# Udacity Self-Driving Car Engineer

## Report

- **Project:** Vehicle Detection and Tracking
- **Name:** Shao Hongxu

**Let me just use some brief sentence to explain the thing I did, basically these are 8 big parts here:**

**Part 1:** import necessary lib
**Part 2:** extract feature from image (HOG, HOD etc.) and combine feature array together.
**Part 3:** train svc model using extract feature from training image
**Part 4:** identify search windows for car detection in image.
**Part 5:** using trained model to predict search window (it is a car or not)
**Part 6:** draw out windows with car detected, and convert into hot window model
**Part 7:** to avoid false positive prediction, give a baseline value & run the code for video.
**Part 8:** Summary

**To be specific as below, you will find some note on Script, key take away and Area could be improved:**

## **Part 1:** import necessary lib

In [1]:
```
import matplotlib.image as mpimg
import numpy as np
import cv2
import os
```

## **Part 2:** extract feature from image (HOG, HOD etc.) and combine feature array together.

```
def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                        vis=False, feature_vec=True):
    # Call with two outputs if vis==True
    if vis == True:
        features, hog_image = hog(img, orientations=orient,
                                  pixels_per_cell=(pix_per_cell, pix_per_cell),
                                  block_norm= 'L2-Hys',
                                  cells_per_block=(cell_per_block, cell_per_block),
                                  transform_sqrt=True,
                                  visualise=vis, feature_vector=feature_vec)
        return features, hog_image
    # Otherwise call with one output
    else:
        features = hog(img, orientations=orient,
                       pixels_per_cell=(pix_per_cell, pix_per_cell),
                       cells_per_block=(cell_per_block, cell_per_block),
                       block_norm= 'L2-Hys',
                       transform_sqrt=True,
                       visualise=vis, feature_vector=feature_vec)
        return features

# Define a function to compute binned color features
def bin_spatial(img, size=(32, 32)):
    # Use cv2.resize().ravel() to create the feature vector
    features = cv2.resize(img, size).ravel()
    # Return the feature vector
    return features

# Define a function to compute color histogram features
# NEED TO CHANGE bins_range if reading .png files with mpimg!
def color_hist(img, nbins=32, bins_range=(0, 256)):
    # Compute the histogram of the color channels separately
    channel1_hist = np.histogram(img[:,:,0], bins=nbins, range=bins_range)
    channel2_hist = np.histogram(img[:,:,1], bins=nbins, range=bins_range)
    channel3_hist = np.histogram(img[:,:,2], bins=nbins, range=bins_range)
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return hist_features
```

```
# Define a function to extract features from a list of images
# Have this function call bin_spatial() and color_hist()
def extract_features(imgs, color_space='RGB', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0,
                        spatial_feat=True, hist_feat=True, hog_feat=True):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        file_features = []
        # Read in each one by one
        image = cv2.imread(file)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # apply color conversion if other than 'RGB'
        if color_space != 'RGB':
            if color_space == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif color_space == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif color_space == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif color_space == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif color_space == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else: feature_image = np.copy(image)

        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
        if hist_feat == True:
            # Apply color_hist()
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        if hog_feat == True:
        # Call get_hog_features() with vis=False, feature_vec=True
            if hog_channel == 'ALL':
                hog_features = []
                for channel in range(feature_image.shape[2]):
                    hog_features.append(get_hog_features(feature_image[:,:,channel],
                                        orient, pix_per_cell, cell_per_block,
                                        vis=False, feature_vec=True))
                hog_features = np.ravel(hog_features)
            else:
                hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                            pix_per_cell, cell_per_block, vis=False, feature_vec=True)
            # Append the new feature vector to the features list
            file_features.append(hog_features)
        features.append(np.concatenate(file_features))
    # Return list of feature vectors
    return features
```

**-> Key take away:** I like this solution very much mentioned inside Udacity lessons this section, it collects more than one angle of image feature for model to do the prediction. It has more chance for model to predict image properly. For instance, at least 1 angel confirmed to be a car, then network may be able to say it is a car.  In the end, feature vector length is 5868 as below:

```
Feature vector length: 5868
```

Why RGB format has been selected: this part for me is a try and error area. I have tried different type of image format, more or less facing some issue while running code for video. Sometime more window detected than needed, others might be not enough window detected.

**-> Area could be improved:** which image format should be chosen for feature extraction is an area I can see a potential for improving the end result.

## **Part 3:** train svc model using extract feature from training image

**-> Key take away:** here I have chosen the same model mentioned inside Udacity for car detection, which is svc. In the end model could give us 97.47% accuracy.

```
16.32 Seconds to train SVC...
Test Accuracy of SVC =  0.9747
```

**-> Area could be improved:** here my feeling is, while doing real car image detection from video, few scenarios like an image contains a car partially can be detected as a car as well. It brings a risk of final car detected window bigger or small than real car, due to difficulty on choosing fault positive parameter.

Maybe it is an area can be improved inside training data, quit interesting topic to explore in future.

**-> Result:**

```
16.32 Seconds to train SVC...
Test Accuracy of SVC =  0.9747
```

## **Part 4:** identify search windows for car detection in image.

**-> Key take away:** this is one of the most important area for whole project. It directly impacts the end result significantly based on experience here. Below is what I did:

```
    windows_1 = slide_window(image, x_start_stop=[720, None], y_start_stop=[400, 477],
                        xy_window=(64, 64), xy_overlap=(0.85, 0.85))

    windows_2 = slide_window(image, x_start_stop=[720, None], y_start_stop=[400, 515],
                        xy_window=(96, 96), xy_overlap=(0.8, 0.8))

    windows_3 = slide_window(image, x_start_stop=[720, None], y_start_stop=[400, 554],
                        xy_window=(128, 128), xy_overlap=(0.7, 0.7))

    windows_4 = slide_window(image, x_start_stop=[720, None], y_start_stop=[400, 592],
                        xy_window=(160, 160), xy_overlap=(0.6, 0.6))

    windows=windows_1 + windows_2 + windows_3 + windows_4
    #print(windows)
    hot_windows = search_windows(image, windows, svc, X_scaler, color_space=color_space,
                            spatial_size=spatial_size, hist_bins=hist_bins,
                            orient=orient, pix_per_cell=pix_per_cell,
                            cell_per_block=cell_per_block,
                            hog_channel=hog_channel, spatial_feat=spatial_feat,
                            hist_feat=hist_feat, hog_feat=hog_feat)
    #print(hot_windows)
    window_img = draw_boxes(draw_image, hot_windows, color=(0, 0, 255), thick=6)
```

Slide_window function has been given by Udacity lesson, and here we are seeking for car detection windows in 4 different size:

64*64, 96*96, 128*128, 160*160

Accordingly, a proper overlap rate is as important as area where we want to extract windows. And below is the design:
Overlap rate: 0.85, 0.8, 0.7, 0.6
Y start stop point: 400, 477; 400, 515; 400, 554; 400, 592

In the end, after list down all window from above architecture, we use a plus plus code to add all windows together for further usage.

**-> Area could be improved:** this is really a try and error area, obviously also some different situations of cars. For instance, where a car could appear with a bigger size, where it can appear with a smaller size etc., it needs to be understood properly so that parameter set up can be done easily.

A lot of parameters at this part which still can be improved further I feel like. Will keep on think about this area as this solution can be used for a large number of issues in the world.

**-> Result:**

Below are windows with car prediction result as Yes:

```
[((819, 400), (883, 464)), ((828, 400), (892, 464)), ((837, 400), (901, 464)), ((1017, 400), (1081, 464)), ((1062, 400), (1126,
464)), ((1080, 400), (1144, 464)), ((1089, 400), (1153, 464)), ((1098, 400), (1162, 464)), ((1116, 400), (1180, 464)), ((1125,
400), (1189, 464)), ((1134, 400), (1198, 464)), ((1143, 400), (1207, 464)), ((1152, 400), (1216, 464)), ((1161, 400), (1225, 46
4)), ((1170, 400), (1234, 464)), ((1179, 400), (1243, 464)), ((1188, 400), (1252, 464)), ((1197, 400), (1261, 464)), ((1215, 40
0), (1279, 464)), ((810, 409), (874, 473)), ((981, 409), (1045, 473)), ((990, 409), (1054, 473)), ((999, 409), (1063, 473)),
((1008, 409), (1072, 473)), ((1044, 409), (1108, 473)), ((1053, 409), (1117, 473)), ((1062, 409), (1126, 473)), ((1071, 409),
(1135, 473)), ((1089, 409), (1153, 473)), ((1098, 409), (1162, 473)), ((1107, 409), (1171, 473)), ((1125, 409), (1189, 473)),
((1143, 409), (1207, 473)), ((1152, 409), (1216, 473)), ((1161, 409), (1225, 473)), ((1170, 409), (1234, 473)), ((1179, 409),
(1243, 473)), ((1188, 409), (1252, 473)), ((1197, 409), (1261, 473)), ((1024, 400), (1152, 528)), ((1062, 400), (1190, 528)),
((1100, 400), (1228, 528)), ((1138, 400), (1266, 528)), ((1040, 400), (1200, 560)), ((1104, 400), (1264, 560))]
```

## **Part 5:** using trained model to predict search window (it is a car or not)

**-> Key take away:**

Next step is to use trained svc model to do the prediction for all windows. You can see code as below:

```
hot_windows = search_windows(image, windows, svc, X_scaler, color_space=color_space,
                        spatial_size=spatial_size, hist_bins=hist_bins,
                        orient=orient, pix_per_cell=pix_per_cell,
                        cell_per_block=cell_per_block,
                        hog_channel=hog_channel, spatial_feat=spatial_feat,
                        hist_feat=hist_feat, hog_feat=hog_feat)
#print(hot_windows)
window_img = draw_boxes(draw_image, hot_windows, color=(0, 0, 255), thick=6)
```

You can check result below.

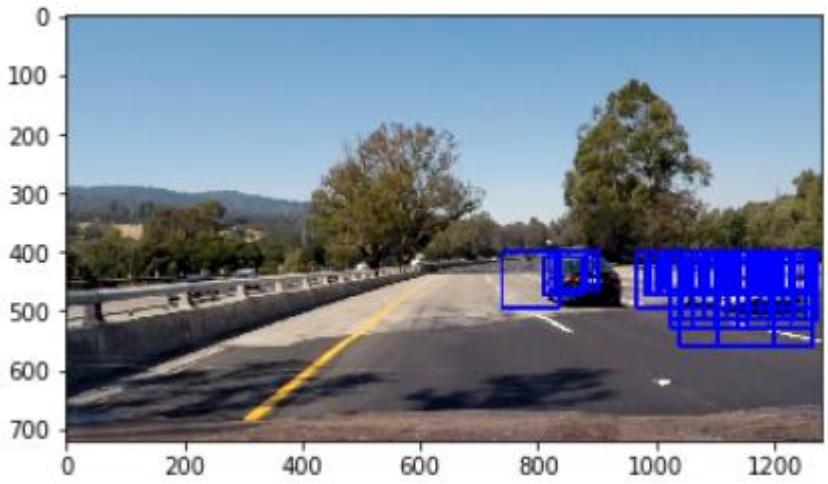
**-> Result:**

```
[((819, 400), (883, 464)), ((828, 400), (892, 464)), ((837, 400), (901, 464)), ((1017, 400), (1081, 464)), ((1062, 400), (1126,
464)), ((1080, 400), (1144, 464)), ((1089, 400), (1153, 464)), ((1098, 400), (1162, 464)), ((1116, 400), (1180, 464)), ((1125,
400), (1189, 464)), ((1134, 400), (1198, 464)), ((1143, 400), (1207, 464)), ((1152, 400), (1216, 464)), ((1161, 400), (1225, 46
4)), ((1170, 400), (1234, 464)), ((1179, 400), (1243, 464)), ((1188, 400), (1252, 464)), ((1197, 400), (1261, 464)), ((1215, 40
0), (1279, 464)), ((810, 409), (874, 473)), ((981, 409), (1045, 473)), ((990, 409), (1054, 473)), ((999, 409), (1063, 473)),
((1008, 409), (1072, 473)), ((1044, 409), (1108, 473)), ((1053, 409), (1117, 473)), ((1062, 409), (1126, 473)), ((1071, 409),
(1135, 473)), ((1089, 409), (1153, 473)), ((1098, 409), (1162, 473)), ((1107, 409), (1171, 473)), ((1125, 409), (1189, 473)),
((1143, 409), (1207, 473)), ((1152, 409), (1216, 473)), ((1161, 409), (1225, 473)), ((1170, 409), (1234, 473)), ((1179, 409),
(1243, 473)), ((1188, 409), (1252, 473)), ((1197, 409), (1261, 473)), ((1024, 400), (1152, 528)), ((1062, 400), (1190, 528)),
((1100, 400), (1228, 528)), ((1138, 400), (1266, 528)), ((1040, 400), (1200, 560)), ((1104, 400), (1264, 560))]
```

## **Part 6:** draw out windows with car detected, and convert into hot window model

**-> Key take away:** with so many car windows detected by model, we need to convert it into a end result as most of the car windows have overlapping with each other. The way to do that, is to use a blank Matrix with all initial value as 0, for area has car window detected, put the value plus 1, after all windows go through the methodology, we will receive another new Matrix with some of the point with value, it could be big, could be small, the whole solution as mentioned by Udacity lesson, we call it hot window. Here we use a picture to display that Matrix, deeper the red is, more window overlapping it is.

**-> Result:**



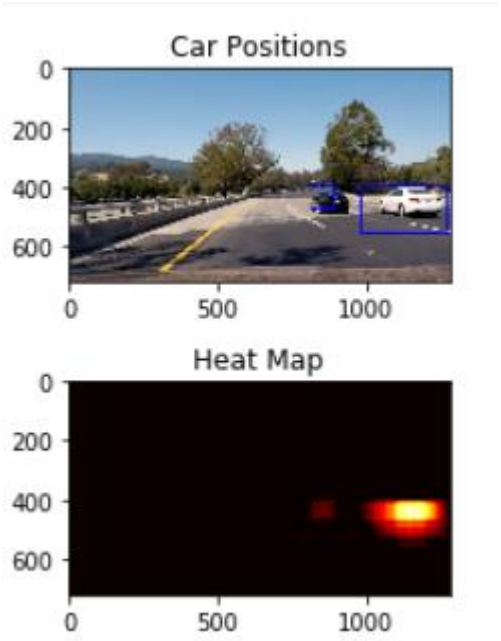

**Part 7:** to avoid false positive prediction, give a baseline value & run the code for video.

**-> Key take away:**

In this section, there is another important parameter. Once we have hot window Matrix, we need to give a number value below how much we consider that point to be a false positive. And we will remove those points, before drawing final car detection windows.

**-> Area could be improved:** Here again is a try and error method for me, in the end, I could avoid the false positive most of the time, but I could see a huge potential here.

**-> Result:**



## **Part 8:** Summary

I personally really like this project, as I remember long time back, the first thing impresses me related to machine learning area is a video from which a human face, car, a lot of objective can be detected as a real time. I can remember the targets with some boxes drawn just like we did in the project. At that moment, I was always think about the solution how it can be detected from live images, until start doing this project, I got a better sense on it using search window and machine learning prediction model.

Still a lot of area could be improved, obviously always from pre-processing, and list of parameters setting up, for instance in part 7, part 6 search window area. Will keep on think about the better solution for this as this is a really important application area in machine learning, which can be used for a lot of different areas. For instance, training data changed into other objective like bird, then whole method will become a detection model for bird in a live video/ camera etc.