Vehicle Detection Project

Yi Zhu

The goals / steps of this project are the following:

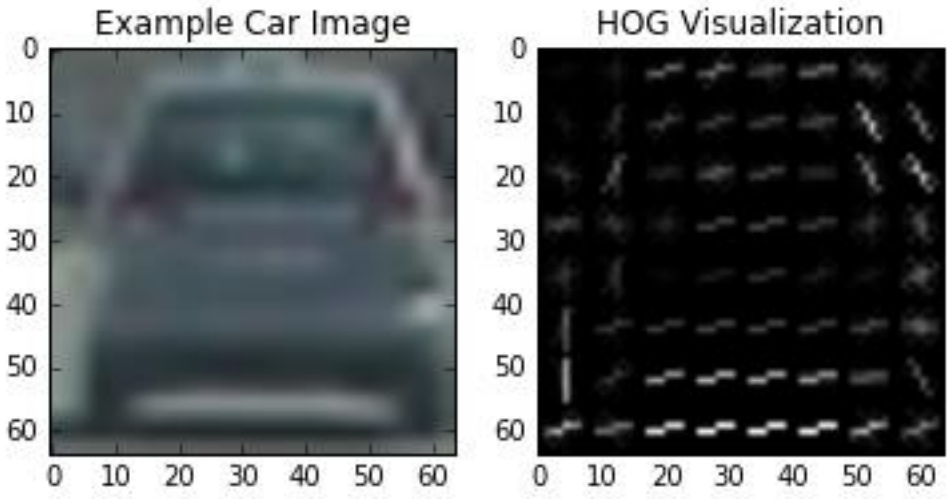
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: 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.
- **###Histogram of Oriented Gradients (HOG)**

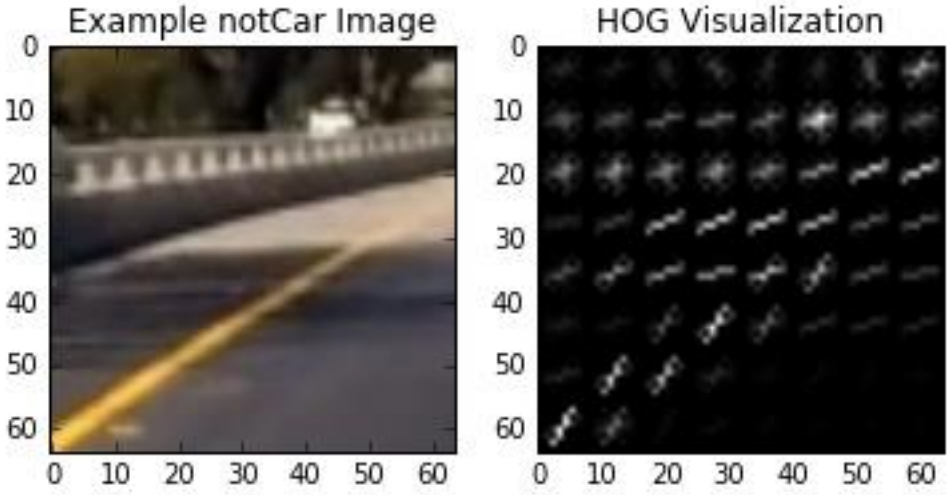
- **####1. Explain how (and identify where in your code) you extracted HOG features from the training images.**

- The code for this step is contained in lines 13 through 30 of the file called `lesson.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:

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 `gray` color space and HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:





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

I tried various combinations of parameters and it turns out colorspace= YCrCb and `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)` works best.

####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 first extend the feature vector using color related features. And then train a linear kernel SVM using 0.8 training and 0.2 test set which is after randomization.

Result of training:

190.91 Seconds to extract HOG features...

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

Feature vector length: 8460

23.73 Seconds to train SVC...

Test Accuracy of SVC = 0.9876

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.]

0.0015 Seconds to predict 10 labels with SVC

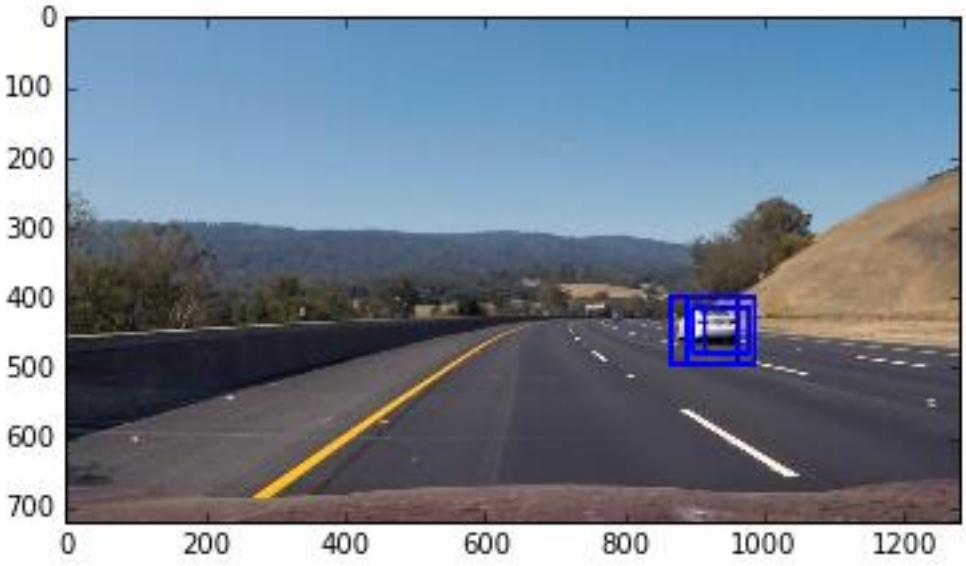
###Sliding Window Search

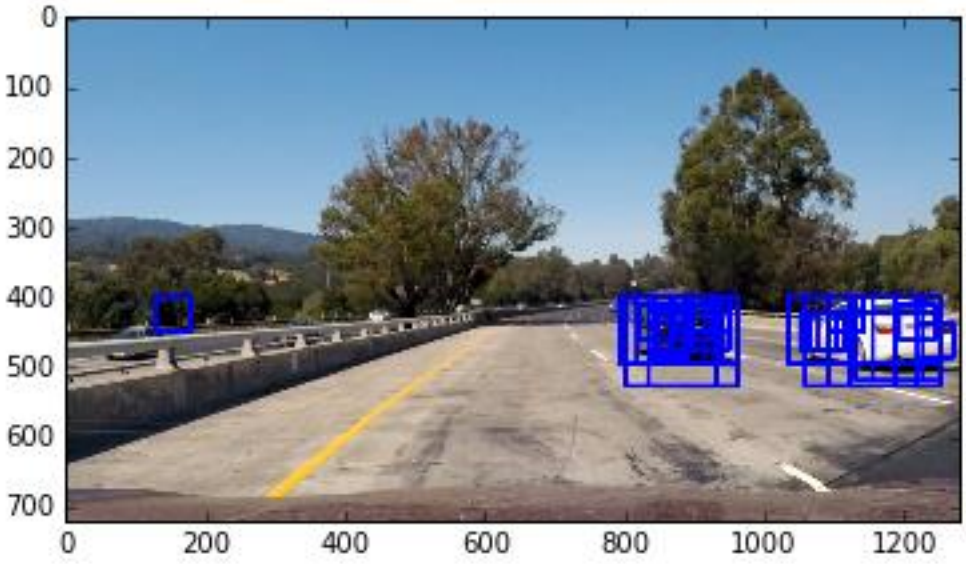
####1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I decided to search window positions using subsample technique introduced in lesson which is shown in line 42-56 in file subsample.py. Here I use window size to be 64. And search windows with different size by adjusting different scaling parameter. Here I search 2 scaling parameters=[1,1.5,2]

####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 two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice re sult. Here are some example images:

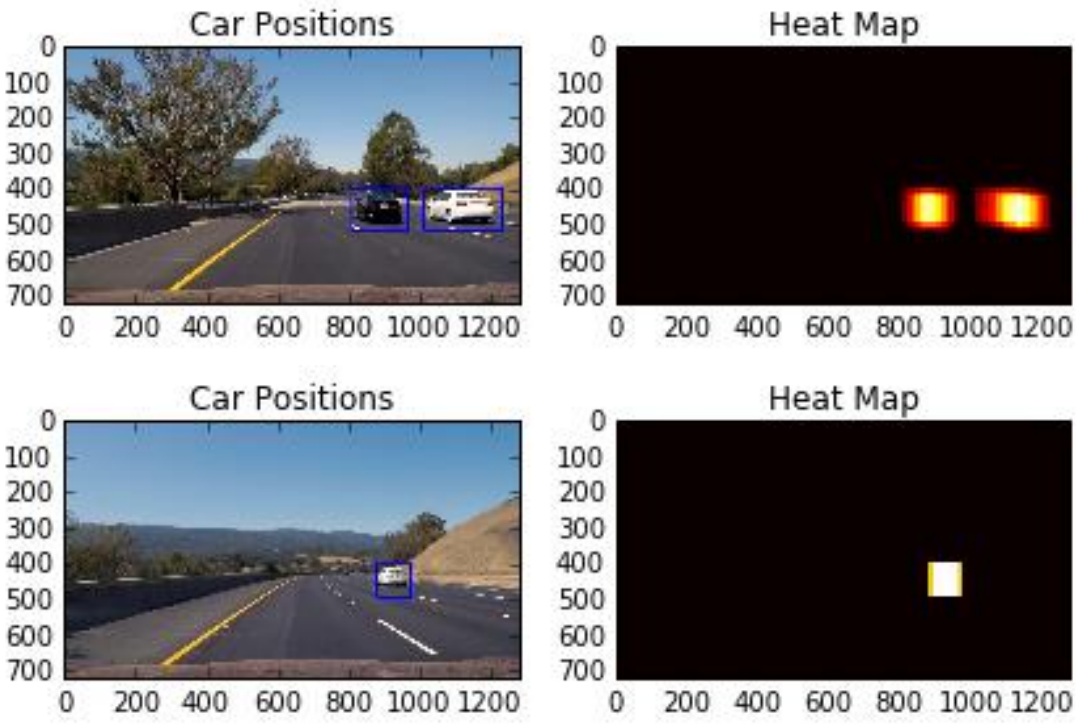




Video Implementation

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.

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:



###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?

Some tuning parameters are chosen manually which might lead to robustness issue in real scenario. For example, the scaling parameter, heat map threshold, etc.

The pipeline can also detect cars on the opposite lane, which in fact doesn't affect the normal driving. However, when implementing on real controller, this might lead to uncomfortable experience for passengers.

The possible solution might be taking advantage of series data to classify if the

car is going same direction or opposite direction and then make decisions to give command to controllers.