Project: Vehicle Detection

Juan Cheng 2017.10.09

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.

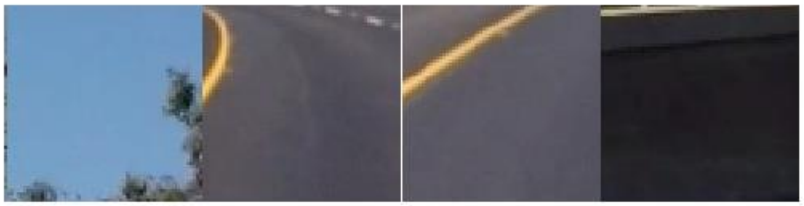
Histogram of Oriented Gradients

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

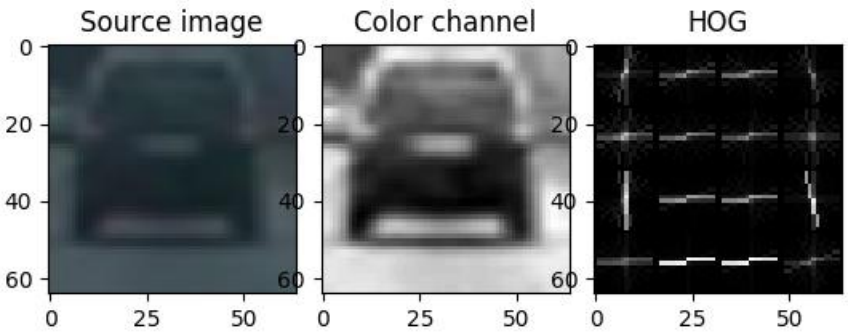
The code for this step is contained in file train_classifier.py, lines 38-51.

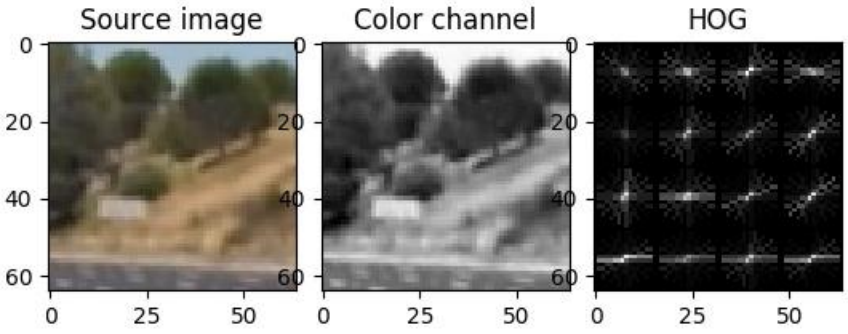
The code for the Histogram of Oriented Gradients (HOG) feature extract is based on the examples given in the project lessons. The same HOG configuration parameters are used: the color space to apply, the color channel(s) to use, the number of gradient orientations, the pixels per cell and the cells per block.

Features were extracted for each of the ‘vehicle’ and ‘non-vehicle’ images provided with the project. A random selection of these images follows.



Some example HOG outputs for car and non-car images, generated by hog_visualizer.py, follow. The stronger shape of the car compared to the more blurred shape of the non-car data is quickly visible.





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

The images were randomly split into a training set (85%) and test set (15%). A linear State Vector Machine (SVM) was trained using the selected features and the accuracy score of the test set used to indicate the performance of the HOG parameters.

Each of the six color spaces was trained at each step. The best accuracy score was found for the LUV, YUV and YCrCb color spaces for all tests. Note that the use of LUV and YUV required the disabling of the HOG square-root transform as this color space contains negative values.

Each of the four HOG channels were trained and the accuracy scores compared. The 'ALL' case was determined to be the best, as it had the highest accuracy, and enabled for the next tests.

The number of orientations, pixels per cell and cells per block were evaluated using three points: default value, lower value and higher value. A small advantage in more orientations was seen, but at the cost of longer run times. There was little observable difference when changing pixels per cell and cells per block.

The code for this step is contained in file hog_training.py together with the results obtained.

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)

The code for this step is contained in file train_classifier.py, lines 192-196. This is the final training code used for the later image testing and video pipeline steps.

A linear SVM, as recommended in the project lessons, was used. Different values of 'C' were tried for the classifier but no noticeable difference was seen in the output test images.

Enabling the spatial and color histogram feature extraction was also tried. This resulted in slower run times and many more false positives in the outputs. These other feature extractions were disabled for the final submission.

Sliding Window Search

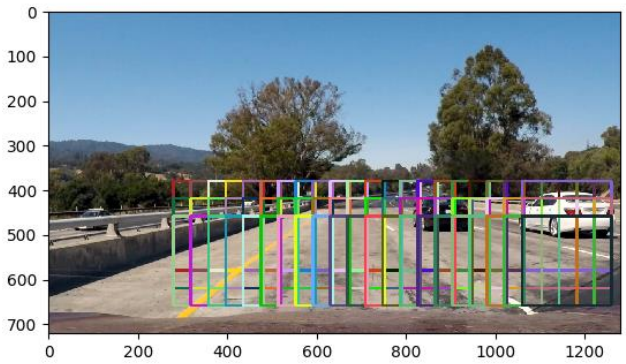
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

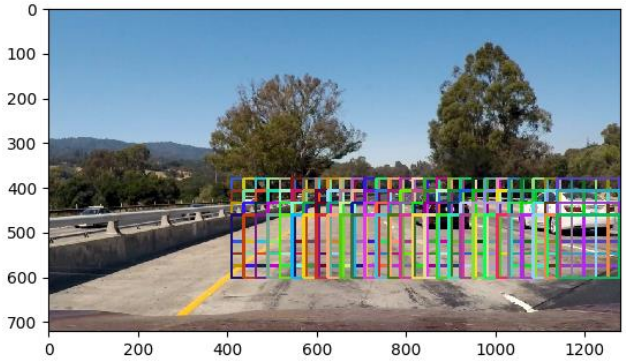
The code for this step is contained in file process_videos.py. The window configuration is lines 37-56, the window generation is lines 165-191.

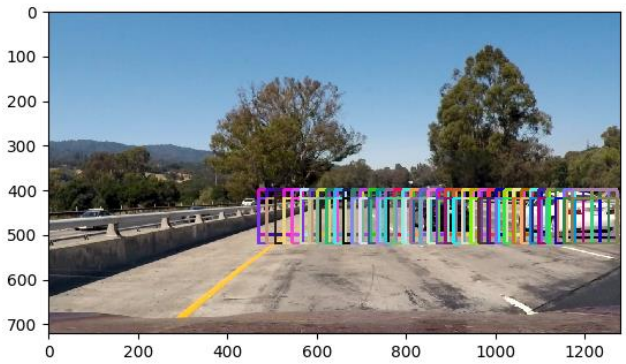
The window sizes were determined by drawing the windows on multiple test images, including those provided and some additional images captured from the video. It was determined that four layers of windows of different sizes were required to account for the vehicle distance. The windows are all centered on the useful road area.

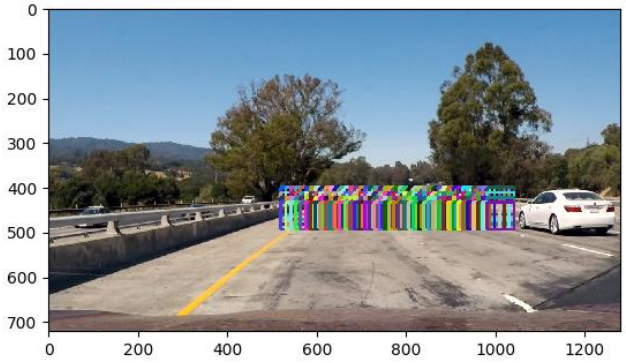
The window overlap was determined empirically based on performance results. The classifier seems to have a low false positive rate and high missed detection rate. A dense mesh of windows is required to reliably detect the vehicles.

The following images show the windows used, closest detection (biggest windows) to farthest detection (smallest windows)

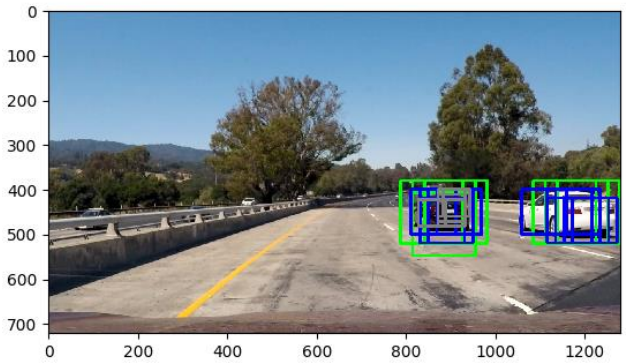

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

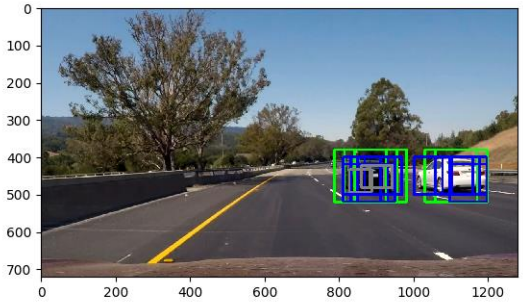

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

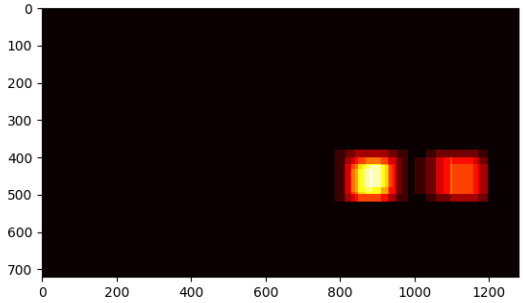
The following images show examples of the detection results for the test images.

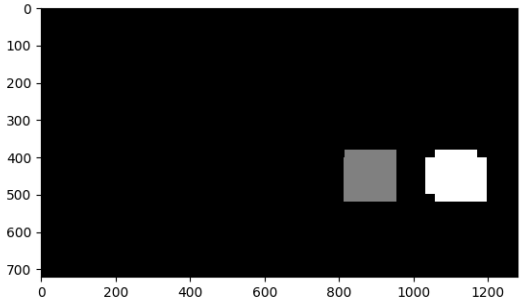


2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

The following images show the detection windows for some example frames from the test and project videos.

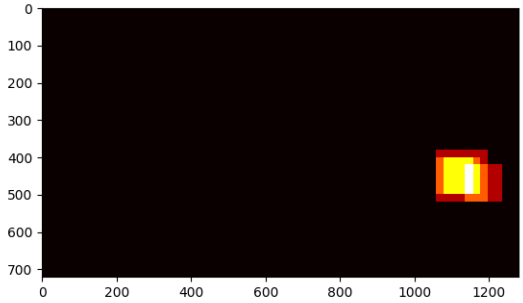

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

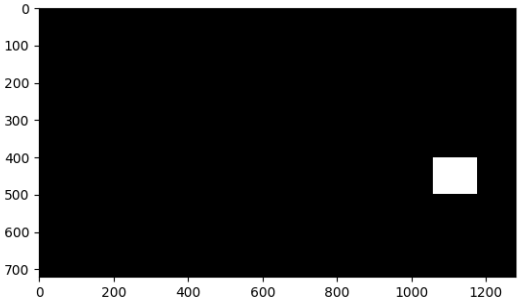

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

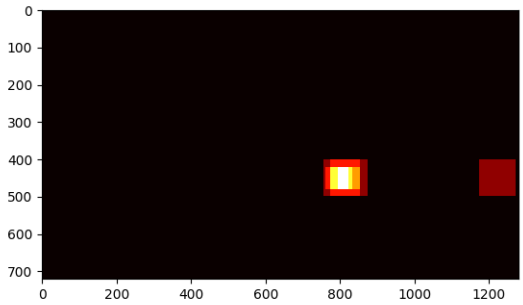
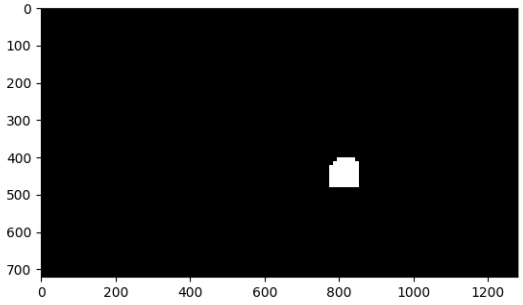

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


The classifier was optimized based on the number of false positives and missed detections in these images by:

- Trying the spatial and histogram feature extraction.
- Trying different values of ‘C’ in the SVM training.
- Modifying the search window overlap.

Video Implementation

1. Provide a link to your final video output.

I’ve included the submission video result, project_video_vehicle_tracking.mp4.

The detection boundary boxes are color-coded to show the confidence of the boundary calculated (green = high confidence, red = low confidence). The confidence is based on the hottest point in the heat map within the boundary box.

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.

A history of the previous 5 heat maps is maintained. These heat maps are accumulated to improve confidence in the boundary box and reduce false positives.

The label method, given in the project lessons, is used to combine overlapping boundary boxes.

Discussion

The video output shows that the vehicles are typically detected well. There's a short area, near where the road surface changes, where the detector starts to fail. This may be related to the search windows or, perhaps, the color change. Dumping the video frames in this area is required to determine the problem and potential fix, a change to the search windows or more data for the classifier.

Improving the classifier:

- Including left-right flipped images, to double the training set, may help improve the classifier. More independent data would be preferable though.
- More investigation into the use of the spatial and histogram features in the classifier is required too and these seemed to add little value in the current implementation.

Improving the search windows:

- The same search windows were used every time. A better strategy may be a fine search around known vehicle locations and coarser search for new vehicles elsewhere.
- No account was taken of the differing perspective for vehicles more to the side of the camera than directly ahead. Wider search windows may help at the lower left/right edges of the image.
- The lane information can be used to guide the search locations.

Extending the project:

- Include the lane lines detection from Project 4.
- Compute relative speeds and distances to other vehicles. This would be the main feedback of this project to self-driving car.