Udacity: CarND-Vehicle-Detection

Author: Bugra Turan

Vehicle Detection Project

The goals / steps of this project are the following:

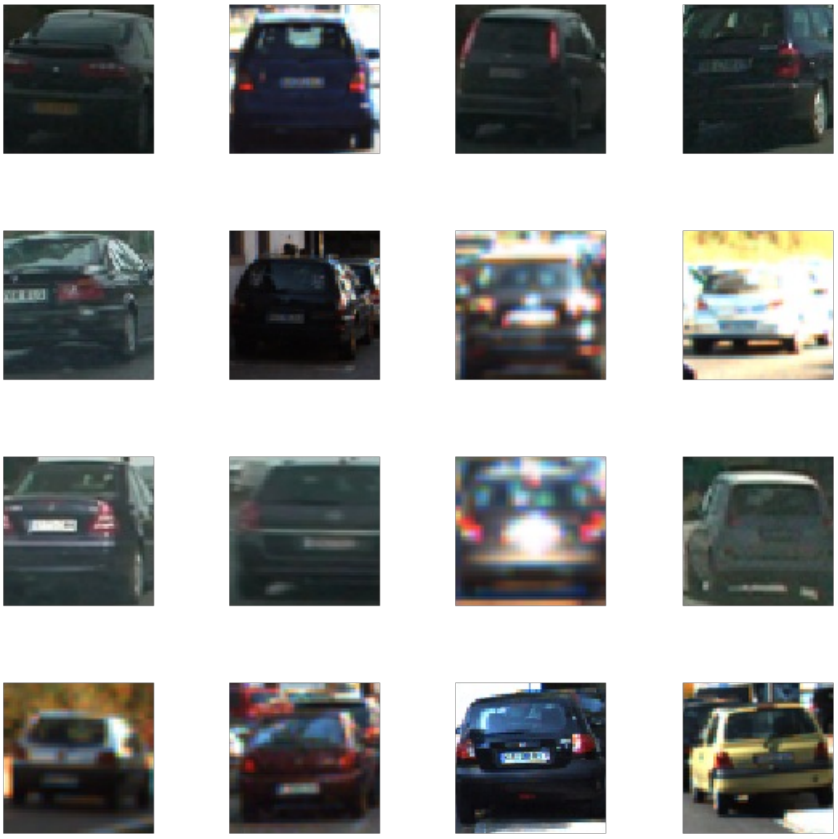
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: 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.

Overview

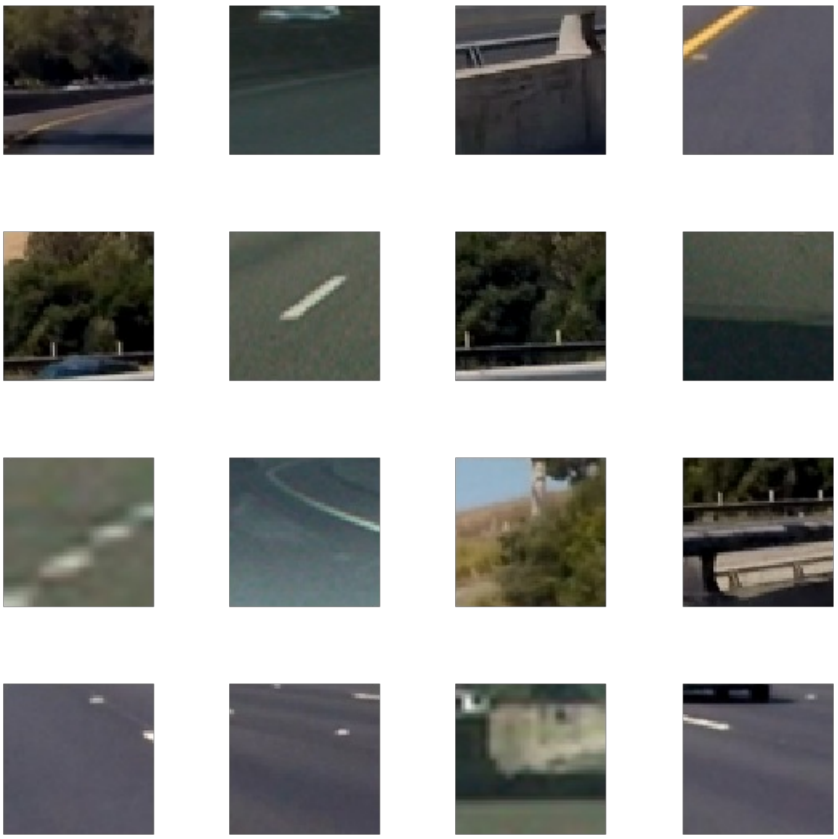
I will go through the steps of the vehicle detection project step by step. Most of the main algorithms were described in the lecture and shown as examples. First, I defined some helper function for plotting and drawing the bounding boxes.

1. Loading Training Data

I have used the provided datasets with cars and background images. I have loaded them into two lists with 8968 background images and 8792 cars. Furthermore, a random choice of images from the two lists were plotted for examination of the set.

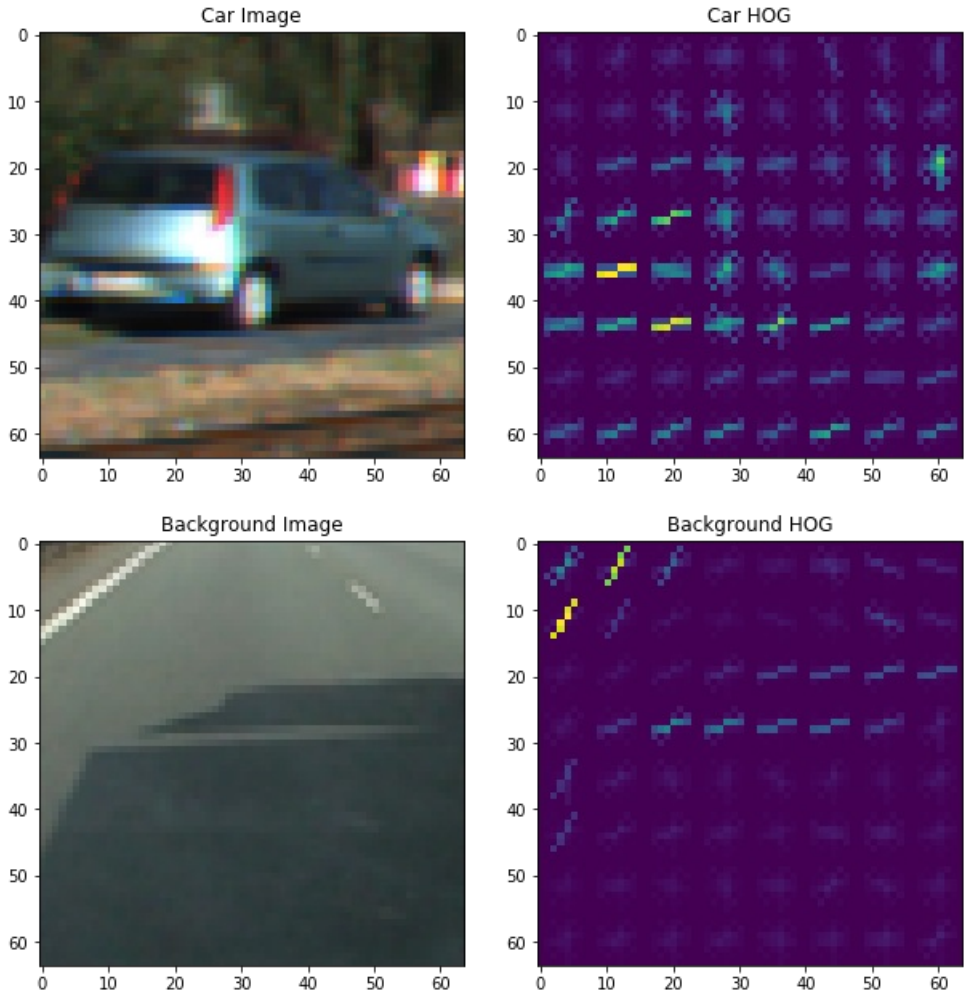


Not cars or background images:



2. Histogram of Oriented Gradients (HOG) feature extraction

For preprocessing of the data the HOG of the images were calculated for feature extraction. The function `get_hog_features()` applies the `hog()` from the skimage library. The output of the function on the random images is shown below:



As expected the significant feature are illustrated by the magnitude and direction of the HOG image. The parameters were chosen as in the the lecture and changes of the pixels per cell etc. did not shown any strong improvement.

3. Additional Features

Along with the HOG processing I also applied spatial binning of the image and color histograms. For this, I have defined two more functions called `bin_spatial()` and `color_hist()`. A third function `extract_feature()` applies all processing steps to extract the final feature vector.

Lastly, I have extracted all features from the images which took around one minute. The feature extraction parameters were:

```
# Feature extraction parameters
colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 11
```

```
pix_per_cell = 16
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
```

4. Training a SVM Classifier

Now that we have all the features and datasets ready we can train a SVM classifier for prediction. The results of the training were:

```
8.89 Seconds to train SVC...
Test Accuracy of SVC =  0.9862
My SVC predicts:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
For these 10 labels:  [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
0.00105 Seconds to predict 10 labels with SVC
```

Different feature extraction parameters which lead to classifier accuracies between 0.96 and 0.98 so these were chosen. Finally, the results were brought together in the function `find_cars()` which does the colorspace conversion, applies the HOG feature extraction on the whole image once, for every channel. Then the windows are calculated for the defined Region Of Intrest (ROI).

The windows are looped over and in every window the HOG features are selected and the spatial binning and color histogram are calculated.

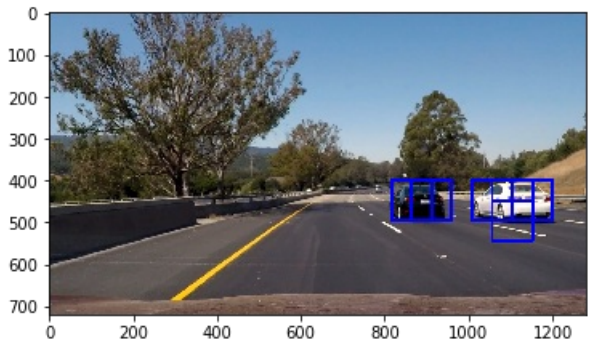
The features are passed to the classifier and when the prediction is correct the bounding box of the cell is appended to a list as well as drawn into the image. So the functions return this image and the list of the bounding boxes coordinates.

5. Test Pipeline

The pipeline is tested on the `test_images` using the following parameters:

```
ystart = 400
ystop = 656
scale = 1.5
spatial_size=(32,32)
hist_bins=32
```

One test image is shown below:



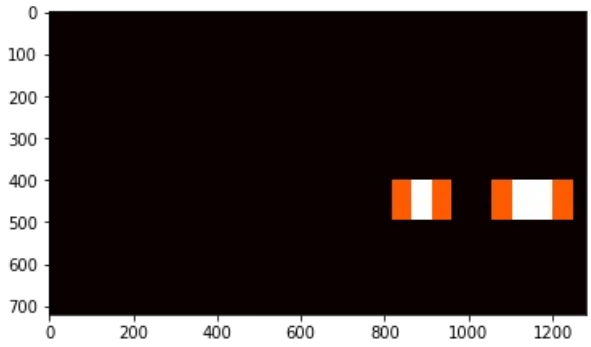
The pipeline yields good results for cars that fit into the window which means it is working well for cars in a certain distance. Therefore, the pipeline is applied multiple times with different ROIs and window scales.

```
for i in range(0,3):
    ystart = 400
    ystop = 464+(i*32)
    scale = 1.0+(i*0.5)
    print(ystart,ystop,scale)
    out_img, _ = find_cars(test_image, ystart, ystop, scale,
                           svc, X_scaler, orient, pix_per_cell,
                           cell_per_block, spatial_size=(32,32), hist_bins=32)
    plt.figure()
    plt.imshow(out_img)
```

This leads to better results for cars that are close. However, there is room for improvement here with a better choice of window sizes.

6. Heatmap

To make use of the multiple bounding box detections a heatmap was used as a filter as it was proposed in the lecture. One example with multiple cars can be seen in the following:



The brightness (or "temperature") increases for multiple detections. Finally, a threshold is applied to reduce the number of false positives and a drawing function is applied to generate bounding boxes around the labeled objects.

7. Define Pipeline Handling Function

The complete pipeline is now defined in the function `process_image()` which takes the images as input and applies image preprocess, classification for multiple ROIs and calculates the heatmap for the total evaluation (see `sum_bbox`). The pipeline is then tested on a short video `test_video.mp4`. Please note that there is no frame-to-frame information but rather every image is processed indepent.

8. Frame-to-Frame Information

In a last step I have implemented a simple frame-to-frame information filtering for a more stable vehicle detection. As I have done this in the projects before I am mainly storing the bounding boxes for the last 20 detections and remove the oldest one if there are more. This time I am using a simple class object instead of a global variable called `Bbox_Detect()`. The method `add_box()` appends the detected bounding boxes.

The new pipeline called `process_video()` makes use of the class which yields a much more stable output as it can be seen in the `project_video_output.mp4`. Please note that the

threshold for the heatmap needs to be adjusted depending on the length of the buffer in the class object.

Discussion

The optimization of the feature extraction parameters was very tideous. Furthermore, there is room for improvement for the windows size and ROI definitions especially for close cars. All in all I am quite happy with the results, although the processing is quite slow. There is alot of feature extraction involved which takes up much processing time. I am wondering if other neural network approaches like semantic image segmentation or bounding box detections like the popular YOLO would be faster and easier to apply. I suppose a combination could be favorable.