Vehicle Detection Project

The goals / steps of this project are the following:

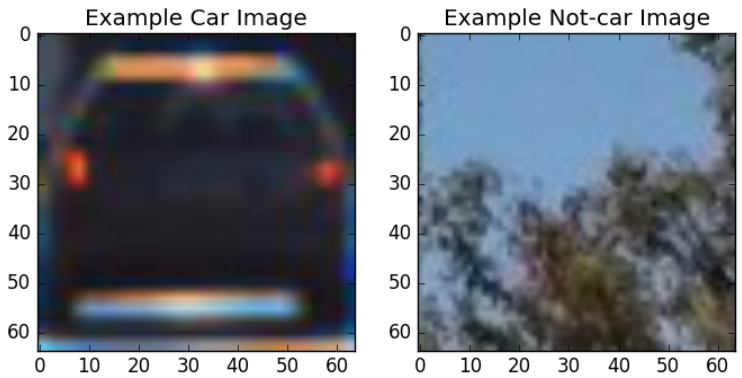
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: 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.

Histogram of Oriented Gradients (HOG)

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

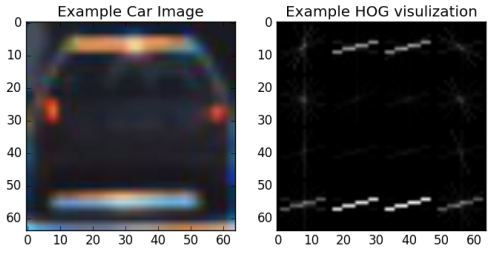
The code for this step is contained (in lines 14 through 31 of the file called help_func.py).

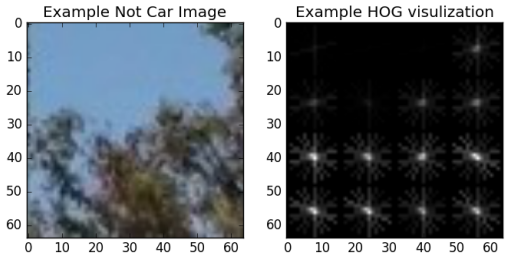
I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:



I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed random images 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=6, pixels_per_cell=(8, 8)and cells_per_block=(3, 3):





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

I tried various combinations of parameters and finally settled down to use orientations = 6, pixel_per_cell = (8, 8), cells_per_block(3, 3) and color space (YCrCb).

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 contained in lines(78 to 96 in detect_cars.py)

I trained a search grid over linear SVM with ‘C’ parameter in range = [10^-3, 1] using features extract from the dataset images with YCrCb color spacing, orientations = 6, pixel_per_cell = (8, 8), cells_per_block(3, 3), spatial size of (16, 16) and histogram bins of (24) with features vector of length (5832), the training process took 2.59 seconds with test accuracy of 99.5%.

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?

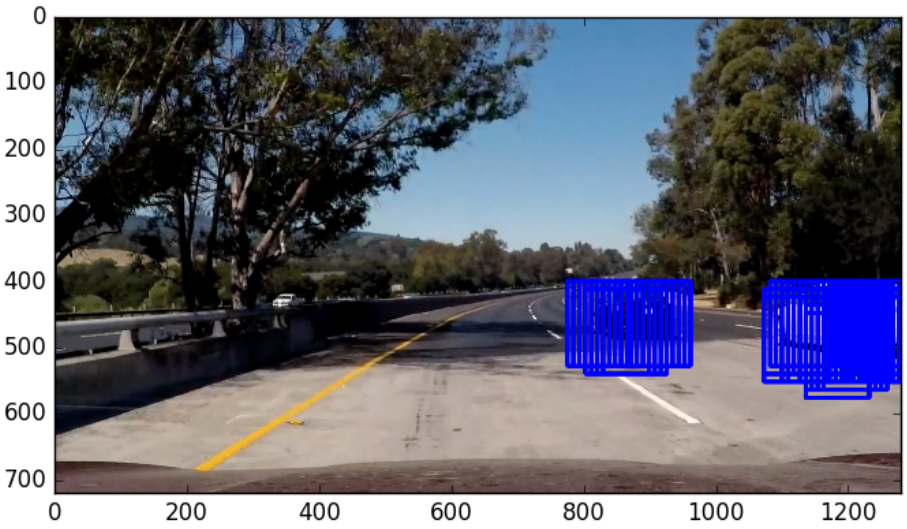
this step contained in lines (108 to 195 in help_func.py)

I started by inspecting the minimum and maximum X and Y values manually by observing the cars in the sample video. I then experimented with several approaches, starting from 0.5 overlap, and approaching 0.9 overlap. Although using windows of multiple sizes with varying offset was helpful initially, using a single window size with high overlap produced the best results in terms of maximizing true positives and minimizing false positives. The size was varied to best fit the expected car size in the video. The downside to having one window size is the possibility of missing cars that are of much larger or smaller size.

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?

Using all the channels in YCrCb color space, and using a smaller bounds for X and Y in sliding windows helped to optimize performance. I also chose a larger size of HOG pixels per cell, and decided to only use one size of sliding window.

From the example results, the pipeline seems to work quite well.



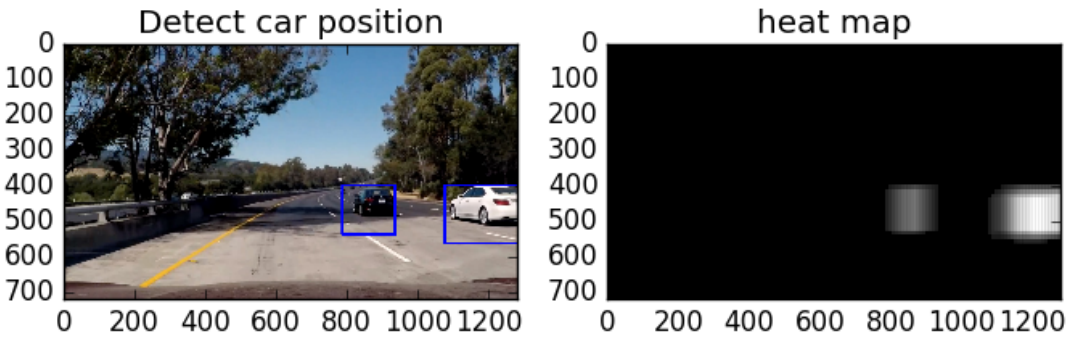
Video Implementation

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.) 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 recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I found that a threshold value of 1 worked best. 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.

Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:



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?

issues with manual tuning the parameters, The classifier also has issues with shadows and other objects with high gradient values

To make the video pipeline more robust, we can look at end-to-end and deep learning solutions.