# Self-Driving Car Nanodegree - Vehicle Detection Project

# 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.

\* Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.

\* Implement a sliding-window technique and use your trained classifier to search for vehicles in images.

\* Run your pipeline on a video stream (start with the test\_video.mp4 and later implement on full project\_video.mp4) 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)

## Explain how (and identify where in your code) you extracted HOG features from the training images.

As for any machine learning problem to solve, we need good features to extract and train our classifier with those. Features can be color, shape, size, orientation, etc, anything that differentiate the certain class from its environment. Histogram of Oriented Gradients proved to be one of the most popular feature extraction methods since 2003 where it was published as a thesis.

The scikit-image package has a built in function to extract Histogram of Oriented Gradient features. The scikit-image hog() function takes in a single color channel or grayscaled image as input, as well as various parameters. These parameters include orientations, pixels\_per\_cell and cells\_per\_block.

The number of orientations is specified as an integer, and represents the number of orientation bins that the gradient information will be split up into in the histogram. Typical values are between 6 and 12 bins.

The pixels\_per\_cell parameter specifies the cell size over which each gradient histogram is computed. This paramater is passed as a 2-tuple so I could have different cell sizes in x and y, but cells are commonly chosen to be square.

The cells\_per\_block parameter is also passed as a 2-tuple, and specifies the local area over which the histogram counts in a given cell will be normalized. Block normalization is not necessarily required, but generally leads to a more robust feature set.

In the sample image below we can see how the gradient histograms of the cells can be used sort of like a signature for a given shape.



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

I tried various combinations of parameters and as the best tradeoff between performance – classification accuracy - and time taken to train the classifier proved to be YCrCb color space, 16x16 pixels per cell, 16 orientations of the gradient, 2x2 cells per block.

## Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVC the following way. To extract features I used the extract\_features() function from Lesson 22 & 29. I did use not only HOG but also spatial color and color histogram features. These features complement each other in the information they capture about the desired object.

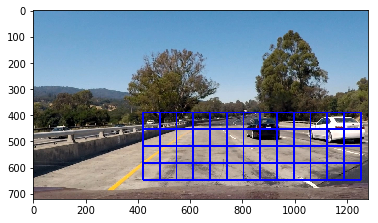
In order one or another do not dominate the feature vector I normalized the concatenated feature vector with sklearn StandardScaler() method. I also used numpy randomize method to split up data into randomized training and test sets.

At first, I tried the combined Color histogram and Spatial binning features and compared the results with only HOG-feature results. In the latter fall I used all three color channels to extract features from. They both performed well on their own – both over 98 % - , so I expect them to perform even better when I apply all these normalized features for my final classifier.

# Sliding Window Search

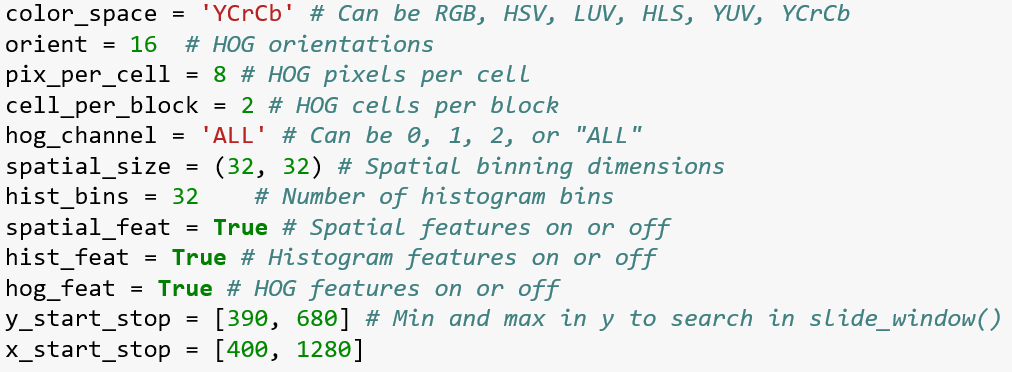
## Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I took the example code of the Udacity Lectures to implement the sliding window function. The parameters like window size, overlap, etc., must be chosen wisely. Normally a 0.5 overlap in each direction is a good choice, but it’s not uncommon to go with 0.6 or 0.7. As we are searching for vehicles in a certain region of the image – e.g. not in the background or in the opposite lane for a highway traffic – we can limit the sliding window run to a smaller region of the entire image.



## Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

I trained my final classifier with the following hyperparameters

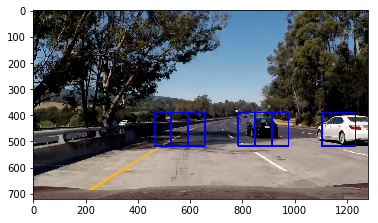


Cutting off the left hand side of the window helped eliminating some false positives. Using all three-channel HOG features, histograms of color and spatially binned color features, I achieved ~99% detection accuracy. The color space choice and other hyperparameter settings do not significantly improve or decrease the detection accuracy, but ultimately, I found the parameters above a good trade-off between accuracy and training time.

## Multiple detections, false positives and drawing bounding boxes

We need to eliminate the multiple and false positive detections. To do that I applied the functions add\_heat() and apply\_threshold() provided in Lesson 37. By building a heat map we combine the overlapping detections – even detections on multiple scales – to find the “hot” regions of the image where the overlapping detections – e.g. the pixels of the predicted bounding boxes – are the most intense and classify these regions by applying a certain threshold. By applying the apply\_threshold() we can reject the regions being “cold” e.g. the number of overlaps don’t exceed a certain threshold.

Another important points is to decide how many cars are on the image and draw bounding boxes around each of them individually. To do that I applied the “label” method described in Lesson 37.



Multiple and false positive detections – apply heat map and thresholding to eliminate these issues

## Final pipeline

In the final image processing pipeline I apply the find\_cars function with all the predefined parameters – which matches the training parameter settings - from Lesson 35. The function extracts the features defined above. The function then makes predictions for bounding boxes to draw around the vehicle. Scanning the image on multiple scales leads to a more accurate final prediction as one can define more dense bounding box regions. I used 4 pyramid layers to run my search on, but I could have used as many as I want as the detection doesn’t have to run real time.

## Video Implementation

<https://www.youtube.com/watch?v=qiB4Y_l75HM&feature=youtu.be>