writeup.md

Vehicle Detection Project 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.

Rubric Points

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

Writeup / README

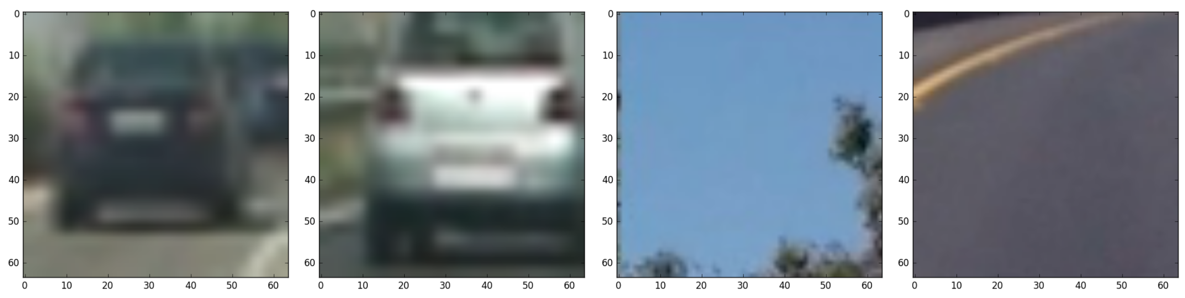
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Histogram of Oriented Gradients (HOG)

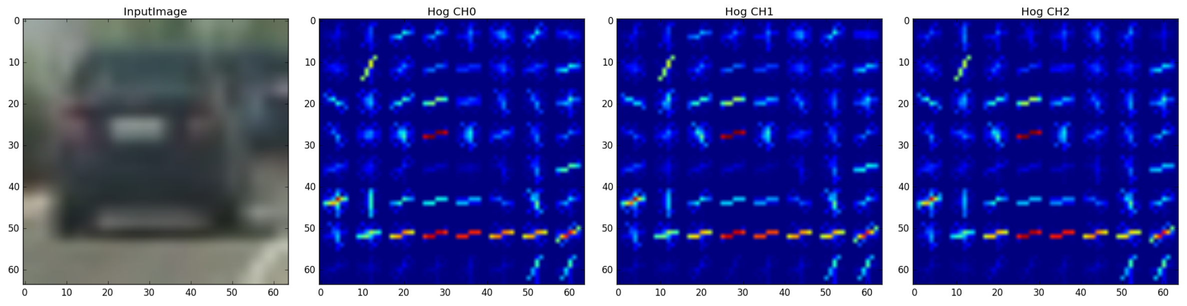
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

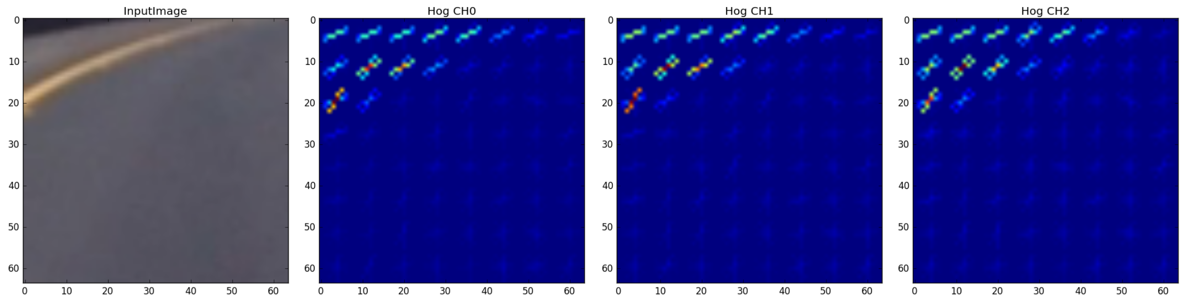
I started by getting the image directory of all images in the vehicle-folder (GTI_Far, GTI_Left GTI_MiddleClose,GTI_Right,KITTY_Extracted) and the non-vehicle-folder (Extras,GTI). Vehicle and non-vehicle samples:



Features are extracted in #87-88 for cars and non-cars in Demo_DetectVehicle.py using the extract_features-function from detectVehicle.py. HOG features are created for all channels in a given color space. I have explored different color spaces (this is elaborated later) and different `skimage.hog()` parameters (`orientations` , `pixels_per_cell` , and `cells_per_block`). I grabbed an image from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=8` , `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)` :





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

The data set is seperated into a train and a test set. Parameters were evaluated by training a classifier on the train data and measuring the accuracy on the test data. Some results are reported in the next question

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).

I trained the classifier using both SVM and Adaboost with decision trees, see line 140-143. Selection of classifier is selected using the classifierType-variable in line 121-122. SVM performed better in accuracy and speed. I tried the following settings using different colorspaces, channels and features. IncludeFeatures specifies if respectively hog-features, histogram and spatial features are used in a setting.

Classfier	Colorspace	Channels	IncludeFeature	Accuracy
SVC	RGB	0	1 0 0	0.9431
SVC	RGB	ALL	1 0 0	0.969
SVC	HSV	ALL	1 0 0	0.9783
SVC	HLS	ALL	1 0 0	0.9755
SVC	YCrCb	ALL	1 0 0	0.9817
SVC	YCrCb	ALL	1 1 0	0.9828
SVC	YCrCb	ALL	1 0 1	0.9879
SVC	YCrCb	ALL	1 1 1	0.991
Adaboost	YCrCb	ALL	1 1 1	0.9718

The highest accuracy is achieved using a SVC classifier, all channels of the YCrCb colorspace, color histogram features and spatial features.

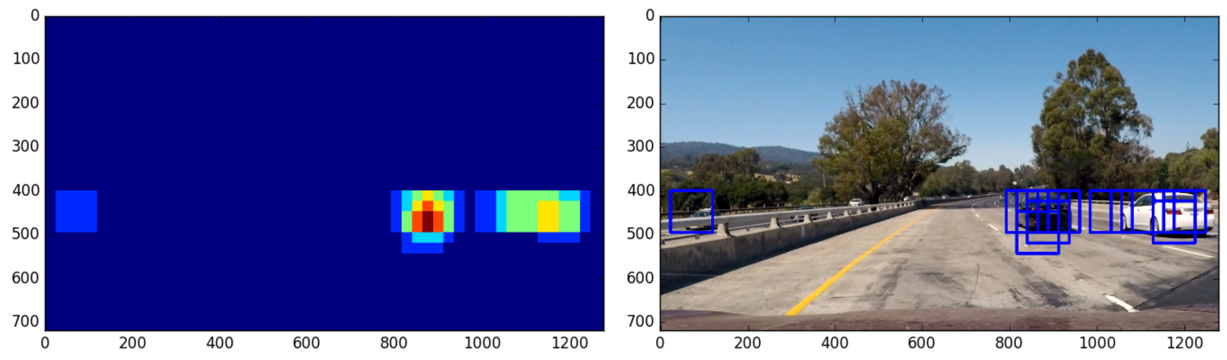
Sliding Window Search

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 sliding window is implemented in the find_cars-function in detectVehicles.py. The code is able to search at multiple scales using the scale-variable. The overlap is fixed by the pixel_per_cell in HOG features. The minimum detection size is set to 64x64 (using a scale factor of 1.0).

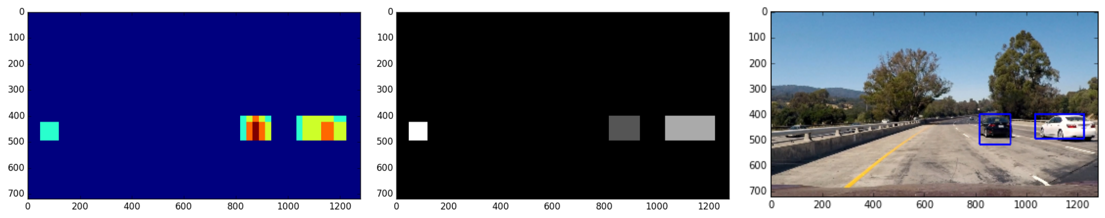
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

This function detects many overlapping car regions as demonstrated below with bounding boxes and a heat map.



From positive detections a heatmap is generated and thresholded to identify vehicles. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Threshold heatmap (below left), individual blobs in the image (below center) and new bounding boxes from connected components (below right)



Video Implementation

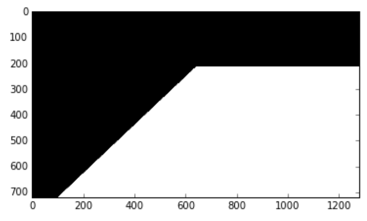
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

The video pipeline combines scales of 1.0 and 1.5 by adding their heatmaps together. A naive tracking procedure is implemented by combining current heatmap with previous heatmaps - each heatmap is remembered by a factor of 0.8. A higher threshold of 1.3 is required as the final heatmaps is a combination of both heatmaps at two scales and previous heatmaps.

Here's a link to my video result

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.

I have reduced the search space, added a threshold and combined heatmaps of previous frames to reduce false positives (This have been described in "Video Implementation" question 1). Finally, I multiply by a mask on the heatmap to filter out false positives. The mask is presented below.



To combine overlapping regions, I have created a heatmap, reduced the heatmap to individual object blobs and created boundings boxes. (This have been described in the "Sliding Window Search" question 2).

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?

To improve robustness of the algorithm, previous heatmaps are remembered by a factor. By increasing the threshold, a single false-positive detection will not result in a detection, thus a detection requires detection of an vehicle in subsequent frames or multiple scales. Furthermore, I have investigated different classifiers and features to improve the classifier. To detect a car at multiple scales (close and far distances), I search for cars at multiple scales. Searching for too small vehicles will however create to many false-positives. Some tweaking of scales and thresholds was required to get a reasonable output.

Where will your pipeline likely fail? What could you do to make it more robust?

The algorithm is not running in real-time and it would be interesting to reduce algorithm complexity. 1) This can be done by selecting a reasonable search space for each scale 2) use a region proposal algorithm 2) Use a depth sensor to provide region proposals. The naive tracking used in my project is not able to handle rapid changes in position of the car or other cars (this is especially critical in hazard situation). More advanced tracking methods should be used to include car movement.

Futhermore, it would be interesting to improve the object/vehicle detector by using fast deep learning obstacle detection algorithms (YOLO, DenseBox, DetectNet, Faster R-CNN) or perhaps train a semantic segmentation network to detect cars and other elements of interest (road, lanes, sidewalks, signs, pedestrians ...)