Vehicle Detection and Tracking

Writeup – Jelena Kocić

1. Introduction

The aim of this project is to make pipeline for vehicle detection and tracking. In order to achieve this we have to learn and apply many techniques, algorithms and approaches. We start with feature extraction from images, e.g. HOG (Histogram of Oriented Gradients), color features or spatial binning. Next, we have to learn about classifiers as SVM, Naive Bayer, Decision Trees, Neural Networks, and properly train chosen classifier. Then, in order to detect the car in the input picture we must apply sliding window technique, or some other approach. Also, it is useful to remove false-positive findings using e.g. heatmap technique. With lot of work, building pipeline, and tweaking hyperparameters we may achieve our goal. I tried to use all of the above, and in this writeup I will present my approach. For the training I am using my PC with Nvidia GPU.

2. Histogram of Oriented Gradients (HOG)

First I downloaded training data, car and noncar images:
https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip
https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip

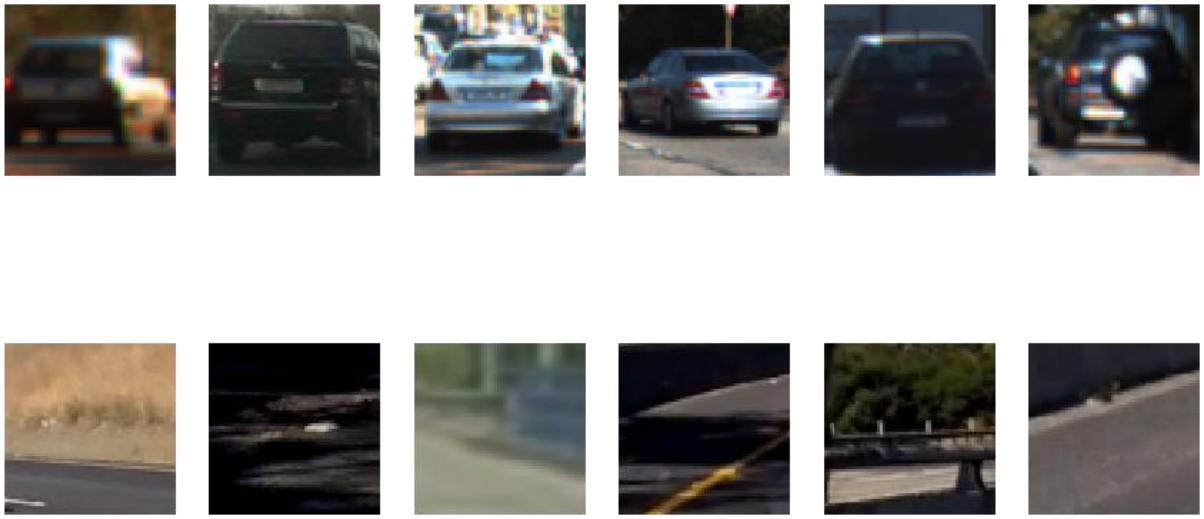
I extracted images using:

```
car_images = glob.glob('dataset/vehicles/**/*.png')
noncar_images = glob.glob('dataset/non-vehicles/**/*.png')
```

As an output I got next data:

```
Number of cars in the dataset = 8792
Number of not cars in dataset = none cars: 8968
Car images shape: (64, 64, 3)
Not-car images shape: (64, 64, 3)
Images type: float32
```

Examples of cars and notcars:



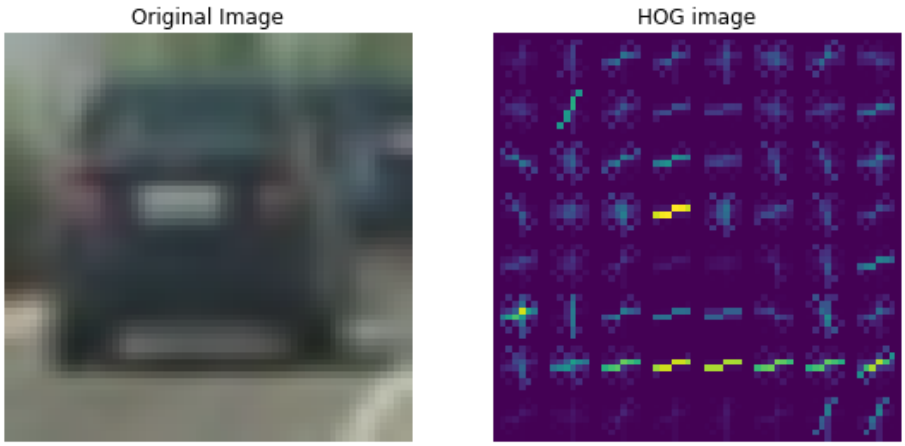
For the extracting features from the images I used HOG (Histogram of Oriented Gradients). Please look at the 2nd code cell, and the function get_hog_features().

HOG parameters that I choose are:

colorspace= 'YUV'
orient= 9
pix_per_cell = 8
cell_per_block =2
hog_channel = 'ALL'

I played with different parameters as well, e.g, orient = 12, pix_per_cell = 16, cell_per_block = 2. However I choose the first set of parameters.

Example of the original image and the image after applying the HOG:



The other techniques that I tried for feature extraction are:
bin_spatial() - a function to compute binned color features,
color_hist() - a function to compute color histogram features.
Please look at the 2rd cell of jupyter notebook.

The function which extract features from a list of images is extract_features(), defined in 2rd cell of the jupyter notebook.

Extraction of the features is done in the 4th code cell of jupyter notebook. Total Number of Feature used in a model is 5292.

3. Sliding Window Search

3.1. Balance, split up and shuffle data set

To normalize your data I used `StandardScaler()` method, refer to 5th code cell in jupyter notebook. Split up data into training and test sets is done as 80% for the training and 20% for the test. Also, I shuffle the data. Snapshot of the code:

```
# Shuffle and Split the data into training and test data
X_scaler = StandardScaler().fit(X)
scaled_X = X_scaler.transform(X)

rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=rand_state)
```

3.2. Training data set

3.2.1. Introduction

Extracts features for each sample in the training set, and supply these feature vectors to the training algorithm, along with the corresponding labels. The training algorithm initializes a model, and then tweaks its parameters using the feature vectors and labels. This is an iterative procedure where one or more samples are presented to the classifier at a time, which then predicts their labels. The error between these predicted labels and ground-truth is used as a signal to modify the parameters. When this error falls below a certain threshold, or when enough iterations have passed, we can consider the model to have been sufficiently trained. Then we can verify how it performs on previously unseen examples using the test set.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

`SVC`, `NuSVC` and `LinearSVC` are classes capable of performing multi-class classification on a dataset.

SVC = C-Support Vector Classification. The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples. The multiclass support is handled according to a one-vs-one scheme.

3.2.2. Implementation

I have implemented linear SVC, please refer to the 6th code cell:

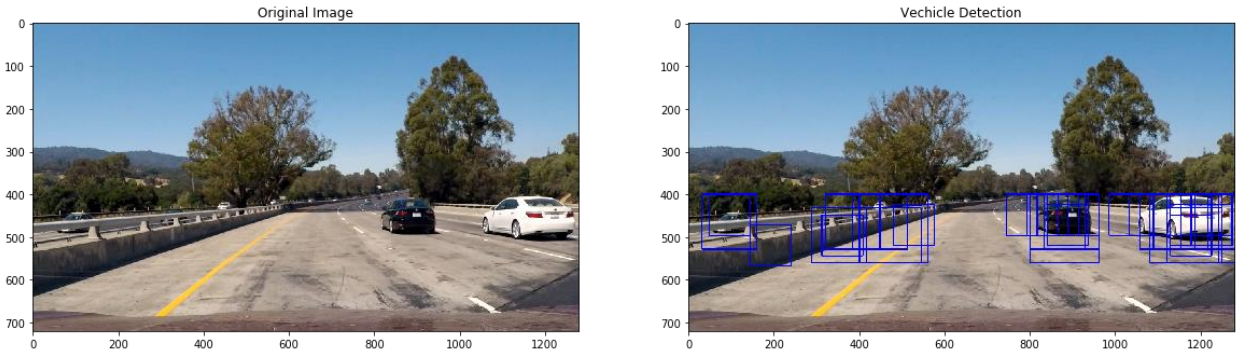
```
# linear Support Vector Classfier to classify the image

from sklearn.svm import SVC

svc = LinearSVC()
# Check the training time for the SVC
t = time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC')
# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
```

Test Accuracy of SVC is 98.4%.

3.3. Sliding Windows

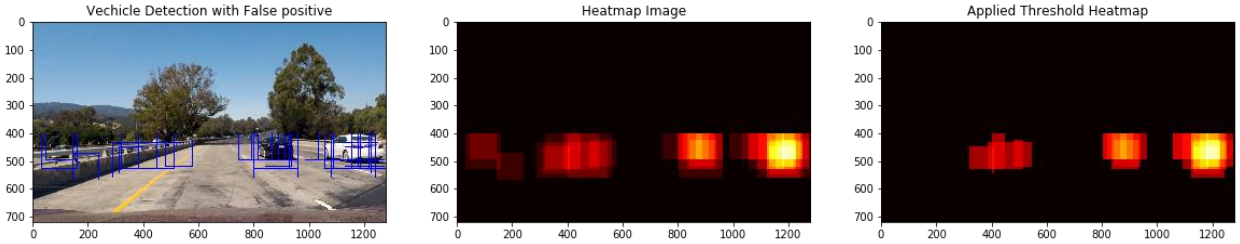
In the 7th code cell of the jupyter notebook is implemented sliding windows technique, draw_img(). I use 3 sliding windows with different sizes, and different positions. Function find_cars() is implemented in 2nd code cell.

Original picture and result of detected cars:



There is lot of false positive results! In order to reduce them I applied heatmap, please refer to the 9th code cell. Function add_heat() is implemented in 2nd code cell. Result:

After applying function for heatmap threshold: apply_threshold(), defined in 2nd code cell, we got result:



This reduces most of the false positive results. But not all of them. Result:

4. Video Implementation

At the end, the code is tested on input video, using Vehicle_Detect() class defined in 15th cell.

Final result is video which can be seen on this link.

5. Discussion

Detecting cars with SVM in sliding windows is appropriate method for this task, but it has some disadvantages. This method does not generalizes well and produces lot of false positives in some situations, it triggers not only on cars but on other parts of an image that is far from car look like.

For the further work it will be good to do better reduction of false positives. Also, we can more with different parameters, than trying other classifiers, like Naive Bayes, Decision Tree or Neural Network. E.g. it is possible to improve the classifier by additional data augmentation. Also, sliding window approach technique for detecting car is one of the possibilities. We can try some other techniques for detecting car in the input image.