Laura Le
20170711

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.

Writeup / README

All relevant codes are in P5_Vehecle_Detection.ipynb. utis.py contains functions provided in class.

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 1st and 3rd code cell of the IPython notebook
I started by reading in all the `vehicle` and `non-vehicle` images.

There are 8792 of car pics and 8968 non-car pics. The ratio of data is about the same so I did not have to augment the data for either car or non-car

I then explored different color spaces and different HOG parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`).

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

At first I train SVC with the provided dataset and the Test Accuracy of SVC is not great. No matter how I twist the HOG parameters (change color space, change , the Test Accuracy of SVC is around 0.97 area. To improve the result, I augment the data set by flip the image and rotate them vertically to generate more data. The result is still the same (around 0.97).
Later when I test svc on a test image (jpg files provided). The result is very lousy.

Ultimately I settle with this set of parameters which give me test accuracy 98.11%. In addition, I found out why the classification worked lousy on my jpg image. The reason is I forgot to scale jpg image by divide bit to 255 (such a bummer! It took me a while to figure out)

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

```
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 = (16, 16) # Spatial binning dimensions
hist_bins = 16    # 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
```

The classifier on full video gives many false positives.
I follow the suggestion to use SVC with kernel rbf. The test accuracy of SVC with kernel rbf is 99.34%. Howerver when test on jpg image or video, the classifier fail on the white car. The reason is that the training data set is biased toward dark-color cars.

Hence I need to augment the dataset with autti data set 2 here [https://github.com/udacity/self-driving-car/tree/master/annotations]
I added additional 3910 car images and 5854 not-car images.
3 random images of car and not-cars from autti data set are shown below
I borrow the code extract not-car frame from autti dataset from parilo at [https://github.com/parilo/carnd-vehicle-detection-and-tracking].
Those code extract not-car frames from autti pics (borrowed from parilo) are in 2nd cell under header "Augment data from Autti" in jupyter notebook.

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).

After augment, data set have:
```
Car samples: 12702
NonCar samples: 14822
```

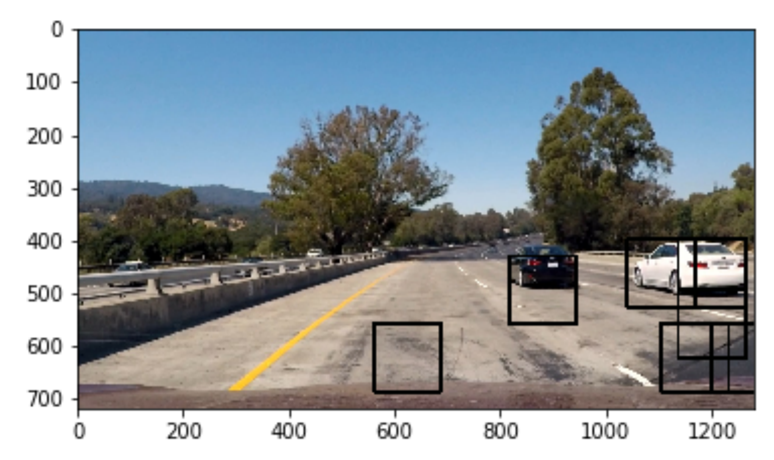
Code can be found in Cell 1 and cell 2 under "Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier" in python notebook.

As discuss above, I use a combine of HOG, spatial and histogram features. I trained a linear SVM using SVC with kernel rbf

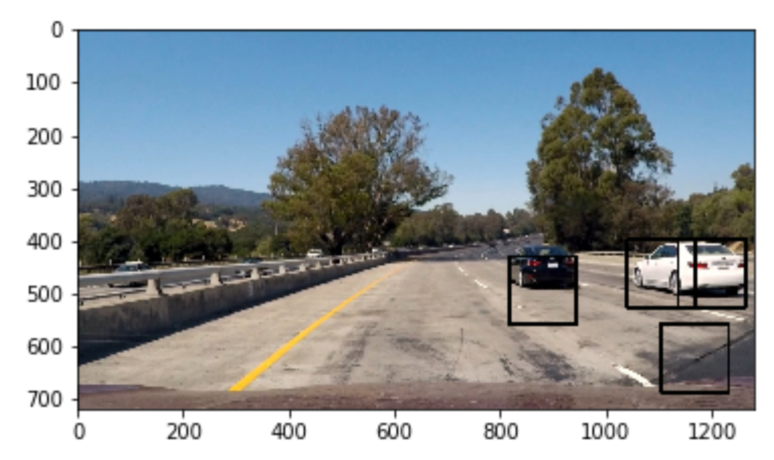
When I still use Linear SVC as classifier, there is a small trick I use in SVC prediction in search_windows() function. When I use svc to predict if a window contains car img, I don't use svc.predict like what the class provided. I used decision_function instead. That helps eliminate from false positive windows

```
test_features = scaler.transform(np.array(features).reshape(1, -1))
    dec = clf.decision_function(test_features)
    prediction = int(dec > 0.75)
    if prediction == 1:
        on_windows.append(window)
```

Example
When I use svc.predict



When I use decision_function with threshold 0.75, it help eliminate the false positive at bottom left



However when I switch to SVC with kernel rbf , the decision_function() did not work well (It does not recognize white car). Hence I use clf.predict() in svc classifier. The training time and test accuracy are below:
```
1540.59 Seconds to train SVC...
Test Accuracy of SVC =  0.9826
```

Sliding Window Search

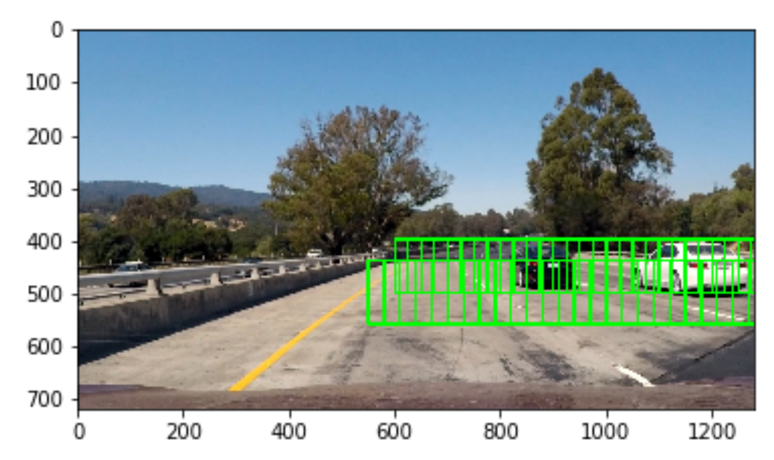
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

To improve speed when we process the whole video clip with multiple image, instead of scan the whole bottom half of the image, we will scan only where a new car can appear and where a car was previously detected (track car) The strategy is in every frame we look for car nearer to us (i.e. higher y) , and new further car (lower y)

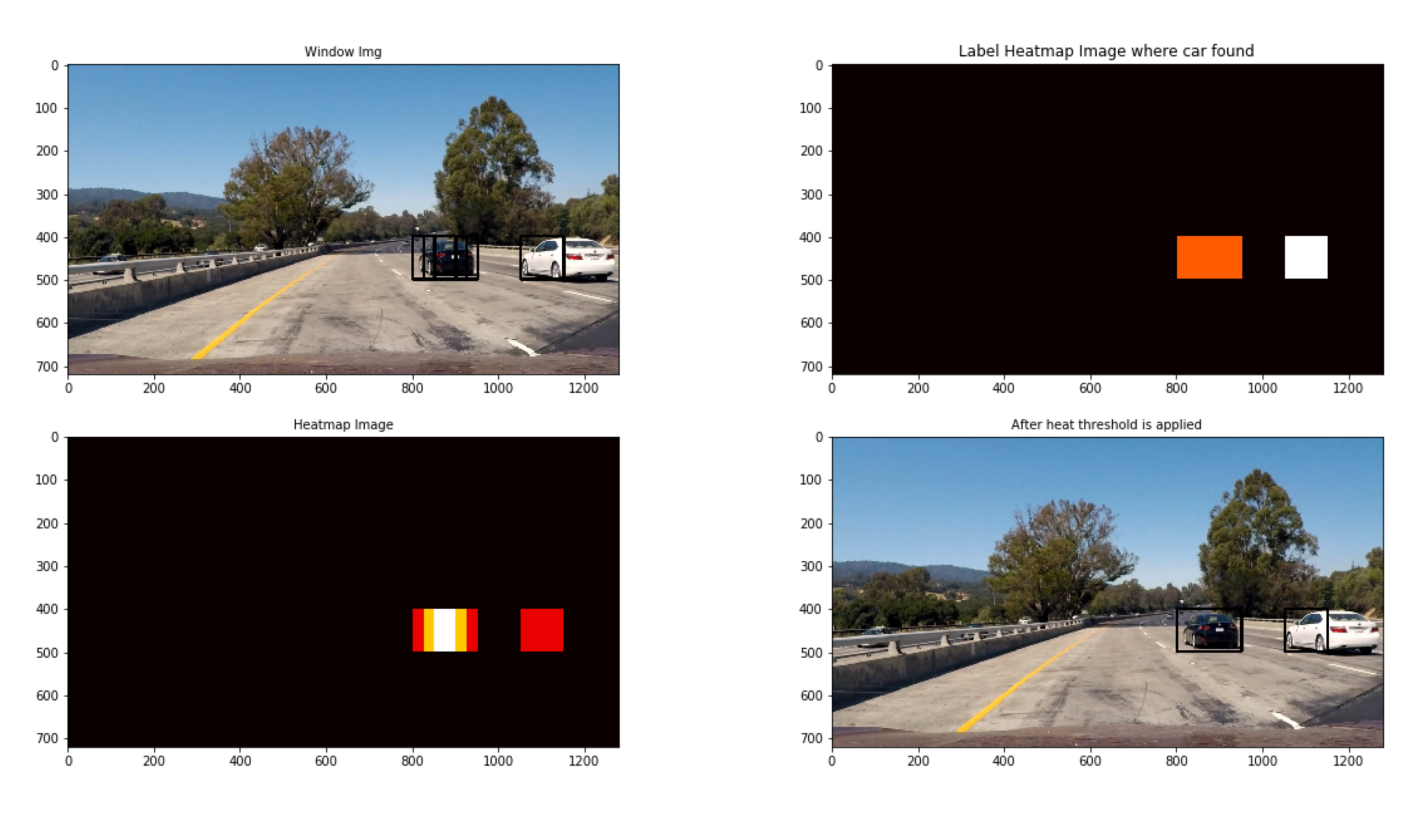
We use multiple scaled box to search for cars in image. The far will have smaller box; the nearer to the picture will have bigger box.The nearer has less boxes and large, the further has more boxes and smaller.

The function doing that is multi_scaled_box_2 in cell 1 under headline "Improve Sliding Window method". I tried a few different version with different X_start_stop and Y_start_stop. That's why the name is ended with _2. Apology for not cleaning up the function name.

Multi-scaled windows look like this with 2 types of window : 1 near and 1 far



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?
heatmap threshold is set to 0.1 in this pic.



The pipeline code in 2nd cell under headline "Pipeline and Video Implementation"
The pipeline is as following
1. Generate multi-scaled windows by sliding window , restricted to region that would have car
2. Use SVC which has been trained , to go through these windows, to predict which window contains car image
3. Combine windows (if there are multiple overlapped windows contains the same car) and eliminate false positive by using heat map

Please note that I only process even frame. For odd frame, I copy the hot windows from its preceding frame. The car will not change its position too much compare to its preceding frame. Use this method will save time tremendously when run the pipeline on the long video.

When process the video, I created a Boundary Box Queue (check class BBoxesQ) to store hot windows of the last 10 frames. I set threshold of heatmap = 7 to eliminate false positives and overlapped windows. I borrow this idea from some discussion on forum (sorry could not find the link to give that person the credit)

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.)

Check out video project_videp_processed_augment_skipframe_scale2.mp4 in output_video folder.

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 10 preceding frames of the video in the queue. 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.

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?

Make the classifier to recognize white car is extremely challenging. Linear svc can recognize white cars but it have too many false positives. SVC with kernel rbf is better (yet take longer time to train and to predict) but it fail to recognize white car. Finally I settle with the approach of augment my data set with autti images (there are a lot more white cars in autti data sets) and use SVC kernel rbf

My pipeline likely fail where it might mistake some of the steel barrier on highway as cars. I fix that by restrict the boxes to avoid the barrier. this is don mannualy when I set X_start_stop =[[550,1280], [600,1300]]in function multi_scaled_box_2(image). However this is not the best way to do in practice.   I should have combine the lane detection in P4 with this project so that the X_start_stop can be set in more dynamic way.

The bounding boxes are not run smoothly as the video elapses. In the Tracking video, the Otto person mention about tracking the centroid of the bounding boxes in which the car is detected with high confidence and then record how centroid move and estimate where it will happen subsequently. I think it's a very good idea but I don't know how to implement yet and don't have much time to test that suggestion. Let me know if you know any github link with a nice solution on this.