

Vehicle Detection

In this project, a vehicle detection algorithm is developed, which utilizes Support Vector Classifiers. The goal is to detect the vehicles on a video stream and show them in a bounding box. In the following section, all of the criterias will be addressed.

The Project Rubric

Readme

Provide a Writeup / README that includes all the rubric points and how you addressed each one.


This document is provided for this purpose

Histogram of Oriented Gradients (HOG)

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.

The HOG feature extraction part is made under the function `extract_features` (can be seen on section 6 on the jupyter notebook), which calls another function which is called as `get_hog_features`. In the end, scikit-learn library is used for calculating HOGs. For extraction, first the image is converted to YUV colorspace. Several colorspace are tried and for me the best test accuracy is obtained by YUV colorspace. All of the color channels are used. The parameters are tuned according to obtain the best test accuracy and by staying as low as possible to decrease the calculation time. `orient = 9 pix_per_cell = 8 cell_per_block = 2`

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

Additional to the HOG features, colorspace and spatial of the images is also used. All of these values are concatenated together and they are normalized via `StandardScaler` function (provided by scikit) to avoid dominant features. The result of normalization can be seen below:  For the training, SVC is used. The images

in time series are seperated manually to not decrease the accuracy of the test accuracy result. Then the training images are shuffled. The result of the training and the test is below. Training Accuracy of SVC = 1.0 Test Accuracy of SVC = 0.9863 This part of the code can be seen under the Train the SVM part of the jupyter notebook (section 9).

Sliding Window Search



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?

For the efficiency reasons, first only a section of the frame is considered for sliding windows, since we know that vehicle is going to be on the road. For detecting further vehicles, narrow field of view with a smaller scale is used

and for the closer vehicles larger field of view with a larger scale is used. For each scale, first the whole HOG is computed to avoid redundant calculations of the same part of the frame. Because of this approach, the overlap is defined by cell per block parameter, which is chosen as 2. Higher numbers resulted as missing vehicles and lower values created more bounding boxes for the same vehicle. The code can be seen on the section 49, under the `find_cars` function, which also includes feature extraction for the vehicles.

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

The result of the `find car` function can be seen below:  Since the `find_cars` function returns redundant

bounding boxes, a heatmap approach is used. First an empty matrix is created with the same size of the image. Then for each bounding box area, +1 is added. The total image is thresholded to remove some of the false positives. The thresholded heatmap is then labeled with the label function from scikit. Some of the examples of an heatmap is below:   To avoid false positives, hard negative mining is also done for bright parts of the

video such as trees. There are about 20 additional non vehicle image for hard negative mining. To avoid false negatives, several scales are used on the same image, and all of their results are added up before the heatmap process. This created more redundant bounding boxes, which helps the vehicle images to surpass the threshold.

Video Implementation

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 result of the project is [here](#). Another result with no false negatives, but includes one false positive is also added [here](#).

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.

Additional to the vehicle detections on images, for the video two heatmap thresholds are used. For each frame, after the heatmap is thresholded with a low number, it is kept in memory to find consistent detections. Then, another heatmap threshold is used on the sum of the past heatmaps to remove false positives. The first threshold helps to remove the clutter from the memory, and the second threshold helps to remove false positives, even if they are consistent but not stable. The memory also stabilizes the bounding boxes and prevent wobbling boxes. This is implemented under the function `process_image`. The thresholds are tuned via trial and error. Some failing parts are also screenshotted and tested alone as an image.

Discussion

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 was avoiding false positives while trying to keep the vehicles. I had to do try various heatmap threshold parameters and that took most of my time for this project. The pipeline might fail on urban parts, because the training set does not have images from the side of a vehicle. It also might fail with ramps since I am only considering a region for a straight road. I would suggest to use deep learning methods to avoid false positives. I would also try to obtain more data for non vehicle conditions, since these images consist of various

different images and it is hard to obtain a good generalized data. Another small improvement would be training the data with RGB images instead of BGR. That would increase the speed of the algorithm.