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.

Data Visualization

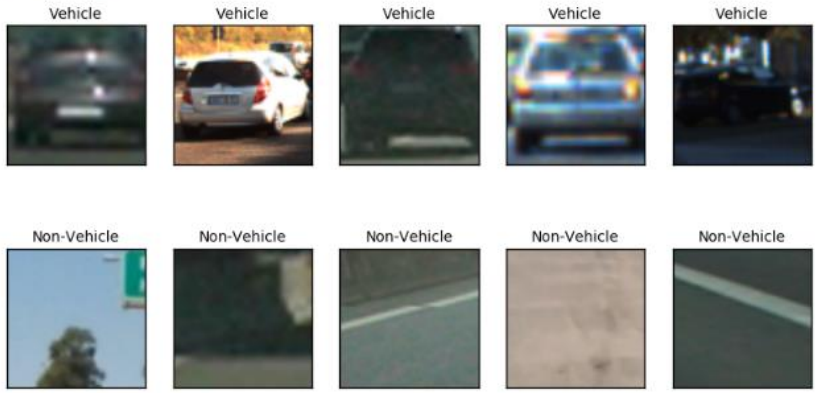
> This section was finished in data_visualization.ipynb. The dataset I used here was the dataset provided by the lecture (vehicle and non_vehicle image data)

Dataset size:

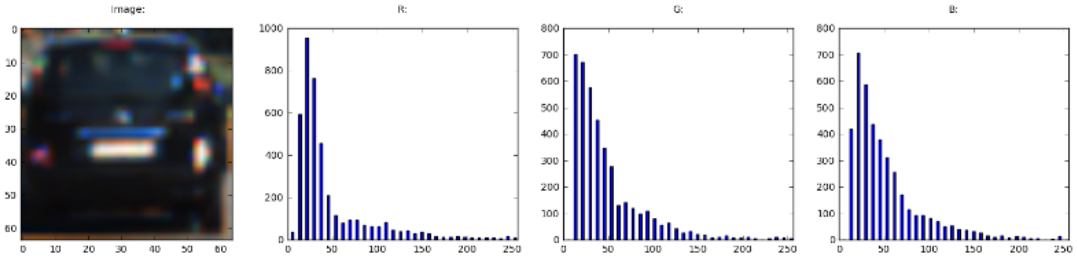
```
len(cars) 8792
len(notcars) 8968
```

Visualization the dataset(code was wrote in Cell4):

```
Total Number of Vehicle Images: 8792
Total Number of Non-Vehicle Images: 8968
Image Size: (64, 64, 3)
Image Type: float32
```

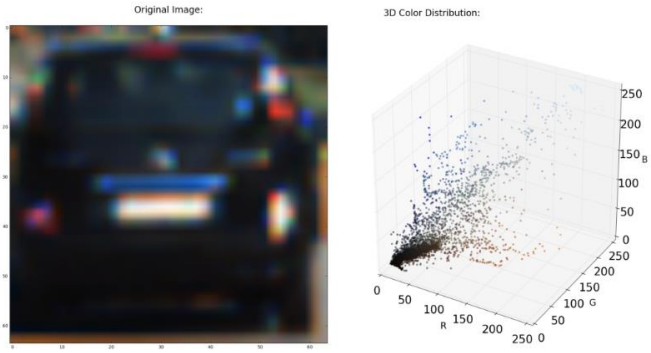


And then randomly chose an image to visualize its RGB data:



(From left to right: origin image, R, G, B)

And then I encountered an function on the forum and I used it here to visualize the 3D color distribution, which would be much more obvious than the data above: [cell 7]



Histogram of Oriented Gradients (HOG):

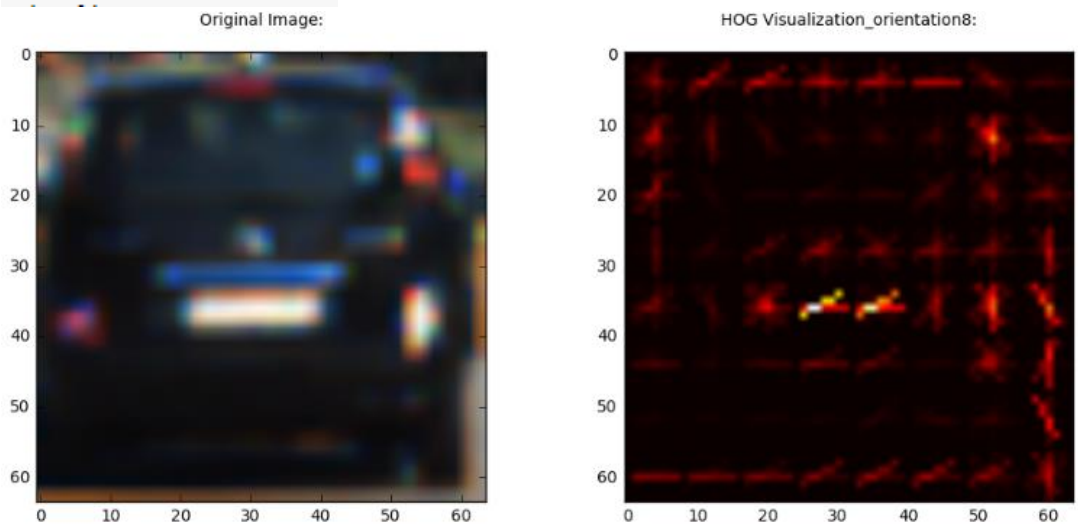
1. I started by reading in all the `vehicle` and `non-vehicle` images. Which was mentioned and visualized above.

2. I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I use the random

image above to test and displayed it to get a feel for what the `skimage.hog()` output looks like:

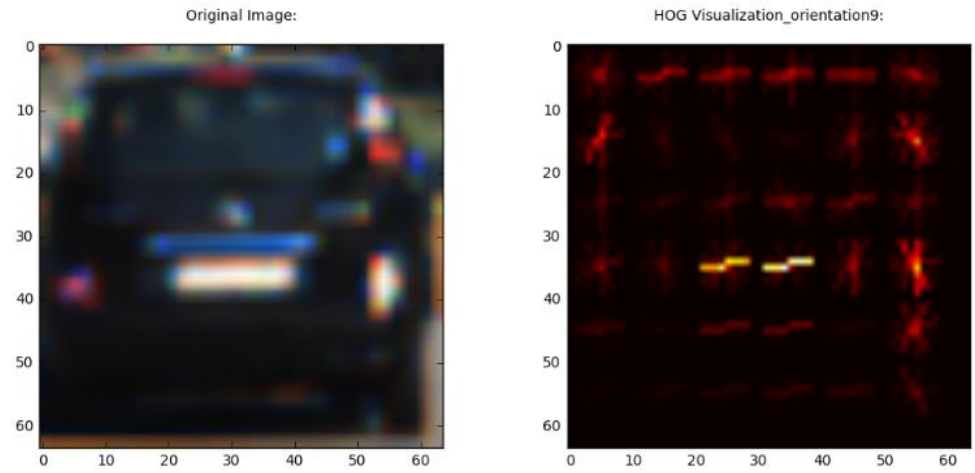
Case1

```
orientations=8,
pixels_per_cell=(8, 8),
cells_per_block=(2, 2),
```



Case 2:

```
orientations=9,
pixels_per_cell=(10, 10),
cells_per_block=(2, 2),
```



I tried various combinations of parameters and I finally decided to use this combination:

```
orientations=8,
pixels_per_cell=(8, 8),
cells_per_block=(2, 2),
```

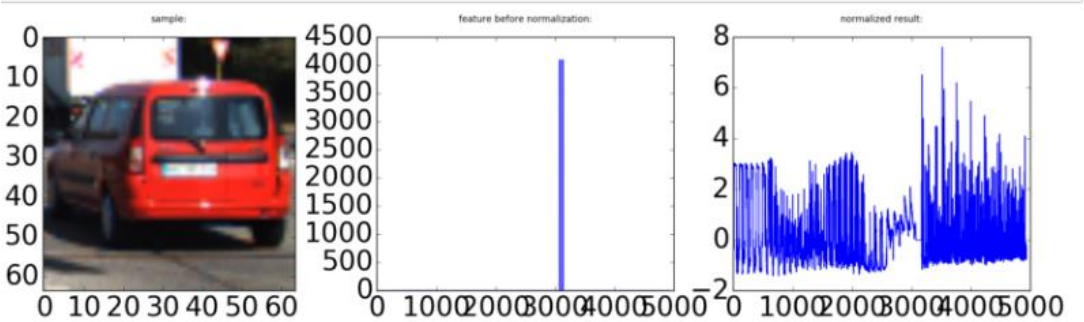
As this combination best detect the vehicle and it is works well when the picture is vague.

Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

This part was finished in video_pipline.ipynb

Firstly, I loaded the data and shuffled them[cell 2]. Then I extract their features[cell 3]. Since the raw data cannot be used to training. I did a data normalization to normalized the data :

Here is an example. The second picture shows the data before normalization and the third picture shows the data that was normalized.



Before training, I split the raw image data into training part and test part by using train_test_split function.

This function has been used many times this term.

from sklearn.model_selection import train_test_split

```
rand_state = np.random.randint(0, 10)
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size=0.2,random_state=rand_state)
```

The ration between training set and testing set is 8:2

At this step, the color space I use is RGB.

And them I trained the data. [cell 6 and cell7]: This is the result I got.

```
svc = LinearSVC(C=0.01)
t=time.time()
print("fitting")
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'seconds to train SVC.')
```

```
fitting
152.07 seconds to train SVC.
```

```
print('Test Accuracy: {0:0.4f}%'.format(svc.score(X_test, y_test)*100))
print('SVC  Predictions:', svc.predict(X_test[0:15]))
print('       Labels:', y_test[0:15])
```

```
Test Accuracy: 96.7905%
```

The test accuracy is 96.79%. And I saved this model in a pickle file named 'saved_svc_new.p" ,related X_scaler was saved in "saved_X_scaler_new.p"

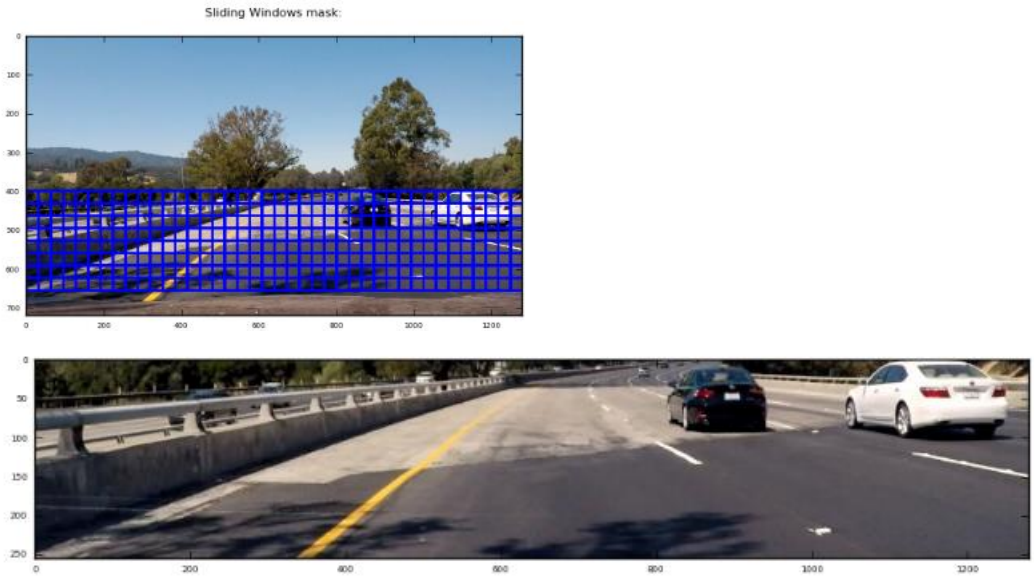
Obviously, 96.79% might not be good. So I tried LUV space. The steps were similar and the final accuracy I got was 98.9%. [Cell 28]

```
Using: 8 orientations (16, 16) pixels per cell and (4, 4) cells per block
color space: LUV
Feature vector length: 1200
3.64  seconds to train SVC.
Test Accuracy of SVC: 98.0856%
```

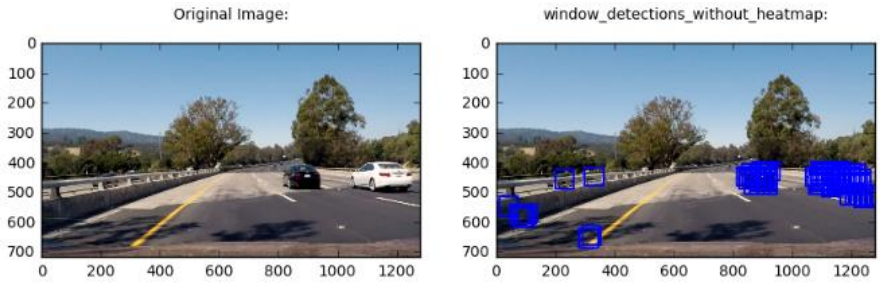
And I saved this model in "saved_svc_LUV.p" and "saved_X_scaler_LUV.p". You can load them if you need. The accuracy is much higher than RGB Channel.

Sliding Window Search

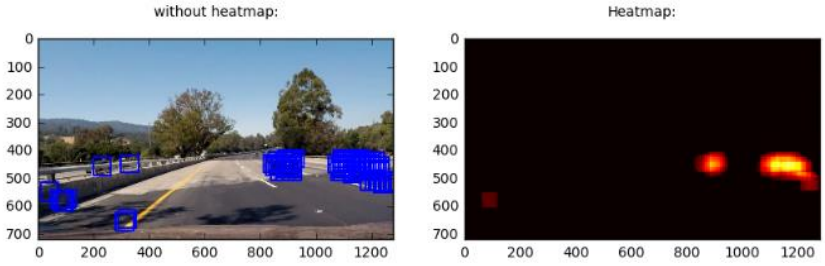
Firstly, I really decided to search random window positions at random scales all over the image. As you can imagine, I totally failed. And then I chose a region of interest to do my sliding window search. I y coordinate I used here is [400,656]. I only focused on this part:



And then I applied the svc I got in previous step(LUV color space) to find vehicles. Here is the result:



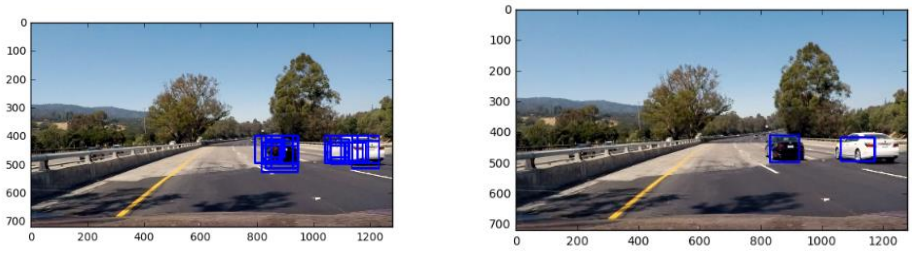
As you can see, the classifier works not well. Some empty area was mark as vehicles. So I add a heatmap to detect images.[cell 31]



By adjusting the heatmap threshold, I can simply control the output.

But this still have some problems when detecting the vehicles. Some empty areas was still mark with boxes(As it was shown above). I tried to change the threshold value but it helps little so I change the color space to YUV(this was suggested by the forum). And I retrained the classifier. and this time I got 98.5% accuracy, which is similar to the LUV space model.

And with similar steps, I got the final output:

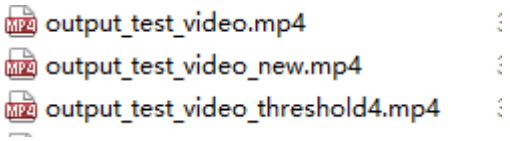


This time it fit well.

Ultimately I searched on two scales using YUV 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.

After trying many times to adjust the parameters, I use [400,656] as my final y-region of interest. My final heatmap threshold is 4.

You can still found these difference in my output test video(I only upload 3 of them, actually I test as least 10 different combinations):



Parameters:

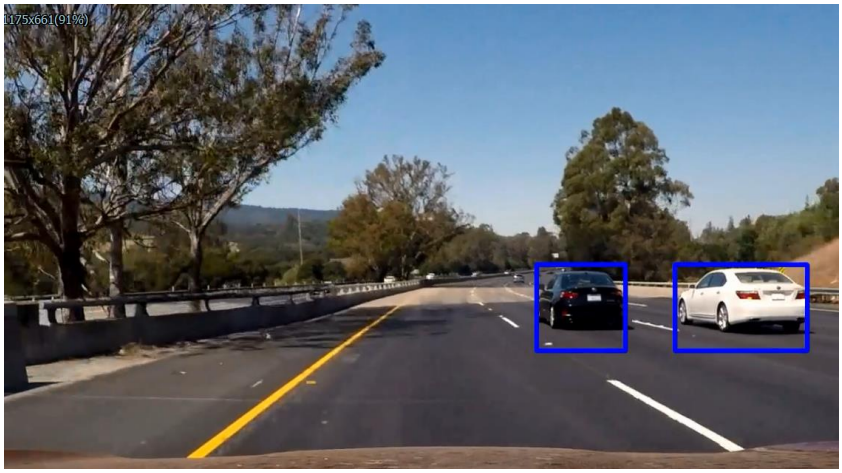
Output_test_video: y-region of interest:[400,500], heatmap threshold:2

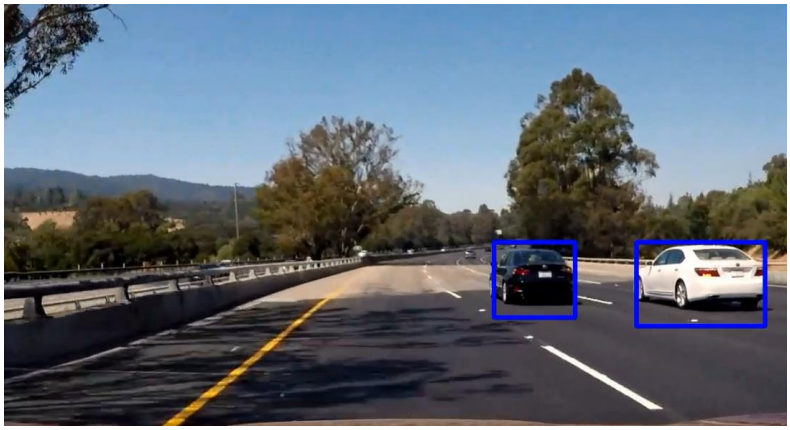
Output_test_video_new: y-region of interest:[400,656], heatmap threshold:2

Output_test_video_threshold4: y-region of interest:[400,656], heatmap threshold:4

And I chose Output_test_video_threshold4.mp4 as my final result.

This is several frame of my video process output:





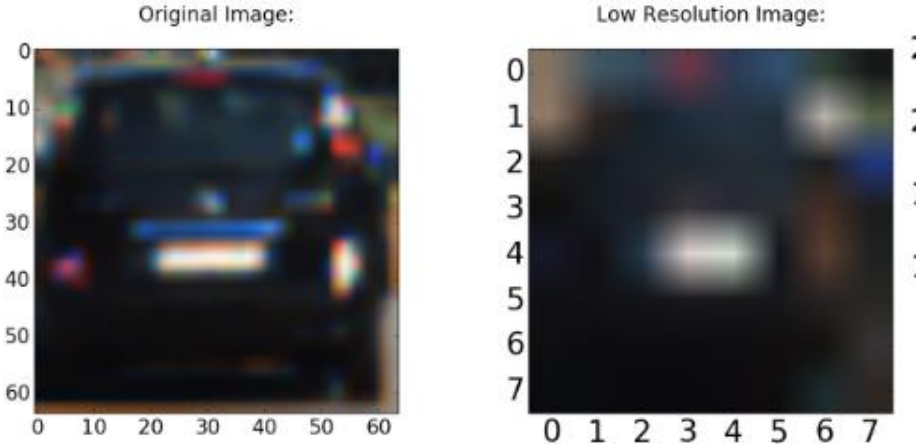
Discussion

1. Problems when finishing this project
   At first, I wrongly chose RGB as my color space channel and I failed to handle the video and I cannot found why it doesn't work. I changed my parameters and tested them many times, it still didn't work. After many attempts, I decided to go the forum and ask for help. I then found that changing the color space can led to different result and I set the color space as LUV and it worked well. And finally I chose YUV because it works better.
2. Problems when training the dataset. Actually I learn intro to machine learning course before I started this nanodegree program. So I didn't get some annoying bugs here. A little problem I encountered here was the dataset size. I firstly use 9:1 as my training set to test set ratio. The accuracy turned out to be not good. By splitting dataset in a right way(8:2), the accuracy I got would be better.

3. My pipline likely to fail when the vehicles are not clear in the image. As I tested in the data visualization part. I found out that when the vehicle sample images were not clear the hog features I got may not fit well.

And when I was building my pipline, I did not consider this kind of occasion. As you can see in my output videos, when the vehicles were too far from the camera, even they are in the region of interest, my pipline won't mark them as vehicles. I am not sure why this problem occurs but I think the reason might be the low resolution. In other words, when it comes to low resolution images, my pipline might fail.

Here is an example of low resolution image:



To solve this issue, some method I think can be implement to promote the performance of this pipline:

1) Use a larger training dataset
2) Try a different classifer
3) Try different parameters.