Tips and Tricks for the Project

Extract HOG features just once for the entire region of interest in each full image / video frame

In one of the previous exercises you extracted HOG features from each individual window as you searched across the image, but it turns out this is rather inefficient. To speed things up, extract HOG features just once for the entire region of interest (i.e. lower half of each frame of video) and subsample that array for each sliding window. To do this, apply `skimage.feature.hog()` with the flag `feature_vec=False`, like this:

```
from skimage.feature import hog
orient = 9
pix_per_cell = 8
cell_per_block = 2

feature_array = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cel
l), cells_per_block=(cell_per_block, cell_per_block), visualise=False, feature_vector=Fa
lse)
```

and down your image in x and y).

So, for example, if you used `cells_per_block=2` in extracting features from the 64x64 pixel training images, then you would want to extract subarrays of shape `(7, 7, 2, 2, 9)` from `feature_array` and then use `np.ravel()` to unroll the feature vector.

Make sure your images are scaled correctly

The training dataset provided for this project (vehicle and non-vehicle images) are in the `.png` format. Somewhat confusingly, `matplotlib image` will read these in on a scale of 0 to 1, but `cv2.imread()` will scale them from 0 to 255. Be sure if you are switching between `cv2.imread()` and `matplotlib image` for reading images that you scale them appropriately! Otherwise your feature vectors can get screwed up.

To add to the confusion, `matplotlib image` will read `.jpg` images in on a scale of 0 to 255 so if you are testing your pipeline on `.jpg` images remember to scale them accordingly. And if you take an image that is scaled from 0 to 1 and change color spaces using `cv2.cvtColor()` you'll get back an image scaled from 0 to 255. So just be sure to be consistent between your training data features and inference features!

Be sure to normalize your training data

Use `sklearn.preprocessing.StandardScaler()` to normalize your feature vectors for training your classifier as described in this lesson. Then apply the same scaling to each of the feature vectors you extract from windows in your test images.

Random shuffling of data

When dealing with image data that was extracted from video, you may be dealing with sequences of images where your target object (vehicles in this case) appear almost identical in a whole series of images. In such a case, even a randomized train-test split will be subject to overfitting because images in the training set may be nearly identical to images in the test set.

For the project vehicles dataset, the `GTI*` folders contain time-series data. In the KITTI folder, you may see the same vehicle appear more than once, but typically under significantly different lighting/angle from other instances.

While it is possible to achieve a sufficiently good result on the project without worrying about time-series issues, if you really want to optimize your classifier, you should devise a train/test split that avoids having nearly identical images in both your training and test sets. This means extracting the time-series tracks from the GTI data and separating the images manually to make sure train and test images are sufficiently different from one another.

For this project, we provide you with a labeled dataset and your job is to decide what features to extract, then train a classifier and ultimately track vehicles in a video stream. Here are links to the labeled data for vehicle and non-vehicle examples to train your classifier. These example images come from a combination of the GTI vehicle image database, the KITTI vision benchmark suite, and examples extracted from the project video itself.

Udacity recently released a labeled dataset of our own, which you are encouraged to take advantage of to augment your training data. You can find the Udacity data here. In each of the folders containing images there's a csv file containing all the labels and bounding boxes. To add vehicle images to your training data, you'll need to use the csv files to extract the bounding box regions and scale them to the same size as the rest of the training images.

The project video will be the same one as for the Advanced Lane Finding Project. The reason for this is that, assuming you already have a working implementation of lane finding for this video, once your vehicle detection pipeline works, you can add it to your lane finding pipeline and do both analyses simultaneously! You can use the test images from the lane finding project to start with, or extract other frames from the video to work on.

NEXT