Writeup report on Vehicle detection and tracking project.

1. Overview

In this project a classifier based primarily on HOG features of car images is used for vehicle detection and tracking on video. So the purpose is to implement detection and tracking pipeline to a given set of project data to detect and track vehicles as accurately as possible.

2. Detection and tracking pipeline

Tracking pipeline consists of the two following steps:

1) vehicle detection on a single image (videoframe);
2) tracking detected vehicle on a series of videoframes.

Each step is describer further.

3. Vehicle detection

a. Data overview

In the project an SVC classifier was used trained on a training dataset for detection. Training dataset consists images belonging to two classes – vehicles and non-vehicles. The example of training data is shown below:

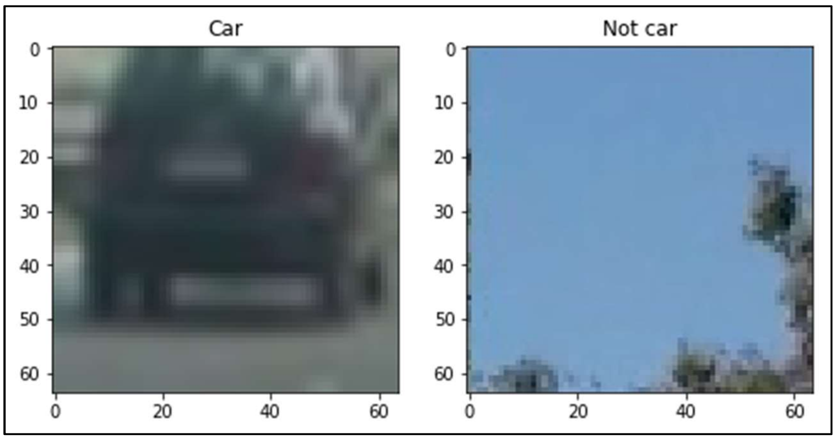

Fig.1. Example of training data

The images are 64x64 size. Total number of vehicles and non-vehicles examples are 8792 and 8968, respectively. This is well-balanced dataset, so we can ignore possible issues with the lack of class representatives.

b. Feature extraction

There are several feature types that could be used for classification task:

1) HOG features
2) spatial color features
3) color histogram features

The main problem with spatial color features is the final number of features – even 32x32 samples yield 3072 features which causes performance problems for real-time detection.

In my work I used two types of features:

1) HOG features of grayscaled image + spatial 32x32 in BGR + 3-channel histogram in BRG with 32 bins for each channel. This yields 4932 features totally for 64x64 pictures

2) HOG features of grayscaled image + 3-channel histogram in BGR with 32 bins for each channel, which yields 1860 features per 64x64 image.

For both cases HOG parameters are: 8 pixels per cell, 2x2 cell per block, 9 orientations.

For each image different types of features (HOG, spatial, histogram) were flattened and concatenated to a single feature vector.

Feature extraction was implemented in FeatureExtractor() class. FeatureExtractor.return_features_and_labels() method was used for getting image features and class labels.

c. Training the classifier.

An SVC classifier was used for classification task. Training set included 14207 training and 3553 testing examples (80:20 split) belonging to both classes. Before training, sklearn StandardScaler() was used for normalizing feature vectors. StandardScaler was fitted on training set only.

For selecting best SVC hyperparameters GridSearchCV with 3 folds was used. The following hyperparameters were tested:

1) kernel: ‘rbf’, ‘linear’, ‘poly’
2) C: 0.1, 1
3) gamma: ‘auto’, 0.001

Finally, rbf-kernel with C=1 and gamma=auto showed the best result.

For the first feature vector type (HOG+spat+hist) accuracy score was 0.989, for the second feature vector (HOG+hist) – 0.983. Although this could seem to be rather good result, it’s worth to mention that for a 1280x720 image splitting to subwindows 64x64 size with 0.5 overlap between subwindows yields about 900 sample images which means that we may expect about 9 false detections per image even for 0.99 accuracy of the classifier. This means that a strong post-processing technique is required to reject false positives. Furthermore, accuracy score may be not the best metrics for training the classifier. As an alternative, precision-recall – type scores may be better characteristics, depending of what is more critical for the detection task – to miss a car of to face a false positive.

d. Sliding window detection.

As the classifier is trained to a fixed image size, a sliding window search on a test images and videoframes is required for detection of vehicles. Furthermore, different scales of sliding windows should be classified for a given image, as the size of vehicles may vary. All this was implemented in Tracker() class. Tracker takes feature extractor, trained classifier and scaler to perform classification task of sliding window. Sliding is performed for a number of different scales (i.e., scale==2 means searching with 128x128 window, as base size of window is 64x64). Detections with different scales are then used to form a heatmap, where overlapping detections make “more heat” on a resultant heatmap. Then scipy.ndimage.measurements.label() is used to contour areas of a frame potentially belonging to detected objects. Then “centers of gravity” found for these areas and these centers are treated as centers of detected objects. Then bounding rectangles are drawn around these centers.

An example of tracker() results on test frames is shown below. Although this is yet detection, not tracking task, class Tracker() can perform both single-frame detection and multiple frames tracking.

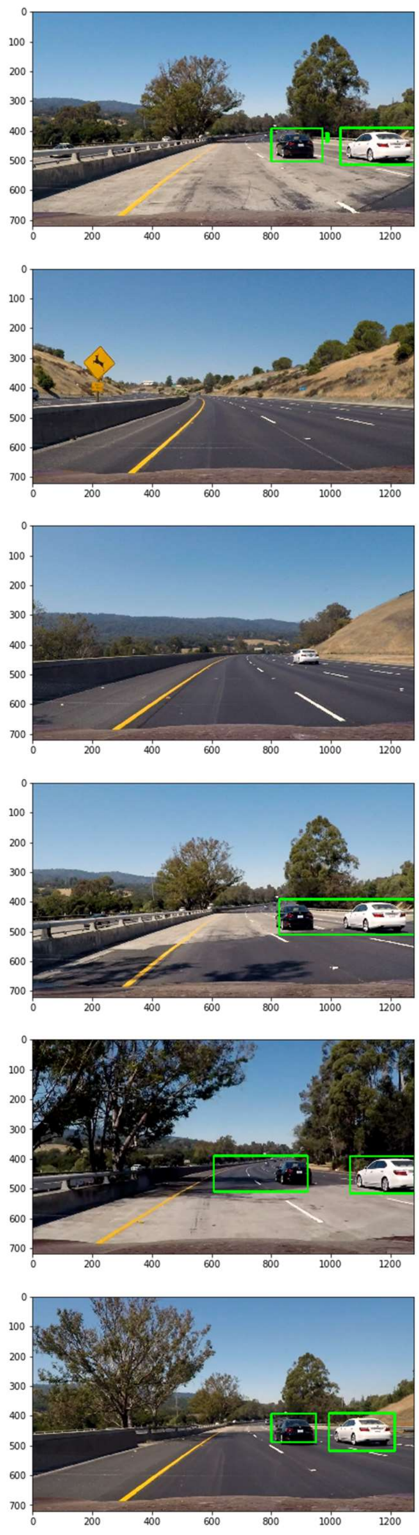

Fig.2. Example of vehicle detection.

On Fig.2 one can see that detection works with moderate quality. In this example an "inter-image" threshold is used in the following manner: for a given image for each scale in (0.5, 1, 2) a weighted sum of heatmaps is found, weight equals scale, than a threshold = 2 is applied. Then label() function is applied to find regions of objects and then bounding rectangles are drawn. We can see one missed detection on the third image and some inaccuracy on 4th and 5th images. This is due to false detections of the classifier that extends bounding boxes and capture near-placed objects into one bbox.

e. Tracking

The same class Tracker() is used for tracking on a series of videoframes. During tracking one more threshold is used for making detection more reliable – once detection for a single frame is performed, a heatmap along with bboxes is returned where pixels equal 1 for areas inside bboxes. These heatmaps are further summed over several frames and “multiframe-threshold” is then applied. Then resultant bboxes are drawn based on a thresholded heatmap over several images. The results of a tracking pipeline can be seen on project videos with detections.

4. Discussion.

Although the pipeline described above works to some extent, I was unable to achieve both robust and real-time result using the techniques mentioned. The problem was that even with only 2 scales and overlap 0.75, processing of one videoframe requires more than 10 seconds, which leads to several hours processing of project video.

Ways to improve the quality and performance:

1) reduce the number of features as much as possible
2) try more different combinations of color spaces and channels for HOG and histogram features
3) try more sophisticated techniques for post-processing the images