Vehicle Detection and Tracking

Project Submission Writeup Report

For the Udacity Self-Driving Car Program – Term 1

Submitted by:

Stewart Teaze

July 21, 2017

Project Goals

The goals / steps of this project are:

- 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.

Rubric Points

The following sections of this document relate to addressing each of the project's rubric points (project requirements), and provide details of how each requirement was addressed in the project implementation.

---

1. Submitted Files

1. writeup.pdf - This document/file is the writeup itself; which, in addition to providing details of how each project requirement was addressed, contains images produced during each of the building block stages of the vehicle detection and tracking application development process.
2. P5.ipynb – Jupyter Notebook with Python code segments, capturing each development step of the evolution of the computer vision processing, model training and testing, and application integration development steps, contributing to (and including), the full implementation of the vehicle detection and tracking processing pipeline.
3. test_video_output – Directory containing video output file (project_video_out), with vehicle detection/tracking "bounding boxes" overlays, produced/output by the code in P5.ipynb, corresponding to the provided test input video file "project_video".

All of the files/directories listed above, can be found at:
https://github.com/blunderbuss9/CarND-P5-VDT-StewartTeaze

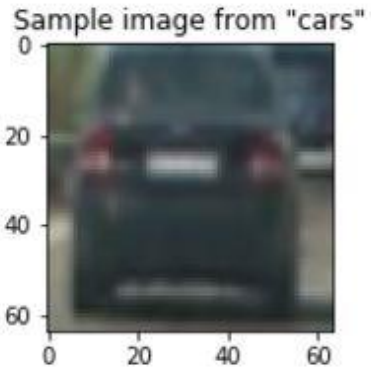
2. Car Image Recognition Classifier

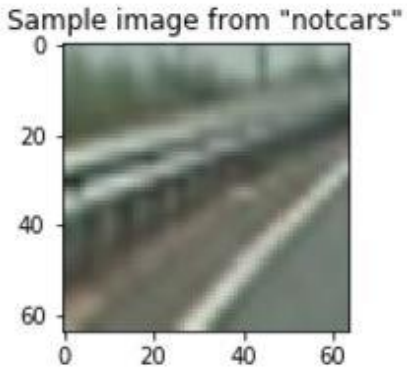
A. Training Data Description

The data used for training the Car/Vehicle Classifier consists of 17760 sample 64x64 pixel training images, in .png format. This data was composed of images from two standardized "benchmark" sources of machine learning training images – the GTI (Grupo de Tratamiento de Imágenes) vehicle image database[1]; and the KITTI (Karlshruhe Institute of Technology/Toyota Technological Institute of Chicago) vision benchmark suite[2]. Roughly half of the images (8792) were of the desired classification type: "Car", and the other half (8968) were of "Non-Car" non-desired types.

A file "glob" structure was set up to allow the Car and Non-Car images to be easily sequentially read in from disk, and these images are separately distributed into the Python lists named "cars" and "notcars", in the project's Jupyter Notebook in the 5th Python code cell.

Here are example images, from the provided training dataset, of the "Car" and "Non-Car" types:





[1] http://www.gti.ssr.upm.es/data/Vehicle_database.html

[2] http://www.cvlibs.net/datasets/kitti/

B. Feature Extraction from Training Data

The car/vehicle feature extraction implementation consists of a combination of 1) Color Channel feature extraction and 2) HOG (Histogram of Oriented Gradients) feature extraction. This combination is orchestrated by the extract_features() utility function, created in the 5th Python code cell and invoked in the 7th code cell of the project submission's Jupyter Notebook. extract_features() in turn invokes specific lower-level utility functions to perform the specific details of the individual feature extractions, as described in the feature extraction details sub-sections 2.B.1 & 2.B.2 below.

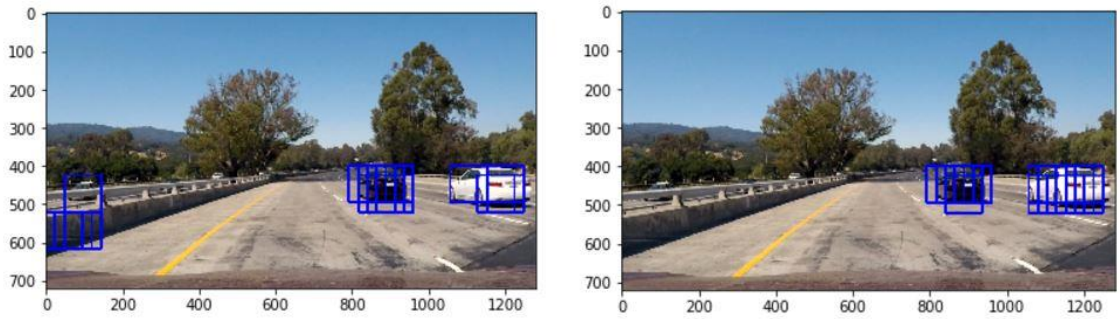
NOTE: All basic utility functions discussed here are implemented in the project notebook's 2nd code cell.

1. Color Channel-based Feature Extraction

The color feature extraction itself consists of a combination of two methods: Spatial Binning of Color, and Color Space Histogram Binning. For both of these methods, the original 64x64 RGB images were first converted to 64x64 YCrCb format "feature images", using the OpenCV function cv2.cvtColor() by utilizing the utility function convert_color() and passing in the cv2.COLOR_RGB2YCrCb parameter.
Here are the details of the implementation of these color feature extraction sub-methods:

2. Spatial Binning of Color –The 64x64 YCrCb feature images are separated into three 32x32 Y, Cr, Cb color channel feature vectors, that are combined by the np.hstack() function call in utility function bin_spatial().
3. Histogram of Y, Cr, Cb Color Space – 32-bin histogram feature vectors are created for each Y, Cr & Cb color channel, and combined by the np.concatenate() function call in utility function color_hist().

Various color channel feature extraction methods were analyzed (in combination with the HOG feature extraction method), by tweaking parameter "color_space" in the 6th code cell. YCrCb color channels were used to create the final trained model, as YCrCb performed better for sample images taken from the project_video. For example, the left image below was made with HSV color channels, and the right image uses YCrCb color channels - as can be seen, more positive car identifications, and less false positives are achieved with YCrCb, vs. HSV (using RGB color channels performed much worse).



2. HOG Feature Extraction

The actual HOG feature extraction [Dalal2005] is performed by utilizing the scikit-image hog() function, as invoked through the get_hog_features() utility function, and passing in a "YCrCb "feature image" (as described in section B.1). As described in the scikit-image documentation[3] and as utilized by the project code, the hog() algorithm returns the Histogram of Oriented Gradients, into feature vectors based on 9 histogram of orientated gradient bins for this implementation (based on the "orient" "tweaker" value set to 9, in the 6th code cell), by performing the following steps:

1. Computing the gradient image in x and y
2. Computing gradient histograms
3. Normalizing across blocks
4. Flattening into a feature vector

The HOG feature extraction parameters "orient" (set to 9), "pixels_per_cell" (8), and cell_per_block(2) were used as they worked satisfactorily in the HOG feature extraction process.

---

[3] See http://scikit-image.org/docs/dev/api/skimage.feature.html?highlight=hog#skimage.feature.hog, and http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html for more details

C. Feature Vector Standardization

In the 7th Python code cell, all the various extracted feature vectors, described in section 2.B, and placed in the list "X" by extract_features, are then standardized (scaled to zero mean and unit variance) in preparation for training the Classifier. sklearn.preprocessing.StandardScaler()[4] is then used to fit a per-column scaler to the stack of feature vectors (into the fitted StandardScaler() class instance named X_scaler), and then applying the scaler-transform on the data in X, using the fitted StandardScaler() class instance X.scaler using the StandardScaler() class method transform(X), returning the scaled feature vector data in the Python list "scaled_X".

D. Train and Test Linear SVC Car Image Classifier Model Using Extracted Feature Vectors

Also in the 7th Python code cell, the scikit-learn sklearn.svm.LinearSVC()[5] class is used to train a Support Vector Machine (SVM) Linear Support Vector Classifier (SVC), specifically to recognize car images, based on the extracted feature vectors described in section 2.B. The data is split into randomized training and test sets X_train/y_train, and X_test/y_test using the sklearn.cross_validation function train_test_split()[6]; then a LinearSVC() class instance is created and named svc(), and the LinearSVC() fit method is used to train that svc instance using the training set, the accuracy of the model is verified with svc.score() (Accuracy values ranged from .9901 to .9918 during various runs for this particular feature vector set and LinearSVC model), and a quick prediction check is run with svc.predict(), and the results of 25 predictions are output using print statements (with the results almost invariably coming out all correct, given the high accuracy of the model).

Accuracy values for the same LinearSVC model, trained with HOG/spatialBin/colorHistogram feature vectors built up from HVC color channel feature images, were seen to be slightly lower than for the same model trained with YCrCb color channel feature images; but, in practice, the YCrCb-based variants performed much better on actual frames taken from the project video. Again, refer to the images presented in Section 2.B.1, for an example comparison of the actual performance on a project video extracted image, when using HVC vs. YCrCb color channel parameters to create the feature images used to train the specific instances of the LinearSVC model. RGB feature image-based trained LinearSVC model variants performed much worse than either HVC or YCrCb variants.

---

[4] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

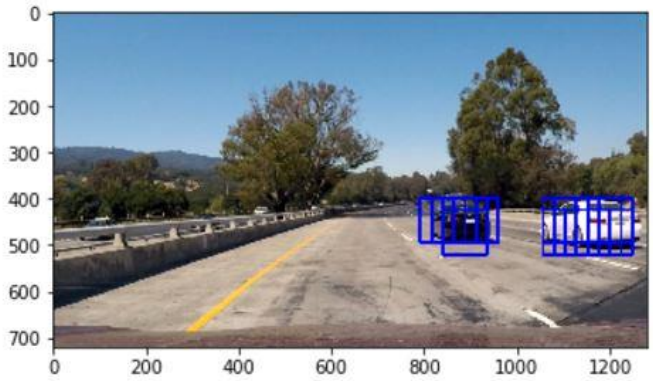
[5] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

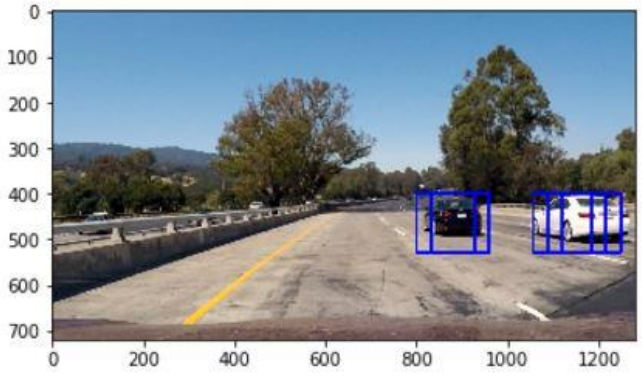
[6] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

3. Sliding Window Search

A. Car Identification in Video Image Frames

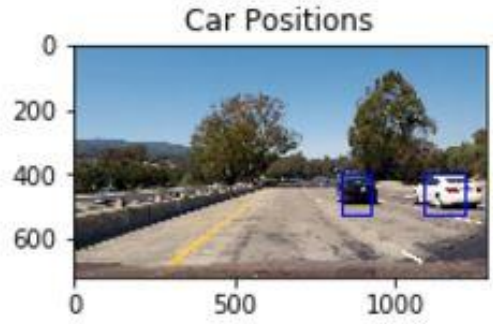
The development of the final method used for identifying cars in each video image frame, started and was initially derived from a modified version of the provided find_cars() utility function. After the modifications I made, find_cars() was able to perform car position prediction identifications of cars, and overlay the prediction boxes onto the provided image, at selectable scales, using my final HOG with YCrCb color channel feature image-based trained variant of the LinearSVC model. The two images below show car prediction identifications made at 1.0 scale (left image), and identifications made at 2.0 scale (right image), which produces a satisfactory overlapping of "tiles" of scaled sub-images to increase the probabilities of positive detections using the heatmap approach described in Sections 3.B and 3.C. These test images are selected from many that I generated by modifying parameters, and performing various test runs, of the code in code cells 9, 10, and 11, using the predictions made by the final trained variant of the LinearSVC model, and these tests drove the final selection of scale and overlap parameters that were chosen.

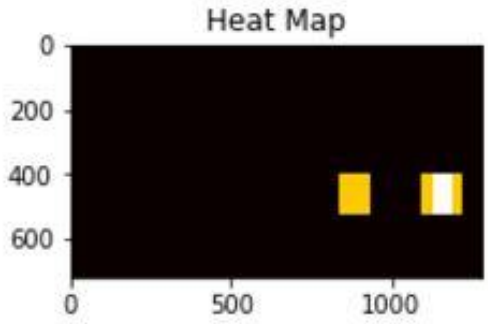




B. Car Position Bounding Boxes and Heat Maps

The next step towards the development of the final method used for identifying cars in each video image frame, was to create a function called find_bboxes, that was similar in functionality to find_cars() utility function, except that it returns a Python list of bounding boxes, instead of an overlayed image, so that all the bounding boxes at various scales that were found on an image, could be combined to generate a "heat map", whose greater utility is related to combining "heat" over multiple frames to re-inforce and solidify positive car positions, and help eliminate "false positive" predictions, a process that will be explained in greater detail in the next section of this document. The two images below, were generated by the output of code cell 12; the left image shows the "solidified" Car positions (prediction bounding boxes), generated by utilizing the extents of the "heat map" bounding box occurrences for that same image, as shown in the right image.





The key functions used in implementation of the heatmap functionality are the provided add_heat() and apply_threshold utility functions, and most importantly the scipy.ndimage.measurements function label().[7]

---

[7] https://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.ndimage.measurements.label.html

C. Using Heat Maps Built Up Over Most-Recent Frames – process_image() pipeline

The next step towards the development of the final method used for identifying cars in each video image frame, was to create a function called process_image(), in code cell 13, to perform the video image pipeline processing. process_image() makes multiple calls to find_bboxes, to create multiple Python lists of car bounding boxes at multiple scales… these multiple lists are combined for the current image, and a global Python list containing the last 4 sets of combined lists of bounding boxes (for the last four video images processed) are used to develop a "more solidified and stable" indication of car bounding boxes using a heatmap, and the apply_threshold() utility function is used with a threshold value of "2", on the heatmap, which virtually eliminated "false positives" in the final output video (occasional false positives would appear, if the threshold value was set to "1"). process_image then draws the solidified bounding boxes on the current image using the draw_labeled_bboxes() utility function, and returns that updated image for inclusion in the output video stream.

4. Discussion

One of the more challenging parts of this project turned out to be the need to modify find_cars() (and it's derivation find_bboxes()) to work with my variant of the HOG/SpatialBin/YCrCb ColorMap feature vector trained LinearSVC model. This became necessary due to the way the feature vector, used to train my model, was constructed (stacked/concatenated/etc.) differently than the model that was used in the version of find_cars() used in the lessons; so, my version of find_cars() had to be updated to use the same feature vector "construction" as my trained model, when using the StandardScalar() class instance for scaling the runtime images' vector data to use with the model to make predictions, otherwise I would get an obtuse "operands could not be broadcast together with shapes" runtime error message, and the program would crash. It took me some time to determine the cause of the error, and fix it; but this diversion had the beneficial effect of forcing me to completely understand the way the feature vector was assembled from all its parts.

Areas for improvement mainly relate to speed optimizations to allow the process to work better in real-time. I attempted a few optimizations, such as trying to use if-then statement logic to manage the index into my ribs_list (list of bounding boxes for recent images processed), rather than using the Python modulus (%) operator (theorizing that the math operation would take longer, and have at least a moderate effect on performance, as it was used during the video loop), but it turned out the speed improvement was almost negligible. I would think that if the algorithm/method used to implement this project in Python, was converted to C++, the speed improvements would be very significant.

Regarding hypothetical cases that could cause pipeline to fail, my greatest concern is the usage of this method to help control an actual car in realtime… I think the usage of Python to develop and initially test the implementation on test videos is worthwhile, but it would be too slow for actual use with humans in the vehicle on real roads. However, once the algorithm/implementation method is proven out in Python, the program could be re-implemented in C++ to achieve acceptable realtime speed performance.

REFERENCES

[Dalal2005] Dalal, N and Triggs, B, Histograms of Oriented Gradients for Human Detection, IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2005 San Diego, CA USA, https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf, DOI:10.1109/CVPR.2005.177

[Geron17] Hands-On Machine Learning with Scikit-Learn & TensorFlow by Aurelien Geron (O'Reilly). Copyright 2017 Aurelien Geron, 978-1-491-96229-9