*Udacity Self-Driving Car Nanodegree*

**Project 5: Vehicle Detection**

Mark Helms, February 12, 2018

## Goal/Summary

The goal in this project is to identify vehicles in a video from a center-mounted, front-facing car camera using a classifier trained on spatial data, color histograms and/or gradient information. Bounding boxes should be drawn around detected vehicles. This report summarizes the activity involved with the project.

## Approach

*Step 1: Dataset Analysis*

Udacity supplied a dataset of 64x64 RGB images corresponding to common street images labeled as either car or not car. All of the images of cars are of the rear of the vehicle. The angle of the vehicle generally vary up to about 45 degrees, which should allow detecting vehicles more than one lane over, although some images are of the side of vehicles. Some examples are shown below:

Some examples of non-vehicle images are shown below:

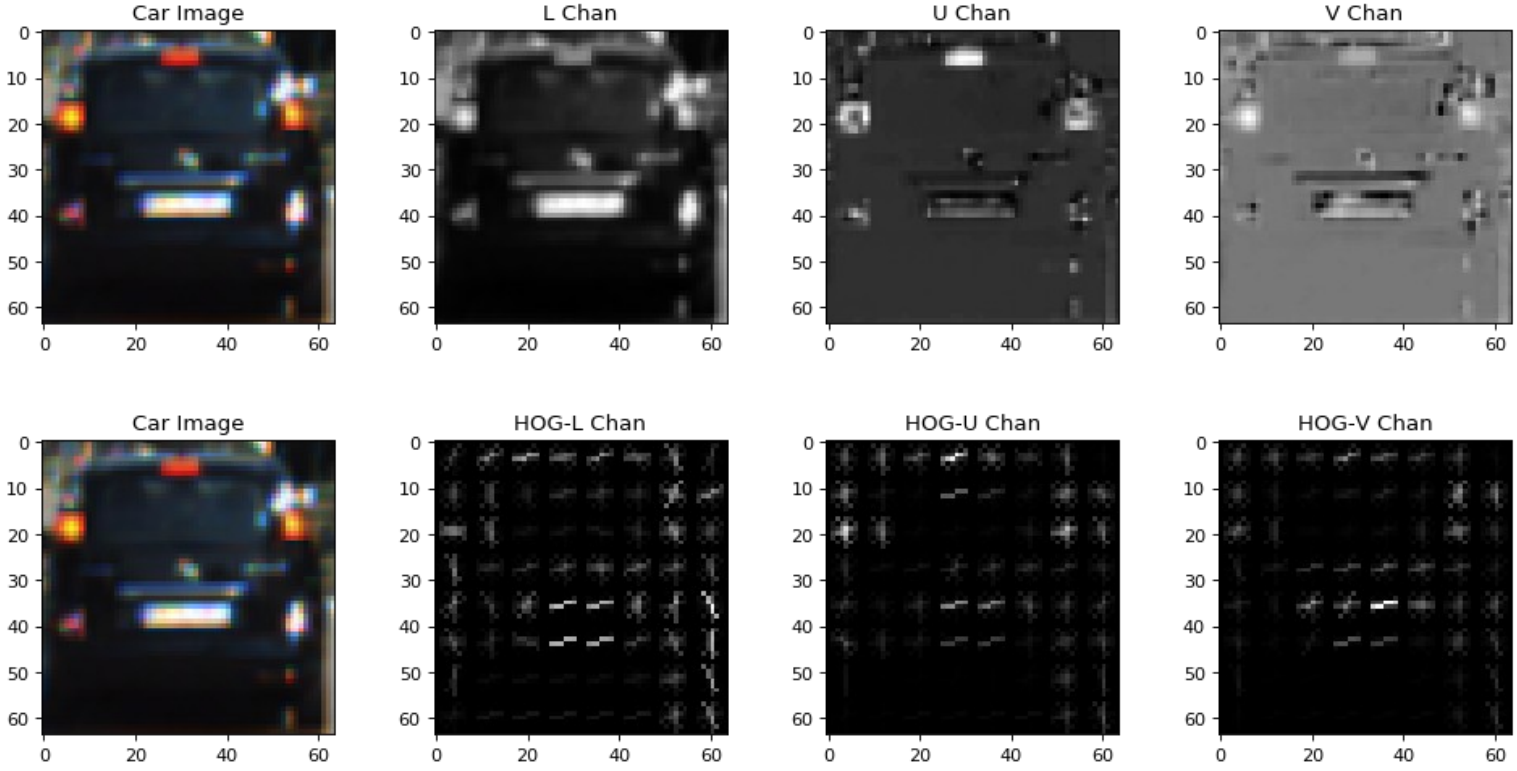
The total numbers of images available for training are 8792 vehicle images and 8968 non-vehicle images. These are fairly evenly matched, which should help the classifier remain unbiased.

*Step 2: Training the Classifier*

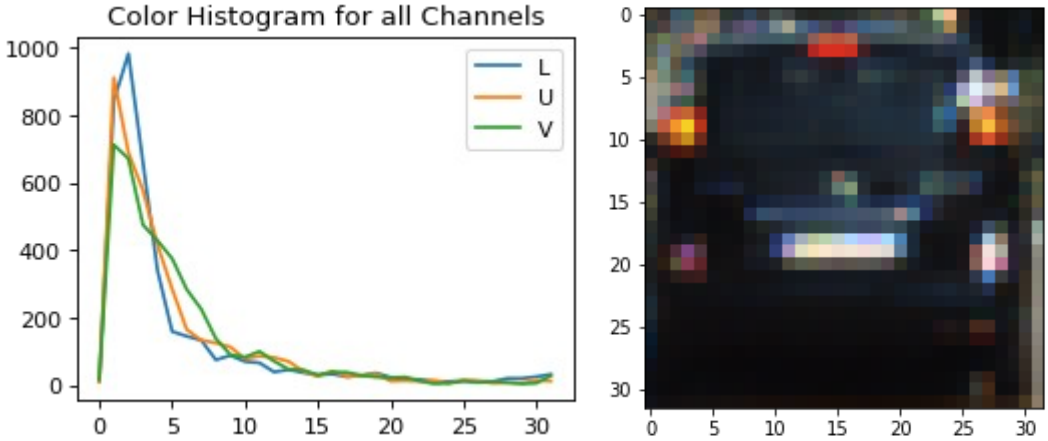
*Picking features:*

The lessons for this section provided good insight into valuable features that can be used, namely: histogram of gradients, histogram of colorspace channels, spatial binning of colorspace channels. The histogram of oriented gradients (HOG) is actually both a histogram and binning technique, whereby pixels are grouped into cells and within these cells, a weighted average of the angle of the Sobel gradient is found, using the magnitude of the Sobel gradient as the weight.

An example image broken into LUV channels and their corresponding HOGs are visualized below. Note how the gradients appear to form a bounding box that has an appearance similar to a capital "A" with a square top. Also note how the U and V channels appear to accentuate the license plate and taillights.




The histogram of the channels and the spatially binned (aka resized to 32x32) image are shown below. Note how a vehicle is still visible in the image.




The LUV colorspace was the colorspace chosen for this project. The L channel alone was used for HOG features, while all channels were used for spatial binning and histograms. The reason for limiting HOG features to the L channel was to increase speed at the cost of a relatively small quality loss.

*Training:*

Now that features have been selected, they are normalized and used to train the classifier. Normalization is an important step for classifiers in order to maximize the generalization of the classification and should help with varying car colors and lighting conditions. The binning, which is part of the HOG features should help with offering some size and position invariance.

The total number of samples was 17760, which was shuffled and split into training (80%) and testing (20%) datasets. Because a linear SVM classifier was used, the only major parameter available to tune was the C factor, a cost factor for miscategorizing features. This was set low, due to the low confidence in the dataset. This should mean only the most clustered/popular feature combinations in the training data will be used to create cutoffs between vehicle/non-vehicle determinations.

The feature vector length was 4932. The total training time was about 12 seconds. The test accuracy was 98.3% and the training accuracy was 100%.

*Step 3: Sliding Window Detection*

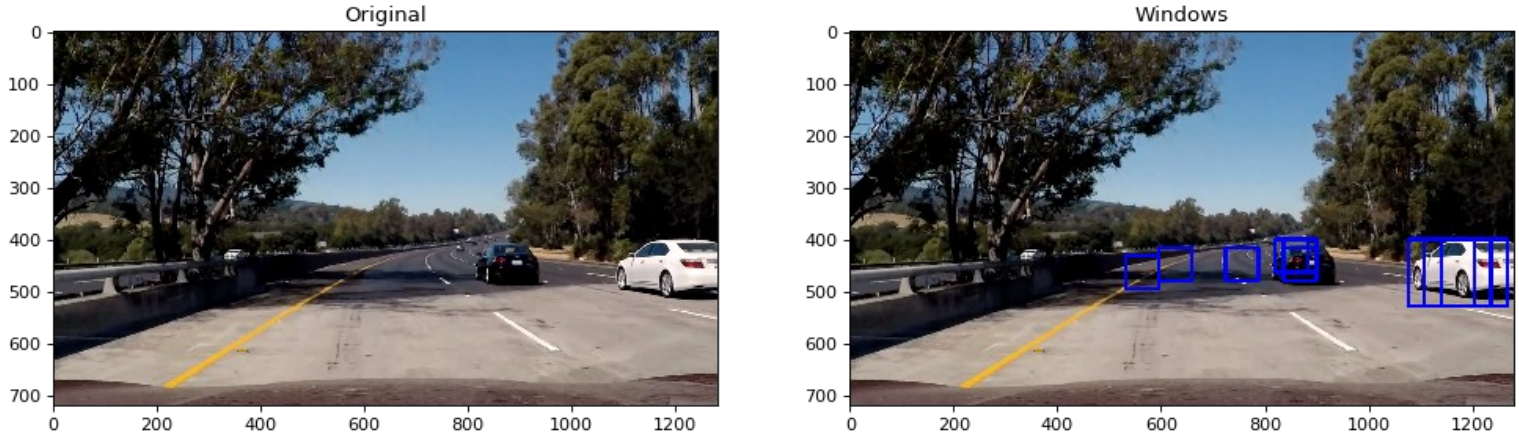
Now that a classifier is available that can operate on images that are 64x64, a sliding window detection algorithm can be used to identify vehicles in subsections of larger images. Two main methods were demonstrated in the lectures:

1) sliding a window of any size across the image and resizing the contents of that window to 64x64 and extracting features from it. This method requires recalculating HOG features for each window position, which can be costly.

2) scaling the entire image and calculating HOG features once for the entire image, then allowing steps of the window which are multiples of the cell sizes used to calculate HOG features. This offers a great improvement in speed, while offering very little restrictions on granularity of detections, assuming that the pixels-per-cell value is a small percentage of the main feature size of 64x64. In this case, the pixels-per-cell value was 8, allowing up to a 87.5% overlap between windows.

The second method was used due to it's being nearly equivalent to the first and being much faster. A slowdown for the algorithm is a requirement to calculate multiple scale sizes, because the classifier performs best when the window contains nearly the entire car. One means of offsetting this extra calculation load is to limit larger scales (smaller window sizes) near the horizon, where cars appear smaller. Combining these search ranges with a general search range of below the sky allows for a great improvement in speed.

An example output of the vehicle detection is shown below, with blue windows indicating positive detections. Note how the shadows on the road easily confuse the algorithm. These false positives must be eliminated.

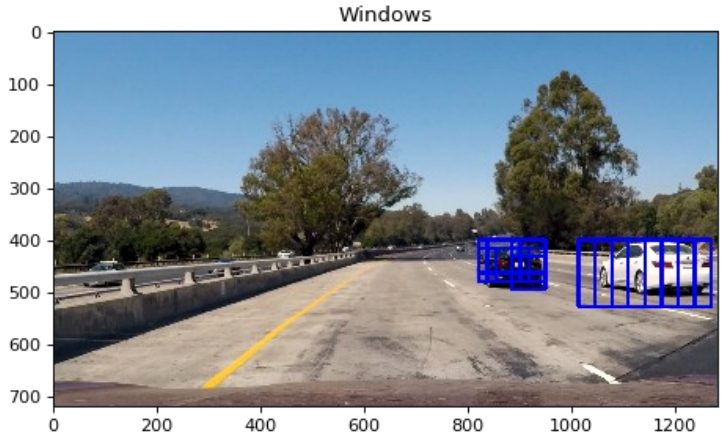


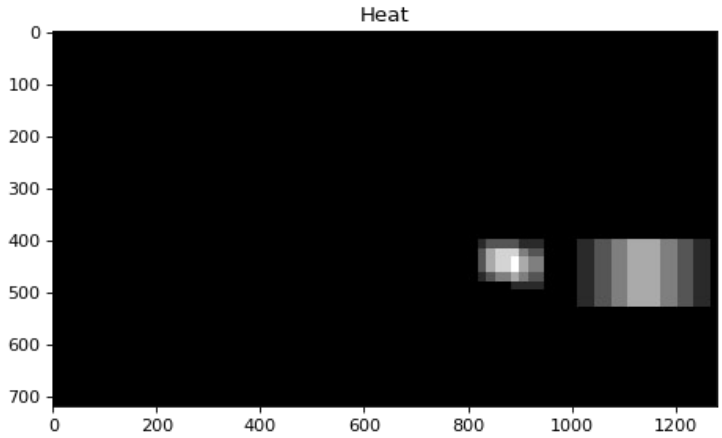
*Step 4: Time-Series Integration*

As seen above in Step 3, false positives can cause bounding boxes to appear in the wrong locations. Luckily, because the image source is a video, it's possible to use multiple frames to make a determine about whether or not the windows actually represent a vehicle on the road. There are many ways to attempt to do this. The lessons suggested the use of a heat map, thresholding and labeling combination, which works very well.

The chosen method in this project used the recommend approach and managed the heatmap in the following way:
0) Initialize heatmap to 0
1) Phase out previous heatmap
2) Detect vehicles as in Step 3 for each frame
3) Add detected regions to the heatmap
4) Check if any areas of the heatmap are above a certain threshold and if so, label and create final bounding boxes
5) Go back to step 1 for next image

An example single-frame detection and heatmap are shown below. Note that as windows overlap, the heatmap from a single frame may be much higher than one. However, in order for the final output to include bounding boxes around any location, that location must have a continuous heatmap presence across multiple frames. This is enforced by having a minimum threshold which is greater than the number of overlapping windows possible at any location.




**Performance**

The algorithm was able to identify the two "main" vehicles in the project video – the black and white cars – while not being able to identify the distant vehicles or the vehicle that shows up on the far right in the last few frames. This performance seems reasonable, although definitely suggests that there is room for improvement.

**Discussion**

Many of the functions from the lessons were used for this project, however some were adjusted in order to provide more functionality, such as *find_cars*, which was adjusted to restrict searches in the x-dimension, in addition to the y-dimension. It was also modified to be compatible with a variety of colorspaces and options of whether or not to include features of one type of another, since these had been hardcoded in the lesson version of the code.

One important observation is that the black car was more readily detected than the white car. This could be an indication that improvements could be made in the feature extraction such that white cars have an equally strong representation.

The car which shows up at the end on the far right may indicate that the integration time is too long. While this helps avoid false positives, it may not be quick enough in certain circumstances when a vehicle suddenly appears. One way to improve upon this is to allow for a lower integration time, which would require a better classifier. One type of classifier that may perform exceptionally well in this situation is that of a neural network. Because neural networks can run on GPUs, switching to using one could speed up the vehicle detection to real-time.

There are other ways to improve the performance of the linear SVM classifier as-is, without resorting to switching it out for a neural network. One of the ways is to use features from multiple colorspaces. This may result in slower performance however. A way to speed up the algorithm is to avoid searching in every frame (not really desirable) or to search in areas where a car was previously detected often and periodically check the rest of the image for a new vehicle.