vehicleDetection

gvenkatasaik

August 2017

1. 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.

Ans. A histogram of gradient directions is computed in each cell. From each of the pixels in the cell the gradient samples are distributed into 9 orientation bins. Local shape information is often well described by the distribution of intensity gradients or edge directions. The cells can be rectangular or can be circular, (R-HOG) or (C-HOG). Capture edge or gradient structure that is very characteristic of the local shape. Each pixel votes for an orientation of the gradient. The vote is weighted by the gradient magnitude. The gradient at each pixel is calculated as $gradImage_y/gradImage_x$. To calculate the HOG descriptor, we need to first calculate the horizontal and vertical gradients. This is easily achieved by filtering the image with the following kernels and the $directionofthegradient = arctan(g_y/g_x)$. Calculating a histogram over a patch makes the representation more robust to noise. Individual gradients may have noise, but a histogram over 8x8 patch makes the representation much less sensitive to noise. 8x 8 patch is a design choice need to be made. The parameters for the HOG feature selection are:

- Bin count
- Number of pixels per cell.
- Number of cells in the given image for normalization.

Block Normalization: Gradients of an image are sensitive to the overall lighting. Ideally, we want our features to be independent of the lighting variations. L2 normalization is done on the histogram of the HOG. But even than normalizing the 8x8 cell, it would be better to normalize over a 16x 16 block for better generalization.

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

Ans. An SVM classifier is used for training upon the data.

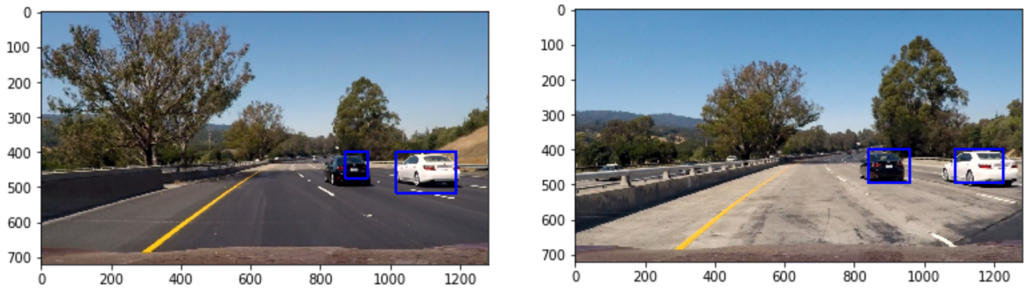
3. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

Ans. Sliding window search is implemented by scaling the images larger or smaller. The window size of searching is kept constant which is 64x64. And the obtained bounding boxes are scaled back to the original image size.

```
## scale: Scale of the image.
xbox_left = np.int(xleft*scale)
ytop_draw = np.int(ytop*scale)
win_draw = np.int(window*scale)
```

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

Ans. Below are some of the test images.



Grid Search CV is used for optimization of the Support Vector Classifier. The parameters used are kernels:(linear, (rbf) Radial Basis Function), C:(1,10).

5. Provide Link to the video.

Ans. The video is uploaded to the Github.

6. 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?

Ans. The following difficulties are faced during the implementation of the pipeline.

- Scene understanding like YOLO or SSD Framework which can regress through all the bounding boxes in the image can work better than this pipeline.
- This pipeline is not suitable for real time applications as each iteration takes place around 4 seconds.
- Object tracking can be used to speed up the time taken for each iteration. For example, by searching in areas, where the pipeline has found some car in the previous frame.
- As the vision systems are not completely reliable and may lead to the false alarm situations, LIDAR data can be used to precisely detect vehicles on the road.