Vehicle Detection Project

Brent Wylie, May 21st, 2017

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.

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

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

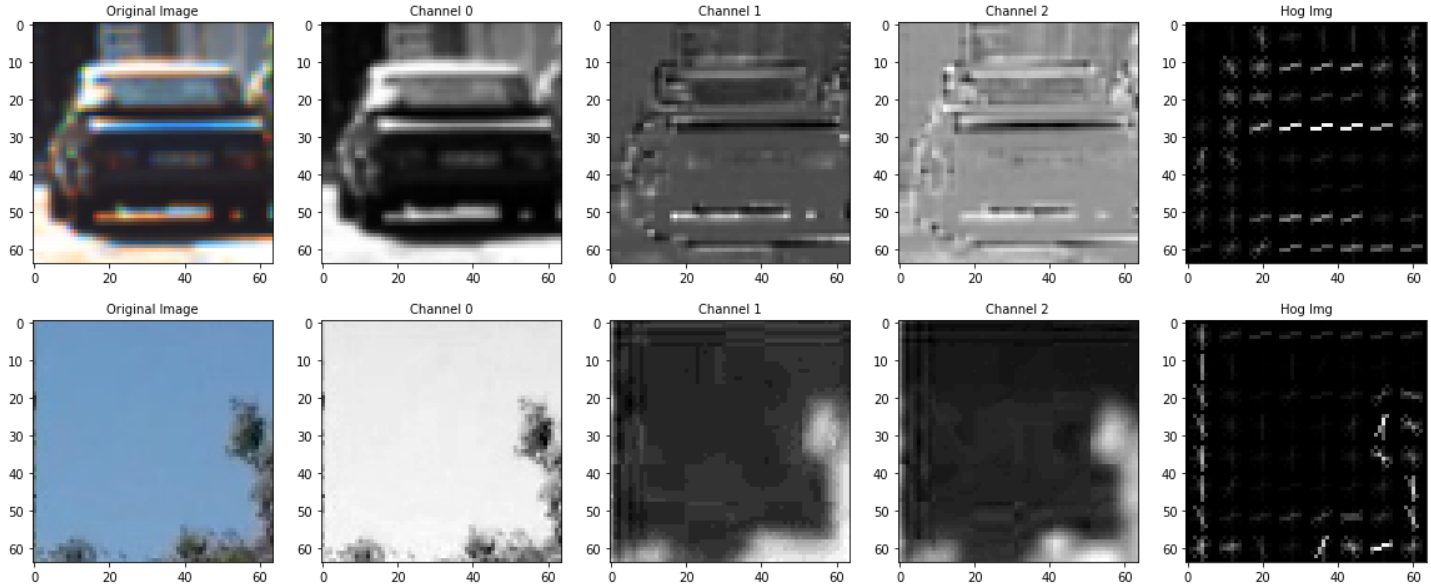
This document will consider all of the rubric points below.

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

In section 1.0 I defined several functions for extracting HOG and other features from input images. The main function for extracting hog features was the 'get_hog_features()', shown in Lesson 29. I encapsulated this function within the 'extract_features()' function along with functions for extracting spatial and color features. This was used in section 2.0 to train the classifier.

I chose to use LUV as the color space for feature extraction, as it seemed to show distinguishable differences between car and non-car features. I experimented with a few different parameters but decided to stick with LUV as the colorspace, orientation=9, pixel_per_cell of 8, and a cell per block of 2. I chose to use all HOG channels as results seemed better that way. Ultimately many different parameters worked OK, so I targeted having the highest training accuracy without significant false positive results. Below is an example of the HOG and Color channel outputs for both a 'Car' example and a 'not car' example, respectively.



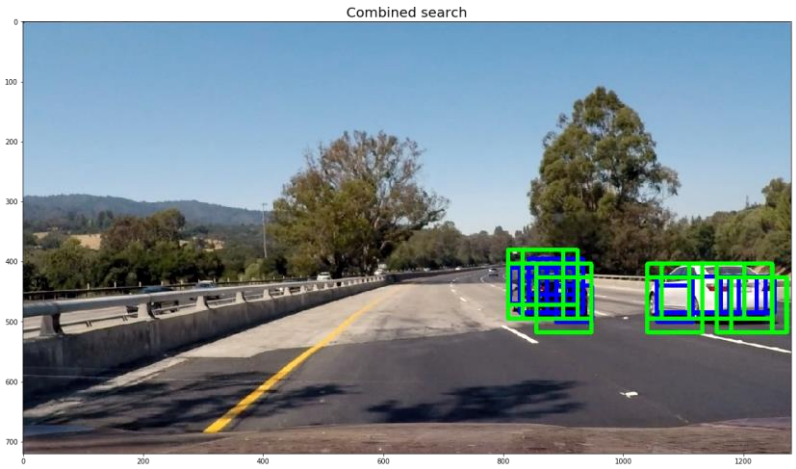
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

In section 2.0, I read the combined GTI and KITTI data sets into sets of 'car' and 'notcar'. Then, I used the 'extract_features()' to extract features and example HOG images from both of these sets. I also extracted LUV channel features and spatial features (of 32x32) to assist in the training process. After extracting all of these features, I scaled the feature vectors, generated the corresponding labels vectors, and fit a Linear SVM to the data. I used GridSearchCV() to explore values for C and in the set of [0.05,0.1,0.5,1.0,1.5,2.0,2.5] and chose C=0.05. I was able to obtain a test accuracy of 99.49%, with 20% of the dataset used as test data. I used test_train_split() to shuffle and split the data.

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?

I implement the sliding window search within the find_cars() function defined in section 3.0. The sliding window search is implemented as a hog-subsampling algorithm. I ended up using the sliding window search twice, once at a smaller scale of 0.85 to catch far features or partially obstructed features (such as out of field of view), and once at a larger scale of 1.45 to catch medium/closer distance cars and whole vehicles. The image below shows one example of this detecting multiple cars. The green boxes are the medium/close range detections and the blue boxes are the far range/small scale detections. Both are overlaid on the same image to help show coverage area.



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?

I tested the find_cars() function in section 3.1 in order to find the optimal search regions and window scales. I limited the region of searching to the road surface in both X and Y directions to reduce the iteration count and improve processing time. I focused on getting as many overlapping detections as possible without causing false positives in the regions of interest, even if it meant a slightly more expensive search algorithm as this led to a more stable final result. Below is another example of the find_cars() test result.

The multiple detections cause the 'heat' to rise in that area which allowed me to raise the heat thresholding to reject occasional false positives while keeping the vehicles around.

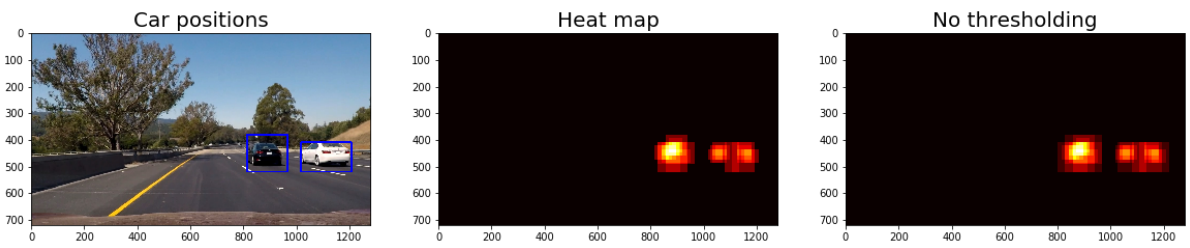
Pipeline (video)

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.)

The video is contained in the same zip file that this writeup was in. It is named 'project_output_submission.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.

In order to reduce false positives and combine overlapping bounding boxes, I defined several functions in section 1.0, provided by Udacity lessons. Then, in section 3.2 I tested these heatmap functions to demonstrate successful filtering of false positives and combination of bounding boxes. Finally, I defined a heatmap class in section 3.3 and tested it in section 3.4. This class essentially keeps track of the heatmaps of the last 10 frames and computes and average heatmap as the program processes frames. This resulted in much more stable bounding boxes generated by the classifier. The test programs in section 3.2 and 3.4 allowed me to choose parameters for thresholding the heatmaps, and monitoring the running average's performance. Initially, I chose to average two frames together with a decay rate for the old frame, but opted to change to a 10-frame average with no decay instead. The Heatmap class returns the overall, unbiased average for use in labeling bounding boxes. Below is an example output of the testing done for verifying heatmap behavior in section 3.2. More pictures are available as part of the ipynb file submitted.



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?

The biggest problem I faced in this project is making efficient and important parameter choices, due to the processing time of the algorithms. Ultimately, I could have experimented more with various feature extractions, with different SVM parameters, and with different tracking algorithms if the most expensive parts of the algorithm were accelerated. I tried to optimize the processing as much as possible to improve turnaround, and generated as many static tests as possible to reduce the need for video testing. Still, video testing was necessary so I made subsets of the project video as sanity checks for better testing before committing to a full video process job. The biggest weaknesses of the pipeline are partially obstructed vehicles. The training set could be improved to allow for smaller windows to detect the features of an obstructed vehicle view. In addition, this pipeline could use smaller windows find these features. Ultimately this application is better suited for a segmentation neural network, but this technique is suitable for less-powerful devices with highly parallel SIMD acceleration for the SVM, HOG and feature extraction operations.

In the future, I would like to implement my algorithm on such an embedded platform, and I would like to make use of multiprocessing to optimize and parallelize parts of the algorithm in order to reach real-time performance.