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 a RandomForestClassifier (as an emsembled version of DecisionTree) 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:

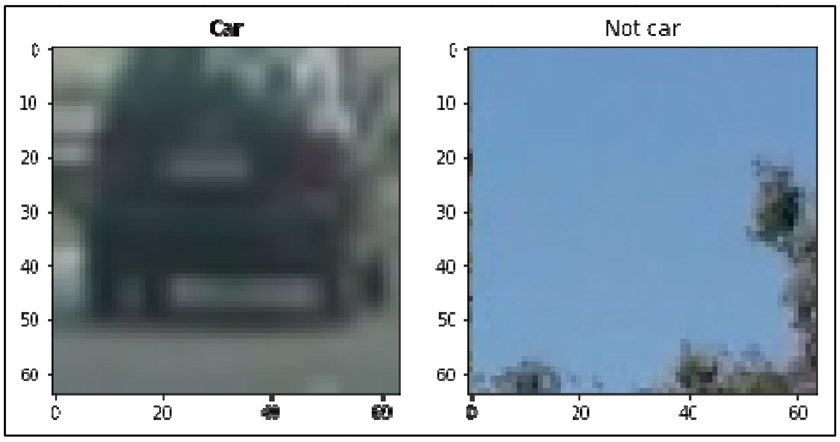

Fig.1. Example of training data

The images are 64x64 size. Total number of vehicles and non-vehicles examples are 8792 and 8968, respectively. Various experiments with the test video showed the tendency of a trained classifier to produce false detections, so hard negative mining was performed to enlarge non-vehicles dataset by 2000 examples. Total size of training set (after train-test splitting) was 15808 training images. Despite after adding 2000 negative examples the balance between classes disturbed, it was decided to do nothing with this as slight domination of negative examples in the training set is compensated with multiple and multiscale detections on video frames.

b. Feature extraction and Training pipeline.

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

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

Feature extraction was implemented in **FeatureExtractor()** class. This class inherits from **sklearn.base.BaseEstimator** class and has **fit** and **transform** methods. This allows using this class in **sklearn.pipeline.Pipeline** class methods for searching best composition of HOG, spatial and histogram features along with different colormap transformations.

The pipeline consisted of the following steps: feature extraction, feature scaling and fitting the classifier. The only hyperparameter of RandomForestClassifier that was tuned using the pipeline was ‘criterion’ – ‘gini’ or ‘entropy’. The number of estimators was set to 100, as a tradeoff between accuracy and performance. RandomizedSearchCV was used with f1-score to get the best combination of parameters of feature extraction. See chapter “Pipeline implementation” in **tracking-pipeline-f1.ipynb** for details.

Finally, the best feature vector consisted of the following sub-features:

1) HOG with 11 orientations on R-channel, 8x8 cell, 2x2 cells per block
2) spatial 16x16 in YCrCb colorspace
3) histogram features in YCrCb with 64 bins per channel

These features were combined in a single feature vector with mode than 3000 features for a 64x64 image.

The best achieved accuracy score was 0.993, f1-score 0.99.

c. 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).

To filter out multiple detections for a given scale a Non-Maximum Suppression algorithm was used (https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/).

To combine the result of detections with different scales on a given image, scipy.ndimage.measurements.label() is used to contour areas of a frame potentially belonging to detected objects.

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.

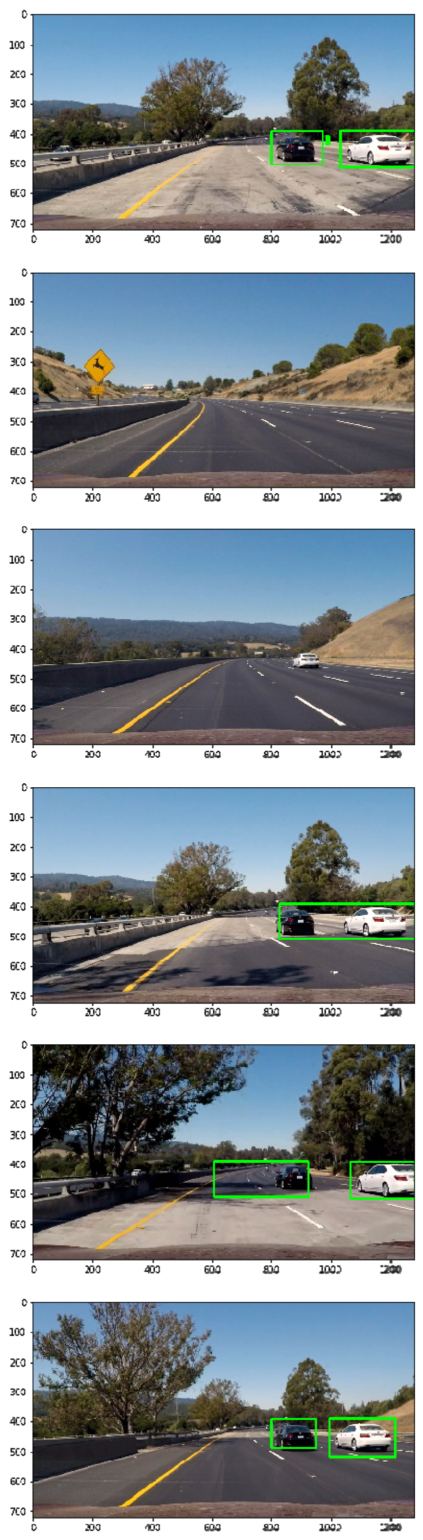

Fig.2. Example of vehicle detection.

On Fig.2 one can see that detection works with moderate quality..

d. 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.

The approach described gives acceptable results (see output_videos/project_video_tracker.mp4). There are several problems that are quite difficult to solve:

1) Acceptable accuracy is only achieved with rather large size of feature vectors – several thousand features. In combination with sliding multiscale window technique this yields very massive number of calculations per frame, so I've only achieved about 0.5 FPS tracking performance on a 8-processor machine, which is quite low result compared to what can NN-based detectors give (i.e., YOLO detector).
2) Bounding boxes are not very smoothly change from frame to frame, I've tried various averaging and filtering techniques, but it seems that some more smart methods should be used.
3) The problem of occlusion is not solved yet completely with this approach (can be seen in the end of the video).