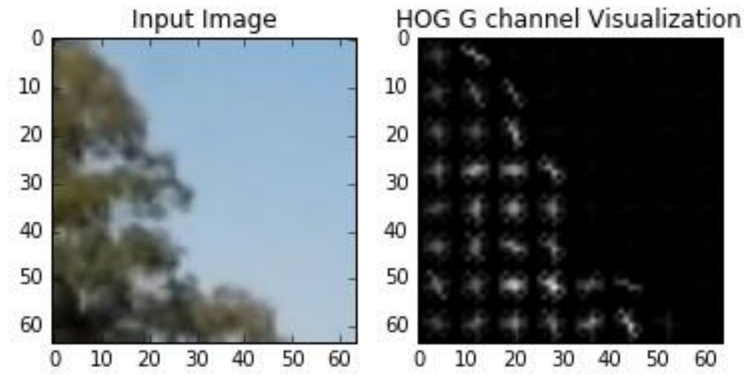
Input Image
HOG G channel Visualization



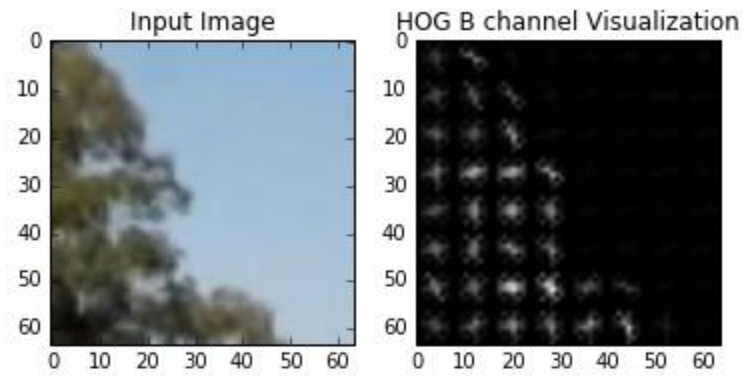
Input Image
HOG B channel Visualization



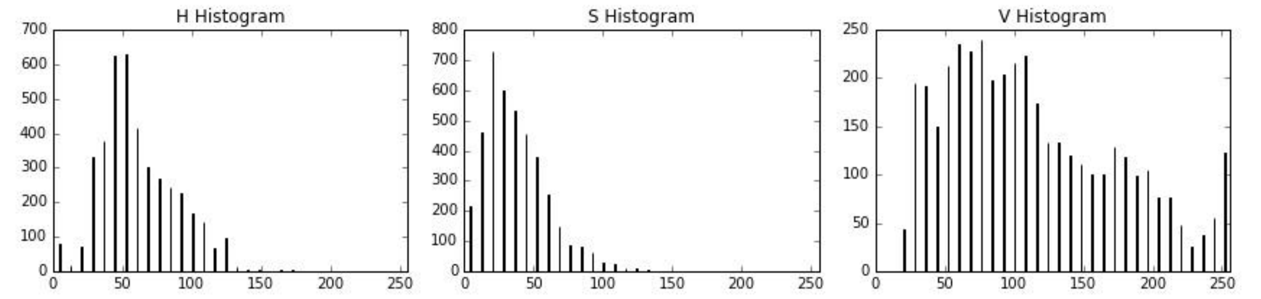
Input Image
HOG R channel Visualization

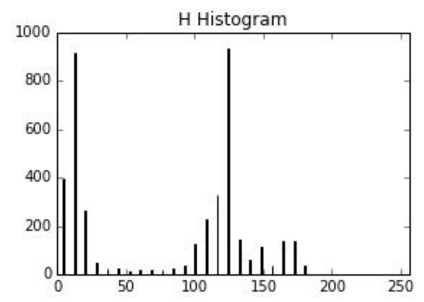
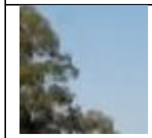
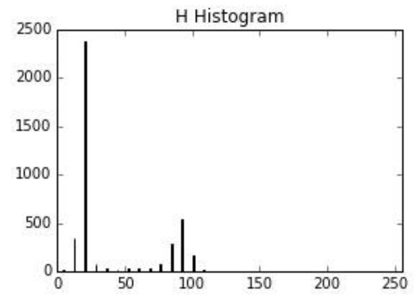
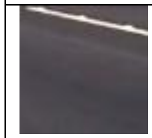
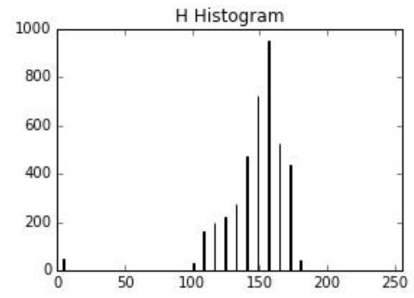


Input Image
HOG G channel Visualization



Color Histogram: Color histogram bins are extracted with size of 32 bins. The color histogram is from HSV color space values. I experimented with different color spaces (HSV, LAB, YCrCb) and used HSV color space based on the quality of results obtained. The size of 32 bins is chosen based on the quality and size compromise. Below figure shows the example images from vehicle and non-vehicle classes and the output feature.

Input image	Color Histogram
	

Gradient magnitude: The gradient magnitude binary for the input image is taken with thresholds of (20,255). The Sobel operator is used along x and y directions to calculate the gradient magnitude value. Below figure shows the example images from vehicle and non-vehicle classes and the output feature.

Input image	Gradient magnitude images

For feature extraction step, process_features() routine extracts the list of input images and calls extract_features() routine to extract the feature vectors. The features are then scaled to zero mean and unit variance using StandardScaler(). The feature vectors are then split into test (80%) and validation inputs(20%).

Training the Classifier:

I chose SVM linear classifier for this project. Decision tree classifier was also tried, but the results were not satisfactory and hence dropped.

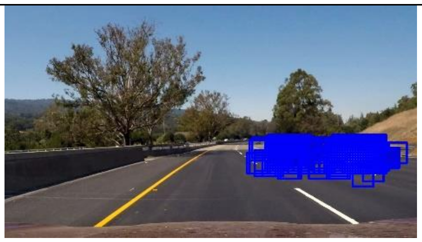
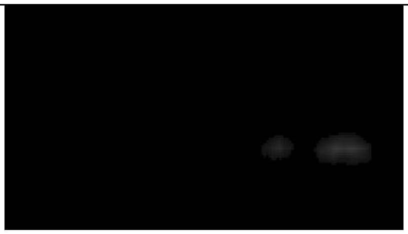
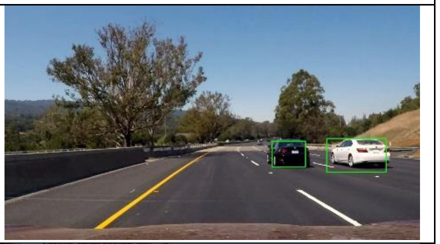
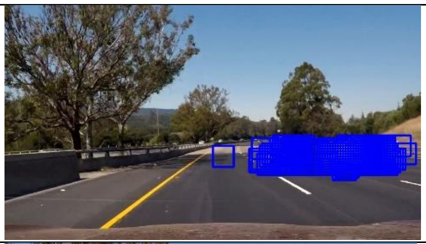
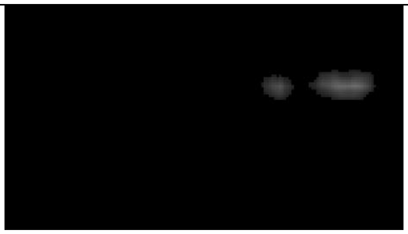
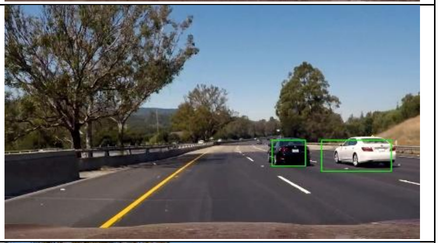
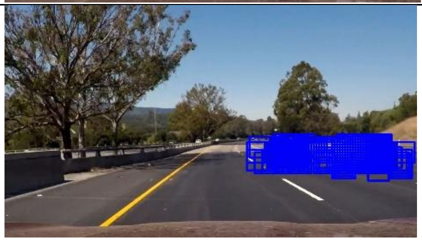
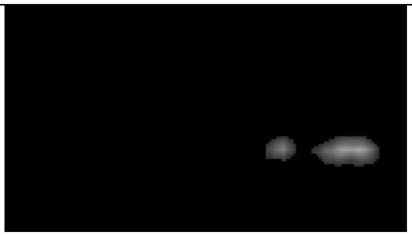
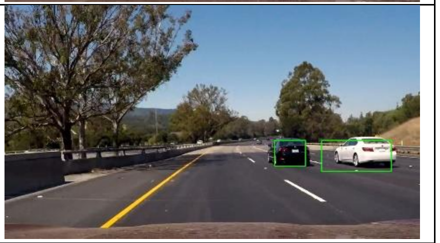
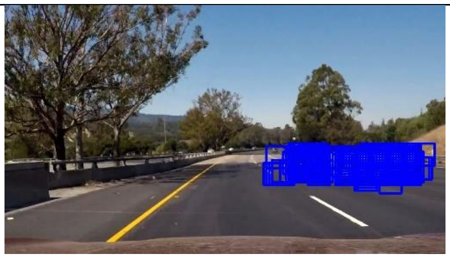
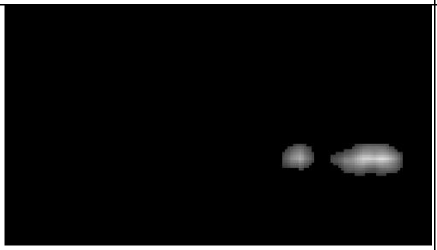
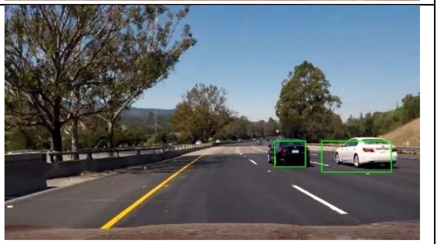
svc = LinearSVC(random_state = 100, verbose=True, max_iter=5000)

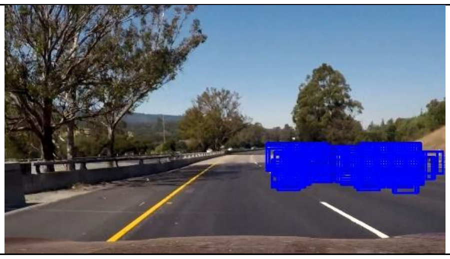
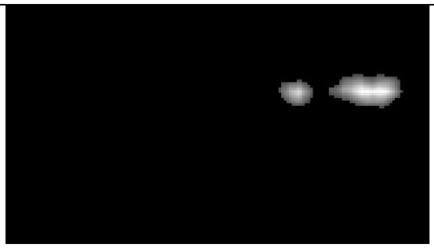
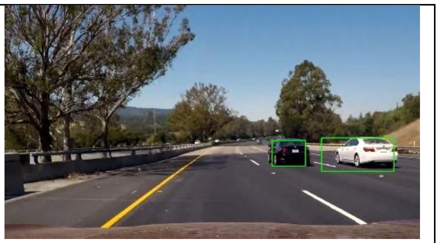
I used a random state (=100) and set the maximum iteration to 5000 to get good fit. Verbose is turned on to display the fit results on screen.

Image & Video pipeline:

The image & Video pipeline use sliding window search to detect vehicles and non-vehicle features. The function find_cars() extracts features from the image and finds vehicles in the image and return the bounding boxes for all the vehicles detected. To minimize false positive detection, based on the bounding boxes found, the routine process_frame() applies heatmap. To further improve the rejection of false positives, heatmaps over a set of 5 frames is averaged to get the final vehicle detection. The threshold used from heatmap to final bounding box drawing is 15 x number_of_frames. This value is arrived after experimentation. Five frames are used for satisfactory averaging.

The sliding window size is chosen to be (64, 64). This was chosen based on experimenting with different values of (64, 64), (128, 128) and (256, 256). The values of (64, 64) gave better results with minimal false positives.

Input images from the Video frame These images are from frame 1, 2, 3,4,5 going from top to bottom in this figure	**Vehicles detected by the moving windows with overlapping of 85%**	**Heatmap (observe the heatmap getting intense as the frames add up from top to bottom)**	**Detected Vehicle window.**
			
			
			
			

To optimize the vehicle detection, only the required portion of the input image is used to detect the vehicles in the image. Below are the final frame co-ordinates used for the vehicle detection:

ystart = 400

ystop = 680

xstart = 640

xstop = 1280

Note that these values may specific for this project video and for generalization, certain tuning may be required. In addition, to improve the vehicle detection, as mentioned in the figure above, the sliding window uses overlapped searching of 85%.

Results Conclusions and further work

The output for the project_video.mp4 (https://youtu.be/xPoWX29qlbU) is shown below which shows minimized jitter with minimal false positive detections.

The project build run output is shown below.

```
(carnd-term1) E:\courses\project5\try2\CarND-Vehicle-Detection>python vehicle_detection.py
Number of training samples:  15525
Number of validation samples:  3882
Number of cars in training samples:  8089
Feature vector length: 10252
[LibLinear]...................................................................................................................
..............................................................................................................................
..............................................................................................................................
..............................................................................................................................
.......
optimization finished, #iter = 5000

WARNING: reaching max number of iterations
Using -s 2 may be faster (also see FAQ)

Objective value = -3.291386
nSV = 1965
344.96 Seconds to train SVC...
Test Accuracy of SVC =  0.9786
My SVC predicts:  [ 1.  0.  1.  1.  1.  0.  0.  0.  1.  0.]
For these 10 labels:  [ 1.  0.  1.  1.  1.  0.  0.  0.  1.  0.]
```

```
0.07 Seconds to predict 10 labels with SVC
[MoviePy] >>>> Building video out_project_video.mp4
[MoviePy] Writing video out_project_video.mp4
100%|#############################################################################9| 1260/1261 [37:51<00:01,  1.76s/it]
[MoviePy] Done.
[MoviePy] >>>> Video ready: out_project_video.mp4
```

The further improvement is possible in terms of further generalization of the detection, better false positive removal can be made. In addition, tracking of vehicle from frame to frame can be made more robust by detecting and collecting and storing the detect vehicle information over the frames. Exploring and detecting the most optimal set of feature vectors can be made that would help run the algorithm real time.