P5 - 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

Training and testing sets

I used all the images given for vehicles and non-vehicles.
After loading the images, I checked that the number of vehicles and non vehicles images were roughly similar (around 8000) so I didn't need to do any data augmentation to compensate one of the two categories.

Instead of loading a full image, I decided to only put in memory the features for those images. In the following section, we will discuss the choice of the features.

The data is then shuffled and separated into a training set (80% of the total number of images) and a testing set (20% of the total images).

Features from the training images

For the data set, I tried several combinations of colors, and features chosen. I didn't want to drop the colors features as I realized during the class that they are important during the classification. However, using color features only didn't seem reliable so I wanted to add the HOG features.

I quickly realized that the best colorspace for the image was YCrCb. The second best option seemed to be the LUV colorspace with an accuracy around 90%).
For the HOG features extraction, I decided to use all the channels. When trying to run one channel at a time, the accuracy of the SVM was dropping by ~2 to 3%.

When choosing my parameters, I've been trying to keep in mind that the feature vector must be of a reasonable length as we don't want to input parameters that are useless but also because it takes more CPU utilization to train the machine learning algorithm.
I tried to increase the number of orientations for the HOG features from 9 to 16. The accuracy of the linear SVM didn't change much (98.8% with 9 orientations versus 99.2% for 16 orientations) and I didn't want to have a feature vector with too many features.
I also played with the number of bins for the color histogram. I settled on 32 as a reasonable number, I could have chosen 16 as the accuracy almost didn't change between 32 and 16.
I tried to keep as must information as possible so i chose to keep the hog features as well as the ones from the color histogram.

The most important of these features is obtained thanks to the HOG method. The HOG method (Histogram of Oriented Gradients) consists of taking the gradient for small windows in an image and extracting the general trend of gradient variation in this window.

Normalization of the data

Once we have an array of all the features for all the images, we need to normalize the data. I used the sklearn method StandardScaler to remove the mean and scale the variance. Once StandardScaler has been used on the training and testing set, we can eventually re-use the same scaler to normalize the images we want to classify.

Training the classifier

I chose to use a SVM (support vector machine) for my machine learning algorithm.
I tried several parameters:

- Rbf kernel: I couldn't get an accuracy greater than 51%
    - Even changing the C and gamma parameters didn't seem to help
    - The rbf kernel was taking a lot more time to train than the linear kernel
- Linear kernel: good accuracy (>80%)
    - The best C parameter I got was C=1. I tried C=0.1 and C = 10
    - I also tried to run the GirdSearchCV to optimize the linear SVM but it seemed to be taking a lot of time and freezing the ipython kernel. The code is commented out in the notebook.

The best accuracy I got was **98.87 %**. I was able to improve this accuracy by adding more fields to the feature vector but I mentioned previously, I wanted to keep it to a reasonable size.

Sliding window

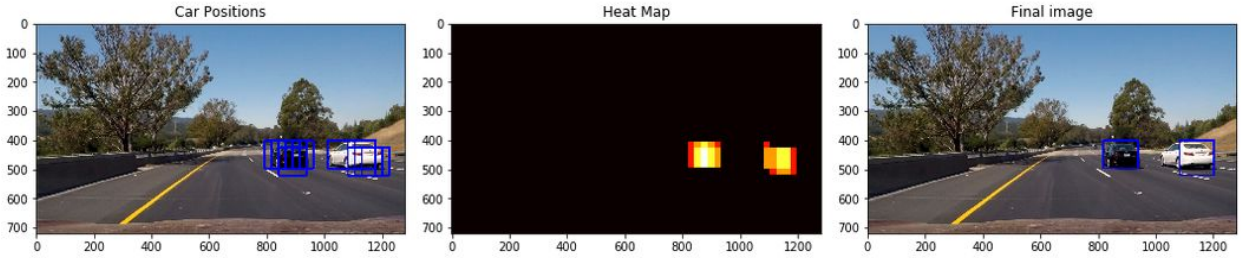
Once an image is extracted from the video, the HOG features for this entire image are extracted. The scales chosen define the size of the sliding window. For each window detected, the color features are computed, normalized, aggregated with the previously computed HOG features and sent to the SVM to check if there is a car or not.
Having different sizes for the windows make the detection more accurate as cars that are farther away will appear smaller and cars that are close will be bigger.

Improving reliability of the classifier

I decided to remove the detections (even if they were not false positives) if they were too much on the left. The reason behind this choice if that we are driving in the left most lane and we shouldn't be concerned about incoming traffic on the highway.
The following capture shows a true positive for the traffic on the other side of the road that was removed by checking that the maximum x value a a box is greater than 150 pixels.

I also created a way of tracking the number of cars in the video and making sure that if we lose a car, we search with different scaling parameters and thresholds.

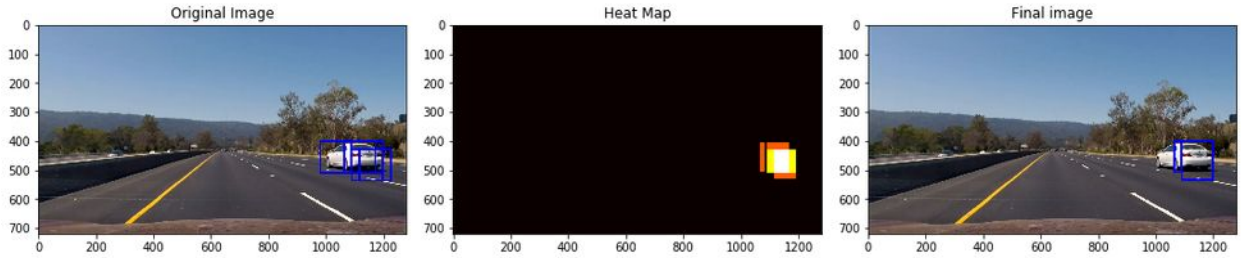
Let me explain in details how it works:
Start with a search that is likely not going to give any false positive. The threshold I chose was 1 and the scales were 1 and 1.7.
These parameters gave me fairly good results, most of the time. However, sometimes the threshold needed such that if there is only 1 window detecting the car it would not consider it as a false positive. However, as there is a high risk of adding false positives to the detections, any of the boxes found must be in a certain radius of the center of the previously detected boxes for this car.
Also, in the background a counter is making sure that we wouldn't look for this car for the entire video if we can't see it anymore. This ensures that if a car disappears from the field of view, we don't keep tracking it.
I also added another search if the previous one was still failing. This time the new search has a different scaling parameter and tends to pick up a lot more false positives. Again the radius method was used to make sure we only pick up the car that went missing in the detection.



Other improvements

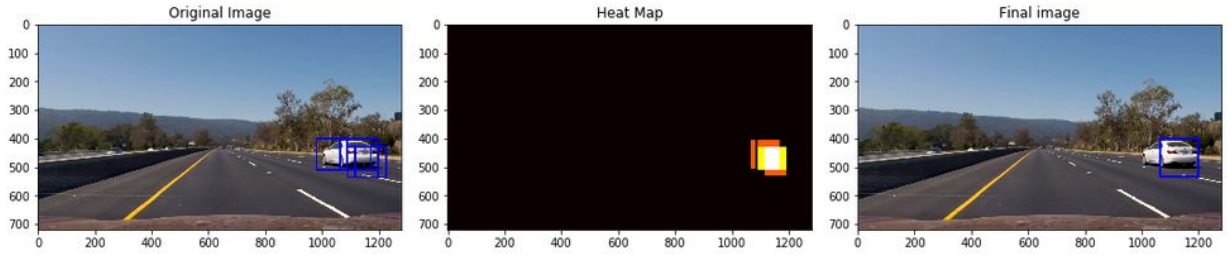
Aggregate the boxes for a same car:

I didn't like that sometimes I had 2 boxes for the same cars, the boxes being very close to one another. You can see in the following image the raw output from the sliding window and classification, the corresponding heatmap and the final boxes. In this case you can notice that the 2 boxes are for the same car.



I decided to implement a function to check that the center of 2 boxes is farther away than a threshold (chosen as 80 pixels in the code).
The functions takes in the list of the boxes found. It transforms that into a list of permutations of 2 boxes, computes the center of the 2 boxes and checks that it is greater than the threshold. If it is not, a new box is created that encompases the 2 boxes.

The final result can be seen in the following screenshot:



Smoothing of the detection over several frames:

In order to give a better visual result for the detection, I decided to smooth the detection over the previous 10 frames of the videos.

Discussion

I would be curious to compare the accuracy and the numbers of false positives between an SVM and a deep neural network. For instance how many more (or less) images would have been needed. I was also wondering if there is a way of combining both SVM and deep neural net.
Running an SVM seems easier, computationally speaking, than running a deep neural net. We could try to have an SVM go through the windows and if a car was detected confirm by running a more accurate neural network.

I think more attention could have been paid to reduce to the strict minimum the number of features needed to be able to run it ‘live’ on a car.

The smoothing could have been improved by tracking the trajectory of the car. Adding this parameter would have helped understanding if a car was getting out of the field of view and thus if we needed to try to find it in our image or not.