Vehicle Detection

Vehicle Detection

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!

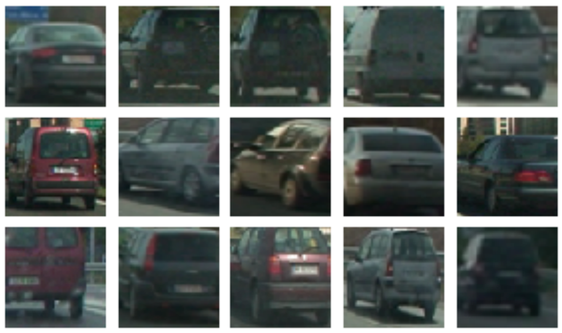
Histogram of Oriented Gradients (HOG)

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

Firstly, I extracted images of all images and non-images.

Here are some samples:

Vehicles

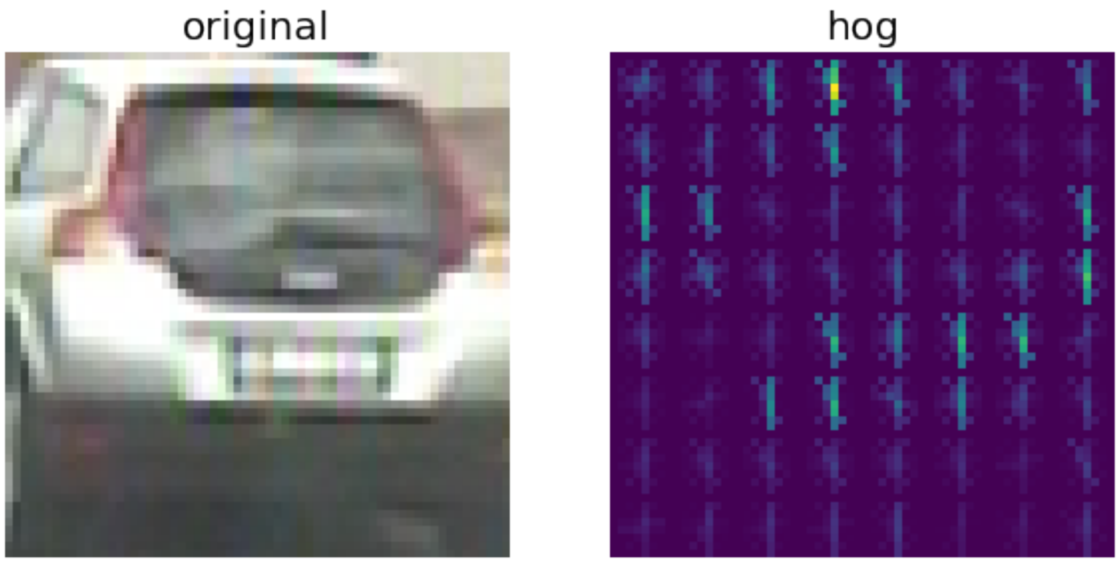


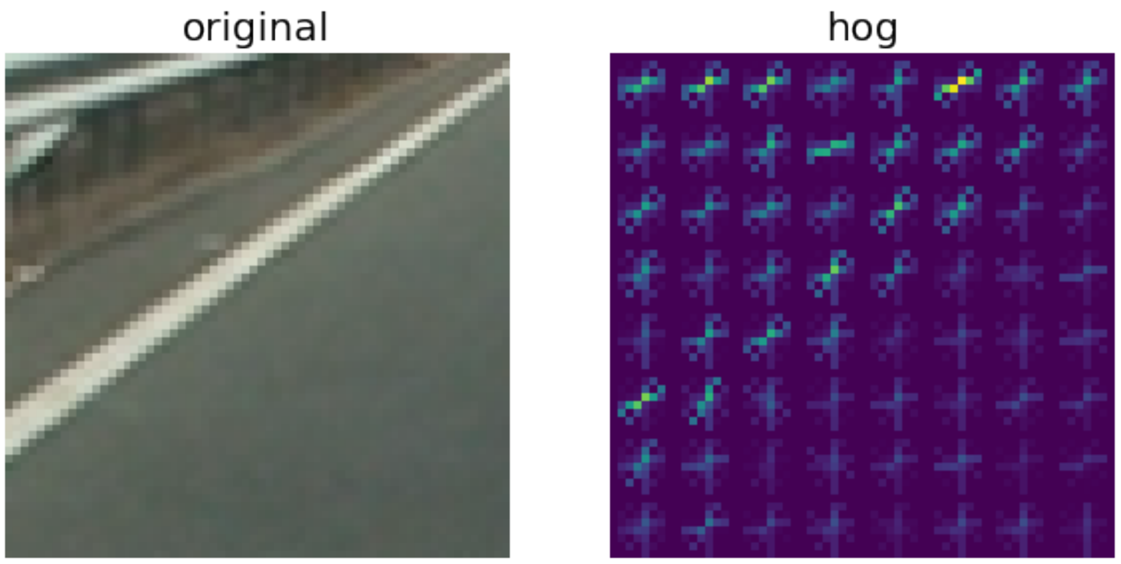
Not Vehicles

Next, I applied the chosen color space to recolor the images.

Finally, I applied the `skimage.feature.hog` method to extract HOG features.

Below is a sample image of the HOG features extracted using above parameters. For comparison, I have included both vehicle and non-vehicle sample.





One can clearly note the distinct shape of the outlines of the car. On the other hand, the image of the road has clear straight lines. These distinctions would serve as great training features for the learning algorithm.

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

Next, I experimented with color spaces and HOG parameters. I found that the following combination worked reasonably well:

- Color space : YCrCb
- `orientations` : 9
- `pixels_per_cell` : 8
- `cells_per_block` : 2

The learning algorithm's high validation accuracy of 98.7% helped me further validate that this set of parameters did not require more tweaking.

The code for extraction of HOG features is contained in `extract_features` function of `feature_extraction.py`.

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 chose Linear SVM as the learning algorithm because of its speed of training and easy of hyperparameter tuning.

The input to the linear SVM (trained on line 71 of `main.py`) using 3 types of features:

- HOG features
- Spatial features
- Histogram features

After concatenating the 3 types of features, each feature vector had 8460 features.

I also used the `sklearn.preprocessing.StandardScaler` to standardize (zero mean & unit variance) features before supplying it to the SVM.

To arrive on the optimal set of hyperparameters, I implemented a function (`find_best_hyperparams` on line 9 of `feature_extraction.py`) to check several C values in the range [0.07,2.6] and also both Hinge and Squared Hinge loss functions.

Finally, the SVM was trained using the following parameters:

- C : 0.07
- Loss function : Hinge
- Penalty : L2

Sliding Window Search

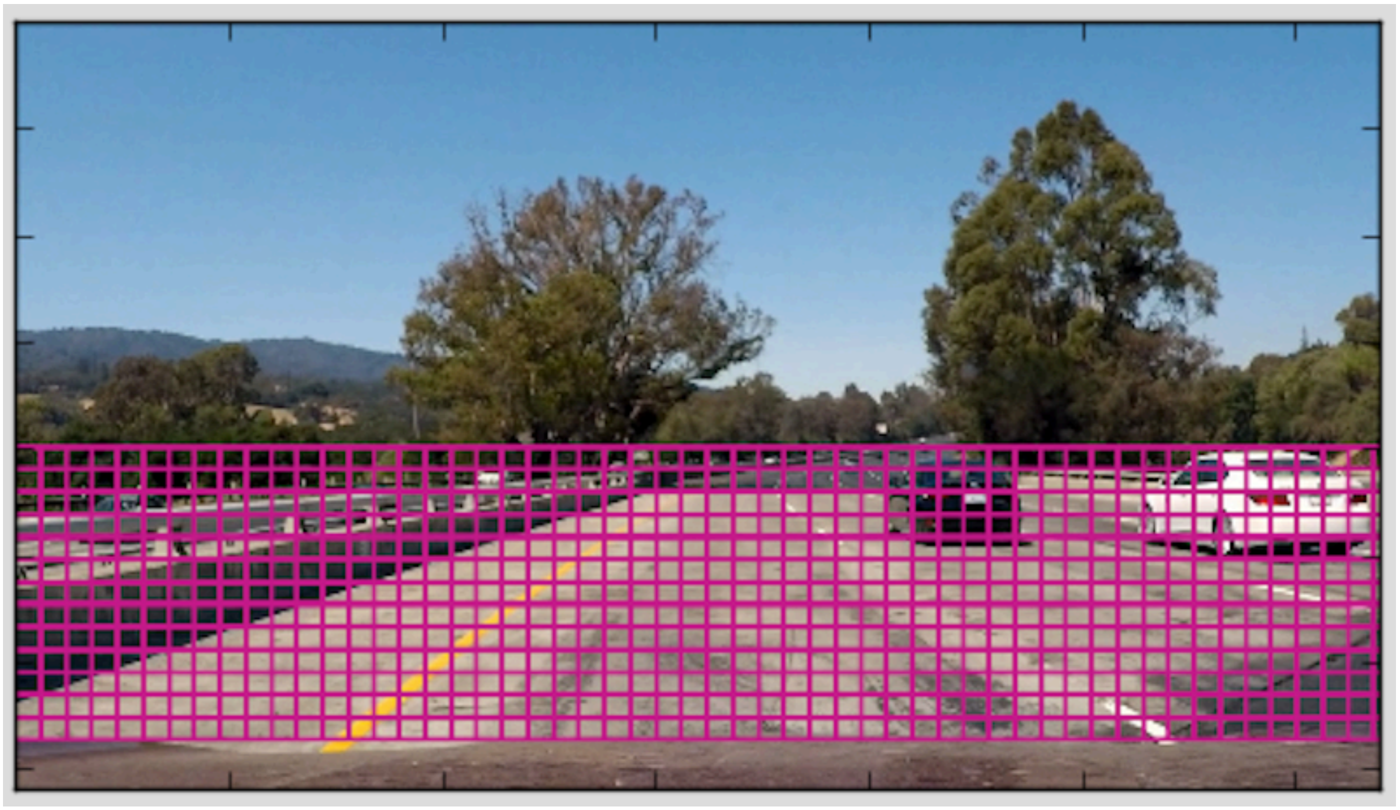
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

Using the sliding windows implementation in the lectures as a starting point, I experimented with a range of values that provided in a stable output.

I settled on using a fixed scale across the lower half of the image. The choice of lower half was a useful common-sense optimization and greatly helped reduce processing time.

1. Window size : 95 x 85 pixels
2. Overlap : 0.75 x 0.75
3. Vertical area of interest : 400 - 600

Here is a sample image visualizing the window size, overlap and area of interest.

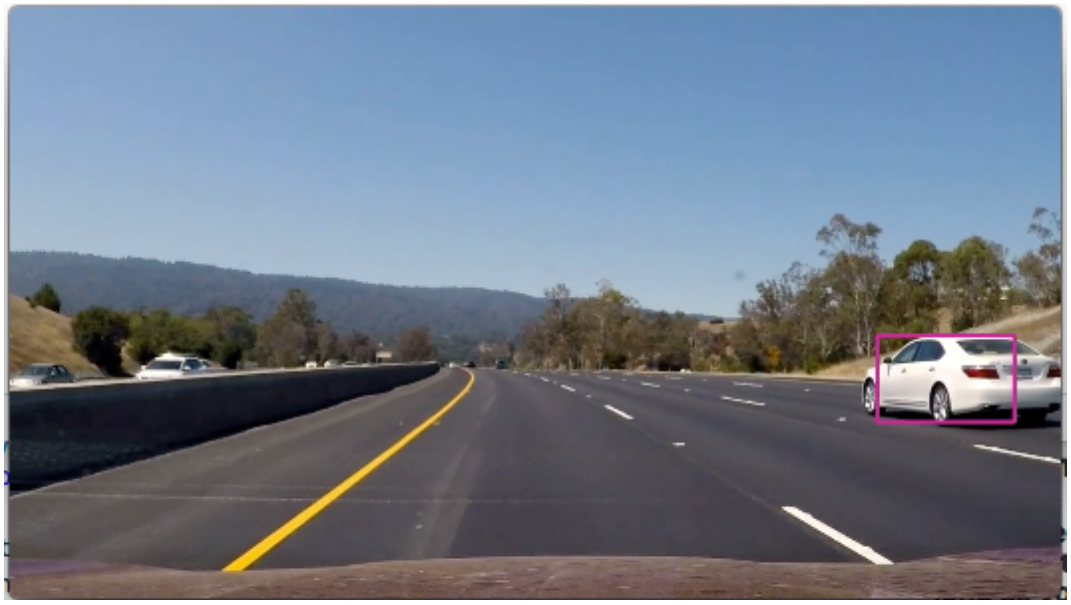


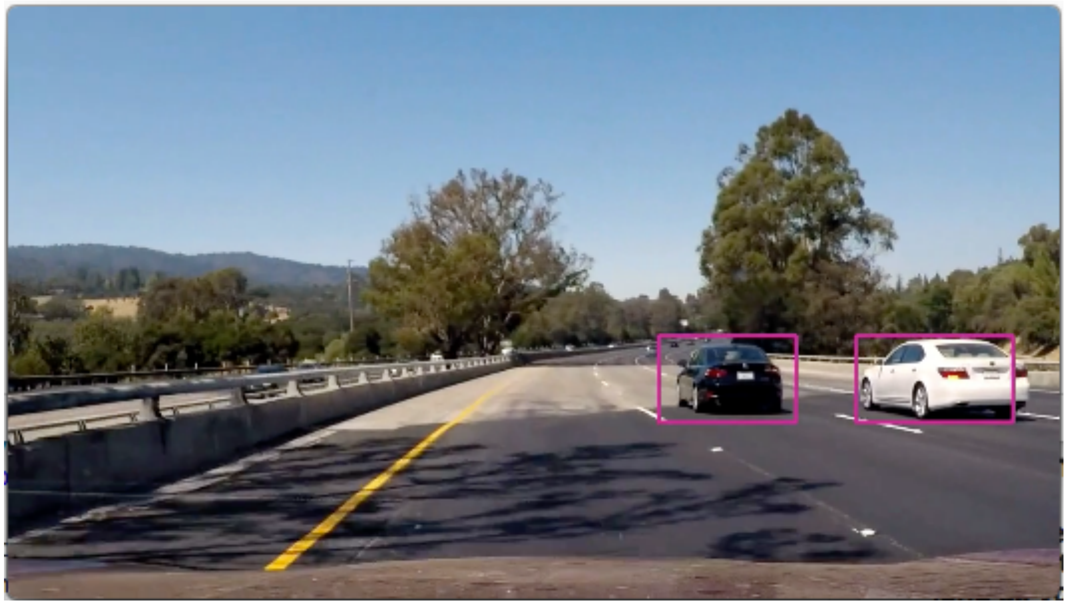
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Some of the optimizations:

1. Combination of HOG, Spatial and Histogram features to maximize the SVM's learning.
2. A constant scale of sliding windows (was faster than using variable scale).
3. SVM hyperparameter tuning (described earlier)
4. Duplicate detection

Sample images showing the pipeline working :





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 : https://github.com/suvir/SelfDrivingCar-VehicleDetection-P5/blob/master/output_project_video.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.

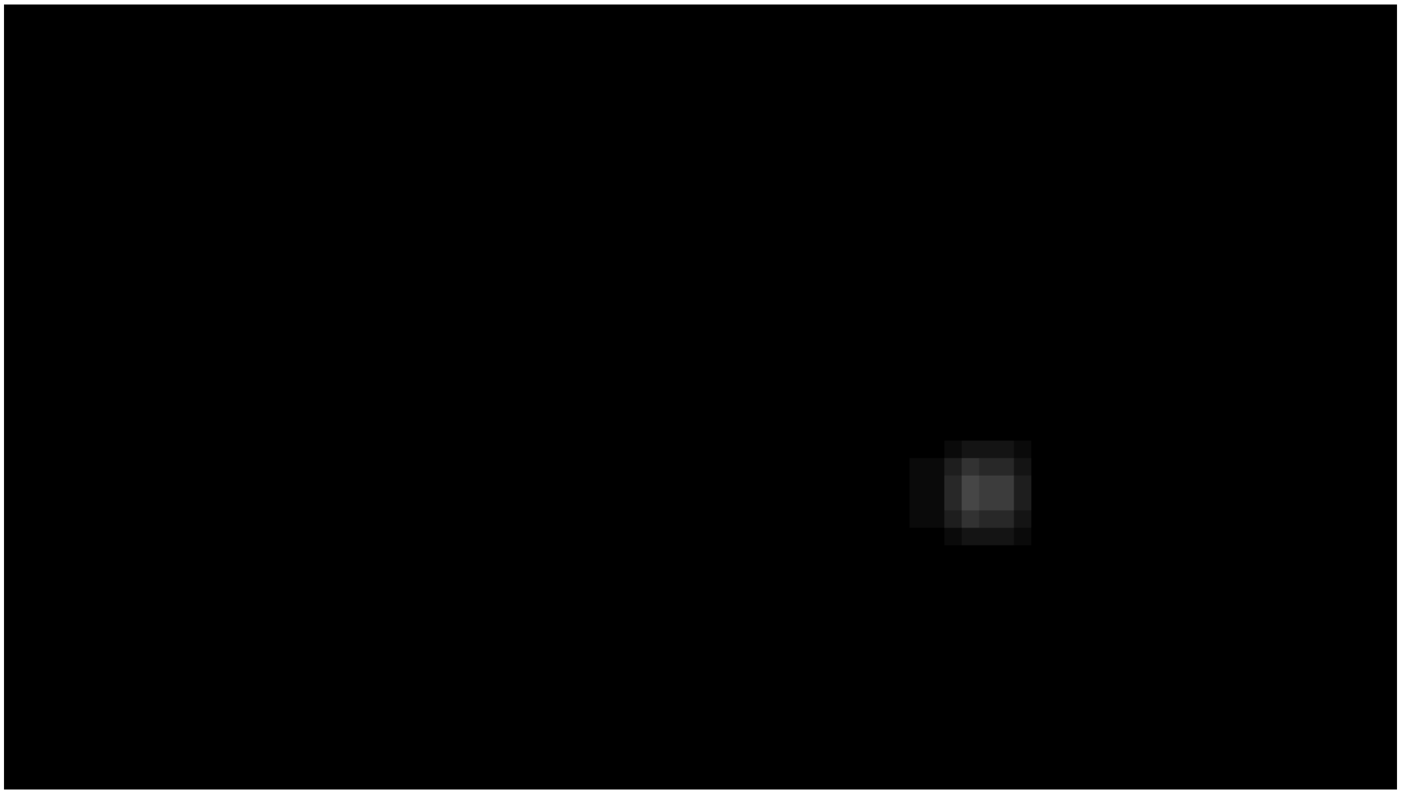
Combining the output of the learning algorithm with heat map detection (using `scipy.ndimage.measurements.label()`) helps provide a useful way to detect false positives.

The false positive implementation works like this :

1. Get output of possible vehicle position from SVM
2. Progressively increase the "heat" of some pixels where the vehicle is found (`increase_heat` method in `vehicle_frame_detection.py`).
3. Apply a threshold (Set to 15) to only keep areas with "hot pixels" (`apply_heatmap_threshold` method in `vehicle_frame_detection.py`) .

The class FrameManager in `FrameManager.py` maintains a queue of last 25 heat map frames. The thresholding function applies threshold to all these heatmaps. This is slightly more computationally intensive but results in smoother bounding boxes.

Below is a sample heatmap.



Discussion

Challenges/Further improvements:

- Performance of the pipeline : Computing the project video takes a long time. There is a lot of room for making the pipeline (almost) real time.

- Learning algorithm : Using a SVM was a great first step. However, having seen the success of deep learning algorithms, I believe that a CNN would improve the accuracy even more.
- Non-vehicle object detection : The pipeline can be augmented to include detection for traffic signs and other objects like pedestrians.