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
- Implement a sliding-window technique and use the 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.

```
[//]: # (Image References)
[image1]: ./examples/car_not_car.png
[image2]: ./examples/HOG_example.jpg
[image3]: ./examples/sliding_windows.jpg
[image4]: ./examples/sliding_window.jpg
[image5]: ./examples/bboxes_and_heat.png
[image6]: ./examples/labels_map.png
[image7]: ./examples/output_bboxes.png
[video1]: ./project_video.mp4
```

Rubric Points

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

Writeup / README

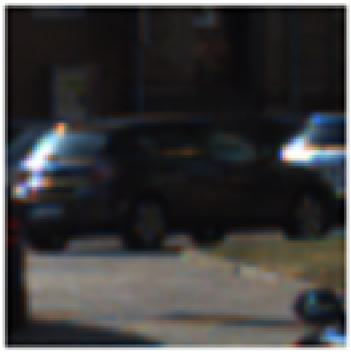
You're reading it! (I hope!)

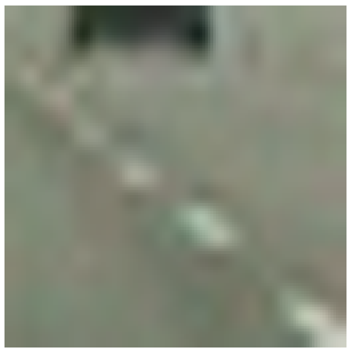
Histogram of Oriented Gradients (HOG)

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

The code for this project is contained in the file called “p5.py”. *DISCLAIMER: I freely acknowledge that at least 90% (estimated) of the code in p5.py is copied directly from either the lessons or the “Project Q & A” video on youtube.* The HOG features were extracted using the scikit-learn library, as described in the lessons, which is imported on line 11. There are two instances in the code where I call the “hog()” function provided by scikit-learn: it is called when extracting features from the training images, and again when extracting features from the images being processed (e.g., the frames from the project video). A wrapper function called get_hog_features() is defined in lines 53-70 of p5.py.

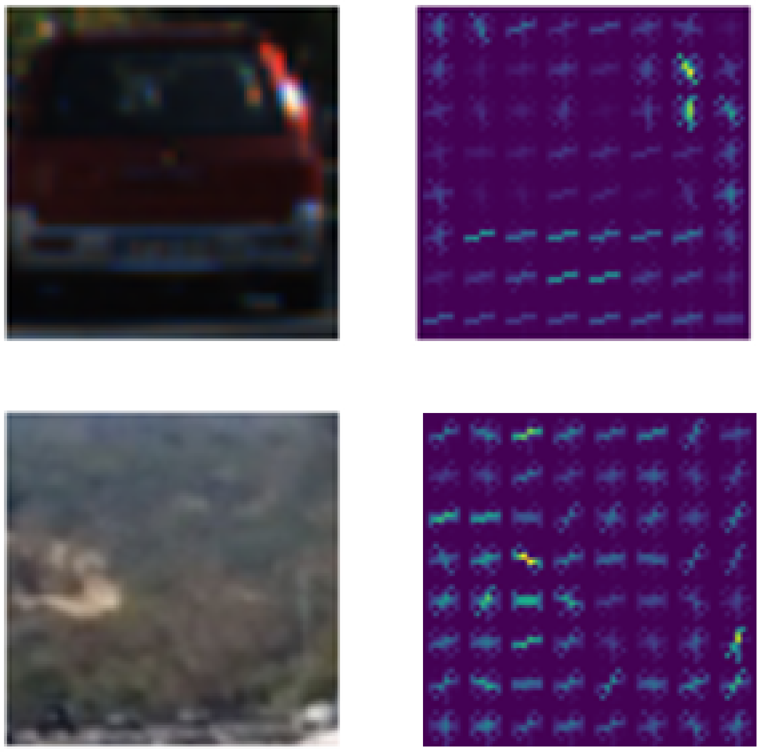
I started by reading in all the 'vehicle' and 'non-vehicle' images. Here is an example of one of each of the 'vehicle' and 'non-vehicle' classes:


A random car image


A random non-car image

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 are examples using the 'YCrCb' color space and HOG parameters of 'orientations=9', 'pixels_per_cell=(8, 8)' and 'cells_per_block=(2, 2)' (clockwise from top left: vehicle; vehicle HOG features; non-vehicle HOG features; non-vehicle):



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

I tried various combinations of parameters during the lesson quiz exercises, and observed the following results:

```
# 2.27 Seconds to extract HOG features...
# Using: 9 orientations 8 pixels per cell and 2 cells per block
# Feature vector length: 1764
# 0.07 Seconds to train SVC...
# Test Accuracy of SVC = 0.94
# My SVC predicts: [ 1. 0. 0. 1. 1. 1. 0. 0. 1. 0.]
# For these 10 labels: [ 1. 0. 0. 1. 1. 1. 0. 0. 1. 0.]
# 0.00112 Seconds to predict 10 labels with SVC

# 1.46 Seconds to extract HOG features...
# Using: 7 orientations 15 pixels per cell and 3 cells per block
# Feature vector length: 252
# 0.09 Seconds to train SVC...
# Test Accuracy of SVC = 0.915
# My SVC predicts: [ 0. 0. 1. 0. 1. 0. 1. 1. 0. 1.]
# For these 10 labels: [ 0. 0. 1. 0. 1. 0. 1. 1. 0. 1.]
# 0.00108 Seconds to predict 10 labels with SVC

# 3.39 Seconds to extract HOG features...
# Using: 5 orientations 16 pixels per cell and 4 cells per block
# Feature vector length: 240
# 0.03 Seconds to train SVC...
# Test Accuracy of SVC = 0.96
# My SVC predicts: [ 0. 1. 1. 0. 1. 0. 1. 1. 1. 1.]
# For these 10 labels: [ 0. 1. 1. 0. 1. 0. 1. 1. 1. 1.]
# 0.00112 Seconds to predict 10 labels with SVC

# 4.01 Seconds to extract HOG features...
# Using: 11 orientations 16 pixels per cell and 4 cells per block
# Feature vector length: 528
# 0.03 Seconds to train SVC...
# Test Accuracy of SVC = 0.98
# My SVC predicts: [ 0. 0. 1. 0. 1. 1. 1. 1. 0. 0.]
# For these 10 labels: [ 0. 0. 1. 0. 1. 1. 1. 1. 0. 0.]
# 0.00108 Seconds to predict 10 labels with SVC
```

Based on those results, I thought initially I would use the last set of parameters shown, since the accuracy reported was highest. However, after watching the project Q&A video & implementing a basic working system, I discovered two things: the parameters chosen by Mr. Keenan in the demonstration worked pretty well, and the set I had intended to use was a bit hard on the CPU of the computational resource that was most convenient to use (i.e., my laptop). Therefore I elected to stick with 9 orientations, 8 pixel square cells, and 2 cells per block.

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 defined my parameters in global variables, initialized in lines 24-29. This way, they could easily be shared by both the training, and the later use of the classifier for predicting. I trained a linear SVM using all of the vehicle images in the set of 8792 provided, as well as all of the 8968 provided non-vehicle images. The training is encapsulated in the function train_clf() defined in lines

444-593 of p5.py. The feature vectors I used contained 8460 elements.

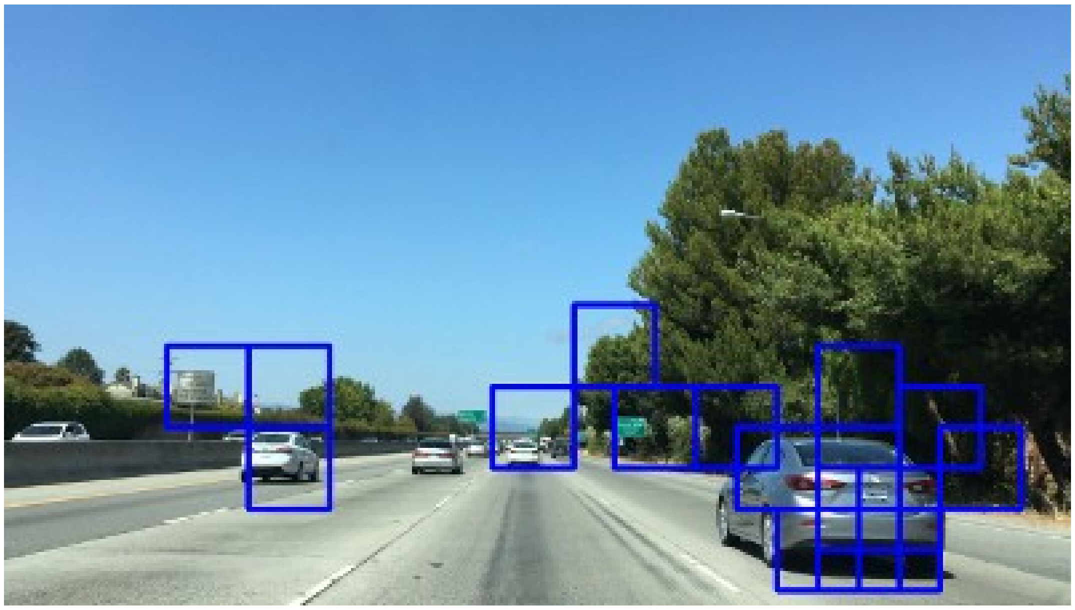
Sliding Window Search

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?

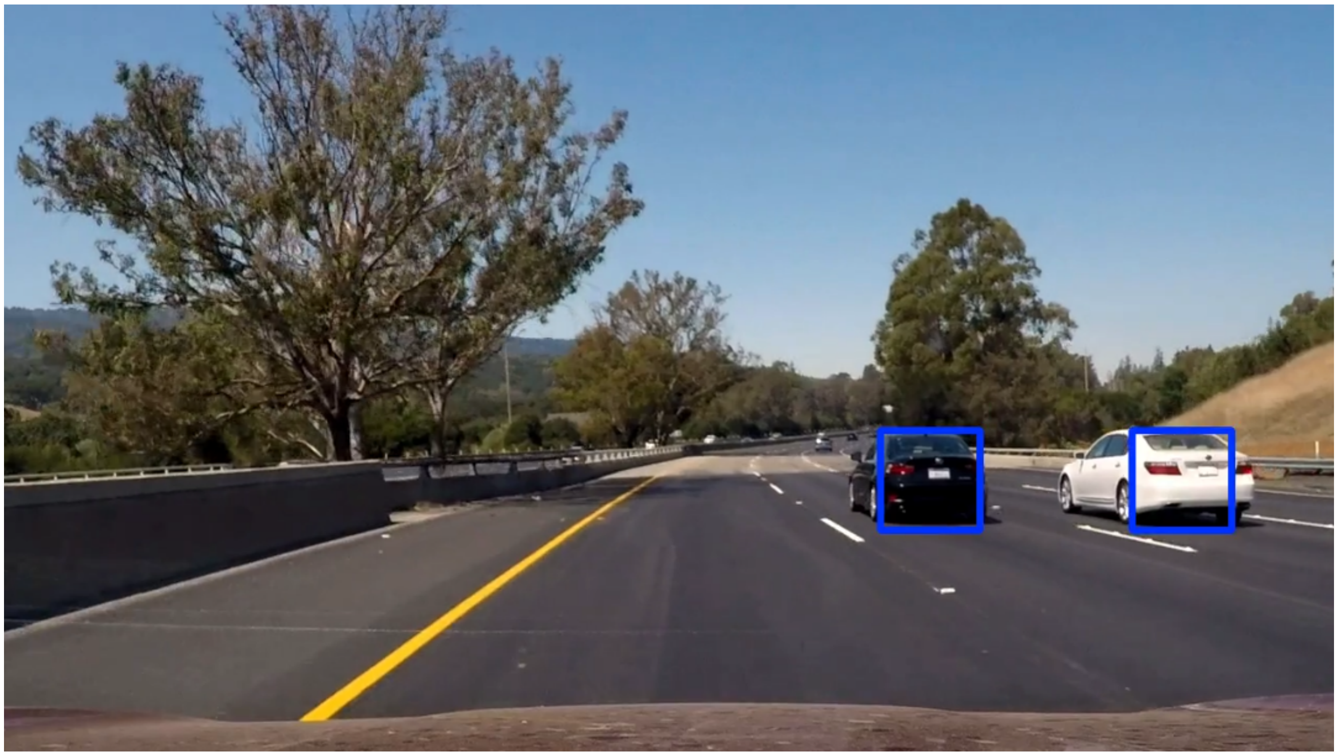
I limited the searched area of each image to avoid looking for cars in the sky and the treetops. I created a region to search in lines 335-341, basically it defines the bottom half of the image. I simply used the values for overlap and window size that were presented in the Q&A video; although I would have liked to implement a scheme that searched smaller windows near the center of the image (i.e., near the horizon), and larger windows near the bottom, as described in one of the lessons, it is the very last day to submit, as I’m typing this, and time is of the essence.

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 using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided an acceptable (I hope!) result. Here is an example image from an earlier stage in development, showing rather poor results:



...and here is an example frame after switching from RGB to YcrCb (and making other changes):



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 processed video, with no false-positive rejection scheme implemented.

In an attempt to reduce the rate of false-positives, I added code in lines 31-33 and a function “integrate()”, defined in lines 417-434. This code adds the heat maps from 5 frames of video, and zeroes out any areas that are identified as part of a vehicle for fewer than 2 out of a sequence of 5 consecutive frames. In other words, areas that are identified as vehicles only once in a sequence of 5 frames are rejected as presumed false positives.

Here's a link to my Project 5 video with false-positive rejection implemented.

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?

The main problem I faced with this project is not having enough time to explore different parameters and approaches, and experiment more. I did not have time to try to incorporate more training data, or to experiment more with different parameter values, or to explore different sliding-window search schemes. I think that by carrying out more of those kinds of experiments, I would be able to make my pipeline more robust, with fewer false positives.