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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Write-up / README that includes all the rubric points and how you addressed each one. You can submit your write-up as markdown or pdf. Here is a template write-up for this project you can use as a guide and a starting point.

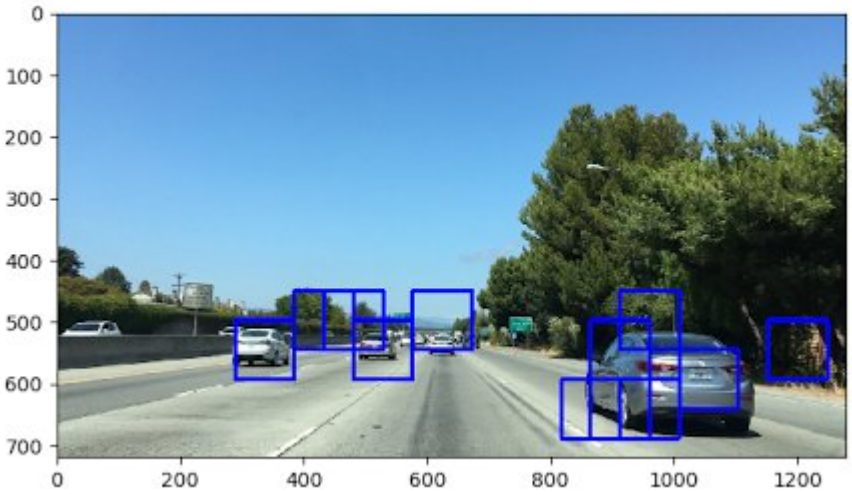
You're reading it!

Histogram of Oriented Gradients (HOG)

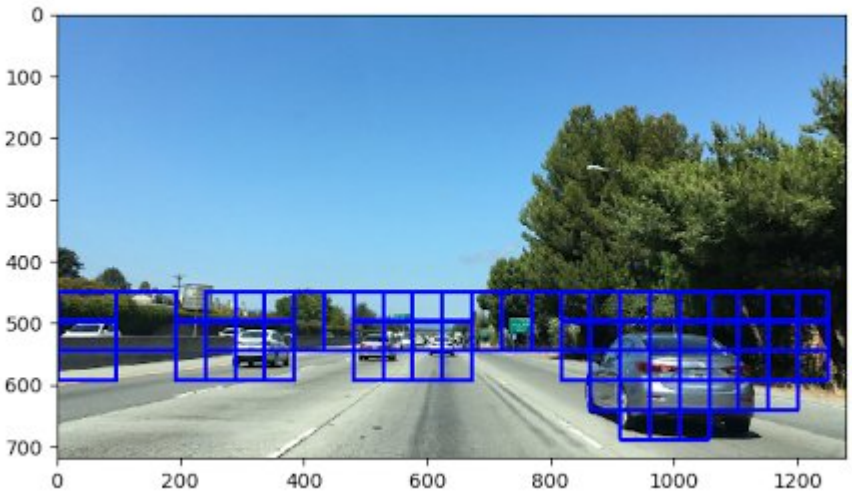
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the get_hog_features() function in the P5.ipynb notebook. I'm extracting separate HOG gradients of the given image's YUV channels. I used all channels since they yielded a richer information, furthermore I was using this color space, because empirical tests showed it's superiority in this case. Some example pictures of using different color spaces:

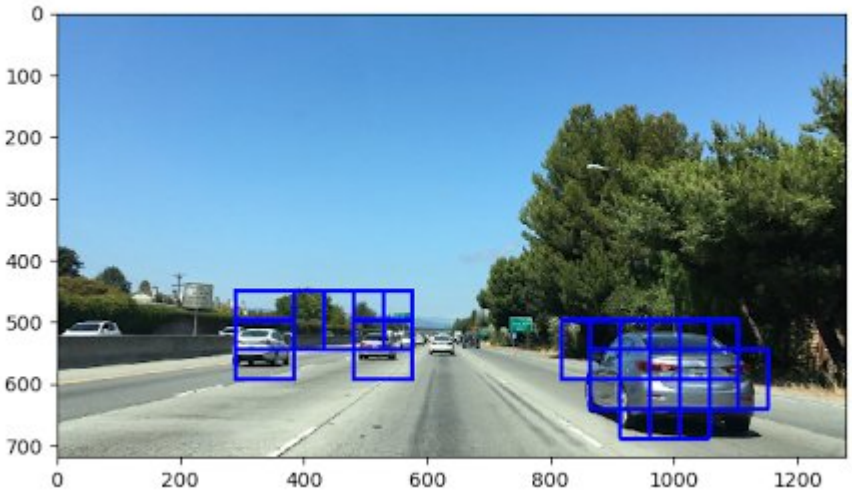
RGB:


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

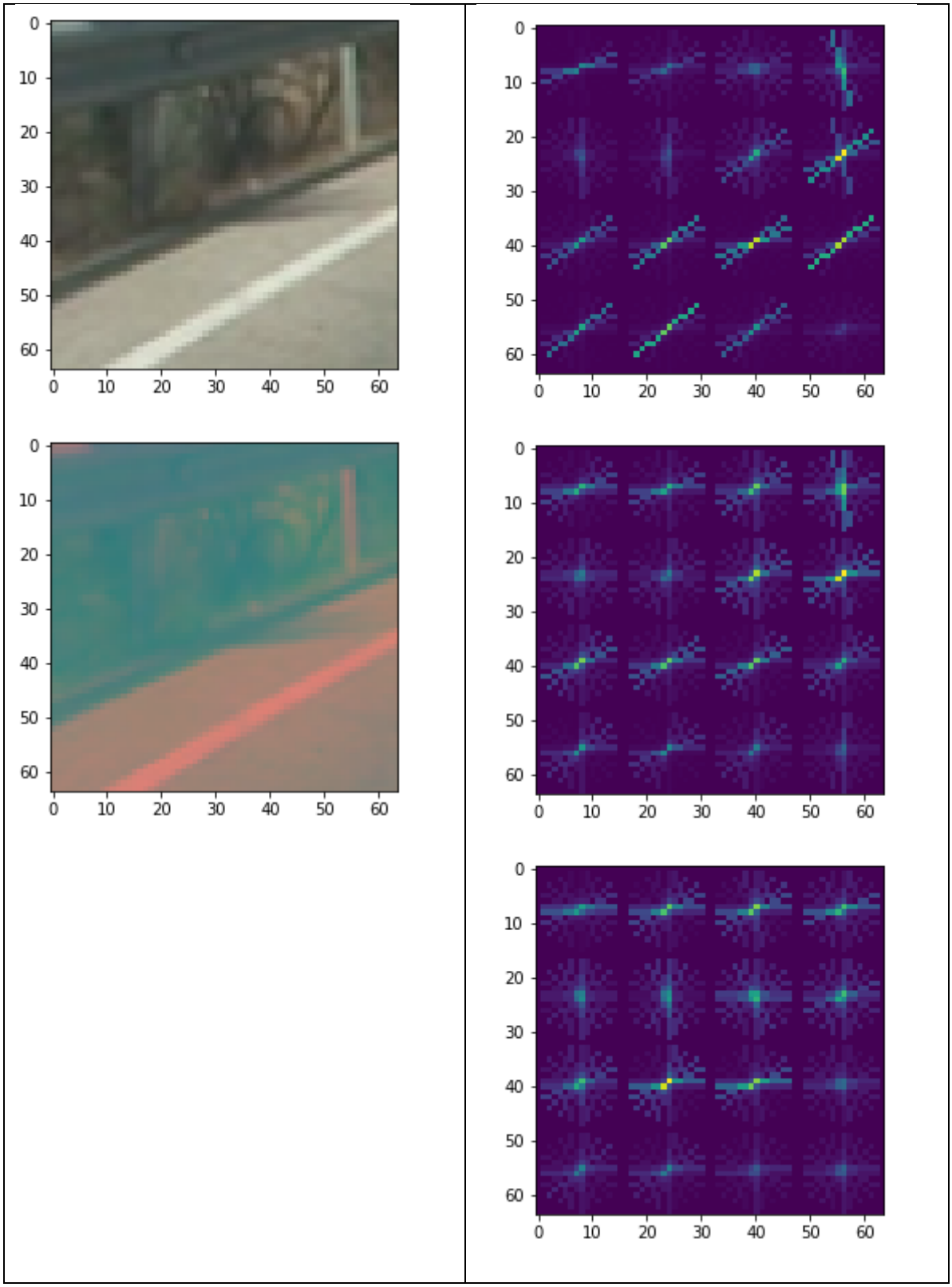
HLS:


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

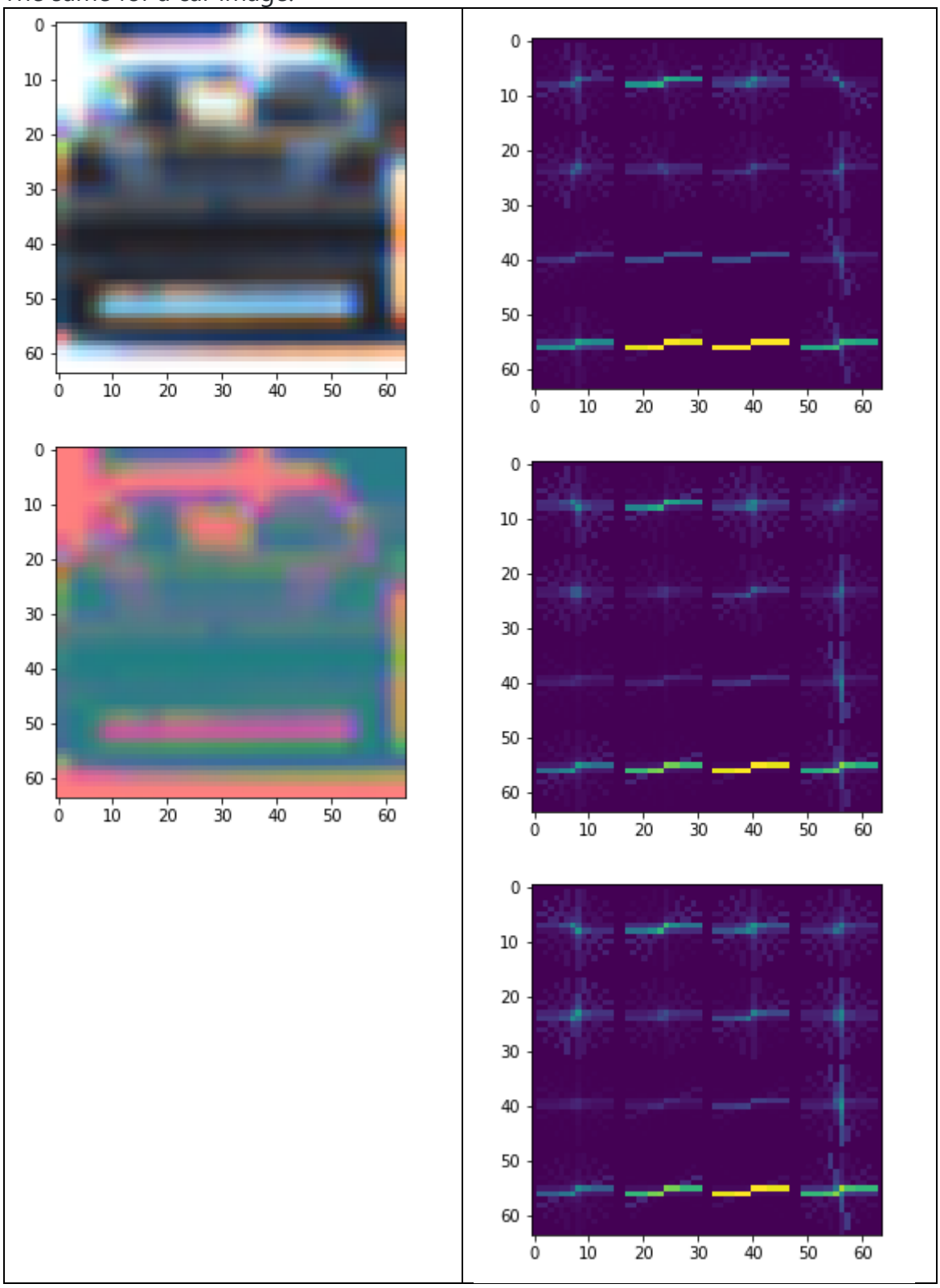
YUV:


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

When experimenting with color spaces I also experimented with orientations, pixel/cell and cell/block values. I came to the conclusion that 11/16/2 was working best for me.
RGB image and YUV image of non-car
…and its YUV component's corresponding HOG gradients:



The same for a car image:



2. Explain how you settled on your final choice of HOG parameters.
Actually, as mentioned before, I was experimenting with them. I choose the that parameter set, that yielded the best accuracy (`from sklearn.metrics import accuracy_score`).

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).

Training is done in section "Training/loading SVM" in the python notebook. I chose sklearn.svm. SVC using rbf with C=10. The reason for this was that I tested several options using GridSearchCV and this performed best.

```
gamma=auto
{'C': 10, 'kernel': 'rbf'}                    |   99.97% / 96.34%
{'C': 7, 'kernel': 'rbf'}                     |   99.94% / 96.28%
{'C': 13, 'kernel': 'rbf'}                    |   100.00% / 96.28%
{'C': 4, 'kernel': 'rbf'}                     |   99.76% / 96.22%
{'C': 16, 'kernel': 'rbf'}                    |   100.00% / 96.22%
{'C': 1, 'kernel': 'rbf'}                     |   99.17% / 95.93%
{'C': 1, 'kernel': 'linear'}                  |   100.00% / 95.69%
{'C': 4, 'kernel': 'linear'}                  |   100.00% / 95.69%
{'C': 7, 'kernel': 'linear'}                  |   100.00% / 95.69%
{'C': 10, 'kernel': 'linear'}                 |   100.00% / 95.69%
{'C': 13, 'kernel': 'linear'}                 |   100.00% / 95.69%
{'C': 16, 'kernel': 'linear'}                 |   100.00% / 95.69%
{'C': 16, 'kernel': 'poly'}                   |   99.50% / 95.16%
{'C': 13, 'kernel': 'poly'}                   |   99.38% / 94.92%
{'C': 10, 'kernel': 'poly'}                   |   99.17% / 94.86%
{'C': 7, 'kernel': 'poly'}                    |   98.73% / 94.75%
{'C': 1, 'kernel': 'poly'}                    |   97.61% / 94.16%
{'C': 4, 'kernel': 'poly'}                    |   98.08% / 93.98%
{'C': 1, 'kernel': 'sigmoid'}                 |   91.68% / 90.50%
{'C': 4, 'kernel': 'sigmoid'}                 |   87.87% / 86.19%
{'C': 7, 'kernel': 'sigmoid'}                 |   86.69% / 84.71%
```

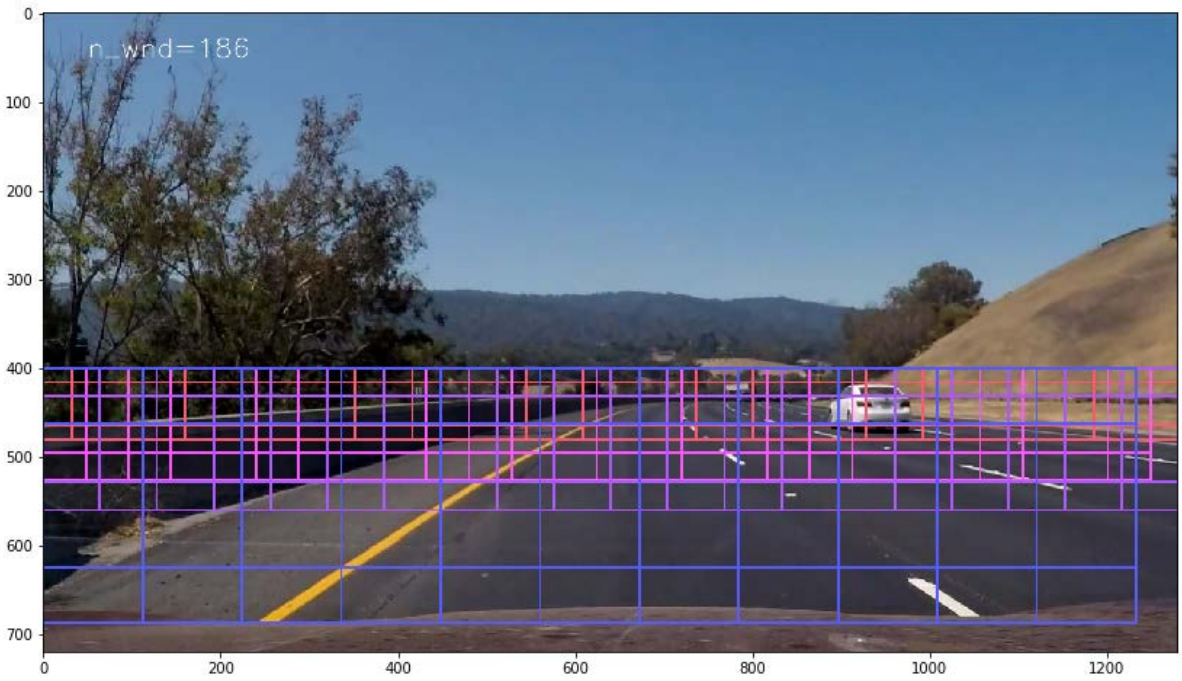
As of the HOG gradiens: I did not use spatial transformation neither color histogram. They did not seem to add much value to the SVM, and just made the handling more buly and slowed down processing the images.

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 covered four overlapping regions with different sizes of windows. They can be seen on the picture below. I thought it was important tu use bigger windows for the nearer objects meanwhile smaller ones for the further ones. Alltogether I scan every image using 186 windows. Overlapping sideways is always 50%.

The function is implemented in "Creating window templates" cell. Function name is slide_window().



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?

The classifier uses limited feature vectors in order to improve speed and reduceme memory use. Furthermore