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. Factory

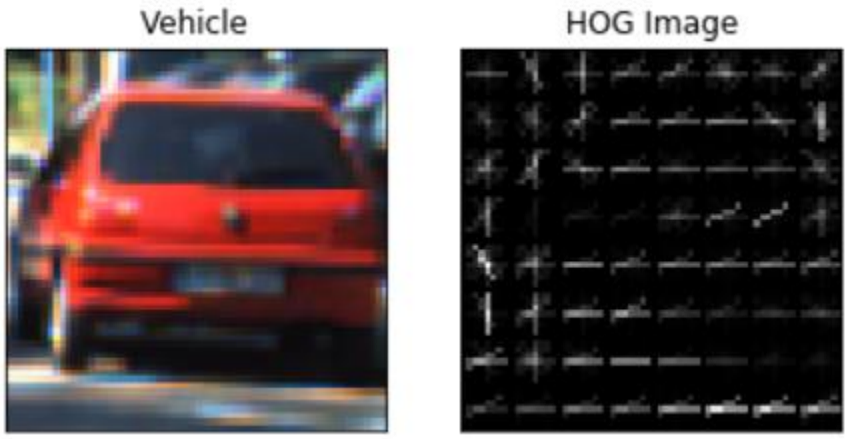
You're reading it!

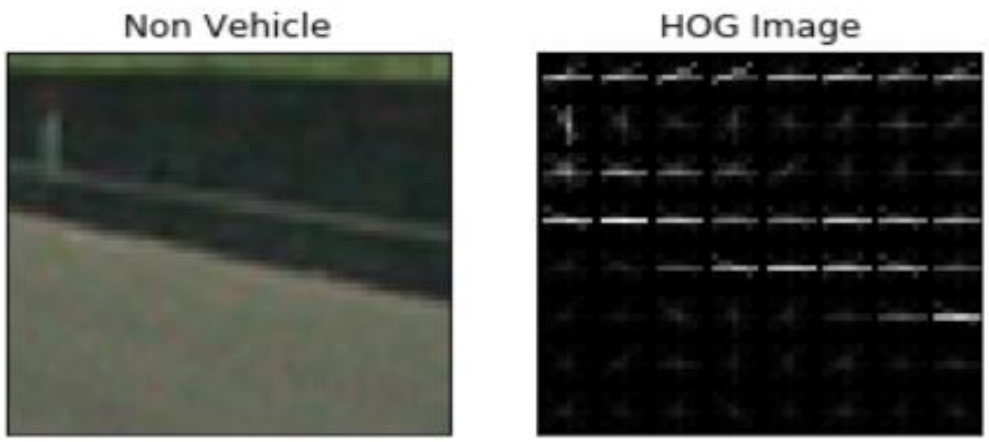
Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images. (Vehicle_Detection.ipynb –Histogram of Oriented Gradient)

- Used the OpenCV function **hogDescriptor()**for extracting Histogram of Oriented features from an image.
- Read all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:

- Picked two random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like.





- For each image, appended the HOG features to the dataset of training features.
- Built the datasets of car and non-car images and labelled it appropriately.
- I also normalized the features to have 0 mean and unit variance by using function StandardScaler (from Scikit-learn)
- Splitted training and testing datasets using 'train_test_split()' (from sklearn.model_selection).

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

I tested the performance of the HOG features from Lab, HSV, YUV, and LUV color spaces and found the **YUV** color space to have the best performance for identifying vehicles. Thus, I chose **YUV** space for feature extraction.

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).

In this steps I tested:

- Linear Support Vector Machine
- Logistic Regression Classifier
- Multi-layer Perceptron

In order to train the classifier to be used to identify vehicles in images. The performance evaluation of classifiers is as under:

Classifier	Training Accuracy	Test Accuracy	Prediction Time
LinearSVC	1.0	0.95411	0.00034
Logistic Regression	0.997	0.95889	0.00017
Multi-layer Perceptron	1.0	0.99183	0.00313

As per above results, I chose the **Multi-layer Perceptron** to identify vehicles in images and in the video stream.

- Between the **Logistic Regression Classifier** and **Multi-layer Perceptron**, **Multi-layer Perceptron**, I chose MLP classifier due to the increase in prediction accuracy.

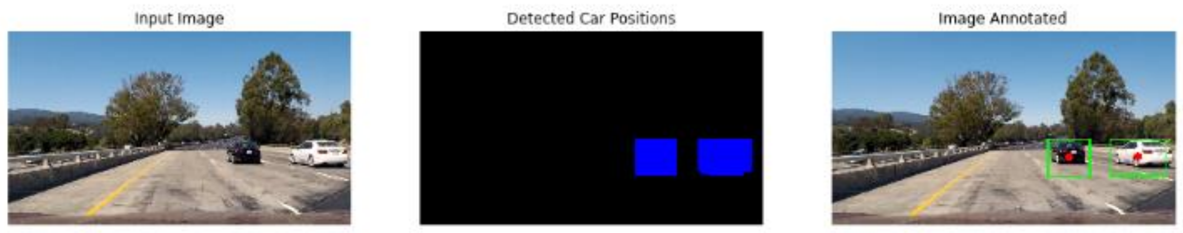
Sliding Window Search

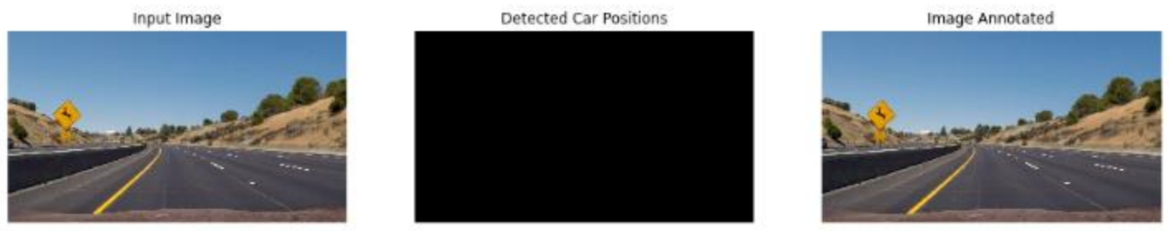
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

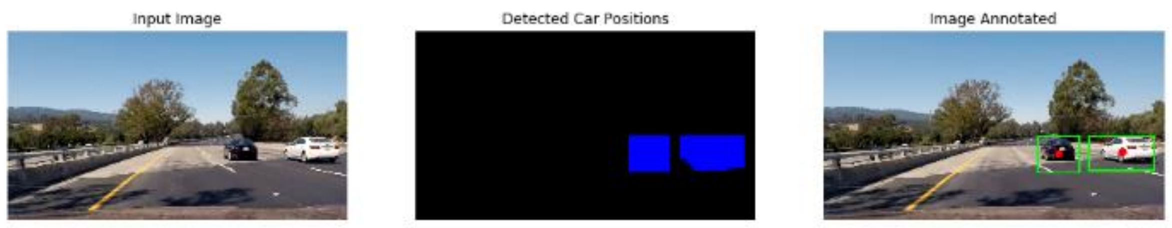
- In order to search vehicle(s) in images, I implemented sliding window approach by taking one slice of the image at a time
- Used the slide to make prediction on the HOG features.
- I searched for car(s) only in the lower half of the image, in order to minimize the search area and speed up the pipeline.
- Searches for vehicles in windows of multiple scales (with an 80% overlap), in order to identify vehicles which can be near or far in the image and thus will appear as different sizes.
- Used MLP classifier's predict_proba() fn., in order to ensure a high confidence for predictions, and used the probability score (picked score higher than 0.99) for each possible class.
- The window coordinates classified as vehicle are appended to a detected_windows[] list and finally calls fn. do_draw_boxes() to draw the boxes across the detected vehicles.

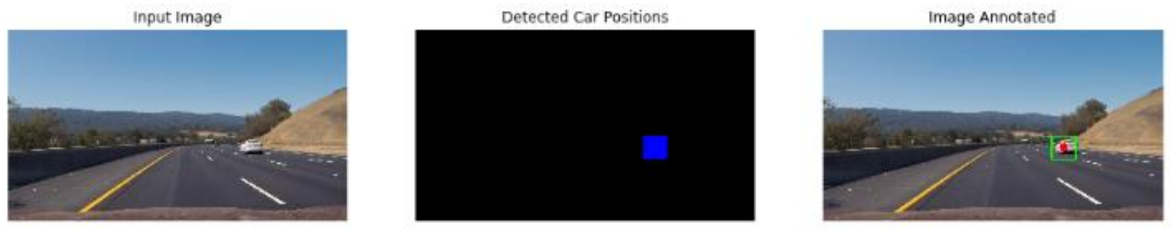
- cv2.findContours() was used to find all of the objects in the mask image to find contours
- cv2.boundingRect() is used on each contour to find the coordinates of the bounding rect for each vehicle.
- At the end, created a copy of the original image, to draw the bounding rectangles.

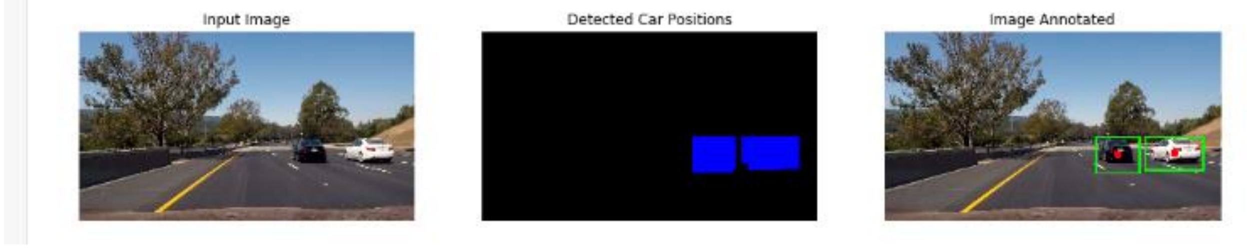
Example:











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?**

Explained above.

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.)

The video is attached with the submission zip folder.

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.

- In order to track vehicles across frames in a video stream, I created a class boxes to store all of the bounding rectangles from each of the previous 12 frames in a list.
- In each frame, I then combine the lists of bounding rectangles from current and previous frames, and then used the OpenCV function cv2.groupRectangles to combine overlapping rectangles in to consolidated bounding boxes.
- Within the group rectangles function I set the parameter groupThreshold equal to 10 which means that it will only look for places where there are greater than 10 overlapping boxes and it will ignore everything else.

- The group rectangles function takes care of the problem of false positives because if any part of the image is classified as a vehicle in fewer than 10 out of 12 consecutive frames, it will be filtered out and will not be included in the final bounding rectangles which are annotated on to the output video.

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?

- Most difficult part of this project was elimination of false positives for detected vehicle, as any wrong detection can wrongly influence the car to make driving decisions which are potentially hazardous.
- Current code pipeline, is still falsely identifying some non-vehicle objects such as trees and traffic signs. To improve the performance, it will be necessary to train on bigger dataset and add more negative features based on the false positives which were identified in the video stream.

Using OpenCV's HOG function from the skimage version greatly improved performance, but it is still not the best. Currently, my pipeline processes videos at about 4 frames per second, and to be useful in real time it would need to be closer to 25 or 30.

- It would be good idea to prune the number of features and reduce the number of windows searched in an attempt to speed up the prediction process.