Udacity Self Driving Car ND
Project 5: Vehicle detection and tracking

The project implements the following key steps:

1) Image feature extraction using Histogram of Gradients (HoG), Spatial Bins and Histogram Bins
2) Train a Linear SVM Classifier
3) Implement an image processing pipeline consisting of the following steps:
    a. A sliding-window technique to search for vehicles (using the trained classifier) on the input images or camera frames
    b. Use a heatmap of recurring detections frame by frame to reject outliers and follow detected vehicles

The code for is above steps is implemented *`vehicle-detection-pipeline.py`*. Each individual stage of the pipeline is isolated into a different function within the file along with an associated test function that allows to test the stage and the overall flow using test images.

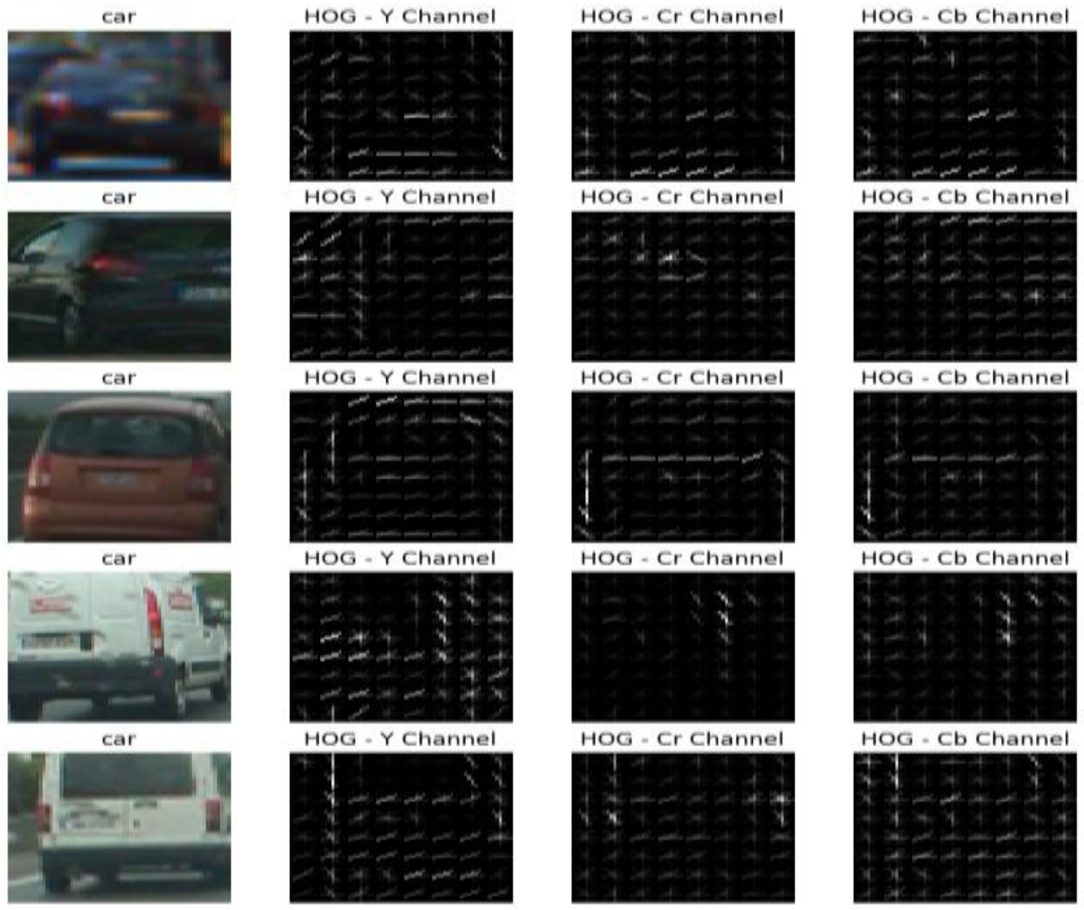
Feature Extraction

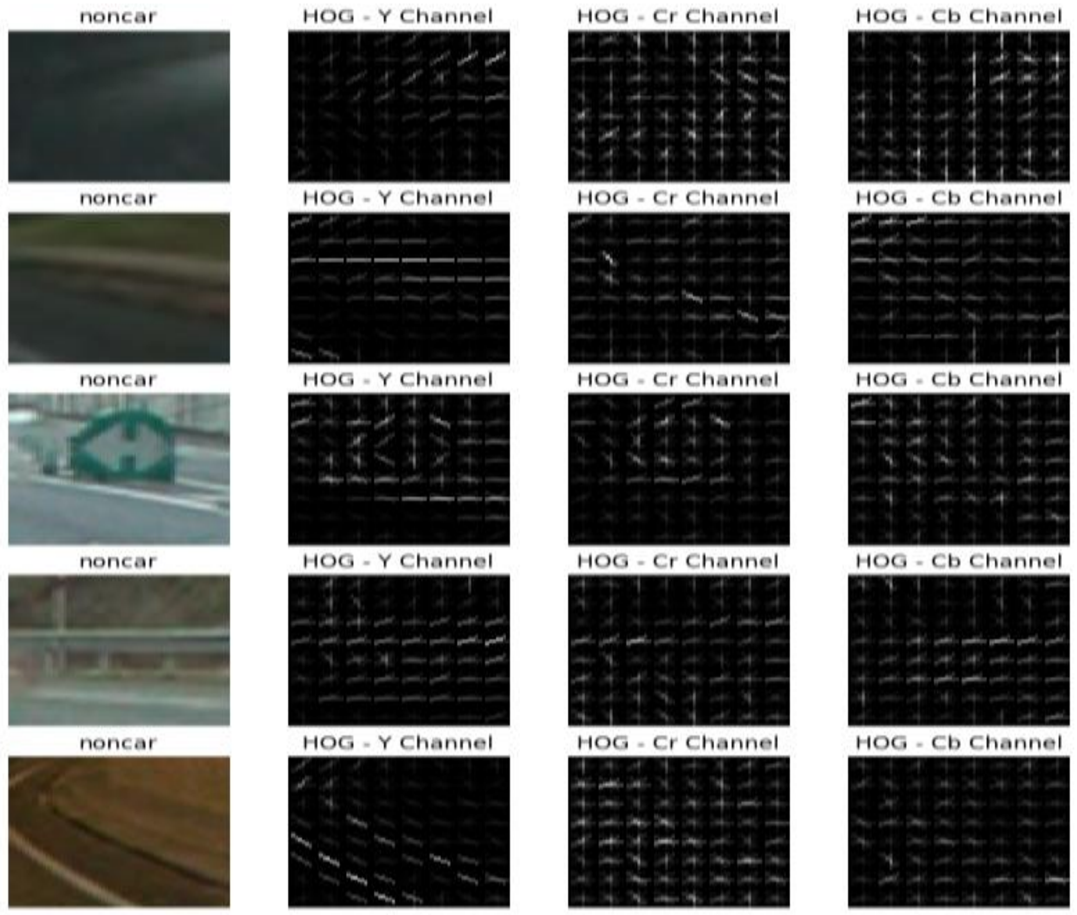
The code for this contained in the following functions:

- `get_hog_features (line# 15)`
- `get_spatial_features (line# 62)`
- `get_histogram_features (line# 67`

Randomly selected images from the training dataset (car and non-car directories) were used and the Histogram of Gradients (HoG) for each of the image individual channel was computed using the `skimage.hog()` function. The following hog function parameters: `orientations, pixels_per_cell, and cells_per_block`, where explored with different values to understand the combination of parameters that will give good result.

Shown below are example HoG feature for both Car and Non-Car images using the YCrCb color space and HOG parameters of `orientations=8, pixels_per_cell=(8, 8) and cells_per_block=(2, 2)`:


car
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
car
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
car
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
car
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
car
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel


noncar
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
noncar
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
noncar
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
noncar
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel
noncar
HOG - Y Channel
HOG - Cr Channel
HOG - Cb Channel

Training a Linear SVM Classifier

The classification accuracy of the Linear SVM classifier was used to decide on the best color space and best parameters.

Color Space	Orientation	Pixels Per Cell	Cells Per Block	Accuracy
RGB	11	16	2	97.65%
	8	8	2	96.44%
HLS	11	16	2	98.23%
	8	8	2	97.23%
YCrCb	11	16	2	98.65%
	8	**8**	**2**	**98.56%**

From the above table YCrCb color space had the highest accuracy, additionally the difference in accuracy between using 8 pixels per cell vs 16 pixels was negligible – hence the color space YCrCb was select with HOG parameters of orientations=8, pixels_per_cell=(8, 8) and cells_per_block=(2, 2).

To further improve the accuracy, Spatial Bins features and Histogram Bins features where also added. The below table show the combination of parameters for Spatial binning and histogram bins and the observed accuracy.

Spatial Size	Histogram Bins	Accuracy
32, 32	32	98.99%
64, 64	32	99.01%
32, 32	**64**	**99.16%**
64, 64	64	99.18%

From the above table a 32 x 32 Spatial size bin with the 64 histogram bins has good accuracy (comparable to 64 x 64 spatial size bins with 64 histogram bins at 1/3 the size of feature vector length 7968 vs 17184)

A Linear SVM classifier was trained using the Scikit-learn library function `LinearSVC()` on the YCrCb image formation using a combination of HoG features on the channel extracted using the parameters orientations=8, pixels_per_cell=(8, 8) and cells_per_block=(2, 2), spatial size of (32, 32) and histogram bins of 64. The training dataset was shuffled and split into train and test set (80 – 20 split) using `the test_train_split()` function from the Scikit-learn library.

Image Processing pipeline

The image processing pipeline to detect and track vehicle consist of 2 main blocks: the vehicle detection in a frame and removing multiple detection and false positives.

For the vehicle detection 2 approaches were implemented and tested: a sliding window technique and a feature sub-sampling. These are implemented and tested using the following functions:

- `detect_using_sliding_window``
- `detect_using_hog_subsampling`
- `detection_pipeline_exploration`

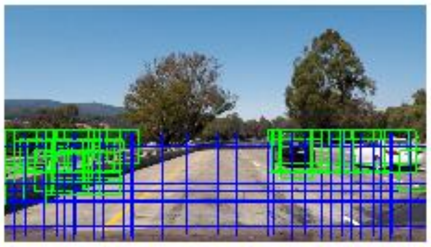
For the sliding window technique, we split the search region within the input image/frame into multiple overlapping windows of sizes 96x96, 112x112 and 128x12. Each of this window is classified using the trained classifier, and if there is match then the bounding rectangle for that window is saved.
For the feature sub-sampling technique, we compute the features of the search region only once and then we sub-sample the features using scaling factors of 1.0, 1.5 and 2.0, and at various overlapping blocks of size 64 x 64.
For the final pipeline the sub-sampling technique was selected due to a significantly reduced run-time (30min vs 120 mins) on the project video.

For the removal of multiple detection and false positive we used heatmap. The key idea here is to keep a track of overlapping regions where classification is positive by adding "heats" to these regions (essentially add 1 to the pixel location). We then track these predicted window pixels over the 16 previous frames, average the heat value of each pixel over that range and finally threshold the heatmap to eliminate pixels which are false positives. This is implemented in both `detect_using_sliding_window`, `detect_using_hog_subsampling` functions.

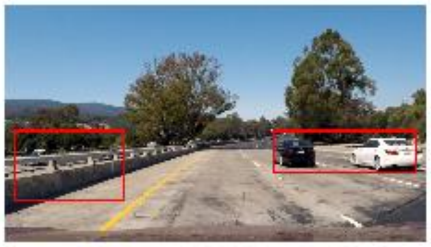
The below image shows the result of searching at various regions, potential classifier positive matches (green rectangle bounding boxes) and the final vehicle detection (red bounding box) after removing the false positives using heatmaps.

test_images\test1.jpg

search windows

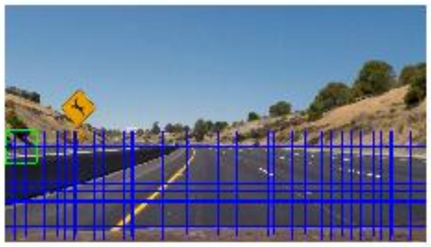


detected cars

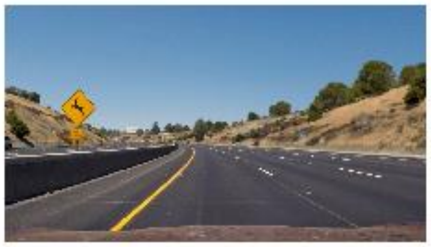


test_images\test2.jpg

search windows

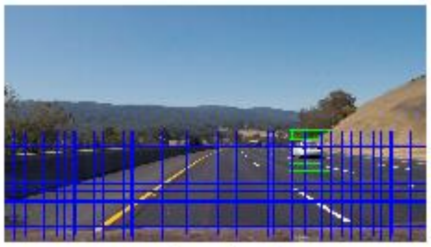


detected cars

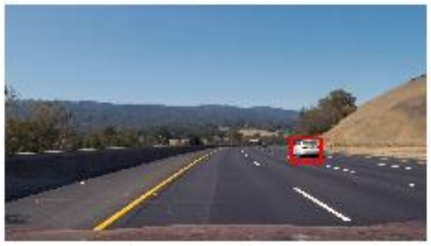


test_images\test3.jpg

search windows

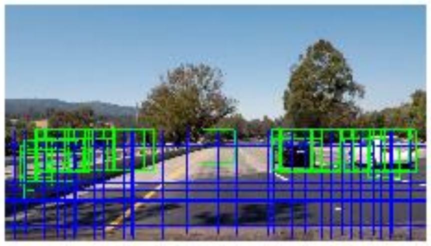


detected cars

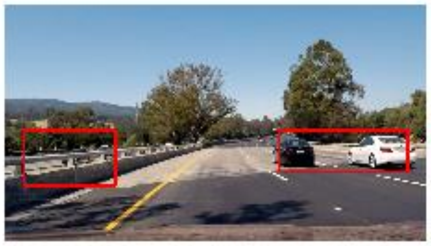


test_images\test4.jpg

search windows

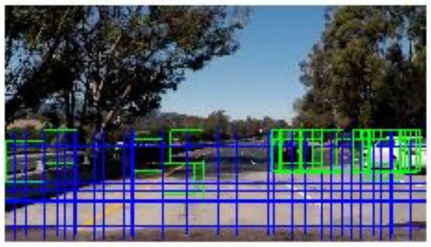


detected cars

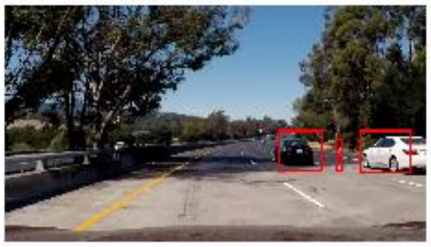


test_images\test5.jpg

search windows

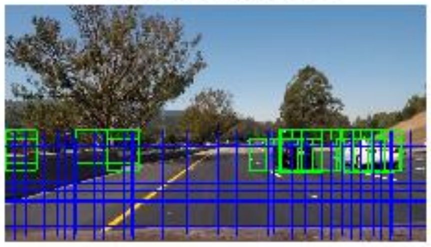


detected cars

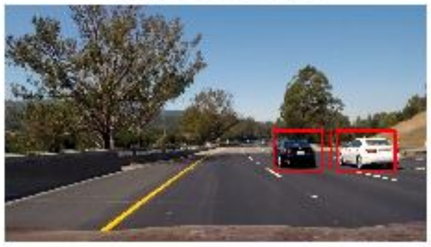


test_images\test6.jpg

search windows



detected cars



Discussion

The most challenging part of this project was to determine ways to find the various techniques (training features and search techniques) to eliminate false positives. This was predominantly caused by the mismatch in the results when using test images versus applying the same techniques on the project video.

I first started off using sliding window technique and contours counting and it seemed to work find on the test images, however it would fail with lots of false positives and missing detection on the final project video. In addition, using the sliding window technique had quite a large runtime (typically 30 – 45 minutes on the final video) resulting in a long delay until the result of any change could be determined (e.g. changing the sliding window size or changing the way heats/overlapping contours were computed and filtered out).

I finally implemented a sub-sampling routine to reduce the run time, heatmap averaging over 16 frames to improve the filter threshold, and added spatial bins and histogram bins to the training features (even though the improvement in accuracy compared to just HoG is <1 %). This resulted into increased false positives on the test images (primarily due to the fact that the average heat value for each pixel could not be used to threshold and filter false positives), however the solution video was much stable in terms of tracking a vehicle continuously and filtering false positives.

As final observation, the use of HoG and additional features to detect and track vehicles does not scale to a real-time implementation (0.3 – 0.5 frames/sec processing speed), additionally this technique still does not solve the problem of night/dark conditions. Some possible ways to optimize the speed to make it more real-time would be to carry forward the location of a potential vehicle (positive classification) from a prior frame and search only within that neighborhood. Some possible ways the problem of night/dark condition can be solved would be using depth cameras and/or LiDAR sensors and using feature extraction on the contour shapes provided by these images (as opposed to color images) as these would be agnostic to environmental conditions.