Vehicle Detection

By: Brendan Kam

The goal of the project was to create a pipeline to detect vehicles in a video. In this report, the steps in designing the vehicle pipeline are discussed. The pipeline was design based on traditional machine learning algorithm, and it was able to detect vehicles in single images and in the video provided for the project. The code for this project is titled 'vehicle_detection.ipynb' which can be found in the zip file for this submission.

Feature extraction

The first part of the pipeline was to extract features from the training images provided for the project. Three different types of features were used in the pipeline. They are spatially binned color features, color histogram features and histogram of gradient (HOG) features. The functions to obtain these features can be found in code cell 4.

Figure 1 shows random samples of the car and non-car training images provided for the project.

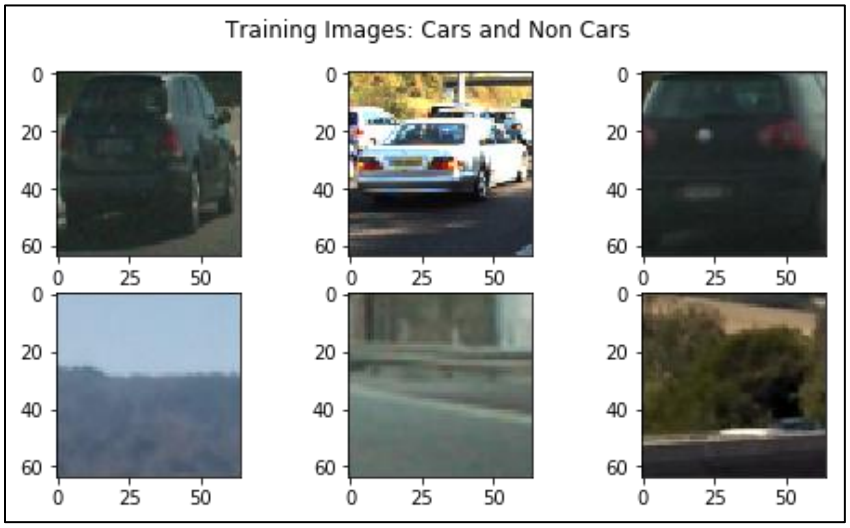

Figure 1: Sample Training Images

Through experimentation, it was found that the Lab color space had the best training performance in terms of accuracy and time needed to train the data. Therefore, the Lab color space was used in this project.

The first feature obtained was the spatially bin color features. From the spatial binning of color features, the machine learning algorithm was able to learn from the shape and color of the images. To obtain this spatial binning of color feature, an image of size 69x69 pixels is reduced to a size of 11x11pixels. By resizing the image, the resolution is lowered, however, certain features of the car can still be recognized. The new size was obtained based on a series of reiteration to obtain the highest prediction accuracy during the training of the images. Figure 2 shows the effect of spatially binning the color of an image.

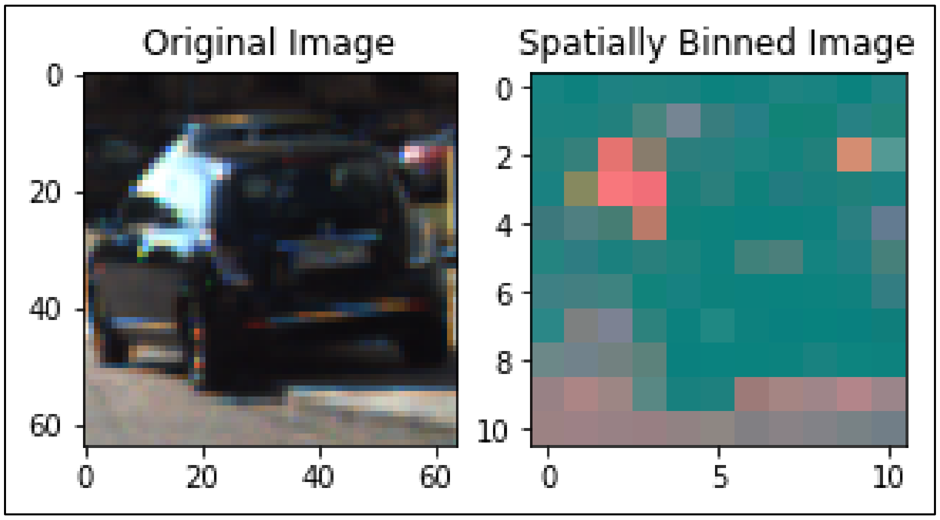

Figure 2: Spatial Binning of Color Features

The next feature obtained was the color histogram features. The color histogram features were obtained by splitting the image into its three-color channels, then sorting the colors in each channel into 32 different color bins and then obtaining their histograms. Figure 3 shows the three channels of the color histogram features of the car image shown in figure 2.

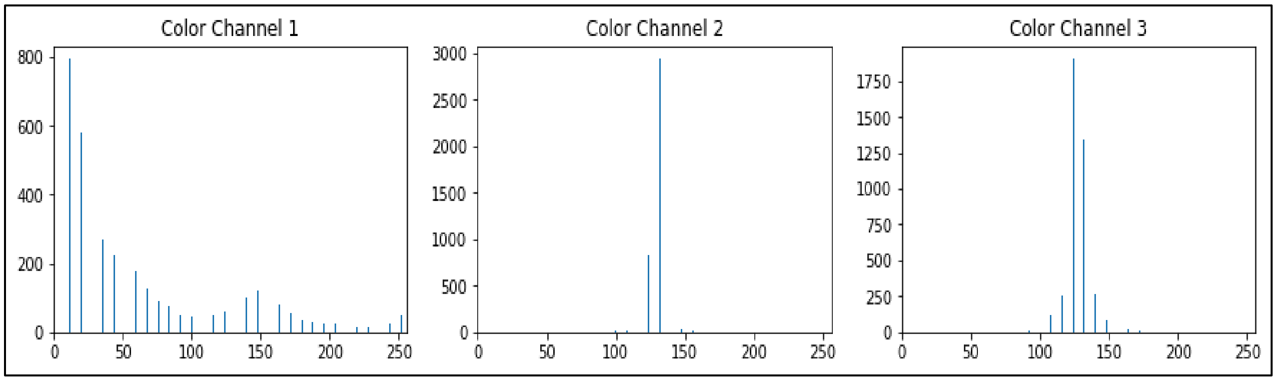

Figure 3: Color Histogram Features

The final features to be extracted are the HOG features. The HOG features were obtained using the skimage.feature.hog module. Three different parameters were used to obtain the HOG features which are the number of orientations, the pixels per cell, and the cells per block. Through trial and error, it was found the number of orientation with gives the best prediction accuracy is between 9 and 11. Since there was no significant increase in prediction accuracy between 9 and 11 orientations, the number of orientation was set to 9. The next parameters are the pixels per cell. This parameter was again chosen through reiterations. If was found that 16 pixels per cell in the x and y direction would provide the best vehicle detection in the single images and video with the least amount of false positive. It also significantly reduced the training time from the 8 pixels per cell used in the class lesson code. Finally, the cells per block was set to 2x2, this was again chosen due to having the best accuracy found through experimenting different values.

Figure 4 shows the HOG features for the 3 color channels of the car image shown in figure 2.

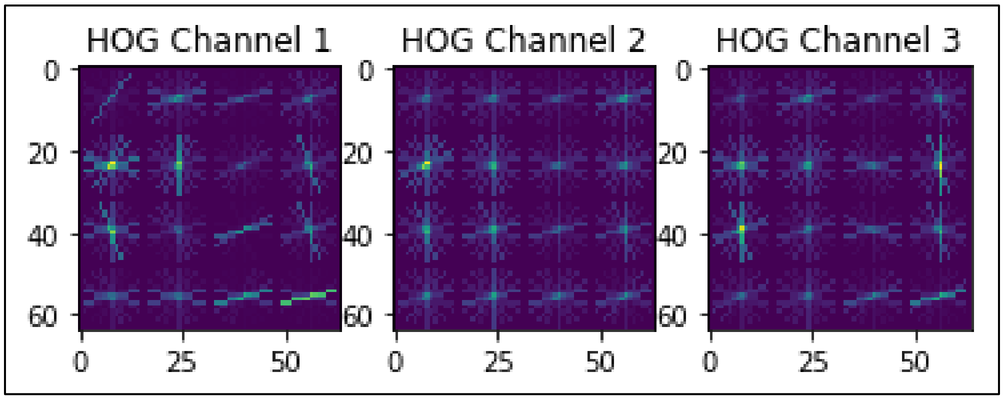

Figure 4: HOG Features

After extracting the 3 features described, the features were concatenated and appended into a list. There were then normalized using the sklearn.preprocessing.StandardScaler module.

Classifier

Having extracted the features, the next step of the project was to use a classifier to train the features so predictions can be obtained. Before training the data, the sklearn.model_selection. train_test_split module was used to shuffle and split the training images, so a subset of the training images can be used for accuracy verification.

The classifier used in this project was a linear support vector machine (SVM). The classifier was obtained from the sklearn.svm.SVC module. A linear SVM and an ‘rbf’ SVM were tested in this project. Since the linear SVM was able to better detect vehicles, it was chosen for the final classifier. Through this classifier, a prediction accuracy of up to 99.3% was obtained. The code for the classifier can be found in code cell 9.

Sliding window search

Having trained the classifier, the next step was to detect vehicles in single images and in videos. This was done by using a sliding window search. This method slides a search window across the image, features of the image are extracted and the classifier is used to predict if a car feature is within the search window. If a car is predicted to be within the window, the window is then saved, to be drawn onto the image or video frame. The functions used for the sliding window search can be found in code cells 5.

For the sliding window search, three different window sizes with different overlaps and search spaces were used. The sizes and overlaps used were chosen for their accuracy in predicting cars while avoiding false positives through trial and error. Through experimentation, it was found that different overlaps could significantly change the results of the car detected and whether a false positive appears. Their starting and end positions were chosen based on the assumption that cars farther away would be smaller in the image, hence would need a smaller search window, while a closer car would need a larger search window. Their starting point was chosen on the road horizon based on the assumption that cars would only go as far out as the road horizon.

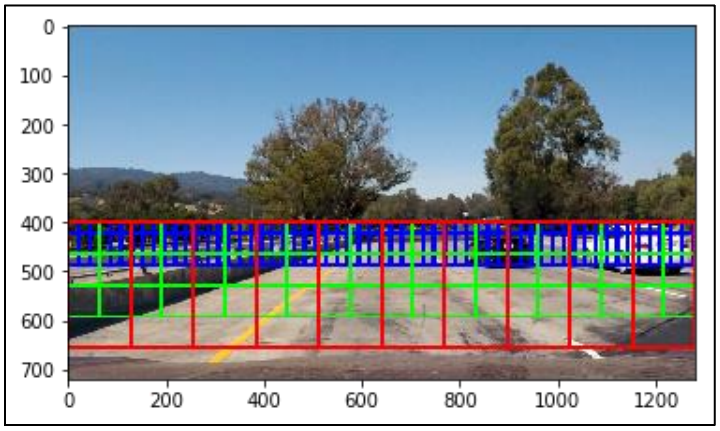

Figure 5: Different Sized Search Windows

Table 1: Search Window Parameters

Window Size	Overlap	Start and End of Search Space (y – direction)
64x64	0.65 x 0.65	400 - 500
128x128	0.5 x 0.5	400 - 600
256x256	0.5 x 0.5	400 -720

Figure 5 shows the different search windows plotted on an image, while Table 1 shows the parameters used for the search window.

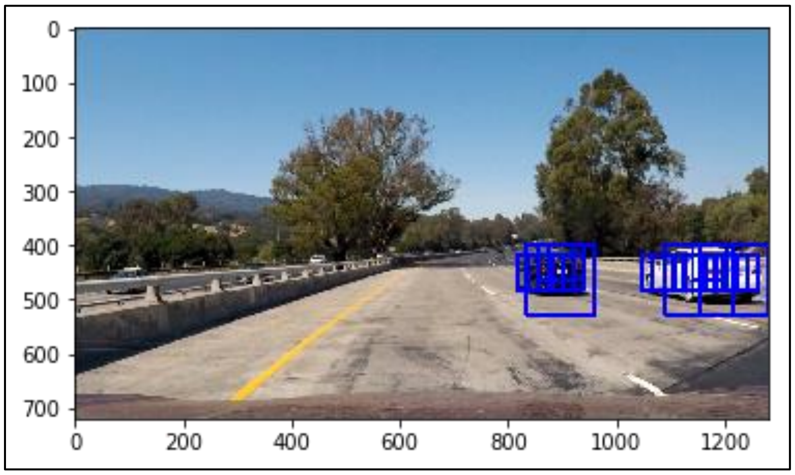

Figure 6: Vehicle Detection (Test 1)

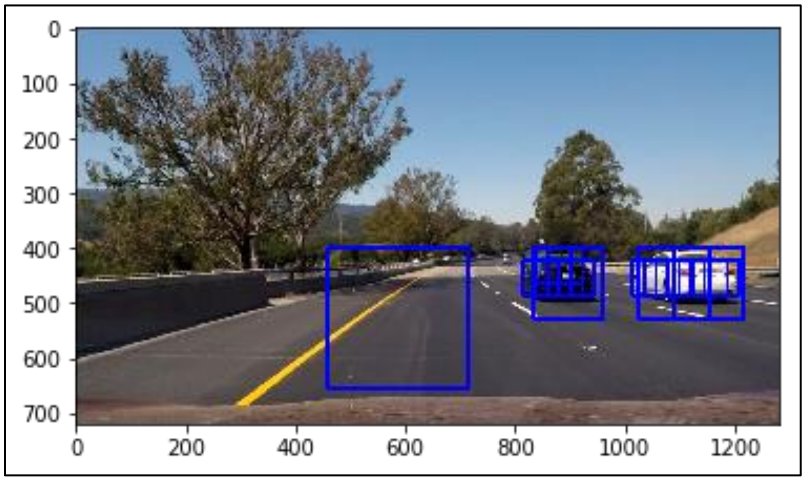

Figure 7: Vehicle Detection (Test 2)

Figures 6 and 7 shows the performance of the vehicle detection pipeline on test images provided for the project. As can be seen in figure 7, there was a false positive. Through trial and error, there were several different ways to reduce the number of false positive. The first way was to find change the color space when training the image. Changing the pixels per cell in the HOG feature extraction also helped in reducing the false positives. Finally, changing the search window size and their overlaps played the biggest role in reducing the number of false positives.

Figure 8 is a picture taken for cars in my city. The vehicle detection pipeline was able to detect the vehicles in the picture accurately.

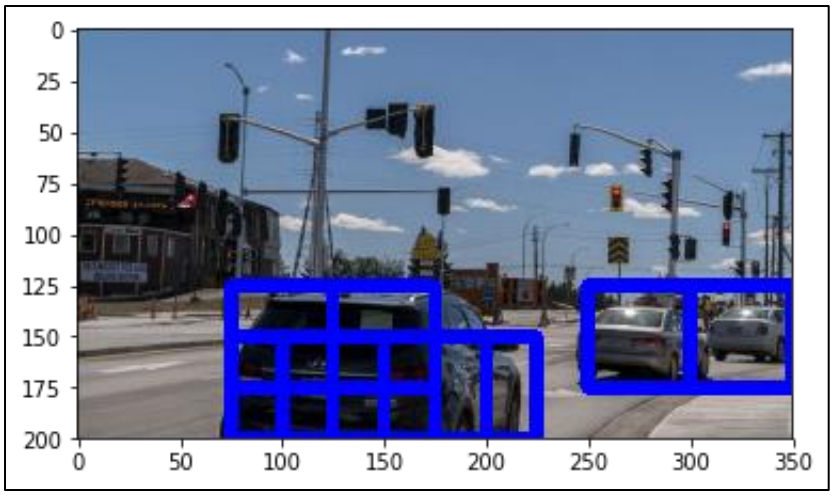

Figure 8: Test image of my hometown

Removing False Positives & Video Implemantation

To remove the false positives from the image and video frames, a heatmap was implemented. The functions used for the heatmap can be found in code cells 6. The basic principle of the heatmap is to find the number of overlapping vehicle detection. The higher the detections, the more likely it is that the image is a true positive. By setting a certain threshold to the heat map, false positives are minimized or removed entirely. The scipy.ndimage.measurements.label module was used to create the heatmap.

Figure 9 shows the heatmap of figure 7. Figure 10 shows the removal of the false positive box that appeared in figure 7 through the implementation of heatmaps.

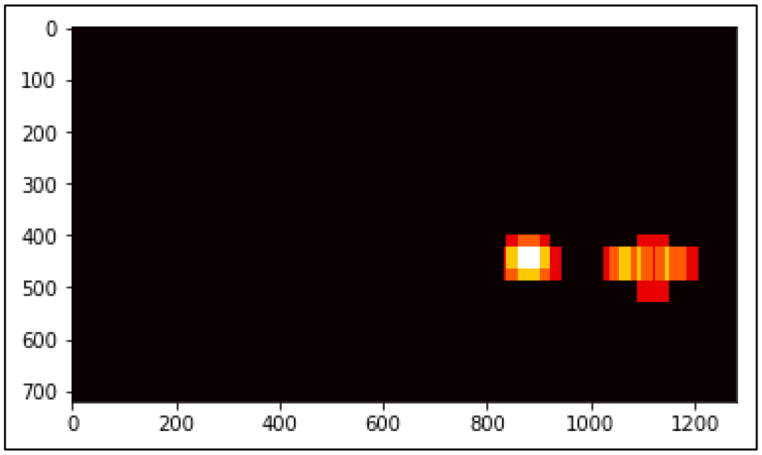

Figure 9: Heat Map of Figure 7

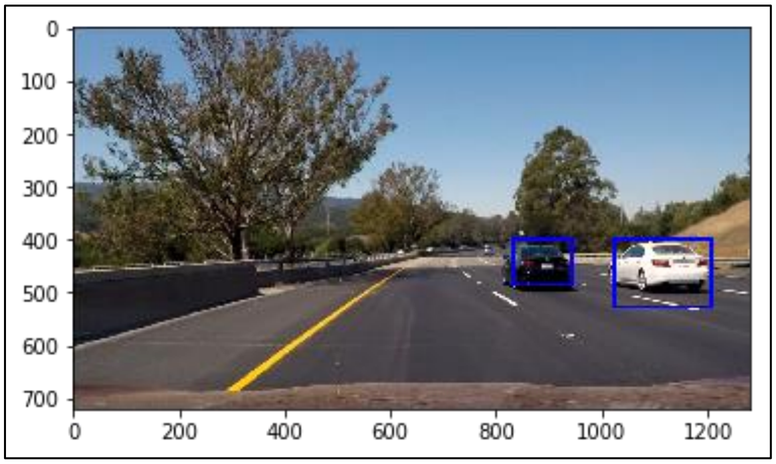

Figure 10: Removal of False Positive in Figure 7 through heatmap implementation

The video implementation of the vehicle detection can be found in the zip file in this submission.

<u>Discussion</u>

There are several aspects where improvements in this pipeline needs to be made. As seen in the video, false positive still appears for the video vehicle detection despite implementing the heatmap. Also, this pipeline was not tested in an urban setting. In an urban setting where it is very “noisy” in terms of colors, more false positives might occur.

One way to improve this might be to use an artificial neural network to train the data. In this pipeline, the features of vehicles were chosen manually, in an artificial neural network, features are chosen network. An artificial neural network might uncover important features to reduce false positives that were not considered in this pipeline.

Another problem with this pipeline is that the bounding boxes are unstable and wobbly despite being able to detect the vehicles most of the time. A way to reduce the instability might be to integrate the heatmap over several frames. This way, multiple detection gets hot while transient false positives remain cool, as described in the class lessons.

A crucial assumption that might cause the pipeline to fail is that it was assumed that cars would only appear as far out as the road horizon. However, in real life, this might not be the case. For example, if a tall truck is driving right in front of the car, the road horizon is block from view. If the truck is close enough, the pipeline will not be able to detect the shape of the truck and might not consider it a vehicle. This could lead to an accident. Also, going up hill or going downhill might change the road horizon.

Finally, this pipeline is unable to work in real time, taking nearly 20 minutes for a 1 minute video.

To make improvements to this pipeline, the use of artificial neural network for training should be explored. A different search algorithm to detect car in an image or video frame, other than the sliding window search, should be explored to find an algorithm that could work in near real time.

References

The pipeline used for this project is based on the codes introduced in the class lessons. The pipeline was also influenced by discussion on the forum with forum mentors.