Pablo Sauras Perez Reflection.

Term 1 – Project 5 – Vehicle Detection

1. Files Submitted

This project submission has the following files:

- **P5_Vehicle_Detection.ipyng** the code of the project.
- **output_images/** is a folder that contains the outputs of the project (images and video).
- **project_video_result.mp4** is the output video of this project.
- **PabloSauras_Reflection_P5.pdf** is this report.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

This is the writeup. Please, read the rest of the sections to see how I covered the project and each rubric point.

Histogram of Oriented Gradients (HOG)

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.

Cell2: I started by reading all the `vehicle` and `non-vehicle` images. Latter in the project I will use a subset of them to train the SVM. I also performed in this cell some preliminary data exploration, such as the total number of vehicle and non-vehicle images, and the number of images in each folder (dataset).

Cell 3 is a function to read some basic characteristics of the datasets.

Cell 4 provides a visualization of a vehicle and non-vehicle example.
Figure 1 shows an example of these images.

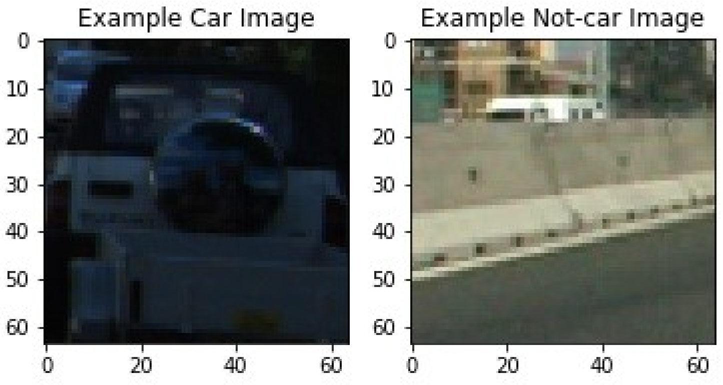

Figure 1: Example of Car image and Non-Car image.

Cell 5 and 6: I then explored HOG feature extraction using `skimage.hog()` and different parameters (`orientations`, `pixels_per_cells`, and `cells_per_block`), so that I can have a feel of how HOG outputs looks like. This time I converted the sample images to grayscale. However, for the real pipeline implementation I used other color space conversion. **Figure 2** shows an output of the HOG extraction function. **Table 1** shows the used parameters for this preliminary HOG visualization.

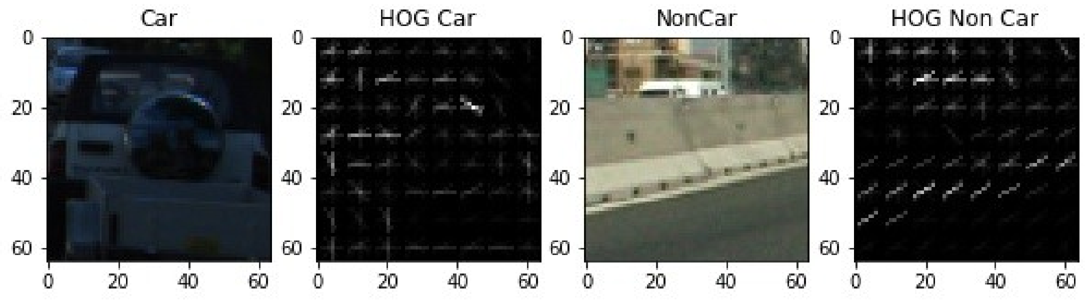

Figure 2: Preliminary HOG extraction visualization.

Table 1. Preliminary HOG extraction parameters.

Color Space	Gray
Orientation	9
Pixels per cell	8
Cells per block	2

Explain how you settled on your final choice of HOG parameters.

Basically, I combined HOG feature extraction with bin spatial and color histogram feature extraction (functions in **Cell 7**) In **Cell 8** we can see the definition of the `extract_features()` function, that combine spatial bin, color histogram and HOG.

As, I will explain latter, the feature extraction was used to train a **Liner SVM** classifier. I tweak the HOG and rest of the parameters so that the classifier testing accuracy was maximized. In this case, a **test accuracy** of **0.9912** was reached.

In **Cell 10 (lines 1 - 7)** I extracted the features of **2000 sample images of cars** and **2000 sample images of non-cars**. I randomized the selection process, so that I am not ‘biased’ by the image selection.

Figure 3 shows an example of this feature extraction. **Table 2** shows the final set of parameters.

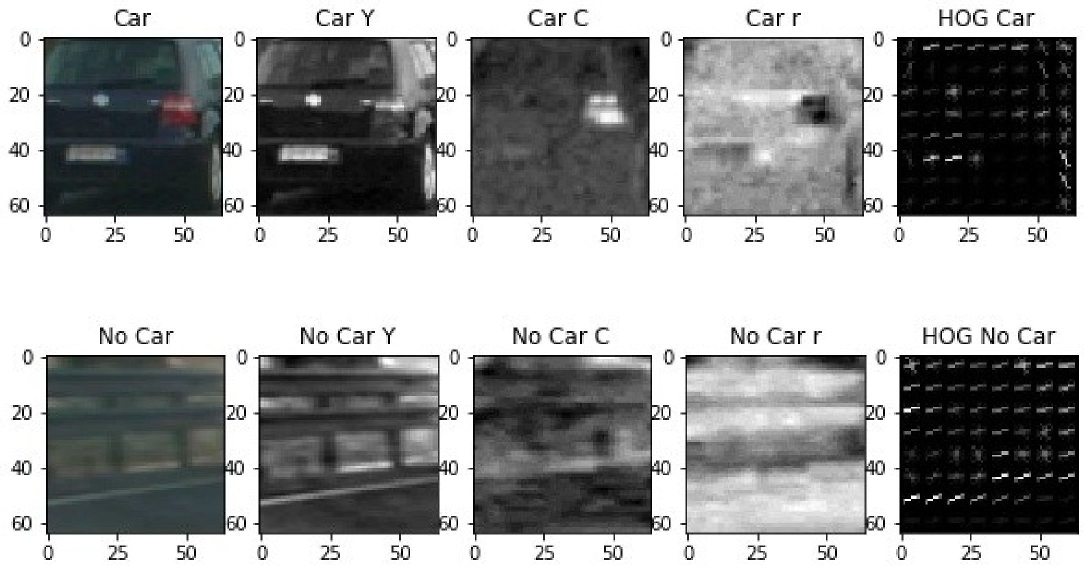

Figure 3: Example of feature extraction. From left-right: RGB image, Y, Cr, Cb channels, HOG. YCrCb colorspace was used.

Table 2. Parameters used for feature extraction

Colorspace	YCrCb
HOG orientations	9
HOG pixels per cell	8
HOG cells per block	2
HOG channel	ALL
Spatial binning dimensions	(32, 32)
Number of histogram bins	32
Spatial features	True
Histogram features	True
HOG features	True

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).

As explained before I used **HOG feature**s, **spatial features** and **color histogram features** for extracting the list of images features. 2000 random images for cars and 2000 random images for non-cars were used to form the feature vectors.

Cell 11: I scale the feature vectors, so that they are **normalized** to have zero mean and unit variance **(lines 2-6).** I also **split the data** into randomized training and test sets (80% training, 20% testing).

Cell 12: I trained a **Linear SVM**. In my case, the testing accuracy was **0.9912.**

Sliding Window Search

1. 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?

Cell 14 and 15: I decided to use HOG Sub-sampling window search. As stated in the lessons, this is a more efficient way for doing the sliding window search, allowing to extract features only once.

The function `find_cars()` in **Cell 14** allows to extract the features at once (remember that I used spatial, color histogram and HOG features) and make predictions.

I used `scale = 1.5` **(Cell 15)** and `cells_per_steps = 2`. This gives a search window overlap of 75%. Although it is possible to run this function with different scale values, I only called it once. I decided the value of these parameters based on the detection of vehicles in test images. As I will explain latter, I also used heatmaps to remove false positives, the correct detection after the heatmap step also influenced the tweaking of the window search parameters.

The values `ystart = 400` and `ystop = 656` define the area of interest of my window search.

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

Cell 16 and 17: To improve the results of my classifier and remove false possitives, I applied **heatmaps. Cell 16** defines the required functions `add_heat()`, `apply_threshold()` and `draw_labeled_bboxes()`.

In **Cell 17** I applied these functions to test images. I used a threshold value of 1 to help remove false positives.

Figure 4 shows a clear example of how the application of heatmap removes false positives.

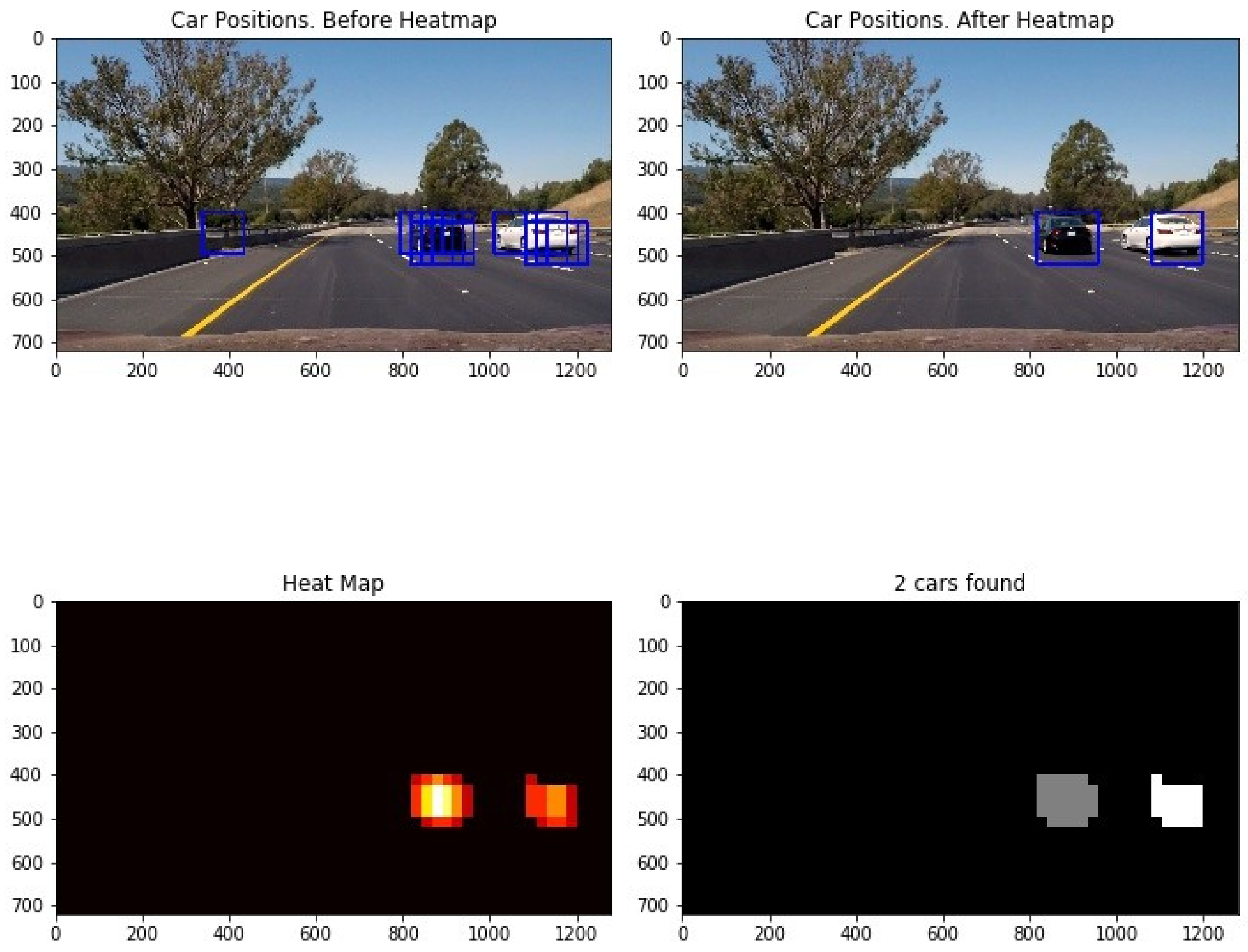

Figure 4: False positive removal by appliying heatmaps.

The rest of the test images results can be seen in the folder output_images.

Video Implementation

1. 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.)

My final video can be seen in this link (https://github.com/pablosaurasperez/CarND-Vehicle-Detection/blob/master/project_video_result.mp4).

As it can be seen the vehicles are identified most of the time. However, there are still some false positives.

2. 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.

Cell 20 defines the function `process_image()` that runs the pipeline over the video frames. In order to improve the results and smooth the detection of vehicles, I used collections of heatmaps to consider several video frames for the heatmap calculation process. For my case, the maximum length of the deque is 20 and the threshold used for removing false positives is 1. It is worth noting that some of the “false positives” in reality seem to be vehicles in the other direction.

Discussion

1. 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?

The results of my pipeline detect and track most of the time vehicles that are close to my ego-vehicle. However, I have noticed that the detection of vehicles further away or in the opposite direction are not properly detected. This can cause problems in the long term, as my ego-vehicle could be interested in events that are further away or in the opposite direction. For example, in a two way road, I may be interested in tracking vehicles in the opposite direction as well. I think this may be solved by providing training data of smaller vehicles. However, even with that, I think the feature extraction may be challenging, as the images are smaller.

Another challenge is the heatmap detection over several frames in order to smooth the video and filter false positives. This part is challenging to tweak and other parameter may be better to have more accurate results.

Regarding the color space, I notice that the one I selected performs well in the project video. However, more analysis should be done for other weather conditions such as rain or fog. Also, I remember the challenge videos of Project 4. There was one video recorded in a narrow two way road in a mountain, with very different lighting conditions (even direct sunlight in the camera). I think this pipeline would not perform well in that video. Moreover, the window search has a fix area of interest in my pipeline. It would be nice to have an adaptive area of interest for sliding window search.