Vehicle Detection and Tracking

Overview

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.

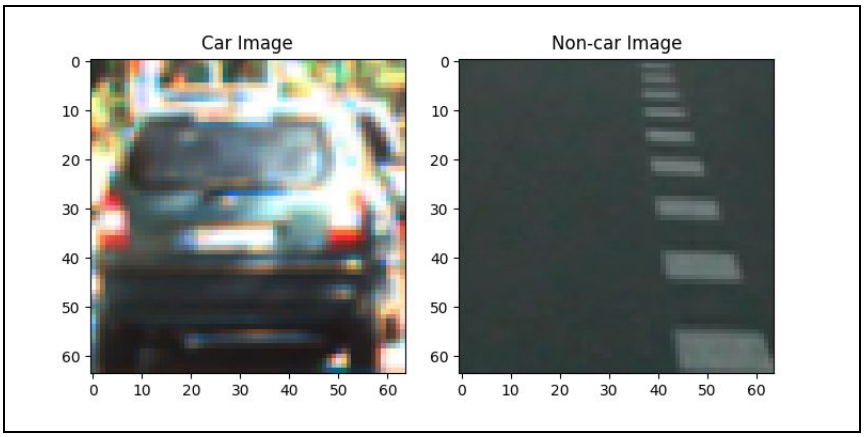
Histogram of Oriented Gradients (HOG)

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.

For trying out feature extraction and training a classifier, I used the dataset provided by the course, which is a labeled dataset consisted of images taken from GTI vehicle image database, the KITTI vision benchmark suite, and examples extracted from the project video itself.

I started by reading in all images contained in the `vehicles` and `non-vehicles` folder using `glob` module. I plotted random images out of the two lists, just for checking out each sub-dataset.

This is an indicative example of what I got:

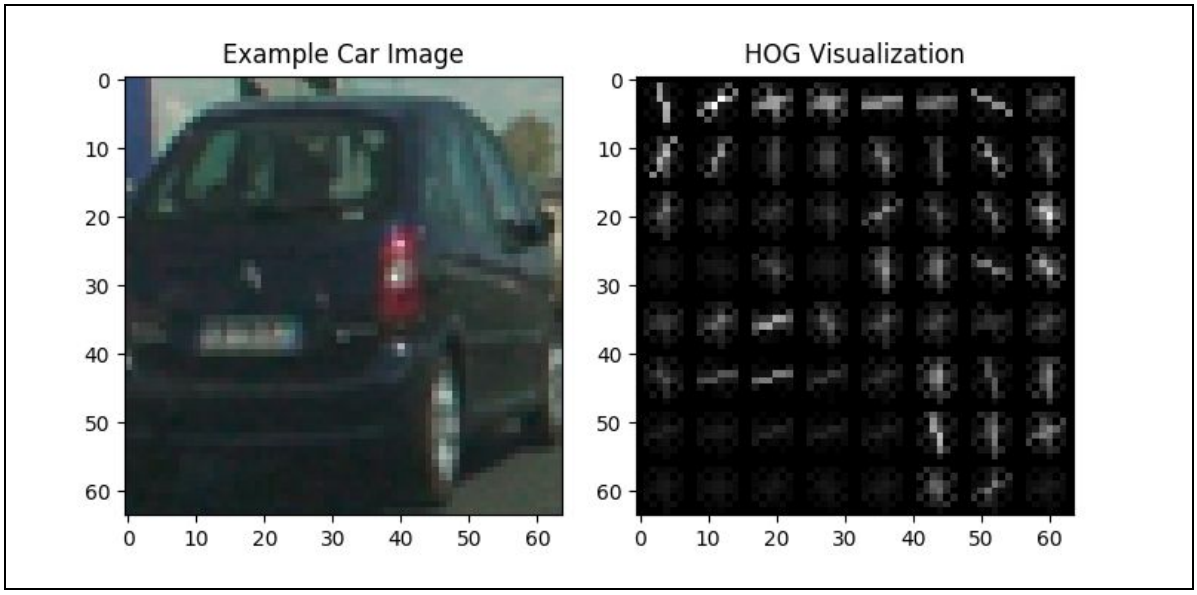


Then I created the `features_functions.py` script which contains functions that help me define the different features I want to extract from my images. Specifically, the `color_hist()` function takes an image and computes the color histogram of features given a particular number of bins (32 in my case) and pixels intensity range, and returns the concatenated color feature vector. In addition, I added the `bin_spatial()` function, so as to scale down the resolution of the images to 32x32 (which is a resolution that preserves the features) and feed my classifier a feature vector easy to train. At last, in order to extract HOG features from the training images, I used the `get_hog_features()` function, in which I pass specifications for orientations, pixels_per_cell, and cells_per_block, as well as flags for whether or not I want the feature vector unrolled and/or a visualization image. All these functions are called by the `extract_features()` function located in the `hog.py` file in tle lines 58-106. I set the parameters the following way and tested between different color spaces, to see the time it takes for HOG features to be extracted.

orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"

Colorspace	RGB	HSV	LUV	HLS	YUV	YCrCb
Features Extracting Time	62.0	65.73	65.72	65.41	62.64	62.63

This is an indicative result of HOG visualization of a car image.



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 features (HOG features, binned color features and color histogram features) extracted from the training data have been used to train an SVM classifier, and you can see this code in the lines 120-154 of the `hog.py` file. Firstly, the data were converted in a numpy array and then the `StandardScaler()` function was applied, in order to normalize the data and by this way preventing from individual features or sets of features dominating the response of my classifier.

`LinearSVC()` (Linear Support Vector Classification) was applied, and training time and accuracy score were printed. The dataset is split into 80% training and 20% testing subsets.

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?

In `sliding_window.py` you can find the code defining two functions that are going to help me do the sliding windows search. Specifically the `slide_window()` function produces a list of bounding boxes for the search windows given an image, start and stop positions in both x and y, window size and overlap. This list is then passed to the `draw_boxes()` function which draws the boxes in the image.

In the lines 294-318 of the `hog.py` file you can find my sliding window search, where you can see I searched at multiple scales such as `xy_window=(64, 64)`, `xy_window=(96, 96)` and `xy_window=(128, 128)` with which I see the cars to appear in my test images.

I also restricted my scanning area to where I expect to see the cars, and that is to the lower half of my images. So, I set `y_start_stop = [395, 620]` to the `slide_window()` function. In addition, in order to prevent from having false positives to the trees I restricted my search area in the x direction as well.

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

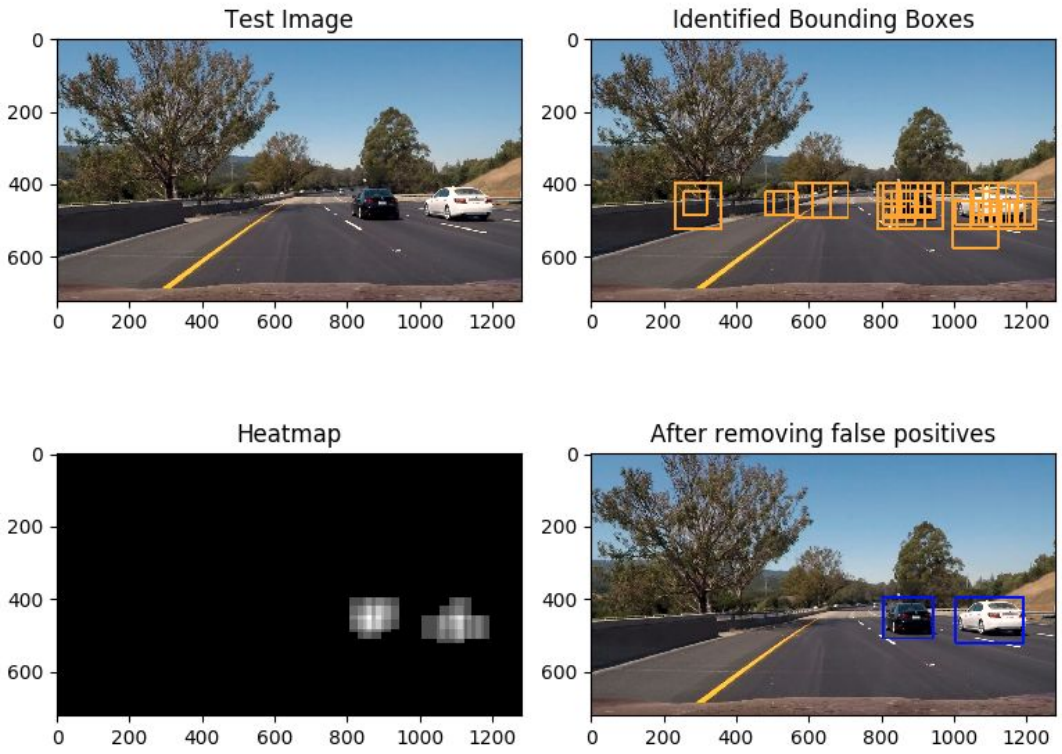
In order to achieve a better result, I retuned my parameters this way:

```
# Define HOG parameters
orient = 13, # I increased the number of gradient orientations to 13.
pix_per_cell = 8
cell_per_block = 2
colorspace = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
# Define sliding window search parameters
```

`spatial_size = (32, 32)` # Spatial binning dimensions, I was testing between (32,32) and `(16,16)` but the performance with `(16,16)` was not so good. It seemed that a further scaled down made me lose features.

`hist_bins = 32` # Number of histogram bins

The dataset is split now into 90% training and 10% testing subsets.
You can find more examples of output images in the `output_images` folder.



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.)

This is a link to my video result; https://youtu.be/gPmQbzrl3F8

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 optimized my pipeline by building a heatmap that combines overlapping detections and removes false positives. In order to do this, I added the functions

located in the lines 261-290: the add_heat() function adds heat to a map for the list of bounding boxes (`hot_windows`) I derived before, the `apply_threshold()` function applies a threshold to the heatmap and thus I can reject false positives, and the `draw_labeled_bboxes()` function draws boxes around the labeled regions of the heatmap. These are some indicative examples of the implementation:

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?

I encountered some trouble having the SVM classify correctly the feeding data. It performed a lot of misclassifications, and I had a lot of bounding boxes drawn on areas that didn't contain a car. Creating the heatmap and applying a threshold made a significant improvement. Investigating different classifiers, like a Decision Tree or GaussianNB would worth the try.

I also noticed on my video that when the two cars approach the two bounding boxes turn into one, when in fact the cars are two. Perhaps, changing the overlapping boxes could improve the result.