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.

In code cell 4 of P5 Python Notebook, I started by reading in all the `vehicle` and `non-vehicle` images. First, I only downloaded the subset of training data, vehicles images:1196 and non-vehicles images:725.

Even testing results were > 99%, I didn't get good results in car detection.

As it might be over-fitting with too few traning data, I downloaded GTI and KITTI images throug h links on Udacity.com. Combining them together, I have close to 10K training, vehicles images: 9988 and non-vehicles images:9693.

One problem is images are mixed in JPEG format and PNG format. I converted PNG images to 2 55 scale to be consistent with JPEG. I used shuffle utility for sklearn to shuffle training data and split 20% for testing.

Here are examples `vehicle` and `non-vehicle` classes:

Fig. 1 images for car and not-car

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=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`. Fig. 2 shows a hog image for a car.

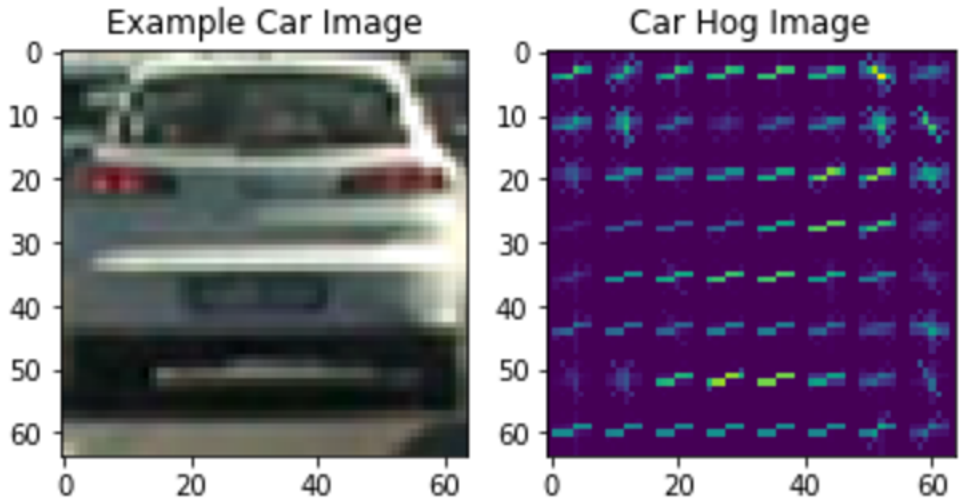


Fig. 2 HOG image for a car

I have tried various combinations of parameters in Udacity sample code in the lecture Table 1 shows the performance difference among different color space. Code cell 6 has the list of parameters setup. Code cell 7 has the function get_hog_features() to get hog features for SVM classifier.

Color Space	Accuracy
HSV	0.955
LUV	0.94
HLS	0.955
YUV	0.96
YCrCb	0.965

Table 1 Performance with respect to different color space

HOG Channel	Accuracy
1	0.96
2	0.93
ALL	0.985

I trained a linear SVM, which is code cell #19.

Here were results when I applied small Udacity training data set and one with additional GTI and KITTI. Both are greater than 99%.

```
0.36 Seconds to train SVC...
```

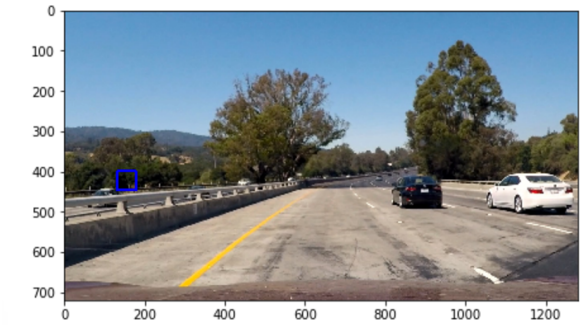
```
Test Accuracy of SVC = 0.9974

17.08 Seconds to train SVC...
Test Accuracy of SVC = 0.9919
```

Sliding Window Search

slide_window() function in the 13th code cell of Python Notebook, which defines a function that takes an image, start and stop positions in both x and y, window size (x and y dimensions), and overlap fraction (for both x and y).
search_windows() is in the 3rd code cell of Python Notebook. It returns a list of windows that match predication.

ystart	ystop	xstart	xstop	scale
400	650	950	1280	2.0
400	500	950	1280	1.5
400	650	0	330	2.0
400	500	0	330	1.5
400	460	330	950	0.75

Table 2 Parameters for sliding windows used in find_cars_boxlist().



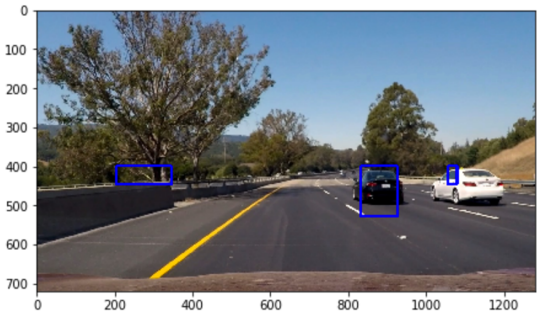



Fig. 3 show images from find_cars_with_heat_map() by using table 2 and heat map threshold three

Ultimately I used detect_car() routine from lecture note (code cell #22) using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Detect_car() is faster than find_car() with heatmap. Here are some example images:

Fig. 4 test images from detect_car()

Video Implementation

Output_images/project_video.mp4 is checked into GitHub under https://github.com/TianHuaBooks/P5. I implemented a simple filter with Python class

```
class MyCarState:
   def __init__(self):
      self.cleanup()

   def cleanup(self):
      self.count = 0  # frame count
      self.prior_boxes = [] # prior boxes
      self.oldest_boxes = [] # older than prior
      self.prior_image = None # cached image
```

When a new image comes in, I call detect_boxes() (code cell #21) to figure out windows. The windows go through simple process to union rectangles if overlapped. Then I check with MyCarState (code cell #31) with process_boxes () (code cell #30) to compare newest windows, middle-age windows, and oldest windows. If there is a window in the middle-age but not newest windows and oldest windows, I drooped it as a false positive. For windows exist among the three lists, I smooth them by average width and height.

To look ahead one image, I save an image in the MyCarState object, prior_image. I return the prior image for video display. As you can see in the result video, the simple filter does a nice job to eliminate false positive images. Smoothing doesn't do that well. However, as I only use 3 elements in width and height average.

The following two debug messages show two windows been dropped out by the filter.

```
lookupBoxes x:((1152, 400), (1248, 496)), n1:None, o1:None
dropped :((1152, 400), (1248, 496))
lookupBoxes x:((336, 592), (432, 688)), n1:None, o1:None
dropped :((336, 592), (432, 688))
```

Discussion

The simple filter I implemented filters out some false positive images. For cars coming from opposite direction, it doesn't filter it out. It needs a more sophisticated way to handle. The blue rectangle on a car has some jitters which should be improved as well.