Vehicle Detection and Tracking

By Erwan Suteau, Aug 27, 2017

# Histogram of Oriented Gradients (HOG)

## Parameter Selection

*Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.*

*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.*

The calculation of the histogram of oriented gradients is defined in the function get\_hog\_features() (line 28 of main.py).

The selection of parameters is based on trial and error. I started with the default parameters defined in the lesson, and then played around those values to see if the classifier would train better. I prioritized accuracy over speed.

I ended up with:

* Color Space: ‘YUV
* Orientations: 11
* Pixels per cell:32x32
* Cells per Block: 2x2
* Transform Sqrt: False
* Block Norm: ‘L2-Hys’

‘YUV’ seemed to perform better or equal to YCrCb. The classifier accuracy was significantly higher with one of those 2 color maps than RGB.

I selected a different norm function that the default one (L1) because the skimage library will use this normalization method in the future. I did not seem to make a huge difference from L1.

I also disabled the square root transform because it was causing problems with some images, returning nan values for small sections.

## Feature Selection

In addition to the HOG features, I also used the spatial features and the color histograms. All combined together, they seem to produce an excellent set of inputs for the classifier to train on. The classifier seemed to perform much better with all 3 combined.

## Classifier

Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

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.

Classifier training can be found in function run\_classifier(), line 322 of main.py

I chose to use Linear Support Vector Classification for this project, as it gave me around 99% of accuracy on the test set.

I used almost the entirety of the smaller training set provided by Udacity. I used the same number of images for each dataset (car and non-cars)

I made sure to normalize the dataset to zero mean and unit variance using the StandardScaler() class. (See function prepare\_feature(), line 306 of main.py)

The dataset was separated into a training set (80%) and a test set (20%). I used the built in score function on the latest to check the accuracy of my classifier.

# Sliding Window Search

## Implementation

Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

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.

My initial approach was the one introduced in the lesson, which for a give set of window size and stride, extracts all the windows and then calculate the feature maps (HOG, spatial and color histogram features).

I realized quickly that this was extremely slow, especially on my 6 year old Sony Vaio ☹. This i

## Examples

Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?

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.)

# Video Implementation

## Results

Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

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.

## False Positive Filtering

Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

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.

# Discussion

Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

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.