Project Writeup

Anirudh AV (anirudh.av93@gmail.com)

---

1. HOG feature extraction

- The key function used to extra t features was the **hog** function imported from the skimage.feature library.
- The arguments in the **hog** function had default values of 9 orientation bins, 8 pixels per cell, 2 cells per block.
- The features for a given set of arguments were extracted in the **getHOG** function. The code uses another wrapper function called the **getFeatures** function to call the **getHOG** function while accommodating for different color spaces.
- The training images namely, the car and non-car images were loaded from the supplied folder using the glob function which is used in the **LoadTrainingData** function.
- Once the images were loaded, the code uses another wrapper function to extract the features and ”massage” the features into test and training feature and label vectors. This is defined in the **PrepareData** function which return the training and test vectors used to build the classifier.

2. HOG parameter selection

- Arpan had suggested using the saturation as a good differentiator for the car vs non-car pixels. I used the YUV colorspace with 11 orientation bins and 16 pixels per cell. These parameters only have an empirical trial and error explanation so far. They could have a more detailed deterministic mathematical proof for coming up with the parameters.

3. Building a classifier

- The code uses a linear svm classifier trained on the training data provided by Udacity. The buidling and training is performed in the **BuildAClassifier** function.
- The classifier gave an accuracy of 98.3% which was obtained on the test data , implemented in the **Evaluate-Classifier** function in the code.

4. Sliding Window Implementation

- The implementation of the sliding window was mostly drawn from the course material.
- The **SlidingWindow** function was copied from the course material and another wrapper function called **CombineWindowSearches** was used to implement multiple sliding window searches with different window parameters on the same image.
- As suggested in the course material, the cars would only appear in the lower half of the image and thus the search region was cropped to the lower half of the image.
- The different scales used in the **CombineWindowSearch** function are 1, 1.5, 2, 3.5.

5. Video Implementation

- A heatmap image was generated in the **HeatMap** function and then passed through the **TesholdImage**. This acts as a way of filtering out the false positive detections of the car in an image.
- Keeping track of where the cars appear in the image seemed like a useful technique. Jeremy shannon a student from another cohort had shown that, similar to the line class in the previous project, we can use a vehicle class to store the rectangle position in an image across different frames in the video.

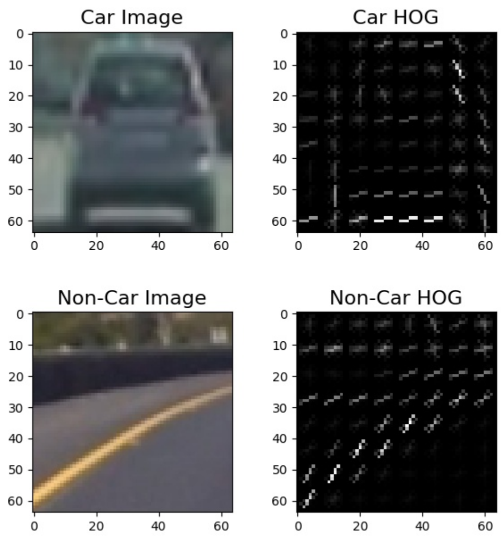

Figure 1: Hog features example

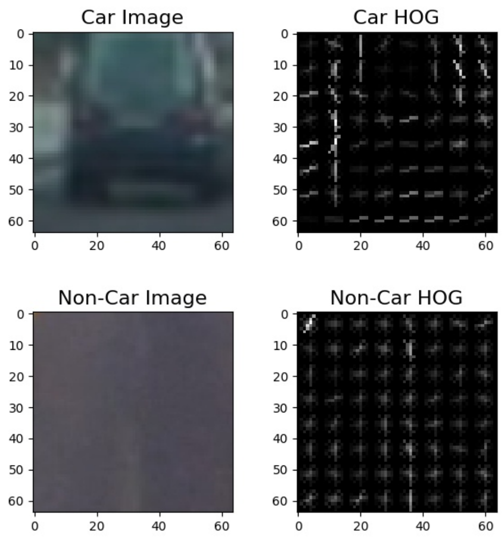

Figure 2: Hog features example

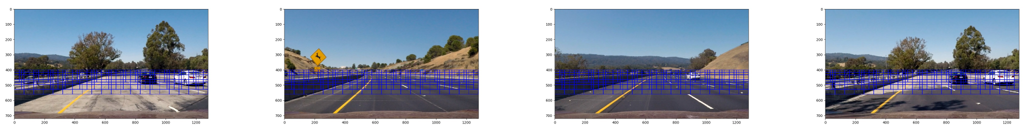

Figure 3: Candidate Windows

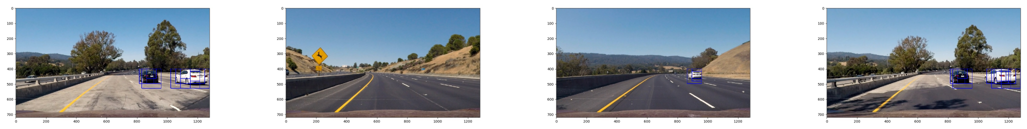

Figure 4: Windows in which Cars are Found

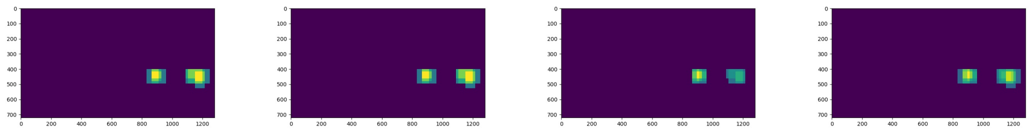

Figure 5: Heatmap Image

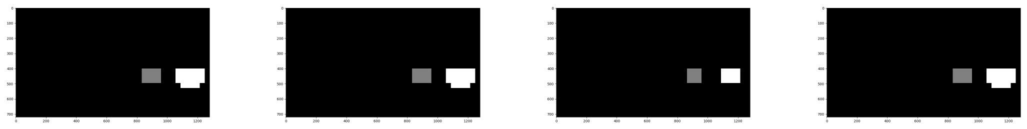

Figure 6: Threshold applied and then labeled

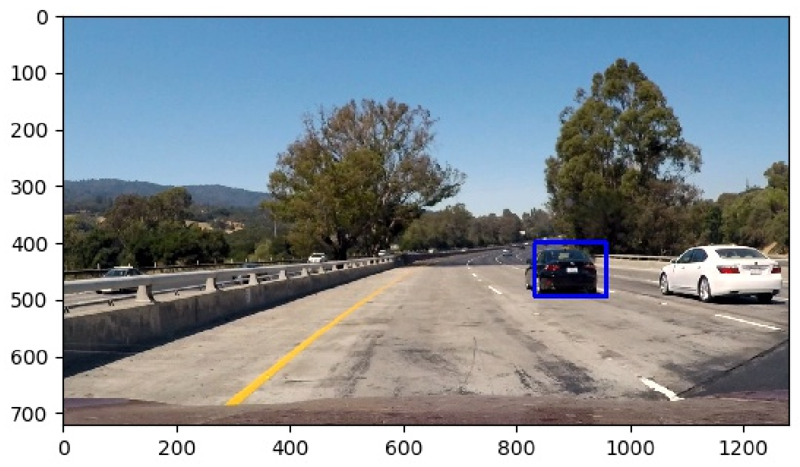

Figure 7: Single rectangle drawn around centroid of all rectangles around the car

6. Limitations

- The images are all in the day - time for which the hog channel selected worked, it may fail during night time / rainy days.
- The detected hog features were for cars, might fail for trucks and bikes (worse than a false positive)
- This algorithm detects vehicles on the opposite direction of traffic even if separated by a divider, something to watch out for, might give false information.
- If cars are on a bridge far away, they fall in the window search region, they might appear as false detections , there should be a filter implemented to detect cars only on the plane of the road.