Writeup Project 5 - Vehicle Detection and Tracking

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.

Files in the repository

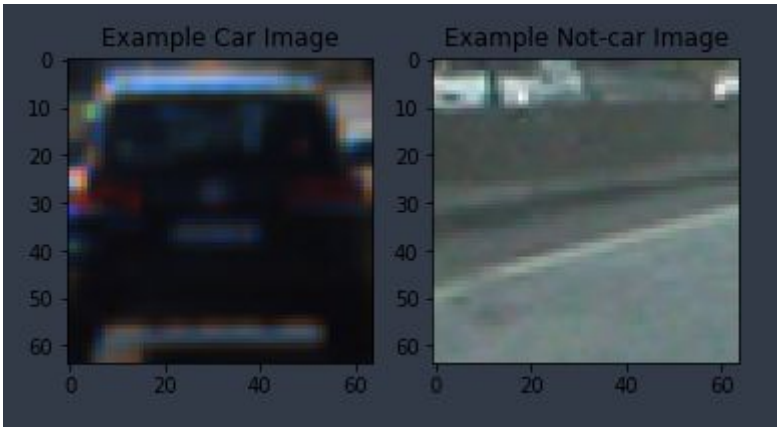
- Main pipeline will be found in the pipeline_P5.ipynb file
- Pipeline that shows how feature extraction works will be find at pipeline_P5_images.ipynb
- All the different training are saved as svc_pickle_"name_of_the_color_space". (If there is another letter, it means that hog features worked only on that feature, if there is noH_noS, it means that was implemented only a 3-channel-HOG feature.

Histogram of Oriented Gradients (HOG)

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

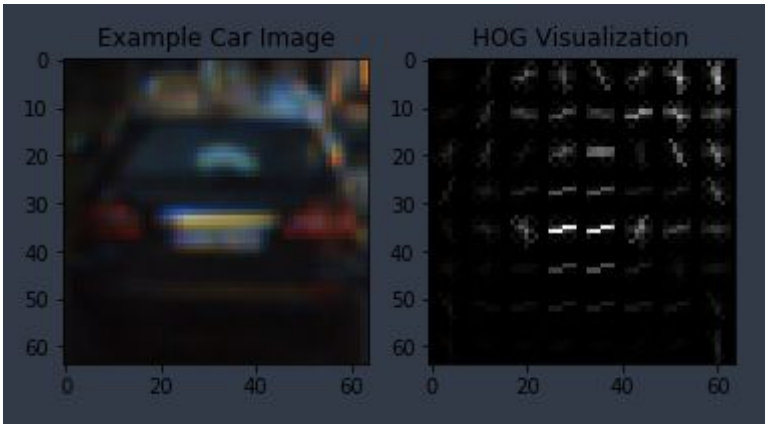
The code for this step is contained in the third point of the file called `pipeline_P5.ipynb`.

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 `RGB` 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.

Starting from color space I experimented with RGB, HLS, HSV and YCrCb. I found that the better results were obtained by HLS and RGB with accuracy higher than 98%. When I considered only one HOG feature (i.e. the S of the HLS color space) I got lower accuracy. So, I concluded that the best choice was considering all 3 channels of HOG parameters. I than experimented eliminating histogram features

and color features, but I ended up with lower accuracy. Accordingly to my results, I considered a concatenation of histogram, spatial and HOG features the best choice for my project.

I experimented a little with the other parameters (orient, pix per cell and cell per block) and I found that 9 orientation with 8x8 pix per cell and 2x2 cell per block were the best choice.

Spatial size is 32x32 and histogram bins are 16. I didn't test other possible options.

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 trained a linear SVM using `from sklearn.svm import LinearSVC`.

```
# Use a linear SVC
svc = LinearSVC()
# Check the training time for the SVC
t=time.time()
#parameter tuning
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svr = svm.SVC()
clf = grid_search.GridSearchCV(svr, parameters)
clf.fit(iris.data, iris.target)
print(clf.fit)

svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
# Check the prediction time for a single sample
t=time.time()
n_predict = 10
print('My SVC predicts:     ', svc.predict(X_test[0:n_predict]))
print('For these',n_predict, 'labels: ', y_test[0:n_predict])
t2 = time.time()
print(round(t2-t, 5), 'Seconds to predict', n_predict,'labels with SVC')
```

I then used from `from sklearn.model_selection import GridSearchCV`.

```
svc.get_params()

C_parameter = np.logspace(-6, -1, 10)
tol_parameter=[0.0001,0.00001]

clf = GridSearchCV(estimator=svc, param_grid=dict(C=C_parameter,tol=tol_parameter))



clf.fit(X_train, y_train)
clf.best_score_

clf.best_estimator_
```

After that, I retrained the model and I got an accuracy close to 99%.

.

Optimized classifier ¶

```
print('Using:',orient,'orientations',pix_per_cell,
    'pixels per cell and', cell_per_block,'cells per block')
print('Feature vector length:', len(X_train[0]))
# Use a linear SVC
svc = LinearSVC(C=0.00016681005372000591,tol=0.0001)
# Check the training time for the SVC
t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
# Check the prediction time for a single sample
t=time.time()
```

```
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 8412
5.03 Seconds to train SVC...
Test Accuracy of SVC =  0.9896
```

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?

At the fourth point of `pipeline_P5_final.ipynb`, I implemented a sliding window search.

I found that a scale of 1.5, it gave me a good accuracy in the final video pipeline. I divided the video with different dimensions of the sliding window. In the bottom center and left (x=[0,780],y=[496,720]), I used a 128x128 window. In this part, cars should be bigger, so a big size window was the better choice. In the middle center (x=[500,780],y=[368,496]), I used a 96x96 window. In this part of the video, cars get smaller. In the right side of the video, I used a 64x64 window (x=[780,1280],y=[368,720]). Here, cars get even smaller.

In the middle left of the video(x=[0,500],y=[368,496]), I used a 128x128 window. In this part of the video, there shouldn't be any car. Using this dimension, I diminished the false positive of the cars in the opposite direction.

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 using HLS 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:

I optimized the classifier using GridSearchCV. It gave a better accuracy in the car recogntion.

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.)

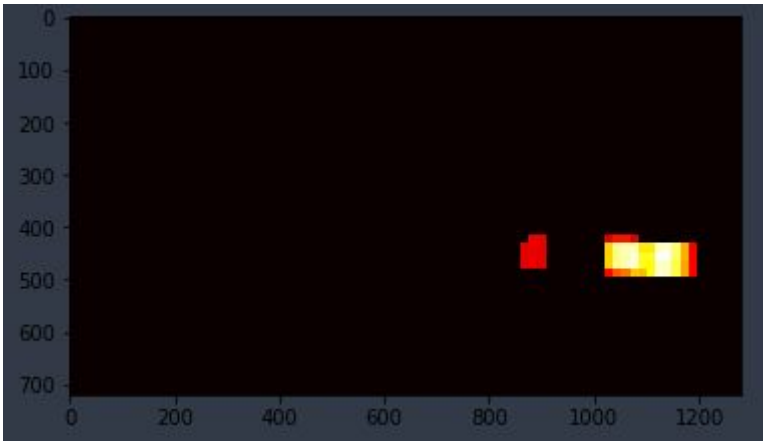
Here's a link to my video result

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.

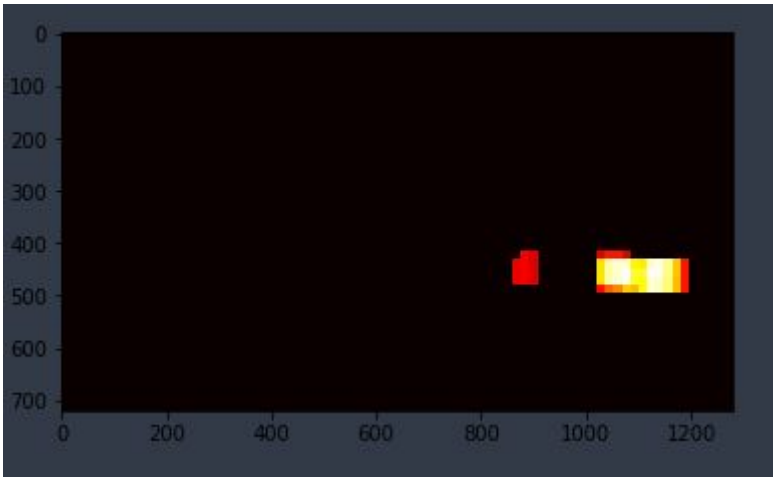
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

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:

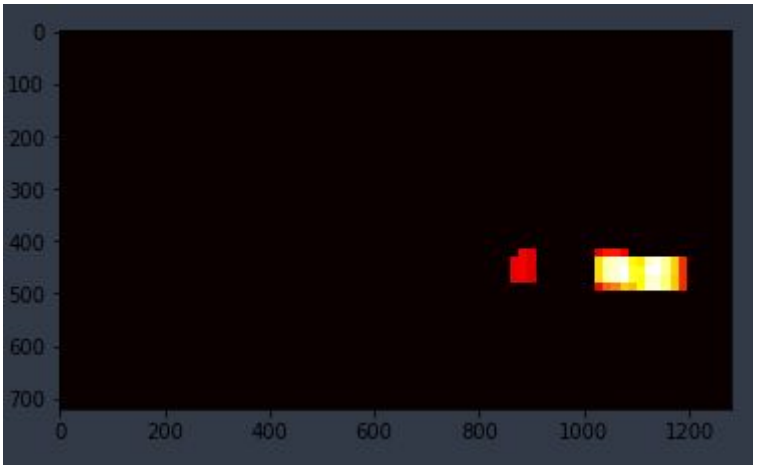
Here are four frames and their corresponding heatmaps:

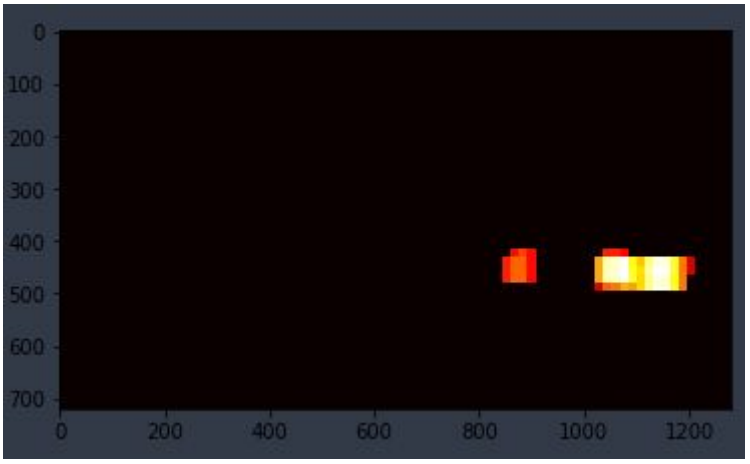



0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



Heatmaps helped to eliminate false positive and increase probability of high confidence detections. I set a threshold to 20 over 20 frames. So, random recognition would be filtered out. At the same time, overlapping detections gave more one point each. So, even if the car was not recognized in one frame, overlapping made possible the recognition.

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?

This was a challenging project. The training of the classifier was the initial part. As described before, I tried different combination and I opted for the HLS 3-HOG channels, with histogram and spatial features. It gave me the highest accuracy, so I think it's the best choice for this project.
The second part was the most challenging part of the project. I first tried the pipeline on some test images and it worked, pretty well. But then, when I had to implement the video pipeline, it was way more difficult. I first tried the pipeline I already used on

the images and I saw it worked poorly. There were a ton of false positive and, sometimes, cars were not recognized.
So, I implemented a heatmap over the last 20 frames and I set the threshold at different numbers. With higher numbers, false positive were gone, but car recognition was not continuous. With lower numbers, car recognition was ok over all the video, but a lot of false positive were not filtered out.
I started trying different color spaces, different combinations of HOG features, but nothing gave me good results. So, I had to come back to my initial approach using HLS 3-HOG channels, with histogram and spatial features.
I, then, tried to mix two different outputs (one using HLS and the other one using RGB). My objective was to get continuous car recognition and eliminate all the false positive using two different channels that gave me different heatmaps. I think the idea was not that bad, but I couldn't figure out a way to make it work properly. At the end, I had to come back to my initial approach and sort out a different soluton.
Watching other Udacity students on the web, I found that changing the dimension of the sliding window over different position of the frame, could improve the quality of the output.
Implementing this new approach, it really made a big difference and I obtained a final output where almost all false positives were filtered out and car recognition was consistent over the video.
If I had to restart the project, I would spend more time on this and, only after a satisfactory setup, I would improve the other parameters.

There are still a lot of possible improvements to this project:

- Having a training data with different weather condition would make car recognition possible even with rain or fog.
- I'm sure that mixing different color spaces would give a better out
- The boxes around cars could be averaged over some frames, to make it smoother.
- I could have improved the pipeline, by focusing research of the boxes, starting from the boxes of the last frame.
- If the car move to the right side of the road, sliding window should change and adapt to the new situation.