Project 5 Vehicle Detection

1. Introduction

The goal of this project is to detect vehicles in a video stream with a traditional machine learning method (Support Vector Machine).

Code files:

Utilities.py: contain all the helper functions such as color_hist, extract_features, find_cars etc.

Feature_train.py: feature training and parameter tuning

Image_test.py: drawing b-boxs on example images

Vehicle_det.py: video process

Feature_parm.pkl: pickle file contains all the parameters as well as the SVM and X_scaler etc.

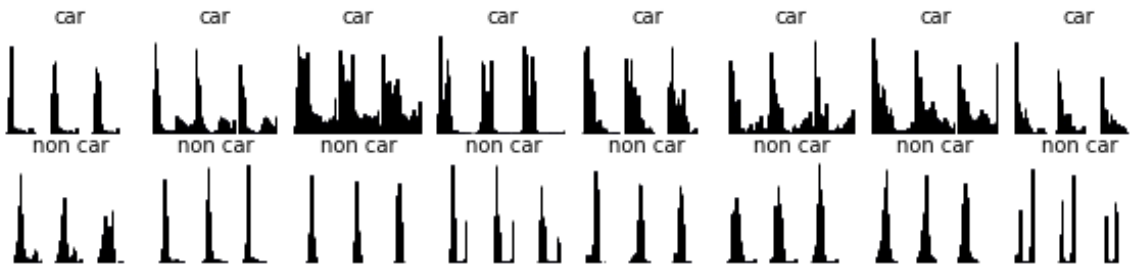
parameter_tuning.xlsx: parameter tuning results

2. Training the SVM

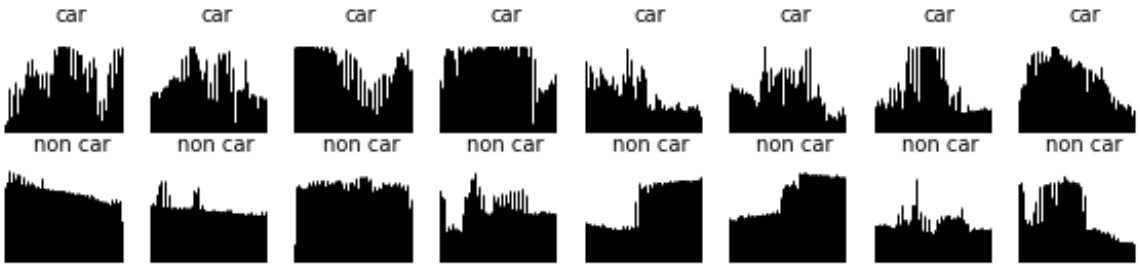
A database of vehicle and non-vehicle contains about 19000 64 by 64 images was given as training dataset. Here is an example of some of the training images:

In order to train the SVM, a feature vector is extracted from each training images and feed along with their label (car/non-car) into a SVM classifier. There are three type of features: color histogram, spatial histogram, and orientation gradient histogram.

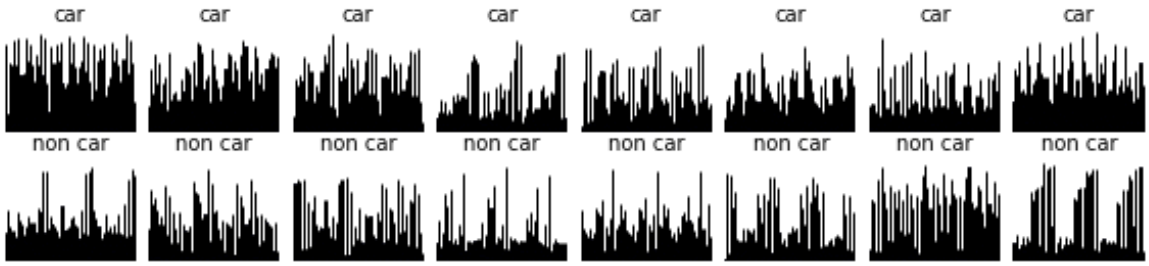
Here is an example of the color histogram in YCrCb color. Notice that the car image has higher standard deviation in each color channel than the non-car



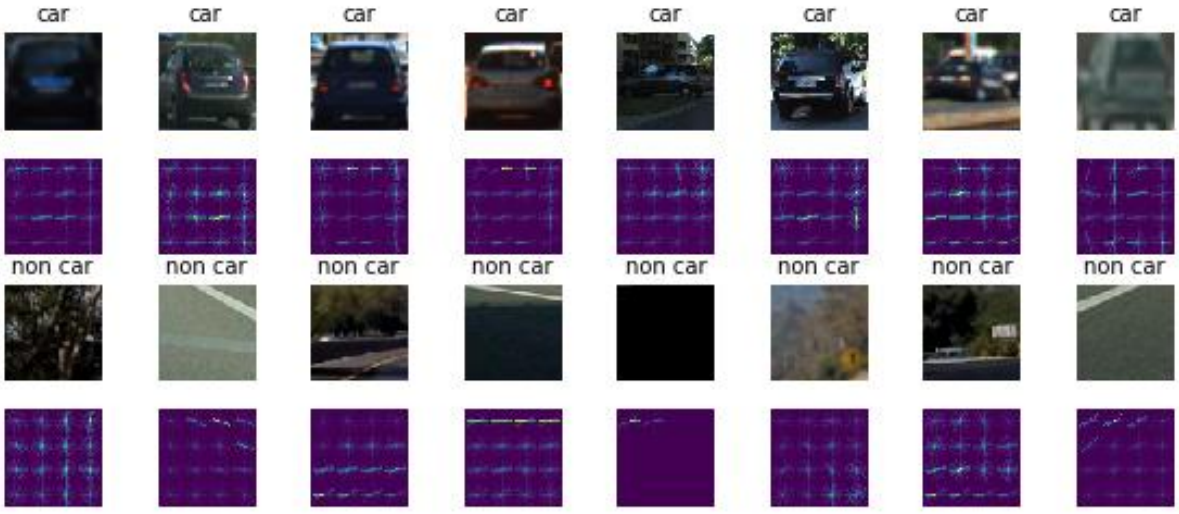
Here is an example of spatial histogram in YCrCb color space. Notice that the non-car's spatial feature distribution is more uniform than the car's



And an example of the HOG feature. Noticing that the car's average amplitude is higher than the non-car's.



And the hog plot, notice that the car hog has more line patterns in it while the non car is more of scattered constellations of dots.



All the features are scaled with the scaler.transform before feeding to the SVM.

3. Color Space selection and Parameter Tuning

This is basically trial and error approach, trying different combinations of parameter and see which render less failed detections (false positive and false negative). The result is documented in the parameter_tuning.xlsx file. Here is the final settings:

```
colorspace = 'YCrCb'
orient = 11
pix_per_cell = 16
cell_per_block = 2
hog_channel = 'ALL'
hist_bins = 32
spatial_size = (32,32)
spatial_feat = True
hist_feat = False
hog_feat = True
```

4. Image Process Pipeline

The pipeline is straightforward, load an image, slide a window in the image and extract the features out of each patch, then use SVM to predict whether a car is in it. After a series of bounding boxes were found, heat map and thresholding were used to decide the final size and positions of the bounding boxs. For the sliding windows search range, I use start = 400, stop = 660. I use two window sizes (scale = 1 and scale = 1.5) to cover the searchable region. This is also a try and error approach, the inputs are the windows size and start, stop positions, and the goal is to minimize false detection in the test images as well as the project video.

Some examples of the test images

5. Video Implementation

As there always some differences in the founded bounding boxes in the sequential video frames, thus the box some time looks jittery, to alleviate this, I skipped every the other two images, this is kind of a cheap way of averaging the box position, make it slightly more stable.

Final Thought

The SVM is simple and easy to train, it gives about 98.5% accuracy on the training dataset. However, there were some false detections on the project video, and I think it is partly because of the small size of the training dataset. The other problem is process speed, it takes about 0.4second to detect vehicles in an image, consider there are many other tasks such as lane detection, traffic sign recognition etc. So this is probably too slow for real time car application, which runs at least 50 frames/sec. I'm looking for ideas to either improve this method by 10fold or some other different ways.