

项目

Vehicle Detection and Tracking

此部分属于 Self-Driving Car Program

项目审阅

注释

Meets Specifications

SHARE YOUR ACCOMPLISHMENT



Great job! Congratulations and good luck on your next project/term!

Writeup / README

The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.

Well done with writeup! 👍

It is always important to understand advantages and limitations of your algorithm and know ways to improve it.

Histogram of Oriented Gradients (HOG)

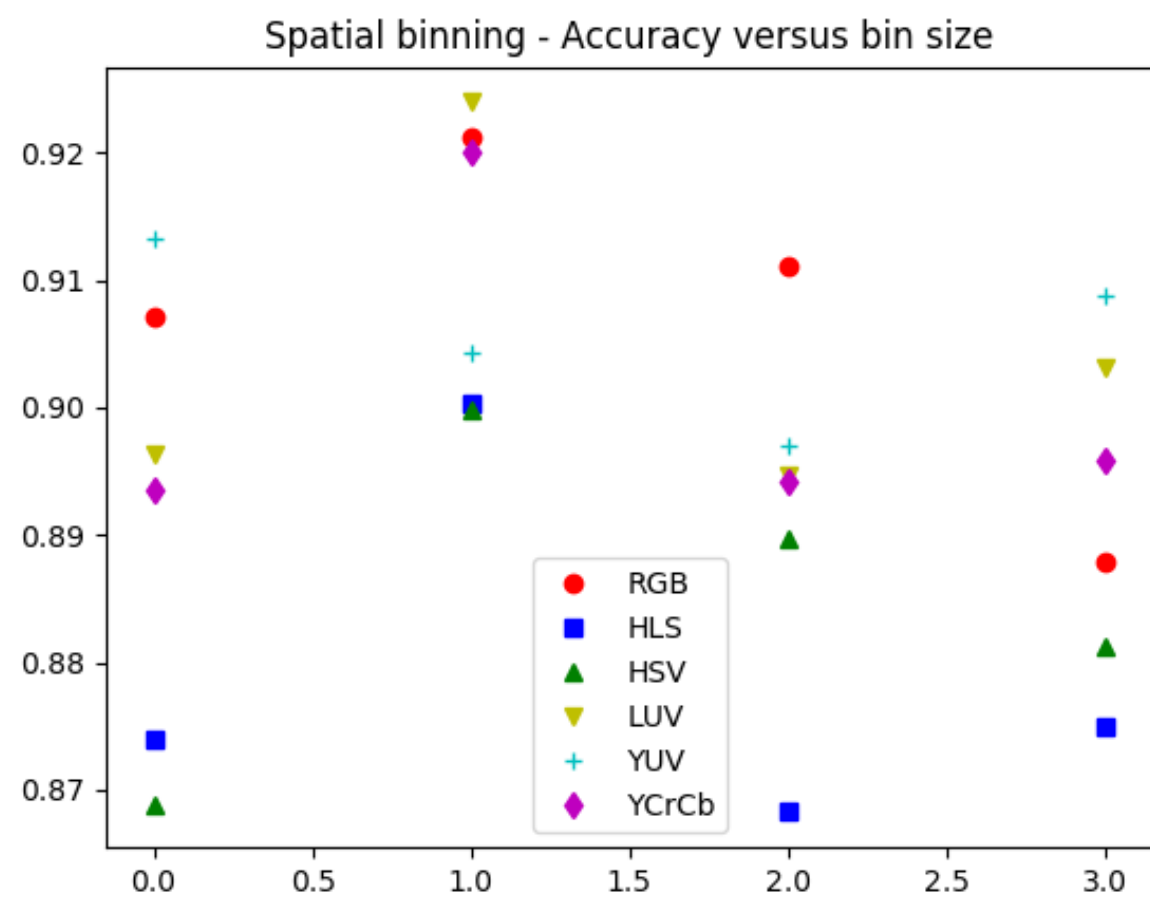
Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels_per_cell, cells_per_block), and why.

Well done with HOG and chosen parameters!

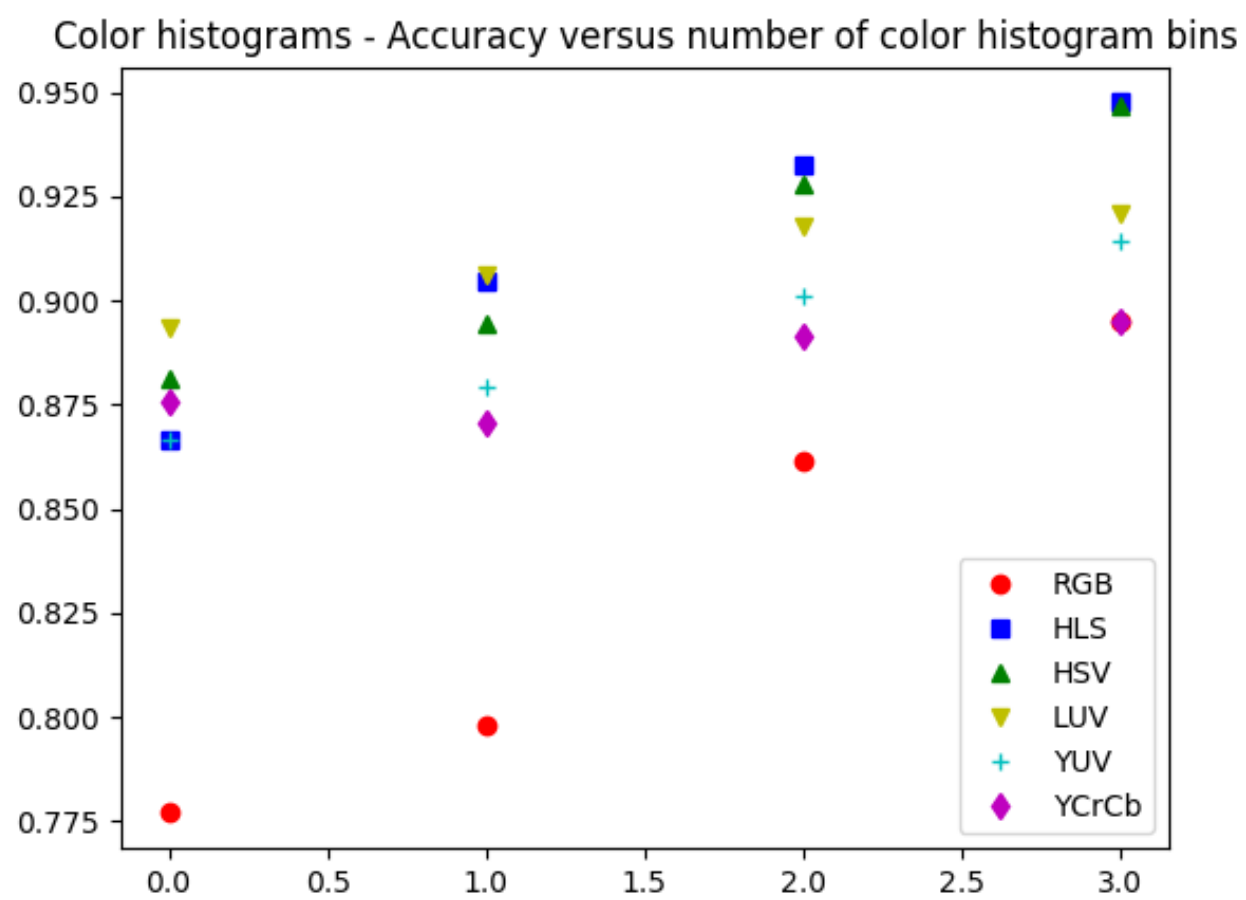
To investigate different input parameters I would recommend you to calculate classifier accuracy for different input parameters with freeze of other parameters. Results you can visualize in a way shown below.

Spatial binning for sizes: (8, 8), (16, 16), (32, 32) & (64, 64) (on x-axis)

给这次审阅打分

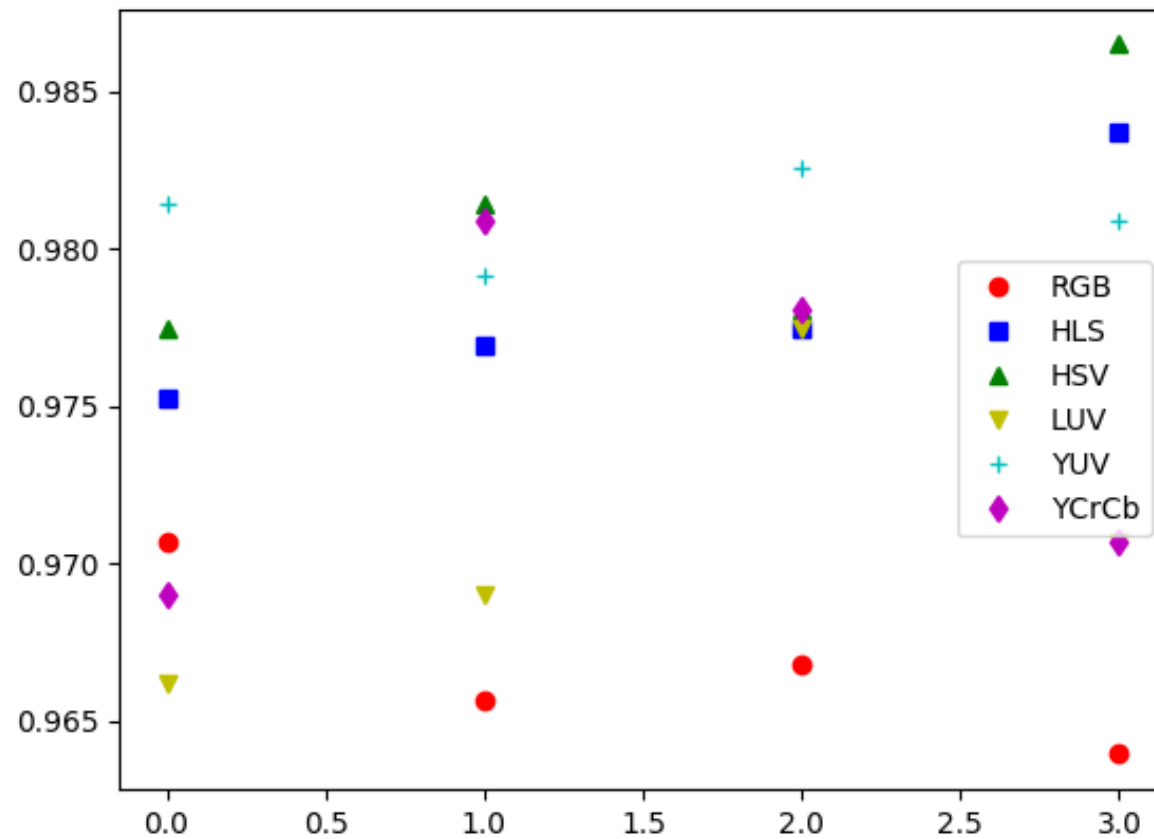


Color histogram features for bin sizes: 8, 16, 32, 64 (on x-axis)



HOG features for orientation histogram bins: 7, 8, 9, 10 (on x-axis)

HOG - Accuracy versus number of histogram bins using all color channels



Usually, the process of choosing values for these parameters is a process of finding a balance between accuracy and performance. So the third dimension we can try to add on these charts is time to train classifier or handle some debug frame.

Excellent articles with the basic description of the algorithm if you need more info besides lectures:

<http://www.learnopencv.com/histogram-of-oriented-gradients/>

<http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/>

The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier.

Well done with SVM!

I would also recommend to add discussion around your classifier:

- why did you choose it?
- did you try others? (you can try also, for example, Naive Bayes Classifier, Linear SVM, Non-Linear SVM, Decision Tree Classifier, MLPClassifier, CNN Classifier, etc)
- what result did you receive with them?
- what was their performance?
- etc

As an improvement you can also use grid search to optimize SVM C parameter:

http://scikit-learn.org/stable/modules/grid_search.html

It is not necessary by the specification but as optional improvement, you can also try to display a confusion matrix for made predictions. It can show how many true/false positives/negatives you have. Here is more info about it:

https://en.wikipedia.org/wiki/Confusion_matrix

Here is a discussion about the difference between several classifiers:

<https://www.quora.com/What-are-the-advantages-of-different-classification-algorithms>

And here are very interesting articles about image classification using SVM:

<https://pdfs.semanticscholar.org/fb6b/a3944cf1e534f665bc86075e0af2d2337eb9.pdf>

<https://www.ijcai.org/Proceedings/05/Papers/post-0001.pdf>

Sliding Window Search

A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen.

Nice sliding window implementation as per lectures!

Also for particular this video you can cut off your image on the x-axis. You are not interested, for example, in part of the image for opposite lane. So on the x-axis, you can cut off image somewhere between 400 and 500 pixels. But note that this will make your algorithm less general and on some type of roads it will not work - for example, if the car is in the central or right lane of the highway.

Also as the performance improvement for video pipeline, you can restrict the area to search the car based on results from the previous frame:

- search only around the previous detection
- and around borders of ROI

Here is an excellent article about sliding window algorithm implementation using python and OpenCV:

<http://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/ction-with-python-and-opencv/>

Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)

Well done with heatmap!

And here are some other methods you can use:

- use decision function with SVC:
http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC.decision_function
- hard negative mining:
https://www.reddit.com/r/computervision/comments/2ggc5l/what_is_hard_negative_mining_and_how_is_it/
https://www.researchgate.net/post/HOG-Is_there_a_relationship_between_hard_negative_mining_and_SVM_C_parameter

Video Implementation

The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video.

Good job with video pipeline! Excellent discussion about possible steps to reduce false positives. Here are some other methods you can use:

- increase threshold for heatmap (but be careful with true positives here, you can loose car detections)
- add more windows/scales to search car and adjust threshold appropriately (this decision is not the best due to performance issues)
- increase windows overlap for small images (this decision is also not the best due to performance issues)
- try different colorspace, channels, HOG parameters (refer please to plots provided in my comment to **Histogram of Oriented Gradients (HOG)** section above).
- process only each third frame, for example (this will also increase performance). The video is recorded in 25FPS. If we assume the car travels with speed of 100km/h, it passes about 28 meters per second or 1.12 meters per frame. Therefore, if you consider only every third frame, when the car moves for 3.3 meters. It is acceptable assumption, you will not notice this in final video, but if you use heatmaps it will decrease false positives and improve performance
- etc

I would recommend also to measure the performance metrics for implemented algorithm (for examples in seconds or milliseconds per frame) - speed is very important in it because all

data should be processed in real-time. Remember that python is usually used for prototyping and algorithm that will be setup on real car are usually written in C/C++ languages to increase their speed.

I like also how you combine your method with lane detector! 👍 Unfortunately, you didn't attach Lane Detection notebook, but as for me radius of curvature looks too small sometimes. Radius with values less than 100 meters can be found only on road with very sharp turns, like mountains roads.

A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur.

Well done with 2 consecutive frames! Your implementation is pretty smooth, however, I think, you can use more frames.

Discussion

Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.


You have an excellent final discussion with some very useful papers provided! Not sure that links below will be useful for you (maybe you already read them), but hope you will find something useful among them.

Here are some interesting articles about object detection with HOG:
<https://medium.com/@mohankarthik/feature-extraction-for-vehicle-detection-using-hog-d99354a84d10>
<http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf>
<http://lear.inrialpes.fr/people/dalal/NavneetDalalThesis.pdf>

And here are described a little bit another approaches to detect vehicles:
https://github.com/andrewssobral/vehicle_detection_haarcascades
http://www.rle.mit.edu/eems/wp-content/uploads/2016/06/dpm_vlsi_2016.pdf
<https://pjreddie.com/darknet/yolo/>

May be the most encouraging and perspective approach is to use Deep Learning to detect vehicles (but note please that in this case we will lose our full intuition on intermediate steps due to the "black-box" behavior of neural nets). Here are some good articles on vehicle detection using DL:
<http://www.cmu.edu/gdi/docs/a-neural-network.pdf> (very old article, may be just to know history of this field of knowledge)
<https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection>
<https://github.com/Lab41/attalos/wiki/SSD>

DL applied for this project:
<https://medium.com/@tuennermann/convolutional-neural-networks-to-find-cars-43cbc4fb713>
<https://github.com/HTuennermann/Vehicle-Detection-and-Tracking>
<https://github.com/maxritter/SDC-Vehicle-Lane-Detection>

Also you can follow this arxiv paper:
<https://arxiv.org/abs/1704.01926>
And here are video with objects segmented using this method:
https://video.twimg.com/ext_tw_video/850263795142336513/pu/vid/640x360/BcalA91yKeHiZ6Qy.mp4


And just for validation of algorithm a harder challenge video:

- what do you think will your algorithm be able to detect cars for example for the following video?
- what do you think you should do to upgrade it to be able to detect cars on it? 🤔



(Yes, it is a motorcycle 😊)

[📄 下载项目](#)

[返回 PATH](#)