Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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!

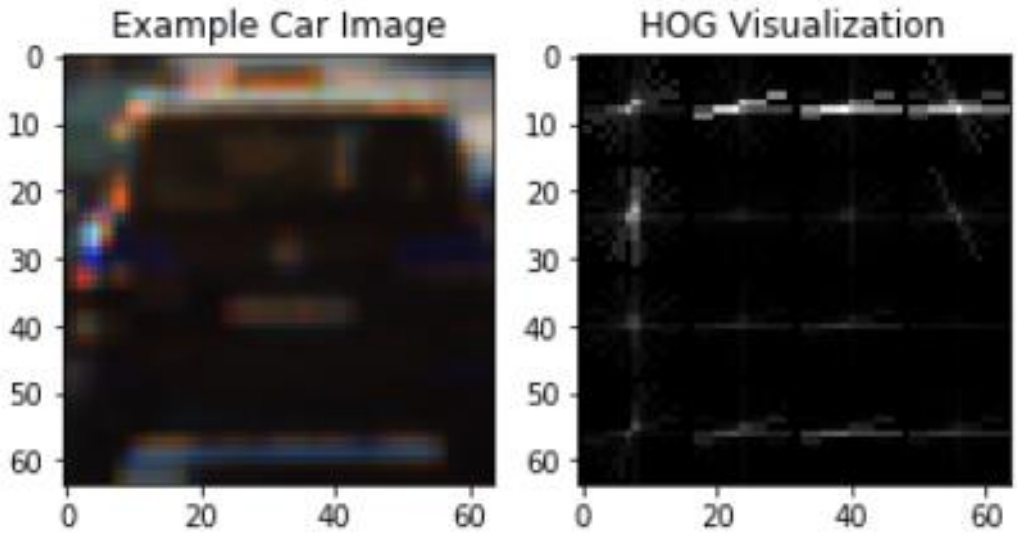
Histogram of Oriented Gradients (HOG)

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

In my IPython notebook, I have a section devoted to HOG. In it I provide a function that is from the lectures to get_hog_features. This function takes an image, orientation, pixel per cell, cell per block, vis, and feature_vec. For our purposes, the image stems from the car and not-car images provided. The unique parameters for HOG are more subjective.

1. Orientation – Number of orientation bins that the gradient information can be split up (e.g. 0->360, with 360/orientations as bins)
2. Pixel_per_cell – Cell size over which each gradient histogram is computed
3. Cell_per_block – Local area over which the histogram counts in a given cell will be normalized

Here is an example of an original vehicle image and a HOG transformed image:



I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`).

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

I evaluated different parameters for Orientation, Pixel_per_cell, and Cell_per_block. I noticed better performance, namely fewer false positives with larger values of orientation and pixel_per_cell compared to the implementation in the lecture.

Final Implementation:
Orientation – 12
Pixel-Per-Cell – 16
Cells-Per-Block - 2

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).

After extracting the features, I stack and normalized the data using StandardScaler(). Next, I split the data into a training and test set with train_test_split(). Now, I perform a fit to a LinearSVC() and check the accuracy.

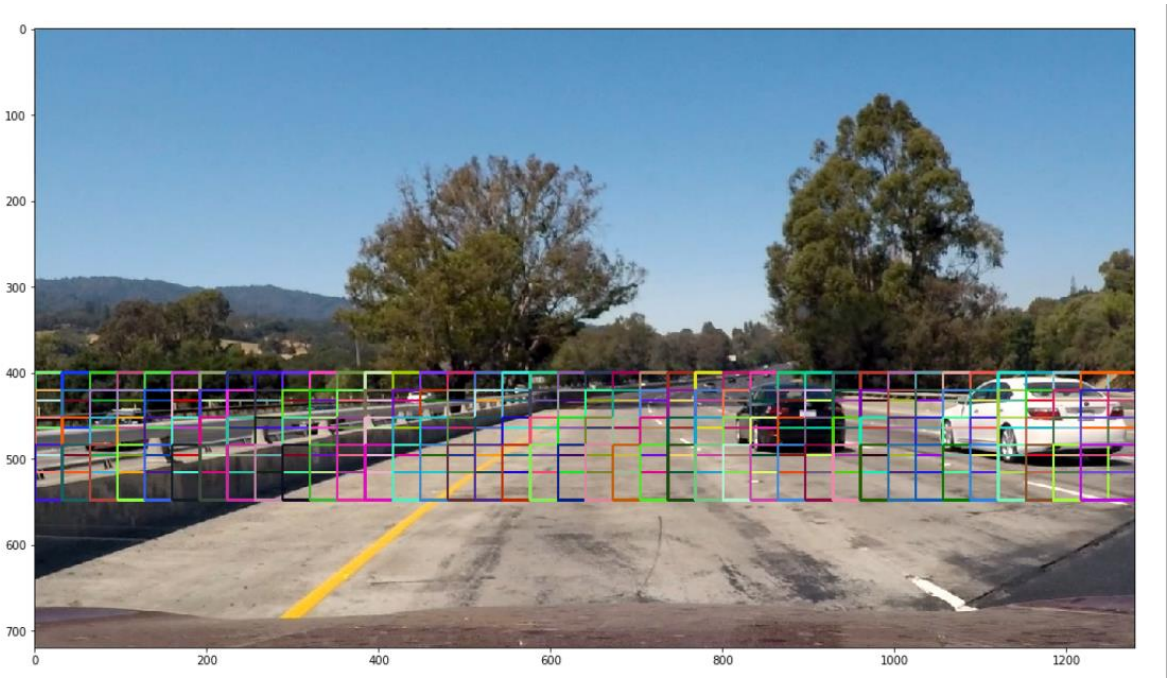
The exact code base can be found in the IPython notebook under the Extract Features and SVM classifier.

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 decided to use a hog sub-sampling window search implementation. This function performs one hog feature extraction and then is sub-sampled to get all of its overlaying windows. I tested with various scaling values with different areas of interest.

An example of one is shown below,



The image above shows two calls for hog sub-sampling windows that are evaluated with a scale 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?

Ultimately, I searched on ten scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here is an example image:

The above example performed well, but suffered from false positives. To mitigate against these false positives, I incorporated a heatmap threshold limit.

The heat signatures would record how many hits for a vehicle detection in a region. A threshold would be defined that would determine if the heat signature was valid. The image below is an example of a heat threshold where at least two hits or more are required.

Further testing showed the need to store previous vehicle detections to smoothen the detection of vehicles. This part of the code is best shown in the video.

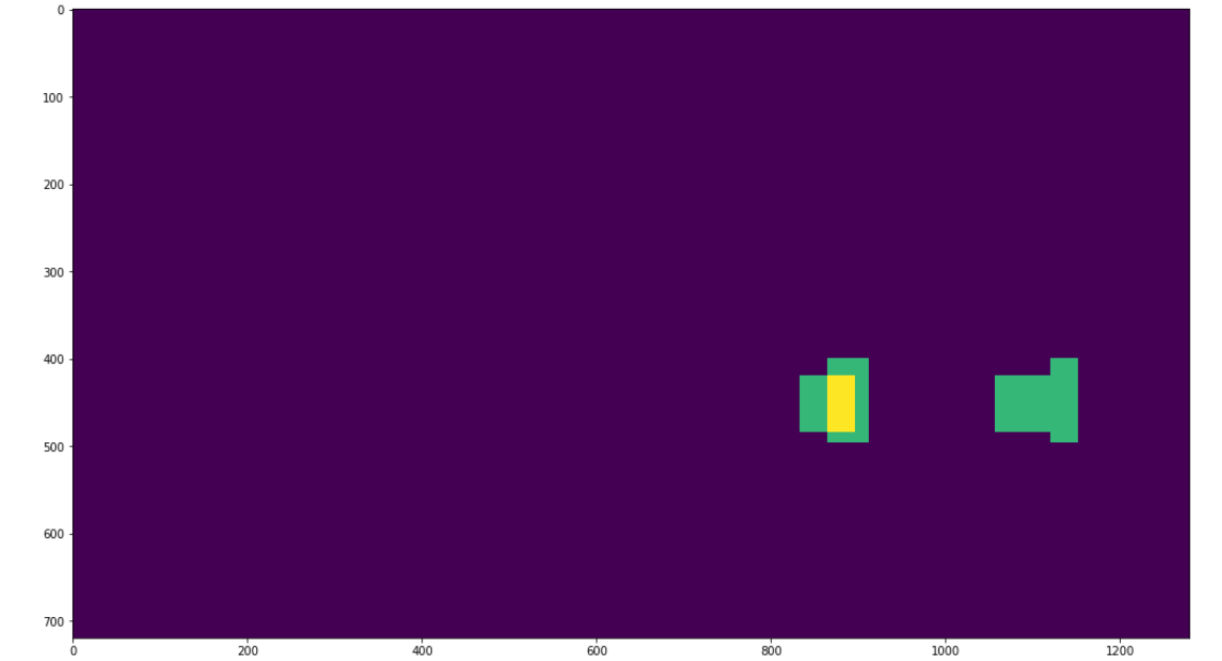
Video Implementation

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 is attached to this project.

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 separate false positives from actual vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I drew a bound over the boxes to cover the area of each blob detected.
Here's an example result showing the heatmap from the previous test image:



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?

I had a problem with mismatched shaped feature vector. I had to practically restart at that point since I failed to isolate the issue to the root cause. After that point I struggled to find the appropriate parameters that would not produce false positives. I found it very useful to use the test video as it shortened testing time significantly.

I iterated through each parameter trying different color spaces, orientations, pixel_per_cell, and cells_per_block until I was satisfied. Then I moved on to the sliding windows which took some time fine tuning.

I was interested in determining a smarter sliding window function for the future which would determine the general perspective transform. I remember in art class the technique for drawing realism and 3D on a 2D plane (paper) you would use a vanishing point. From there you can create lines that permeate out to the edges of paper. If we

could establish a horizon and a vanishing point then we could estimate the windows of interest.

We also could have tweaked the lane lines to find the road. Then we could limit the space to the road instead of the lower half of the image captured.

If I would pursue this further I would also look at using another camera. Then I could interpret the distance based on similar detection. If two cameras could identify the same point then it is possible to triangulate a distance.