# Project: Vehicle Detection

Akhil Waghmare

September 2017

## Histogram of Oriented Gradients (HOG)

1. *Explain how you extracted HOG features from the training images.*

   I started by reading in all the vehicle and non-vehicle images. Here is an example of each:

   The HOG features are extracted in the get_hog_features function in helpers.py. It takes in all of the relevant parameters (orientation, pixels per cell, cells per block), and uses the hog function in skimage.feature

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

   I tried different HOG parameters and tried to train a classifier. The accuracy of the classifier was my metric to measure how goo the HOG parameters were capturing a vehicle vs non-vehicle frame. I played around with the parameters until I was able to achieve over 97% accuracy on the classifier.

3. *Describe how you trained a classifier using your detected HOG features.*

   I took each of the images and retrieved the HOG features. This acted as the input data into the classifier. More specifically, I stacked the vehicle and non-vehicle feature vectors into one large vector to input to the trainer. I also zero-meaned and unit-varianced the data before sending into the classifier for training. The labels were simply 1 or 0 to denote vehicle vs non-vehicle. I randomly chose 20% of the training data to be my test set. I then trained the SVM using the svm.fit(X_train, y_train) to train the SVM. This code is in the classifier() function in helpers.py

## Sliding Window Search

1. *Describe how you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?*

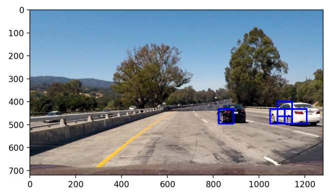
   To help avoid false positives, I restricted the window search to the bottom right half to the image (so the sky or trees wouldn't be searched).

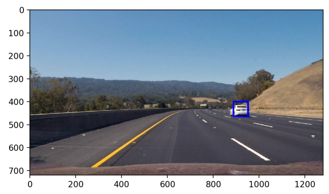
   We loop through all the windows. For each window, extract the features and feed into the classifier. If the classifier predicts there's a car in that window, then we save that window as in the "hot" state. The scale was chosen to be the same size as the input images in the classifier (64 X 64). Since we chose the windows to be the size of the input images, I thought not much overlap was needed, so chose a nonzero value of (0.5,0.5).

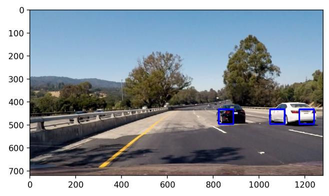
This behavior is controlled in the find_windows() and search_windows() functions in helpers.py

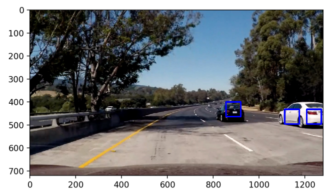
2. *Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?*

   To optimize performance, the search region was restricted to not include the sky and trees.









## Video Implementation

1. *Link to the final video output.*

   `https://youtu.be/FpceQCCttsg`

2. *Describe how you implemented some kind of filter for false positives and some method of combining overlapping bounding boxes.*

   Heatmaps were used to reduce false positives and accurately track cars as they moved. These heatmaps were generated by the "hot" windows from the past 20 frames. A threshold was then applied on the heatmap to ensure only very "warm" spots were considered. This logic is in the add_to_heatmap(), threshold_heatmap(), and generate_heatmaps() functions.

   To construct the single bounding boxes, I first labeled the image using scipy.ndimage.measurements.labels; using these labels to classify each detected car, the min and max x and y values of the corresponding windows were used to construct an encompassing bounding box. This logic is in the draw_bounding_box() function.

## Discussion

1. *Briefly discuss any problems/issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?*

   There are still some frames where the cars aren't being detected. This is often due to the presence of shadows.

While doing the project, I also tried to use spatial and histogram features. However, I found the classification was better with just using HOG features. The lesson here is more features aren't always better.