Vehicle Detection

By: Tyler Zamjahn

APPROACH

For this project I began with the various functions provided in the lessons, and then tuned the parameters until I had a result that successfully output bounding boxes around each vehicle. My approach was then to ensure that I minimized false negatives to roughly zero and then filter out the false positives through frame to frame averaging. I fine-tuned the parameters and produced all the images used in this report in a jupyter notebook and then transferred the essential code to a python file for execution on the videos.

DATA SET

The data set consisted of 8792 images that are cars and 8968 images that are not cars. After compiling a list of the images in both categories, I then saved the information using the pickle method for efficiency.

CODE

Histogram of Oriented Gradients (HOG)

Parameters and Color Space

In order to choose parameters for the HOG features and color space, I used the quizzes in the lesson to determine which parameters gave the best accuracy initially and then tuned the parameters slightly in order to maximize accuracy and minimize vector length. I tested the accuracy of a linear SVM holding spatial and histogram bins constant and varied the color space in the Color Classify Section of the Object Detection Lesson. The accuracies are shown rounded to the nearest tenth of a percent.

Color Space	Accuracy Run 1	Accuracy Run 2
RGB	97.7	97.1
HSV	96.9	97.7
LUV	98.8	98.5
YUV	98.3	99.0

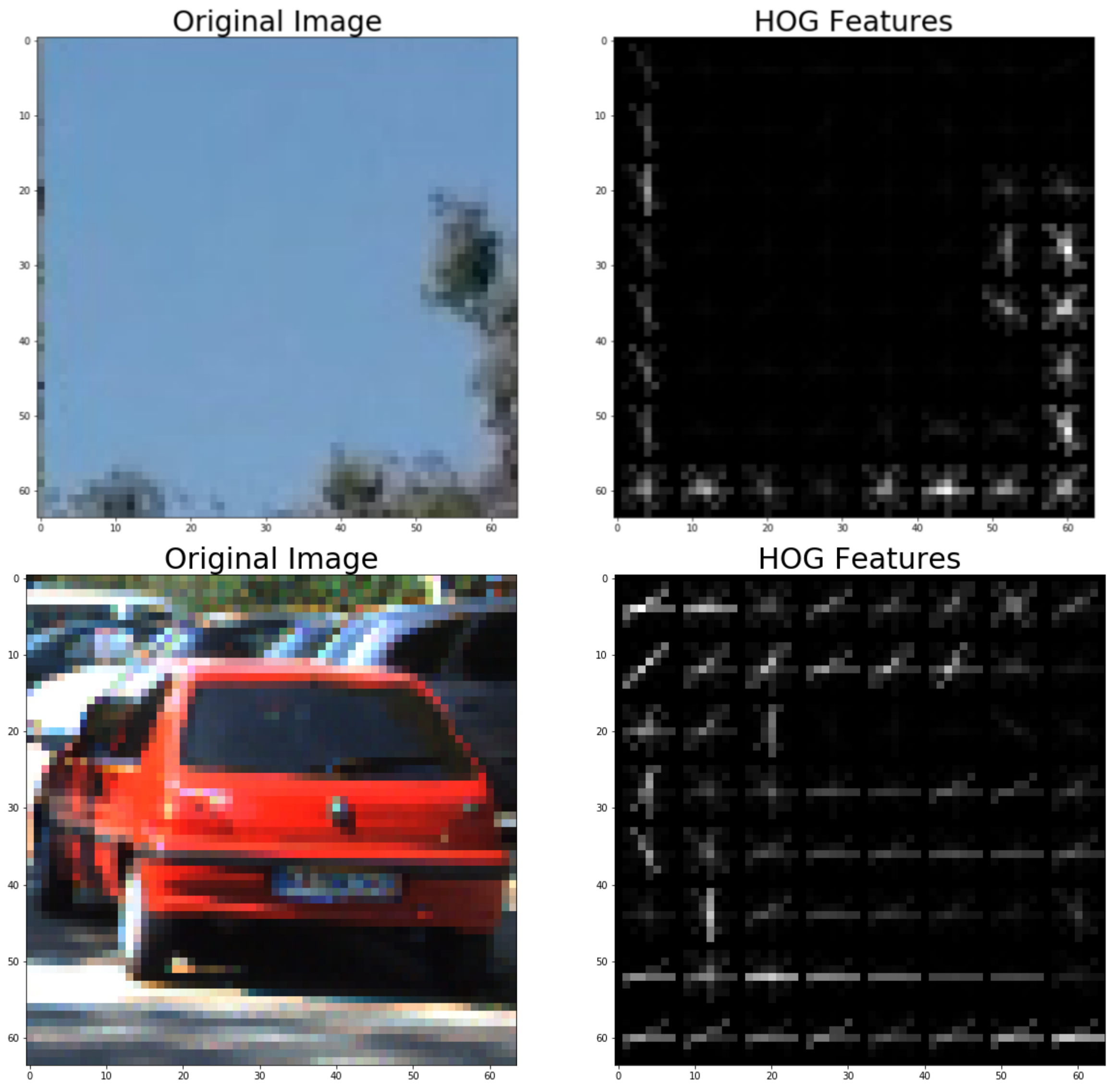
Additionally, I then modified the number of spatial and histogram bins, holding the color space constant and found that 32 worked well for both parameters.

I then tested the accuracy of a linear SVM holding gradient orientation bins, pixels per cell, cells per block, and the hog channels used constant and varied the color space in the HOG Classify Section of the Object Detection Lesson. The accuracies are shown rounded to the nearest tenth of a percent.

Color Space	Accuracy Run 1	Accuracy Run 2
RGB	92.5	90.5
HSV	94.0	90.5
LUV	95.0	96.5
YUV	96.0	97.5

YCrCb	96.5	96.0
HLS	91.0	94.0

There were some variations from run to run because the sample size was set to 500 images of cars and non-cars, but YUV stood out as the one of the top choices in both tests. Additionally, I tested the pixels per cell, cells per block, and number of orients in the sci-kit-image HOG section and found that 7-9 orientation bins, 8 pixels per cell, and 2 cells per block produced images that were readily recognizable as cars. The following images show resulting HOG Features for the chosen HOG parameters on a grayscale image.



Training the Classifier

After fine tuning the parameters I then trained various linear SVM models using slight tweaks of the parameters I found above to maximize the resulting test accuracy and minimize the length of the feature vector. In order to achieve a satisfactory accuracy level I used the following parameters:

Color Space	YUV
HOG Channels Used	ALL
Pixels Per Cell	8
Cells Per Block	2
Gradient Orientation Bins	8
Histogram Bins	32
Spatial Bins	32

It took 108.15 seconds to find the feature vectors for all of the car images and 133.4 seconds to compute the vectors for the non-car images. It then took 121.94 seconds to train the linear SVM with a training accuracy of 99.04 percent. I then saved the X_scaler and svc files needed to execute the linear SVM using the pickle method to be called for further testing.

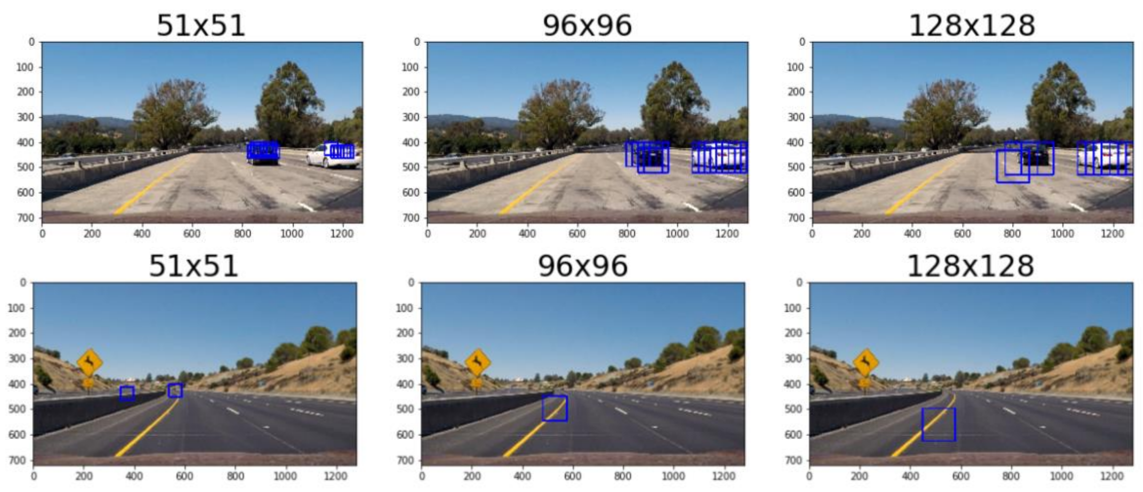
Sliding Window Search

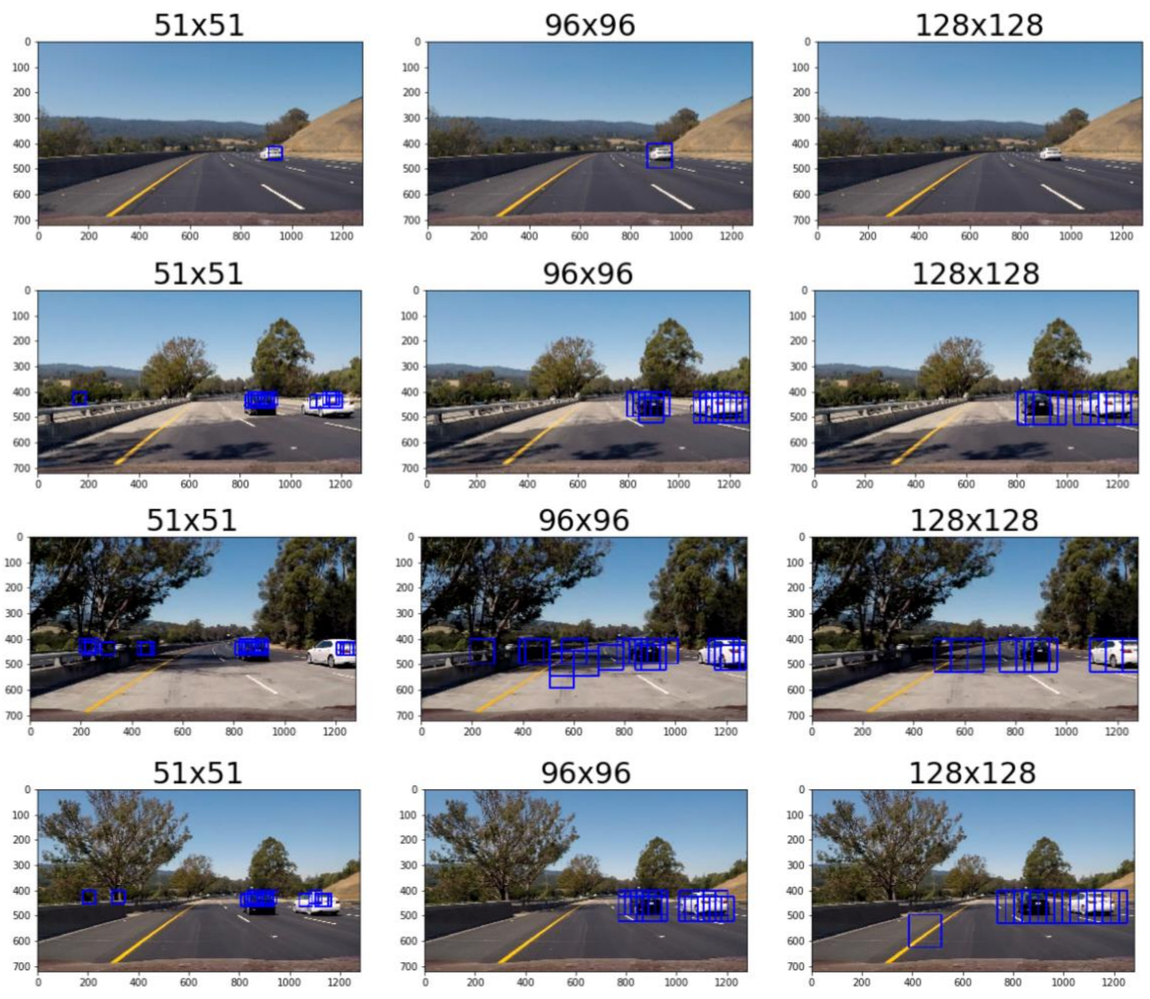
Implementation

For efficiency, I used the find_cars() function and scale parameter to set three different sliding window searches. The windows were chosen based on the different scales that cars appear in the images so that each window would traverse the frame and be roughly the size of a car. The three scales chosen were .8, 1.5, and 2 which correlate to a window size of 51x51, 96x96, and 128x128 respectively. Each search was limited to the region where cars of that size could appear. The first search was limited in the y direction to pixels 400 through 464, the second search was limited from 400 to 592, and the third from 400 to 656. The cells per step was kept constant at 2 to create a 25 percent overlap between windows.

Examples

The following images show the resulting bounding boxes produced by each search on the six test images given in the repository.





There are some false positives in the images which will be filtered out in the next section, but the resulting bounding boxes from each search window are combined in to one bounding box list for that individual image.

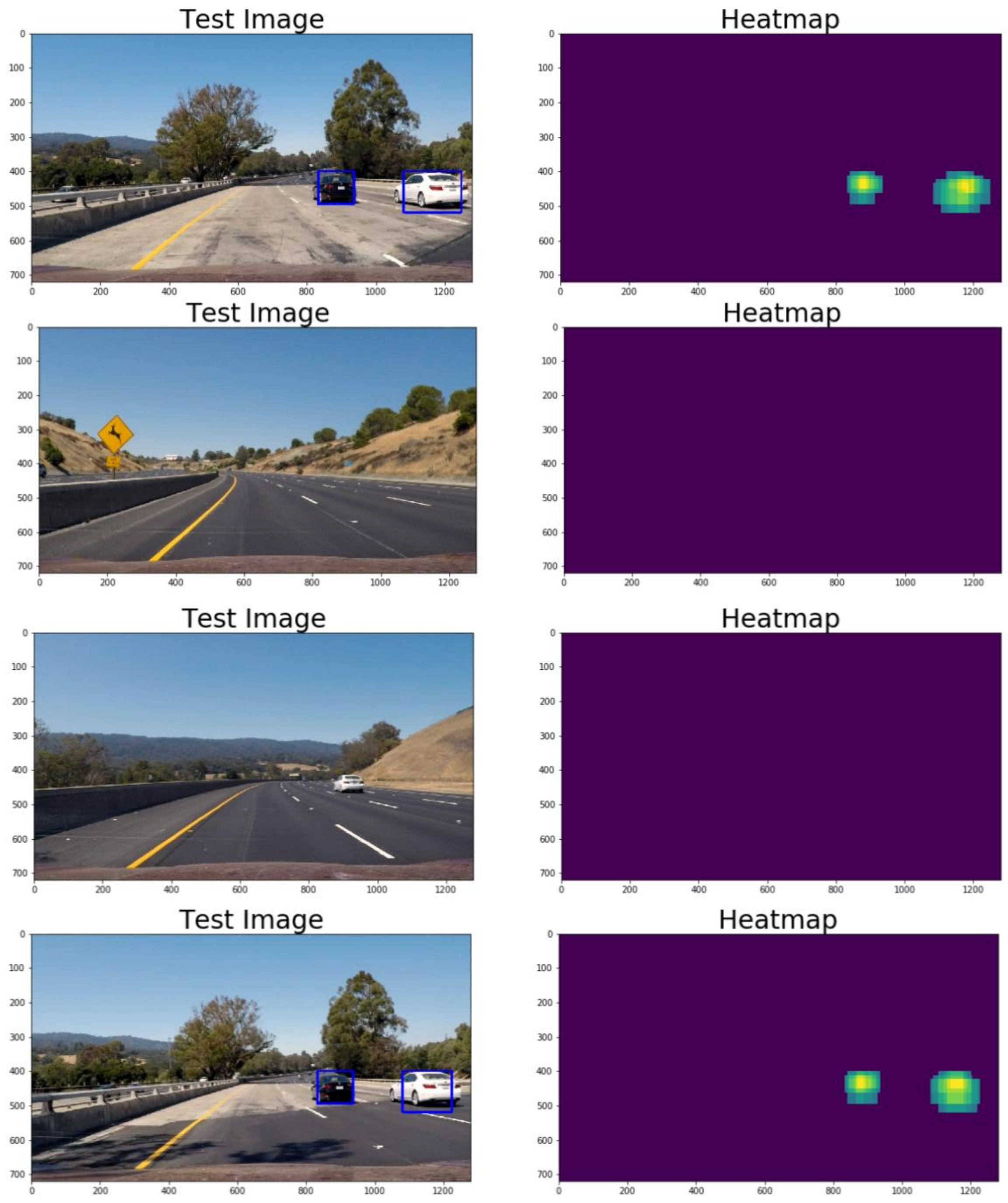
Video Implementation

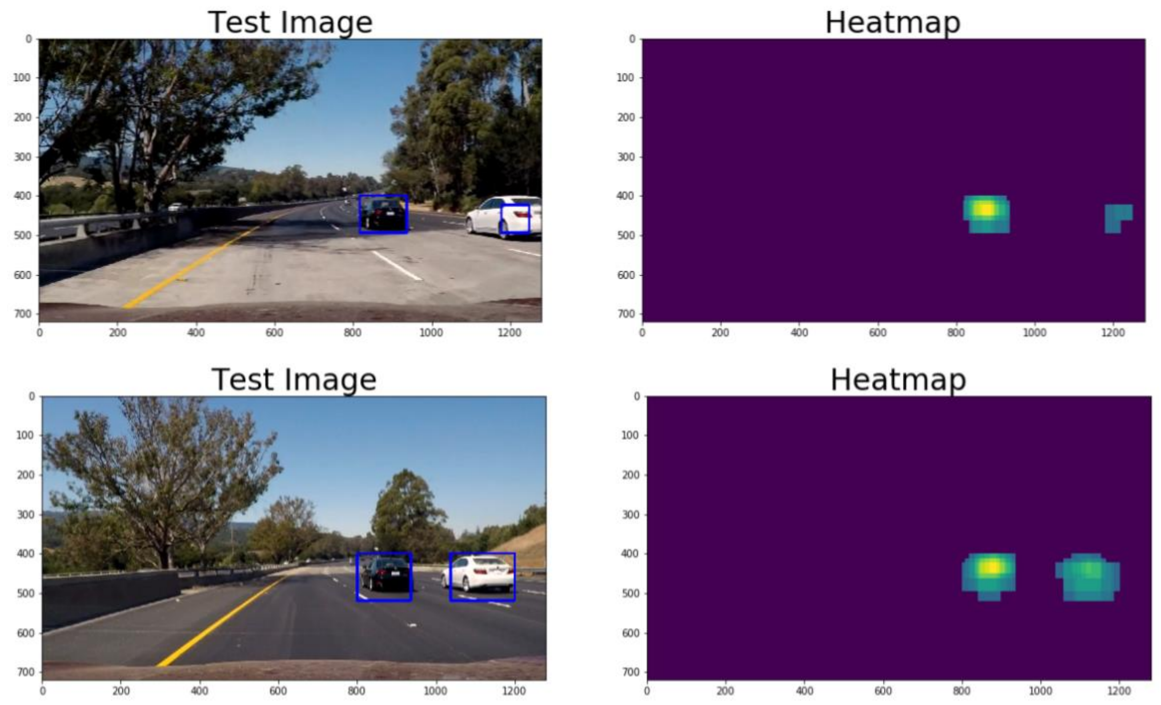
Process

After implementing the sliding window search and outputting the list of bounding boxes representing positives for each image, the positives then go through two filters to eliminate false negatives.

Eliminating False Negatives

The first filter eliminates any false negatives on the individual image using heat map thresholding. This image then should contain solely bounding boxes that represent cars with a small allowance for small false positives. The resulting heat map is then put into a queue, built using a class function called last five, with the last four images in the video sequence. The five heat maps are then summed to form a new heat map and the new heat map is then filtered using a higher threshold. The outputs of the dual thresholds are shown below for the test images:


Test Image
Heatmap
Test Image
Heatmap
Test Image
Heatmap
Test Image
Heatmap
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



The threshold for the new image is 5 times the number of images in the queue, while the threshold for each individual image is 4. If a car fails to cross the threshold in a single image, it can still be represented by a bounding box if the previous four images have formed a dense bounding box around the car.

RESULTS AND ROOM FOR IMPROVEMENT

The resulting video correctly identifies the other cars in the immediate surrounding area of the car that is videoing and does not include any false positives that would cause the car to suddenly stop. The classifier and subsequent thresholds have a hard time identifying cars that are farther away and near the middle of the y axis of the image because of the limited amount of hits from the sliding window search. Further fine tuning of thresholds and possibly even creating different thresholds for the smaller window search would make the classifier more accurate in picking up vehicles that are further away.

An additional improvement in the efficiency and possible accuracy of the classifier would be to use a CNN to allow for faster classifying of each window. Using a CNN would allow for improved speed and scaling by allowing for GPU usage. Another possible improvement would be to change the cells per step size to reduce the number of windows searched.

Finally, the threshold values could be lowered to allow for more robust tracking of cars in the horizon and edges of the video, but this could also produce false positives which can be dangerous. This project focused on finding a pipeline that eliminated all false positives and produced an acceptable tracking of the cars in the image. This pipeline could be combined with other logic, path estimation, or hardware tools to produce a safe representation of the other cars on the road.