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 https://review.udacity.com/#!/rubrics/513/view)

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](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

You're reading it! 😊

I type my writeups in Word because I am not in love with markup languages and end being very time consuming for me. Hope I will speed up my skills soon!

Histogram of Oriented Gradients (HOG)

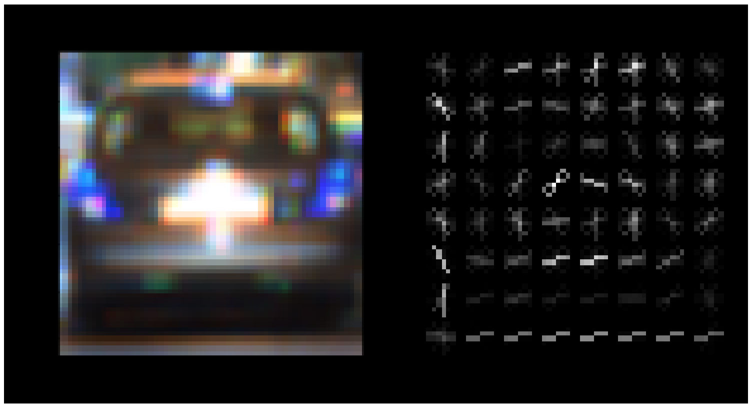
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

Look at file SVC_model_training.ipynb

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, 8)` and `cells_per_block=(2, 2)`:



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

By error trial I guessed what works best.

Block normalization L2 at sklearn HOG function is a true problem, gives me A LOT false positives. So the most important parameter for me is being sure is set to L1 which seems to be deprecated and not the default option anymore.

I also set HOG at channel 0, 9 orientations, color space set to YCrCb, pix per cell 8 pix, cells per block 2 and hog channel 0.

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).

Check SVC_model_training.ipynb file

dataset is augmented flipping the images, like using your selfie camera. More labeled data for free! A small fraction of 20% is reserved to measure training performance, and to be sure classifier has no obvious problems like overfitting. Images are normalized as suggested at lecture.

I trained a linear SVM using default parameters. Then I learn C is the most important you can fine tune.

As suggested, I used sklearn GridSearchCV and tune C parameter to 0.01 instead of default 1. That made the accuracy over the metrics a bit better.

However, the HOG features extraction parameters, did a much more remarkable effect when using the sliding window. Not choosing good HOG features makes the pipeline to have A LOT more false positives.

I ended having SVC tuned with 0.9821 accuracy which is fairly good.

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?

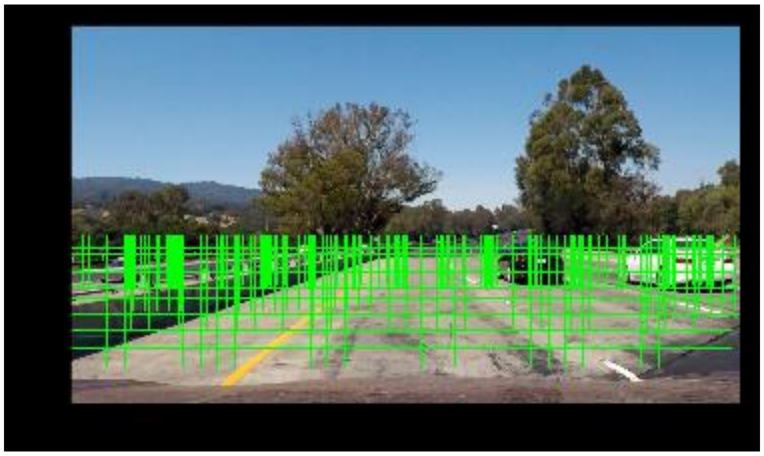
Now go to Vehicle_detect.ipynb and check the section Sliding Window Technique. The function is similar to the provided at the lectures. I used in the lower half of the window. I set horizon about y=400 and don’t use X limits. The X limits when sliding, can prevent you from detecting the oncoming cars, but the X point has to be calculated, is not fixed. So for the sake of simplicity I just use Y axis limits.

Windows are from 180x180 to 64x64.

The biggest 180x180px windows are applied in the proximity of our vehicle since the more near, the more bigger other cars will look.

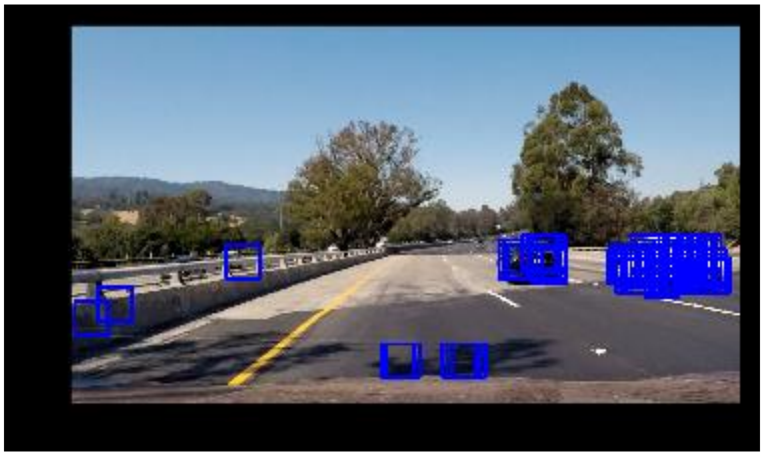
Then windows with sizes 150x150, 120x120, 80x80 and 64x64 are applied as you can see at this image.

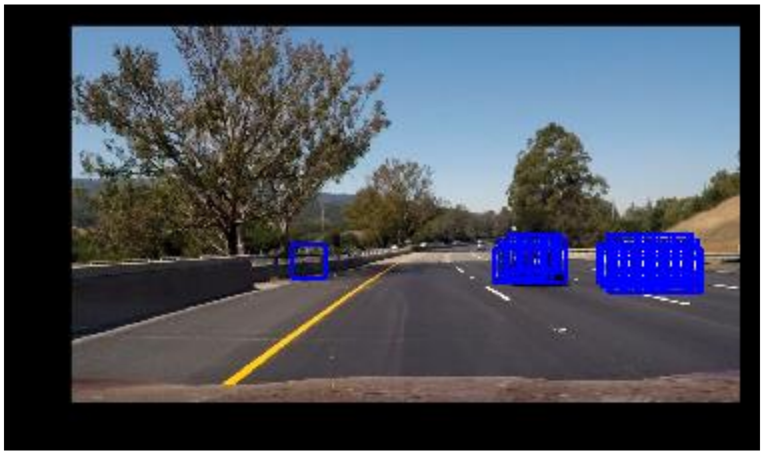
Overlap values I tested gives me good results for 0.8 to 0.85

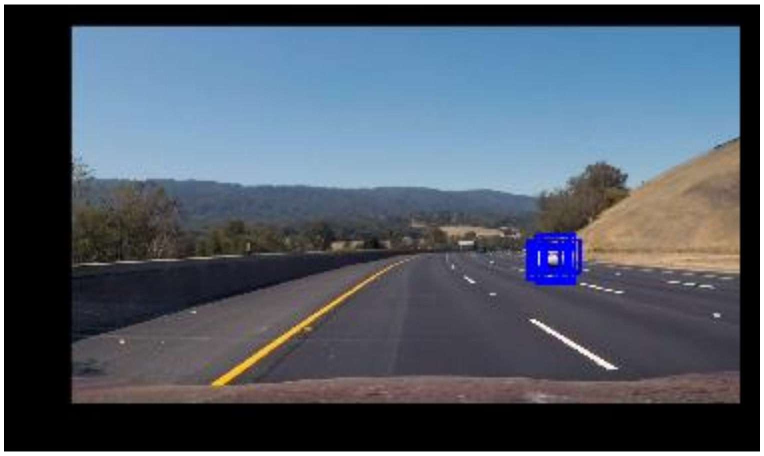


2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

I used all HOG features by trial / error, and then the filter for false positives.







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.)

See out_video.mp4 at my github repo

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 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.

Check the video_process function for moviepy frames. There is a threshold of three frames containing heatpoints before drawing a box.

I have not coded manually a function to check manually the heatmaps since it is already working, and didn't have to debug it. (Check the video)

If absolutely needed I can code it.

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 used the tips provided at lectures, nothing fancy, nothing exotic. L1 – L2 block homogeneization at sklearn HOG computing function is very weird for me since changes a lot the performance of the pipeline.

Feeding the SVM with correct values is the most difficult part for me along with the sliding window and heatmaps.

I have learnt how to feed data to a machine learning classifier by appending all the interest features. I liked it a lot.

I have read a bit about HOG, and seems to be a historic milestone at machine learning. The CNN used at traffic sign classifier are more easy to feed up, and there are also the new YOLO architecture which I suspect can fit very well solving this exercise in half or less code lines and at a faster FPS.

For real world I think this algorithm wont work as is, I got 1 FPS at most with a fairly new laptop (i7 8th gen) .