****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 Report

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 lines 14 through 54 of the file called `car_classifier.py`.

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


car


notcar

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. I finished this step in another python file called test_hog_imgs.py.

Here is an example using the `RGB` color space and HOG parameters of `orientations=6`, `pixels_per_cell=8` and `cells_per_block=2`:


car hog features

 not car hog features

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

I tried various combinations of parameters and finally, I used 'YCrCb' color space and HOG parameters of 'orientation=9', 'pixels_per_cell=8' , 'cell_per_block=2' and 'hog_channel=ALL'. Our teacher in Udacity chosed these parameters, so I think maybe they would work. Actually they worked.

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 a linear SVM using spatial features, color histogram features and HOG features. You can see it in line 65-113 in functions.py file.

I used StandardScaler function to normalize the training data. Test size is 0.2. Code line is 56 – 73.

Finally, I saved the training result in svc_pickle.p file. Test Accuracy of SVC is about 99%.

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?

Sliding window code line is 11-75 in image_det.py file. Image_det.py file is used to test pictures in test_image files and also provides functions to support Vehicle detection in video.

In find_cars function, I use cells_per_step to decide how much to overlap

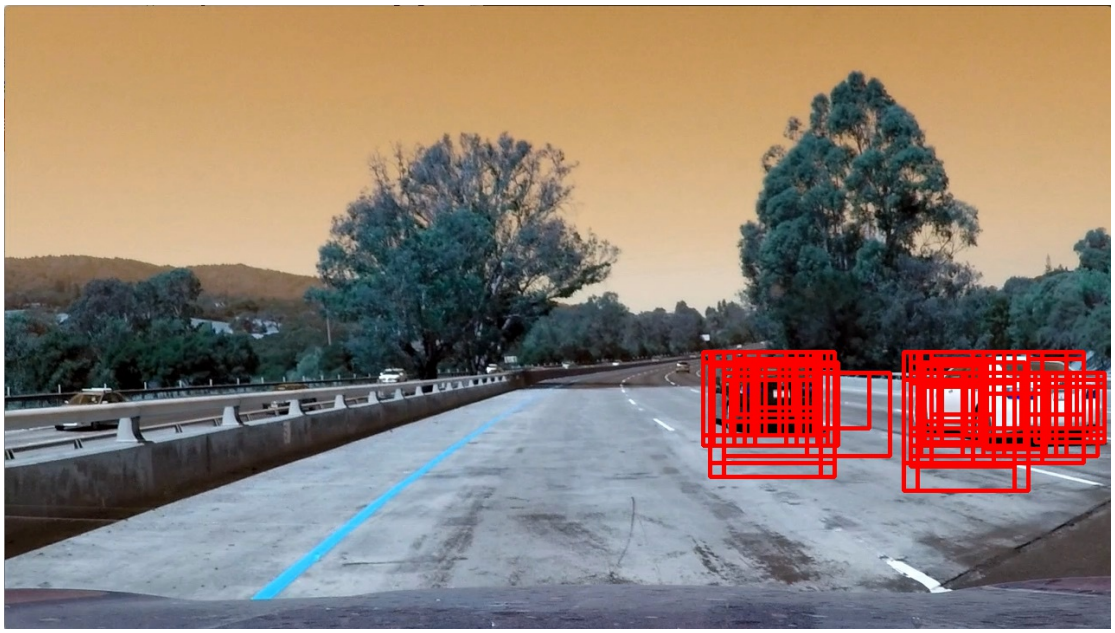
windows. Then slide windows through x axis and y axis(code line 43-73).

Scales can be decided by how large you want your windows be. If you want to use big windows, you chose a big scale, such as 2. Finally, I used 3 scales together.

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?

I decided to search with the range X: 640-1280 Y: 400-656(code line 185-188 in image_det.py file) window positions all over the images.

Ultimately I searched on 3 scales(1, 1.5, 2.0) using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Scales code line is 84-93 in image_det.py.
Here are some example images:



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.)

My video result is called project.mp4. I also provide test video called test.mp4.

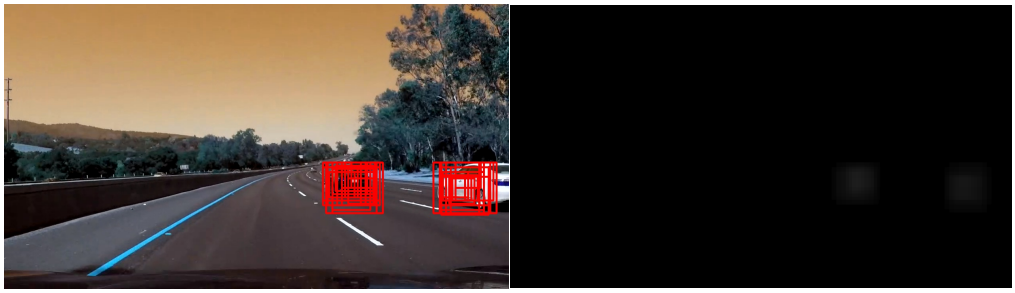
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

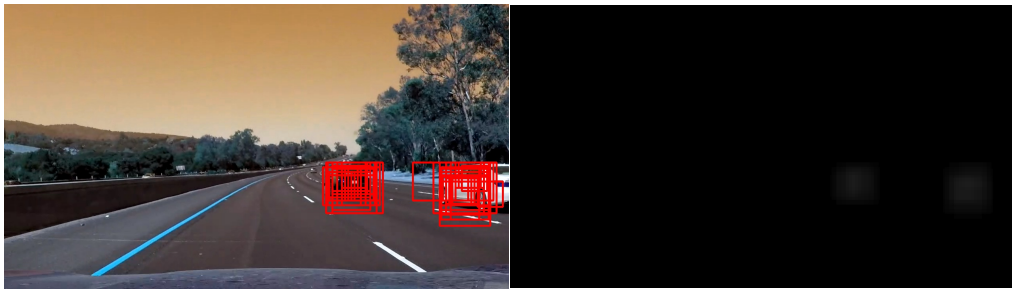
First, I restricted the image into a small one using xstart, xstop, ystart and ystop. I searched vehicles in a reasonable area of the image.

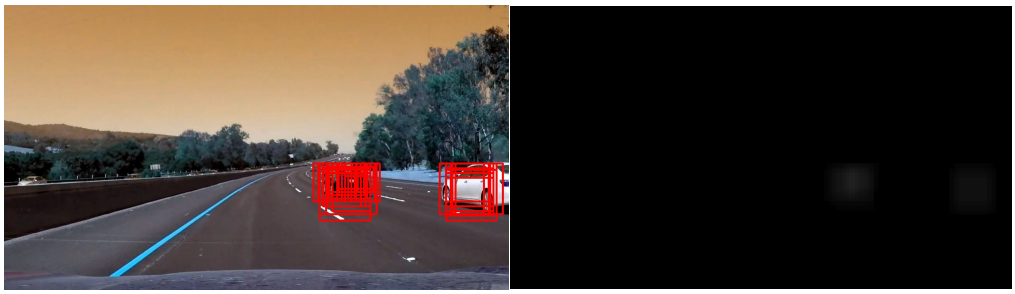
Second, I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions(I used 3 as my threshold, code line 100 in image_det.py). I then used `scipy.ndimage.measurements.label()` to 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 test_images, the result of `scipy.ndimage.measurements.label()`:

Here are six frames and their corresponding heatmaps: (heatmaps are not very clear and I don't know how to change white to red to make it clear)







Here the resulting bounding boxes are drawn onto the last frame in the series:

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?

Sometimes I cannot detect vehicles correctly in the video. I don't know why. I guess maybe the reason is train method or training set. But I am not sure.

When a car first time appears, my pipeline cannot detect it correctly. And the draw boxes are not quite stable.

I want to average the x and y positions, but the boxes are not correct position of the car. There is a tracing problem. But when I use the real position I found in each frame of video, the draw boxes were not stable. Hope better solutions.

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

I used class vehicle to save cars that I have already detected. But the biggest problem is that how can I know which exist cars the new car belongs to. I will improve the method to decide how to trace the same car.