**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**](https://review.udacity.com/#!/rubrics/513/view)**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'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.

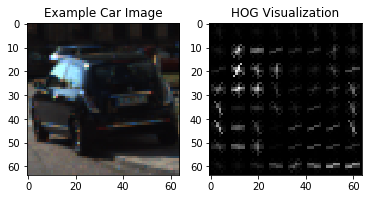
The code for this step is contained in the first and third code cell of the IPython notebook.

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=9, pixels\_per\_cell=8 and cells\_per\_block=2:



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

Using these parameters the trained SVC on HOG had a test accuracy of .9811.

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 two linear SVCs separately using color for one and HOG for the other. The color SVC statistics are:

Using spatial binning of: (32, 32) and 32 histogram bins

Feature vector length: 3168

47.72 Seconds to train SVC...

Test Accuracy of SVC = 0.9155

My SVC predicts: [ 1. 0. 1. 0. 1. 0. 1. 1. 1. 1.]

For these 10 labels: [ 1. 0. 1. 0. 1. 0. 1. 1. 1. 1.]

0.002 Seconds to predict 10 labels with SVC

The HOG SVC statistics are:

215.0 Seconds to extract HOG features...

Using: 9 orientations 8 pixels per cell and 2 cells per block

Feature vector length: 4932

17.77 Seconds to train SVC...

Test Accuracy of SVC = 0.9811

My SVC predicts: [ 0. 0. 1. 0. 1. 1. 0. 0. 0. 0.]

For these 10 labels: [ 0. 0. 1. 0. 1. 1. 0. 0. 0. 0.]

0.002 Seconds to predict 10 labels with SVC

Then after many iterations on the test video I landed on the following parameters:

color\_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb

orient = 9 # HOG orientations

pix\_per\_cell = 8 # HOG pixels per cell

cell\_per\_block = 2 # HOG cells per block

hog\_channel = "ALL" # Can be 0, 1, 2, or "ALL"

spatial\_size = (32, 32) # Spatial binning dimensions

hist\_bins = 32 # Number of histogram bins

spatial\_feat = True # Spatial features on or off

hist\_feat = True # Histogram features on or off

hog\_feat = True # HOG features on or off

ystart = 400

ystop = 656

scale = 1.4

Combing HOG, color channel, and special size I got the following results.

Using: 9 orientations 8 pixels per cell and 2 cells per block

Feature vector length: 8460

12.67 Seconds to train SVC...

Test Accuracy of SVC = 0.9893

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 started by jumping two pixels for the window search but I was having issues with accuracy so I changed the overlap to one pixel which caused the video to take much longer to process. I chose a window size of 8 x 8 because it seemed to be around the a good average size for the car close up or fare away and did a good job detecting.

2. Show an example of a test image to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on two 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: 

**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](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/project_video.mp4)

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 with a value of 2 to identify 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 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:

**Here are six frames and their corresponding heatmaps:**



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?

This project was a balance between having accurate bounding boxes that completely encompassed the car and getting rid of false positives. An easy solution that would have made the project much more robust in relation to this project video would be restricting the area of the window search to only be in the lane that the cars were driving in. That would have gotten rid of the areas where almost all of the false positives were. However that would make the vehicle detector not be robust to any other video. There was also a balance between the time taken to process the video vs accuracy. We could have sampled at many different window sizes and done a parameter search to ensure accuracy of the SVC. The pipeline might fail if cars were turning or the video was under different video/weather conditions. It would probably be ideal if there was some time of smoothing of the bounding boxes from frame to frame so that they were not so jerky and jumpy.