Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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

You're reading it!

Source Code

There are four source files provided with this report:

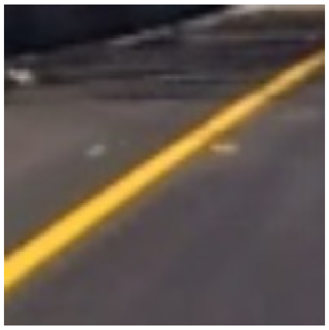
- `utils.py` : contains supplementary functions that were provided in the lectures
- `train_model.py` : contains the code that extracts features from the images and trains a classifier (Linear SVM)
- `test_model.py` : contains code that tests the classifier against the test images
- `video_pipeline.py` : contains the pipeline function that is executed for each window frame. For each image it extracts features, searches for cars within sliding windows and uses a threshold function in order to lower the false positive rate.

Histogram of Oriented Gradients (HOG)

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

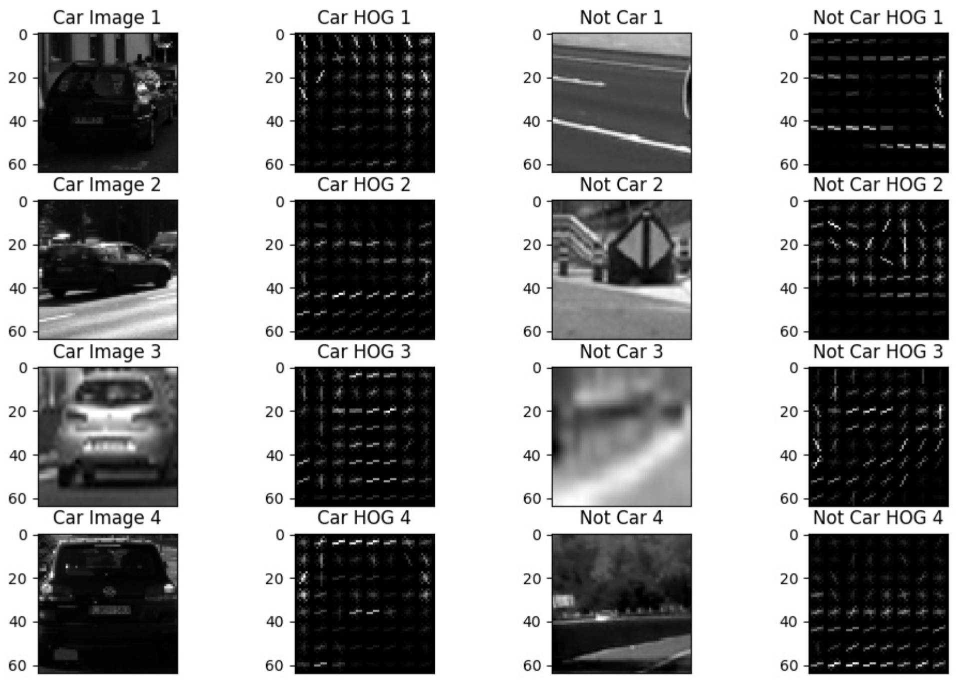
The code for hog feature extraction is contained in lines 31 through 48 of the file called `utils.py` in function `get_hog_features()`. In addition to HOG features, I also used spatial features (function `bin_spatial()`) and color histogram features (function `color_hist()`.

I started by reading in all the `vehicle` and `non-vehicle` images from the larger dataset. 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 `YCrCb` color space and HOG parameters of `orientations=9`, `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. I mainly varied the number of orientations from 8 to 12, the different color spaces and the number of HOG channels. I finally settled with 9 orientations in order to avoid having a huge feature space. Also, the 'YCrCb' color space gave better results in the initial version of the classifier.

The final selection of parameters was:

```
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
y_start_stop = [380, 670] # Min and max in y to search in slide_window()
```

These parameters gave a final vector length of 6108 and in the first pass on the classifier showed promising results.

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).

Before starting the training the data were scaled to zero mean and unit variance using `sklearn.preprocessing.StandardScaler()` (lines 52 to 56 in `train_model.py`).

I trained a `LinearSVC` using the default parameters as well as a `RandomForestClassifier` with 50 and 250 estimators and a `GradientBoostingClassifier` with the default values. The ensemble methods were slightly better in test accuracy as well as in the false positives but they took longer to train. The random forest with 250 estimators provided the best results but it was very slow during the prediction phase (~6 seconds per frame). For that reason I went back to the Linear SVM and tried to tune its parameters using Grid Search. I tried several values for parameter C (penalty term) and also used `fit_intercept=False` since all features were already scaled. The code for the Grid Search can be found in the `train_model.py` (lines 76 to 84).

The final parameters of the best estimator can be seen below:

```
LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=False,
intercept_scaling=1, loss='squared_hinge', max_iter=1000,
multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0)
```

The final test accuracy was 0.9916 and the table below shows the confusion matrix, where the false positives were reduced to just 18.

Confusion matrix	**Predicted True**	**Predicted False**
Actual True	1781	18
Actual False	12	1741

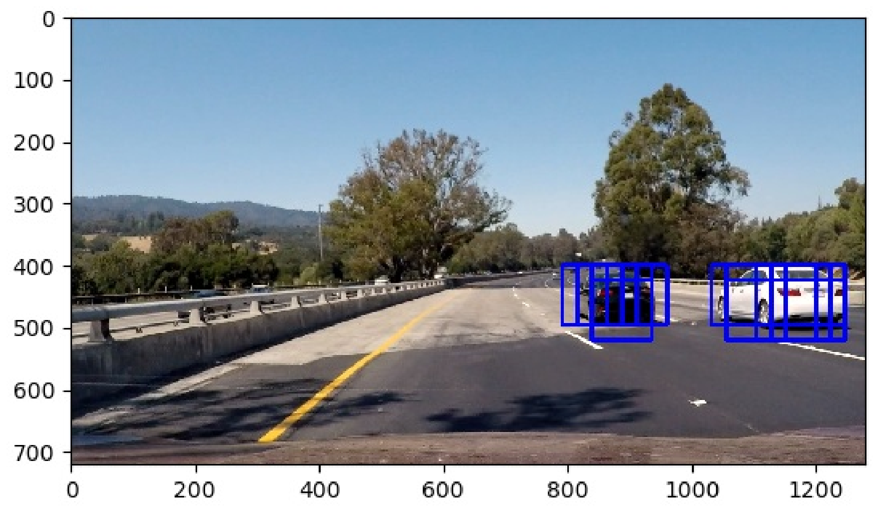
Sliding Window Search

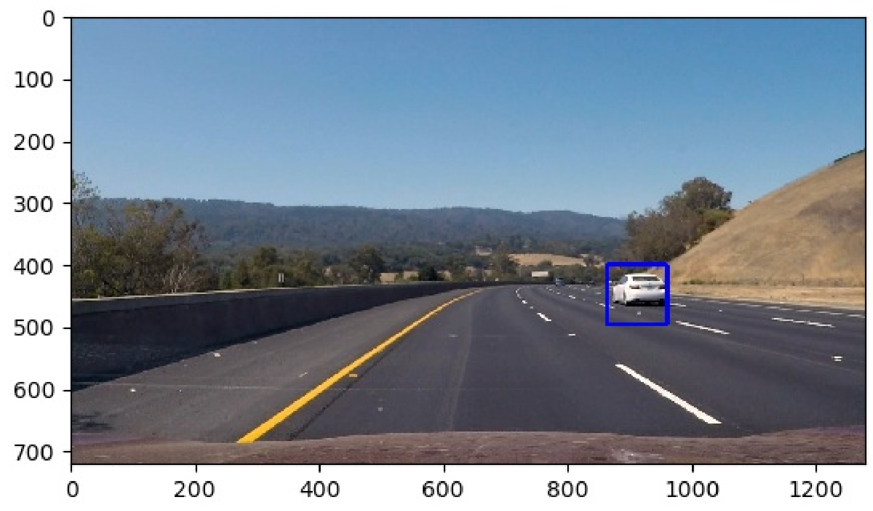
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

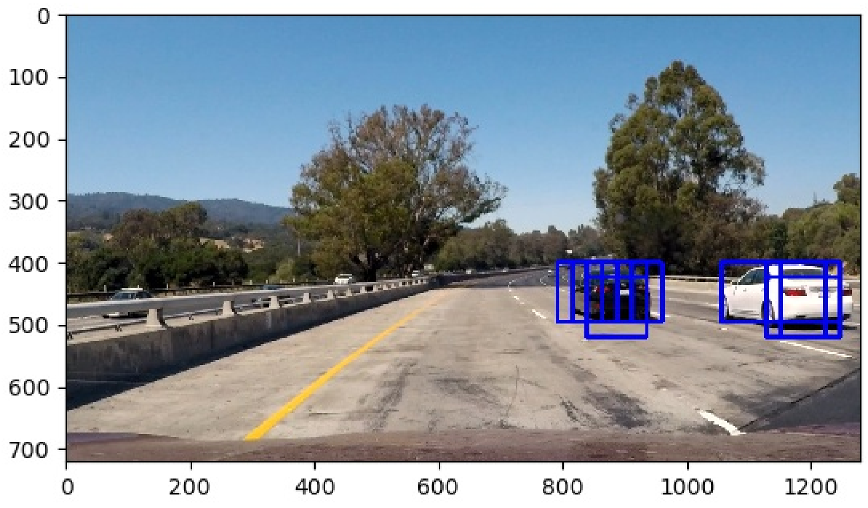
I initially used sliding windows of different sizes (64, 64), (96, 96) and (128, 128) with functions `slide_window()` and `search_windows()` within `utils.py` in order to generate sliding windows and perform predictions in each one of them. Overlapping was set to 1.5. In the end I used HOG subsampling in order to avoid calculating HOG features for each window and this way the pipeline became faster. I kept the window size to 64 and this worked out quite well. The code for the HOG subsampling which contains also the window search functions can be found in `utils.py`, lines 181 to 246 (function `find_cars()`).

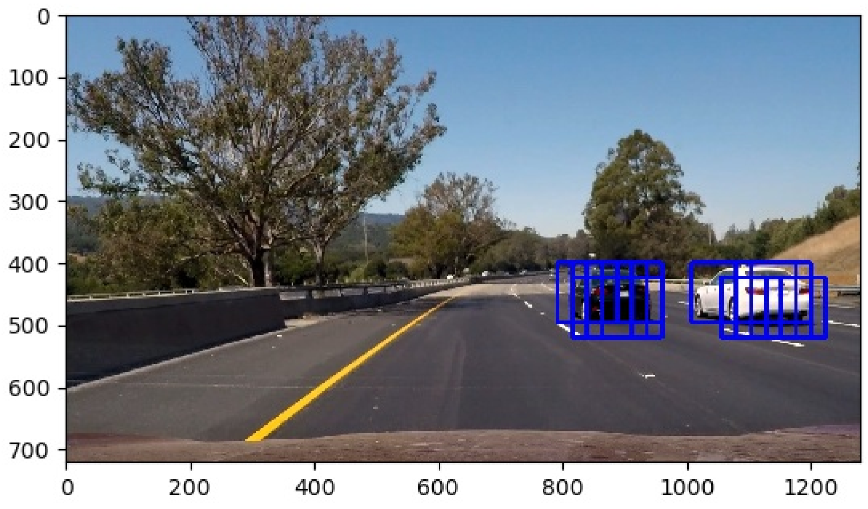
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on one scale using YCrCb 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:


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





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.)

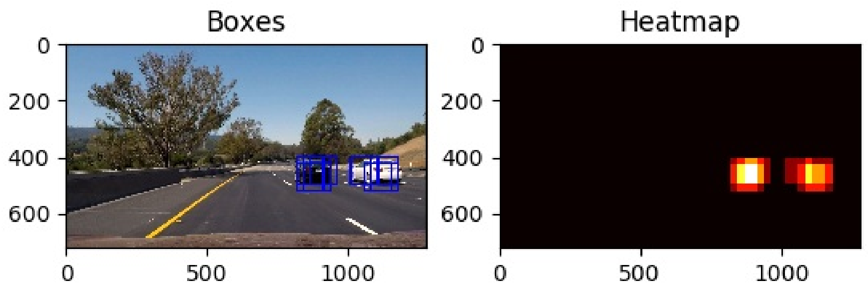
Here’s a [the video](#) which is also included as a file in the zip archive provided for the project (test_video_out_svc.mp4).

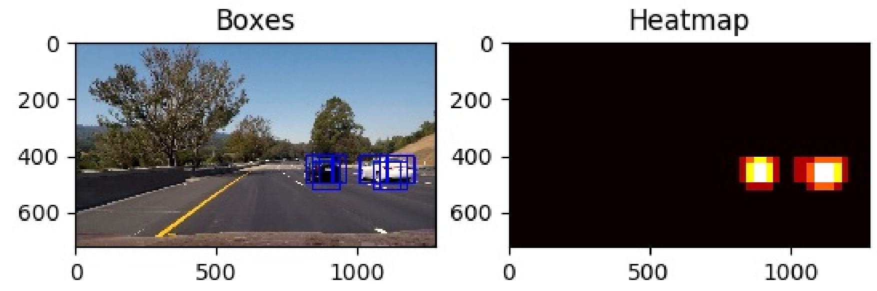
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

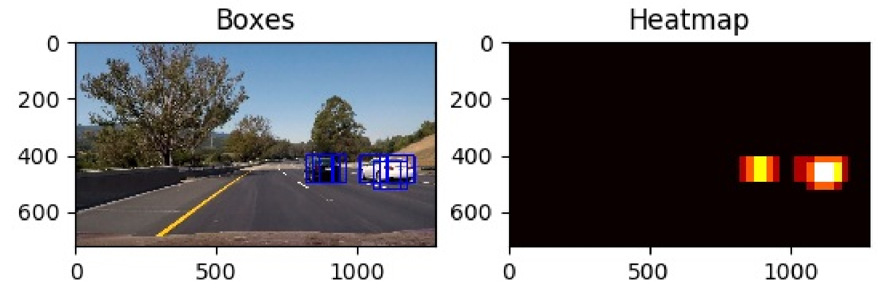
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap for each image and then stored it in a list that served as a queue. In order to avoid false positives I averaged the last six heatmaps and applied a threshold of 2 in order to identify vehicle positions and to avoid false positives that appear in individual frames. 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.
The code for this part can be found in `video_pipeline.py` inside the `process_image()` function.

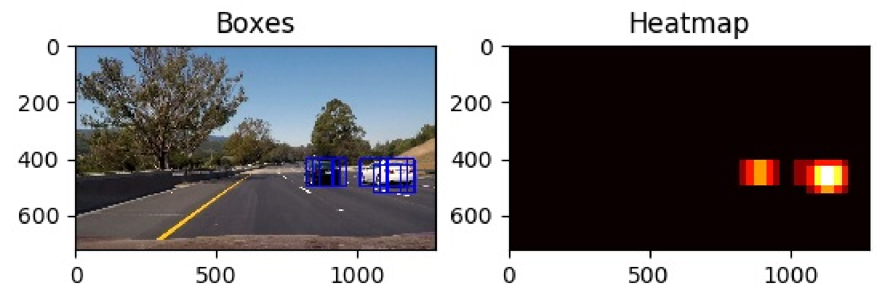
Here’s an example result showing the heatmap from a series of six frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video.

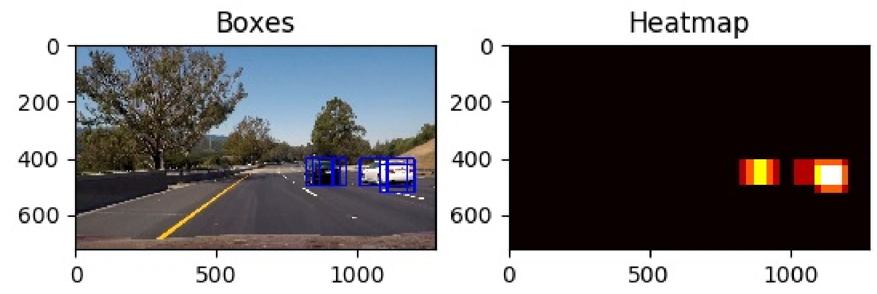
The six frames and their corresponding heatmaps:

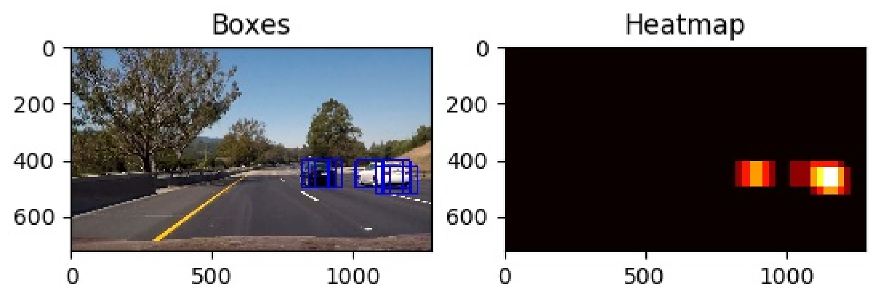



Boxes
Heatmap
0
200
400
600
0
500
1000

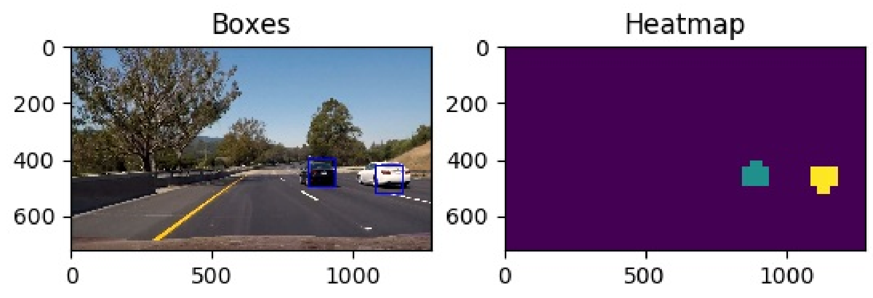

Boxes
Heatmap
0
200
400
600
0
500
1000


Boxes
Heatmap
0
200
400
600
0
500
1000


Boxes
Heatmap
0
200
400
600
0
500
1000



Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames and the corresponding bounding boxes onto the last frame in the series:



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?

I faced a number of issues that led to the various design decisions presented in this report. The initial problem was the limitation on the time it takes to process each video image frame, since this affected the type of classifier I used. The second major issue I faced was the number of false positives that were produced. I managed to minimize that in the final video by using a threshold in the overlapping boxes as well as the previous frames, but there is still some false positive frames (one or two). Another challenge is the lack of detection of cars under certain conditions. For example in some part of the road where the asphalt is white, the white car is not detected. Of course this happens momentarily, but it might be a problem in a real system.