P5- Vehicle Detection

Udacity SDCND Term 1

Daniel Tobias

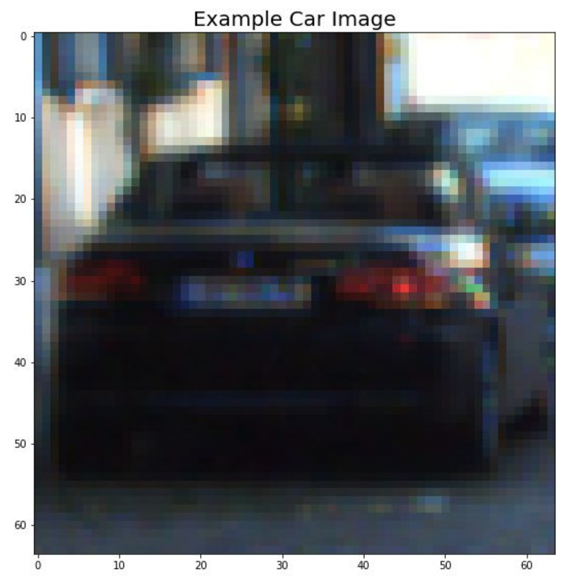
The goals / steps of this project are the following:

- The goals / steps of this project are the following:
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.
- Run the pipeline on a video stream and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Histogram of Oriented Gradients (HOG)

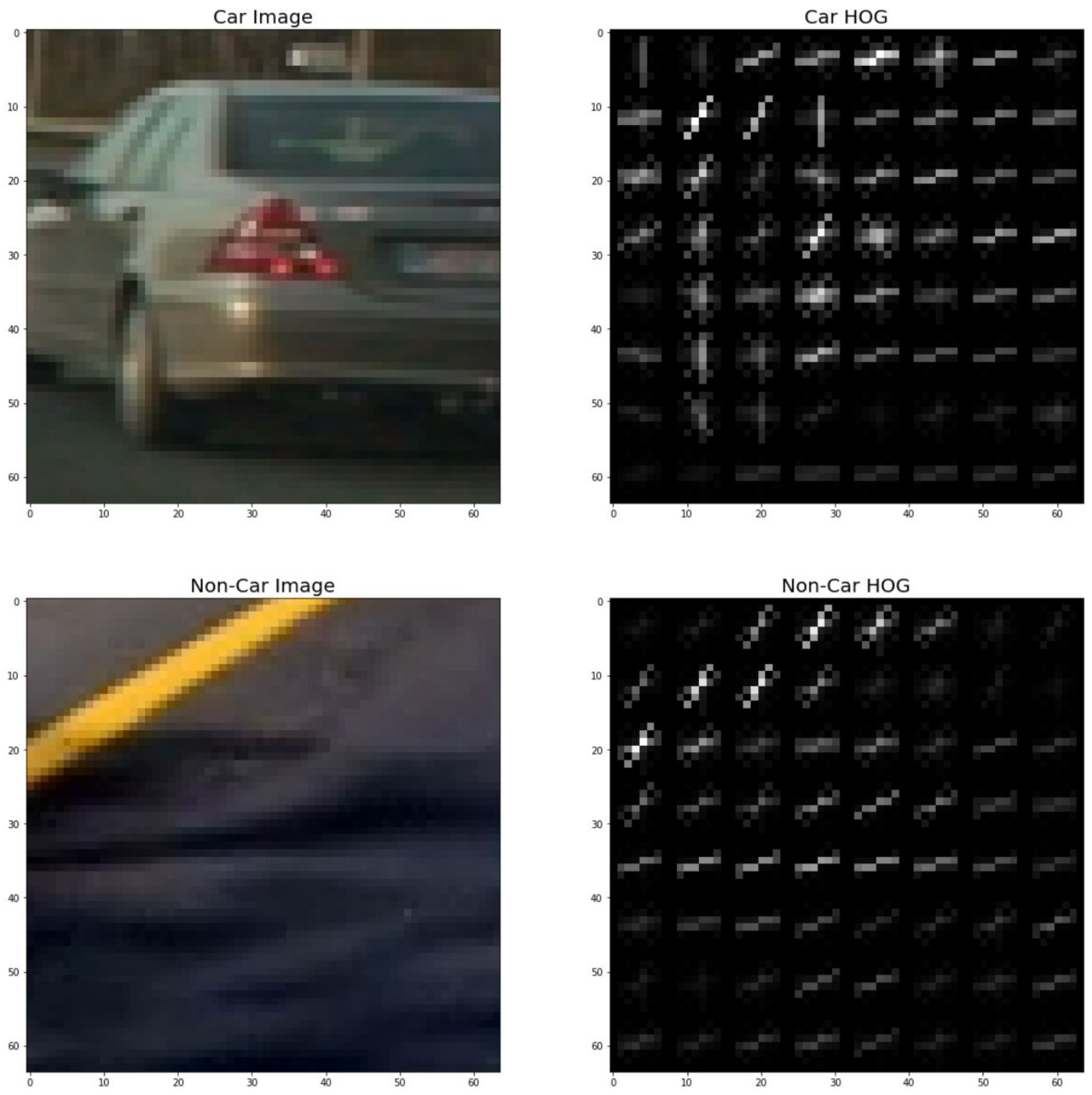
1. Load in images

The first step consisted of loading in the locations of all the images in the labeled image dataset vehicle and non-vehicle. Then the same process was done for the test images, which can be seen near the end of the report. Here are a few of the images contained in the vehicle and non-vehicle dataset, this happens in the 3rd cell block of accompanying notebook.



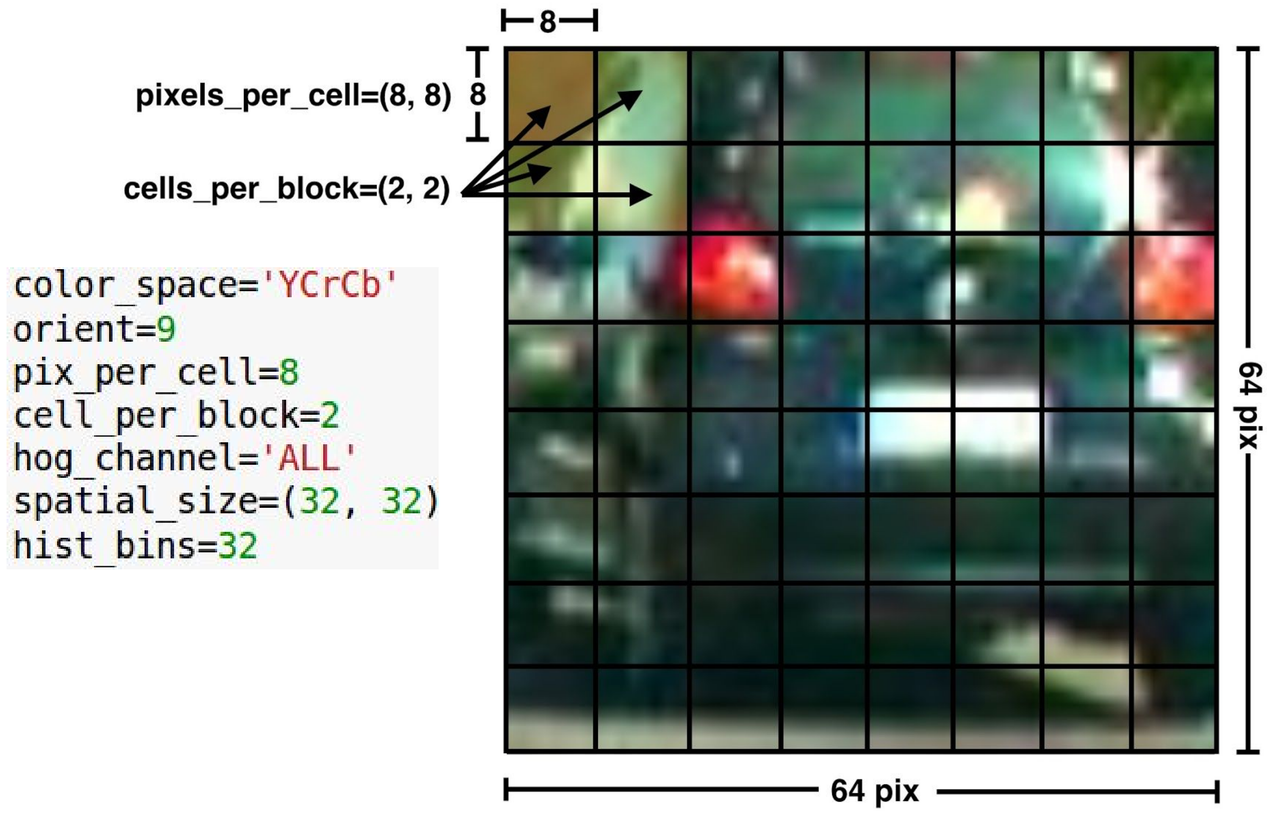
2. Visualize HOG features

To visualize the the HOG operating, images from vehicle and non-vehicle dataset were selected and loaded in. Then the HOG operation was performed on both images, below is the visualization of the original image and the opposing HOG image. This happens in the 5th code block in the notebook.



2.b Exploring HOG parameters

After trying a few different parameters, it was noticed that using the YCrCb color space and all three color channels of the original image worked well. Most of the other settings like were borrowed from the lesson with the exception of spatial size which was shrunk. Increasing the amount of orientations to 18 didn't produce any noticeable difference in the video, but slightly increased the hog extraction time.



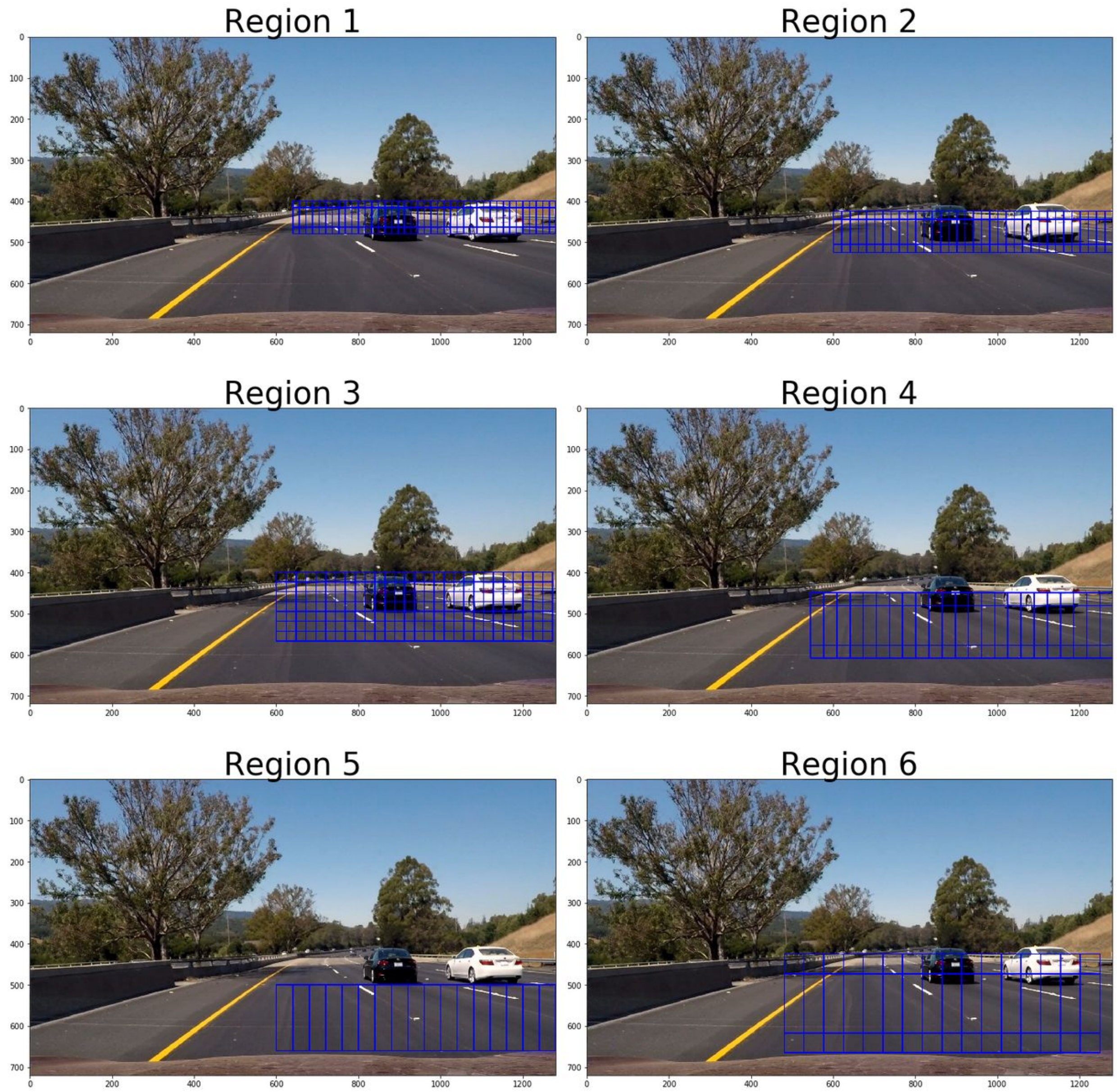
```
color_space='YCrCb'
orient=9
pix_per_cell=8
cell_per_block=2
hog_channel='ALL'
spatial_size=(32, 32)
hist_bins=32
```

3. Training the classifier.

First was to extract all the hog features from both datasets, this can be seen in code block 6. Then the images were normalized, shuffled and split into training and testing sets. These sets were used to train a linear SVM to classify between vehicle and not vehicle. Due to the shuffling of the datasets, the accuracy of the classifier based on a testing set of 20% fluctuated around 99.2% ± 0.0-0.2%. The training and results can be seen in code block 7 of the notebook.

4. Sliding window regions

The main approach used to pick regions to search for a car was to stack multiple regions of different height on the Y axis and use different box sizes. This can be seen in the image below. Additional parameters were added to the find_car function which can be seen in code block 4 line #154. This let us set the starting X pixel position for defining the search regions.

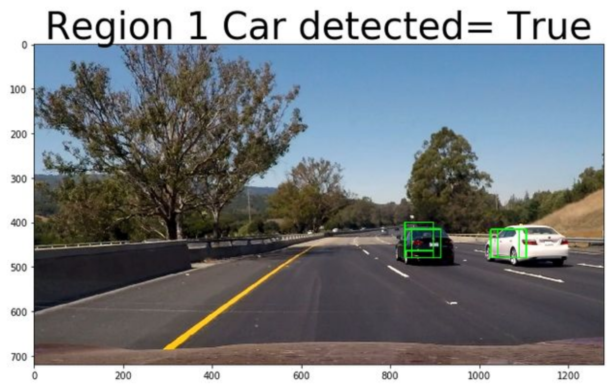


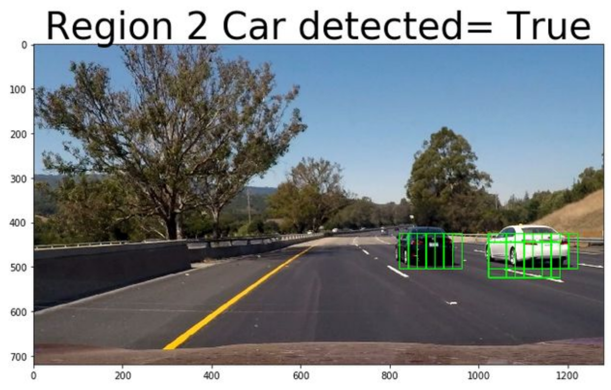
The overall idea of how to define regions was to increase the size of the box search region as you descended on the Y axis. Adding additional regions would improve the accuracy of the pipeline at the cost of calculation time.

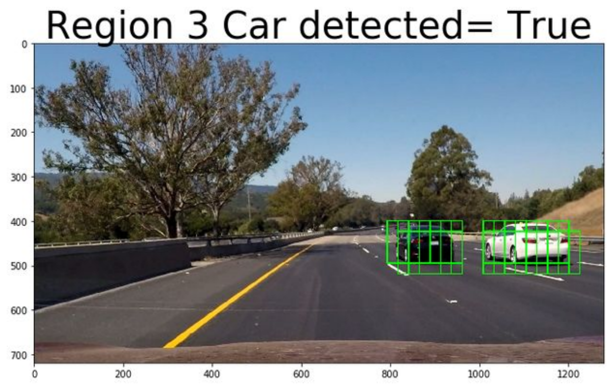
P5- Vehicle Detection
Udacity SDCND Term 1
Daniel Tobias

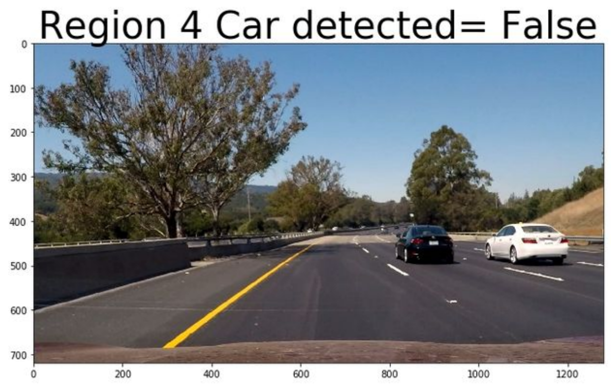
4.b Sliding window regions

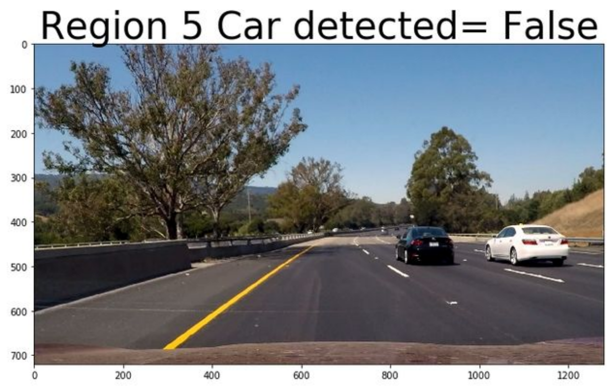
In the image below we can see the squares that detected a car denoted with a green bounding box. Also it is import to note that region 4,5,6 did not detect a car. This is because if we look at the image above and compare it with the image below we can see that the regions in question do not fully overlap the car. Code block 9

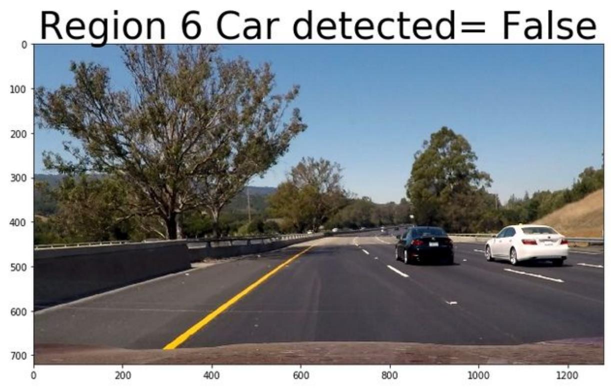






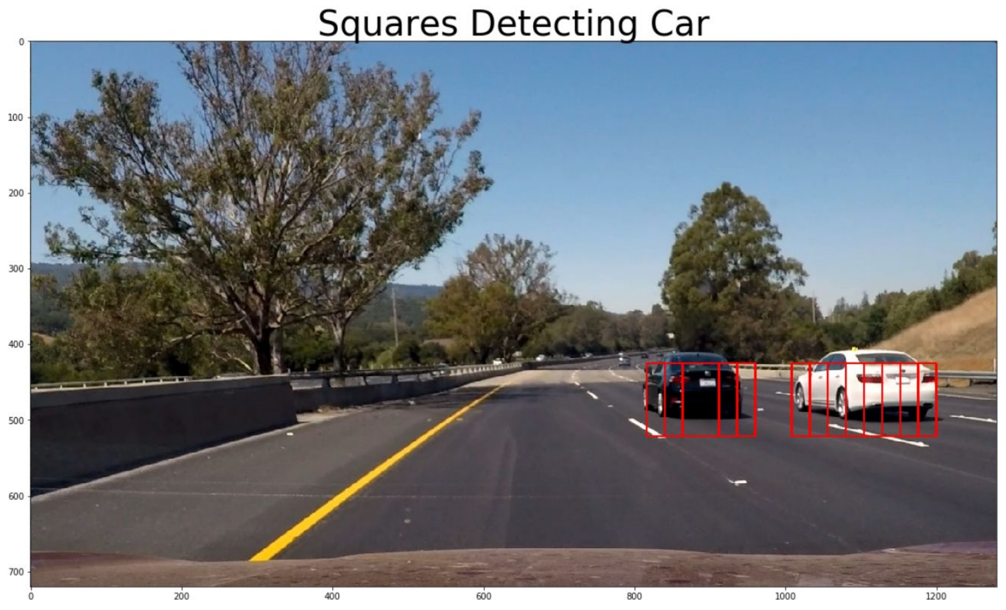


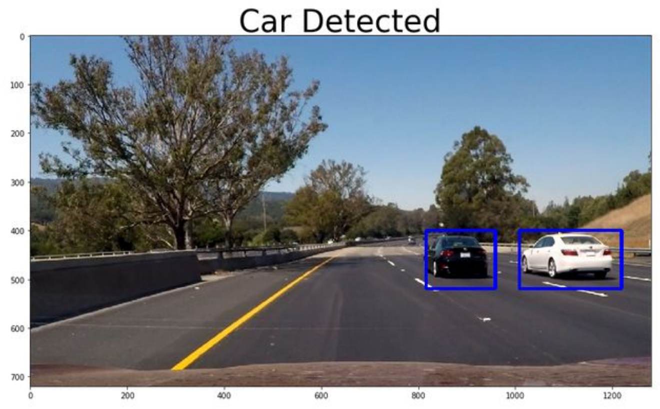


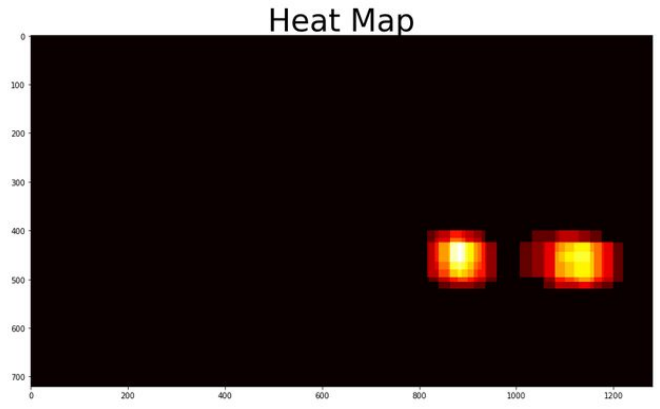


4.c Sliding window regions heatmap

Using each region, a heatmap was produced that showed the overlapping of the individual boundary boxes, this can be seen as the right image Heat Map. Then by calculating the min and max values for x,y pixels a box was drawn, this can viewed as left image below. A threshing operation was done also to essentially remove pixels that did not overlap other pixels. This helps remove false positives by allowing only areas with multiple detection regions get counted. Code block 10-15



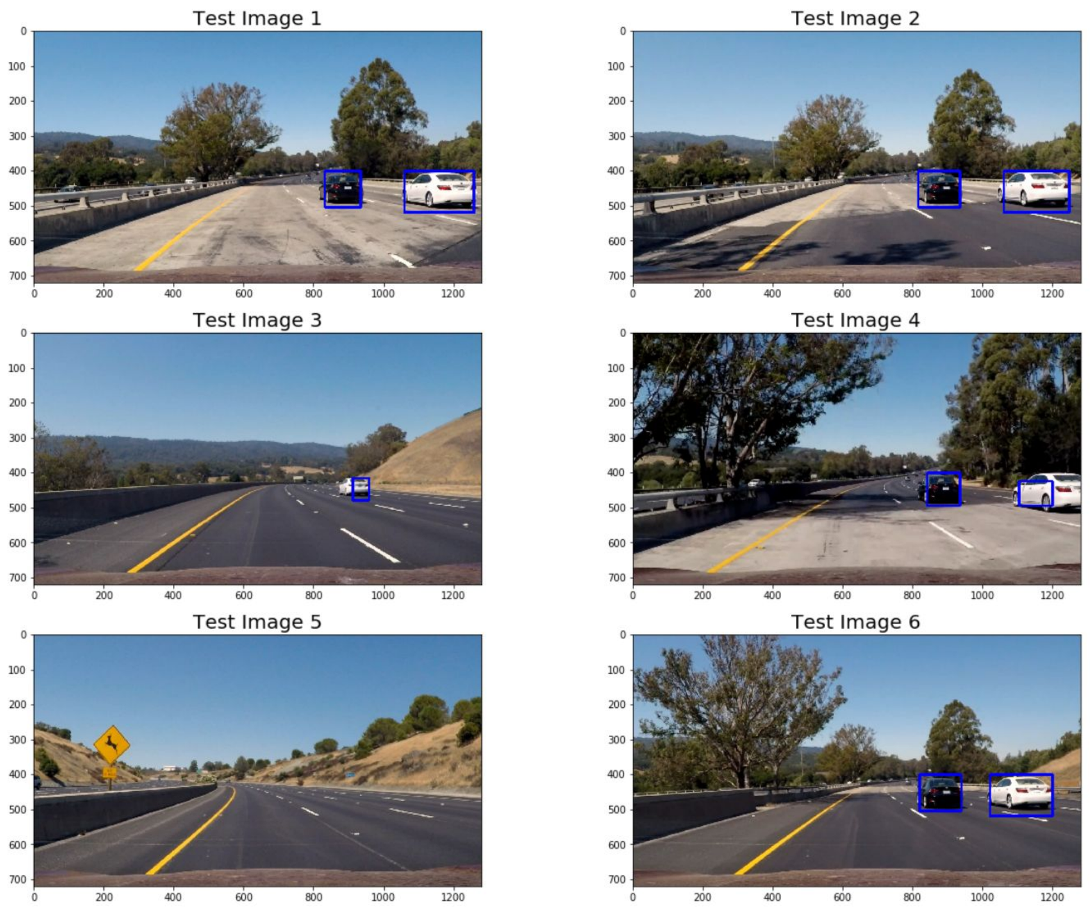




5. Pipeline single images and video

The single image pipeline consisted of using the trained classifier on the different overlapping regions described in **sliding window section** of this report. Squares that detected a car were then transformed into a heatmap with the points of the rectangles being saved. The heatmap was then filtered to remove weak classification pixels, and the min and max x,y pixels of the heatmap were used to define the final boundary box seen in the image below.

A modified version of the single image pipeline was used for video. The difference were the creation of a memory system that remembers a few previous frames and subsequent heatmap. In the case of this project it remembers the past 8 frames and used the sum of this to define the final heatmap. This helped to smooth out the boundary region for detecting the cars. There was also a more rigorous attempt at threshing the images since multiple false positives would stack up faster in this version of the pipeline. This can be seen in code block #19



The Pipeline used on the test examples.

P5- Vehicle Detection

Udacity SDCND Term 1

Daniel Tobias

Briefly discuss any problems / issues you faced in your implementation of this project.

You could see a difference in results regarding the classifier and the test images/final video depending on the randomization and split of the testing and training data. Over multiple attempts to finding a good set of parameters it was noticed that sometimes the white car in test image 4 was not detected correctly. Sometimes in the video there is a false classification that can be seen in the highway median. This was solved by reducing the region so that it only be searching areas where a car could potentially be.

Where will your pipeline likely fail?

The pipeline has problems detecting vehicles that are far ahead. This is mostly due to the regions not having sufficiently small squares to detect, you can see this in my video where the white car pulls further ahead then pack again. Adding a memory system helped so that even if there was a brief disconnect it would handle it. Might not work well in different lighting conditions and weather. It may get confused on signs of similar height, due to the shape.

What could you do to make it more robust?

An approach would be to algorithmically define the regions and add more ‘layers’ to the overlapping regions. A feedback system, or ML approach that would use the good detection heatmap region to focus fire more detection regions in that general area might work well.