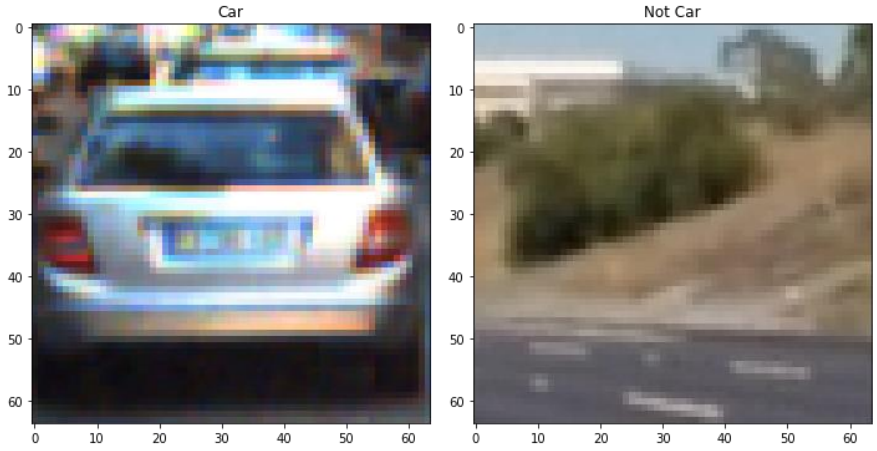
Vehicle Detection Project

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
- Apply a color transform and append binned color features, as well as histograms of color, to the HOG feature vector.
- Normalize the features and randomize a selection for training and testing.
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- Run the 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 Data

I used all images provided for the project to train and test my classifier. I imported a total number of 8968 not-car and 8792 car images. Here is an example of one of each of the "Car" and "Not-Car" classes:



Feature Extraction

In this project, I decided to use all HOG, binned, and color histogram features to train the classifier. The code for this step is contained in the section 1.1 of the IPython notebook.

In order to find the optimal features, I started with the default parameters provided in the course and trained the support vector machine classifier using the extracted features. I used the test accuracy as the metrics to evaluate the quality of my features. I tried this procedure for several times until I got a satisfactory prediction accuracy. The final values I used for feature extraction parameters are:

Spatial binned features:
spatial_size = (32, 32)

Color histogram features:
histogram bins = 32

HOG features:
Orientations = 9, pixels per cell = 8, cells per block = 2, color channels = ALL (RGB)

Using these parameters, I obtained a feature vector with the length of 8460. The final accuracy of my classifier for the test data was 0.9848.

It's worth mentioning that, we could have gotten the same or even higher accuracy with not using spatial and color histogram features, which would essentially increase the speed and efficiency of the prediction as well.

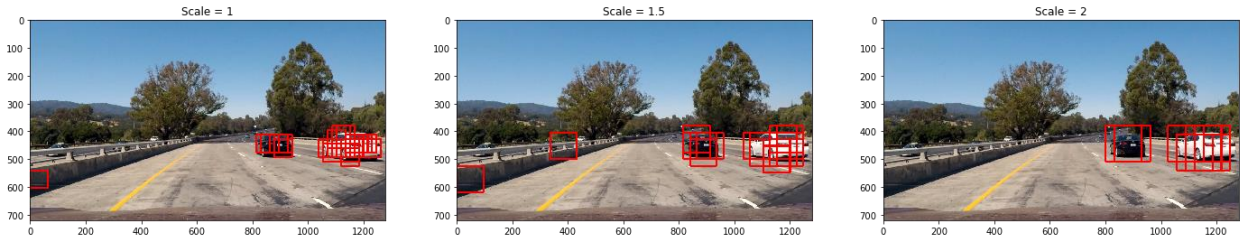
Classifier

In order to avoid overfitting, I decided to use a simple linear support vector machine for my classifier. The code for this part is provided in the section 1.3 of the notebook. To train the classifier, I first extracted my features and scaled them using sklearn.preprocessing.StandardScaler(). Next, I split the data into training and test sets using sklearn.model_selection.train_test_split() function.

I defined a linear SVM model using sklearn.svm.LinearSVC() and trained the model using the extracted features and labels. The final test accuracy obtained from the model was 0.9848.

Sliding Window Search

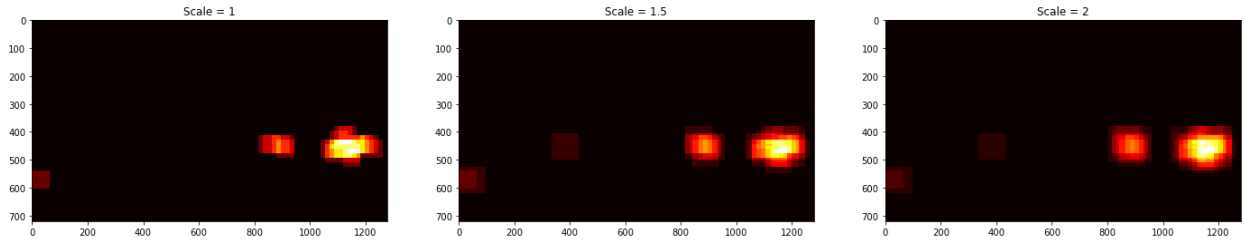
I implemented a sliding window technique to search for the cars on a specific region of the road image. The size of the cars on the road would depend on their distances from the camera. Therefore, we need to perform this search with multiple scales in order to be able to detect all cars with various sizes in the image. The function that I wrote for this search uses a default value of 64x64 pixels for the sliding window. Using a scaling parameter we can change the size of the window and use the same function to perform window searching for various scales. After some playing around, I found that by using three scale values of 1, 1.5 and 2, I can optimally identify cars of various sizes on the road with minimum number of false positives. Picture below shows an example of window searching for the three scales.



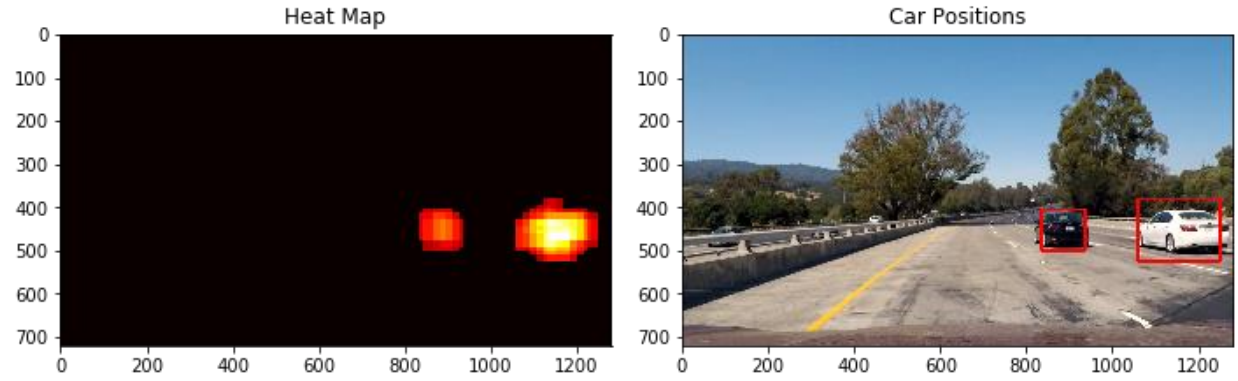
Multiple Detections & False Positives

The pictures above show all the bounding boxes for where the classifier reported positive detections using each sliding window size. It can be seen that overlapping detections exist for each of the two vehicles, and in the first two pictures, I find a false positive detection on the guardrail to the left. In order to combine overlapping detections and remove false positives, I built a heat-map from these detected boxes.

To make a heat-map, we're adding "heat" (+=1) for all pixels within windows where a positive detection is reported by the classifier. The heat-map of the three picture above is shown in the following.



If the classifier is working well, then the "hot" parts of the map are where the cars are, and by imposing a threshold, we can reject areas affected by false positives. So I wrote a function to threshold the heat-map. In practice, I add all boxes detected using each window scale to a python list called box_list, create the heat-map from this list and apply the threshold to this heat-map. I can then draw boxes around the hot thresholded areas to determine the position of the cars. The result is shown below for the threshold value of 4.

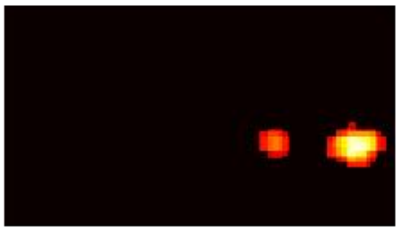


As can be seen, the false positives on the left guiderail have been removed after thresholding, and only the regions with the number of four or more overlapping detections have remained in the heat-map, which are the true locations of the cars.

Image Processing Pipeline

I wrote a function named detect_cars() to implement the entire pipeline described above on each test image. The function takes an image, runs the classifier on image patches using a sliding window with three scales, creates a list of detected boxes, produces a heat-map of the detected areas and thresholds the heat-map to remove the false positives and combine multiple overlapping detections. The function returns the thresholded heat-map and the original image with boxes drawn around the detected cars on the road. The results of applying this function to some test road images have been shown below.

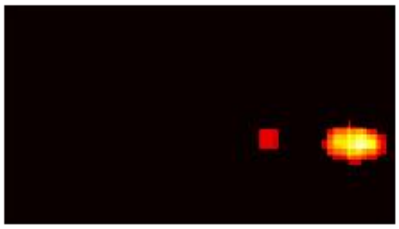
1



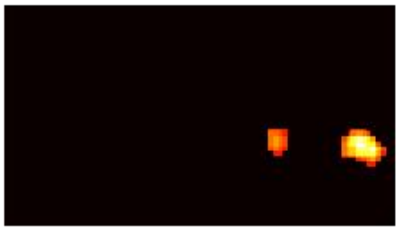
2

3

4



5



6

Creating the Video

I used the pipeline function to process the frames of the videos and detect car at each frame. The output videos for both “test_video.mp4” and “project_video.mp4” are available in the project folder with names “test_video_output.mp4” and “project_video_output.mp4”, respectively.

Discussion

The output videos show that the developed pipeline is performing very well. There are a few false positives in some frames which appear and disappear very quickly. In order to improve the robustness of the pipeline even further, I would suggest playing around with some parameters including, feature extraction parameters, the number and values of sliding window scales, and the threshold value for the heat-map.

I used RGB channels to extract HOG features in this project. The extracted features showed a good performance, and therefore, I did not try using other color channels. Extracting HOG features from other color channels would potentially increase the prediction accuracy even further.

In addition, I used a linear CVM model as the classifier. Playing around with other kernel choices and tweaking parameters such as Gamma and C, could potentially lead to a better prediction. One way to obtain the optimal CVM parameters is to use *exhaustive grid search* or *randomized parameter optimization* tools from sklearn.