# Vehicle Detection

Yuda Wang

Self-driving Car Nanodegree

Udacity

August-September 2017

# Contents

*Chapter 1*

# Introduction

This is the fifth project of self-driving car nanodegree, term 1.

In this project, several functions should be included accroding to the writeup template :

" 1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier.

2. 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.

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

4. Run your pipeline on a video stream (start with the test$_v ideo.mp4 and later implement on full project_v ideo.mp$

5. Estimate a bounding box for vehicles detected.

"

However, just these steps are not enough to make the output accurate. I made several improvements:

1. Augment and balance the data to make the raw data better quality.

2. Two svms, one svm for predicting whether it is a car or not, the other for predicting whether it is on the right side.

3. Extract the HOG once for each scale.

4. Multiple scales of sliding windows.

5. Use pipeline with history, feed the history svm information to the heat map.

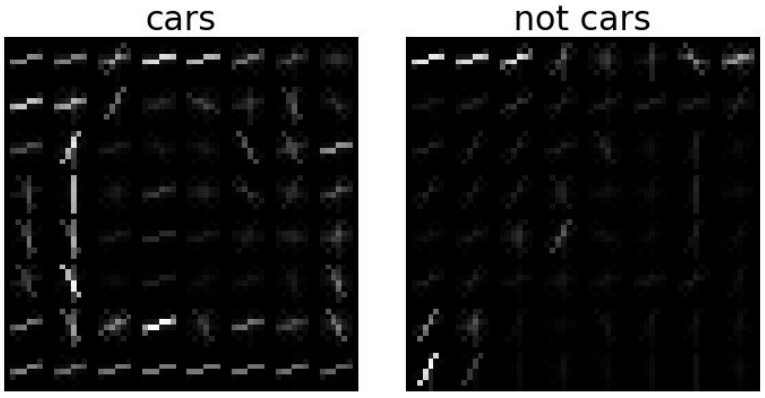
*Chapter 2*

# Histogram of Oriented Gradients (HOG)

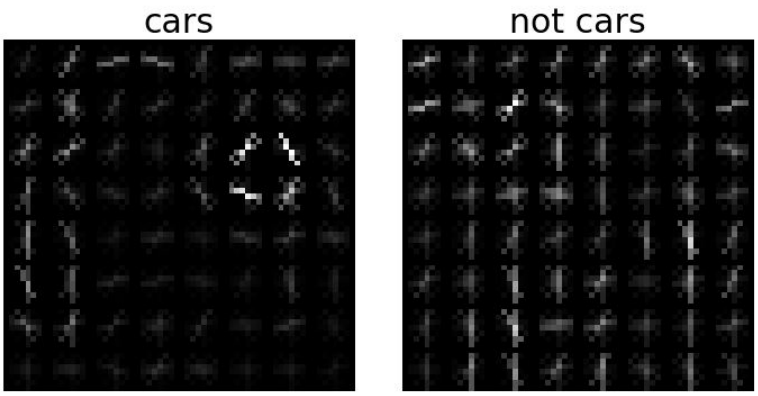
## 2.1 HOG

I choose 3 channels of HOG extraction. Because all the three channels have the information that maybe helpful for prediction.

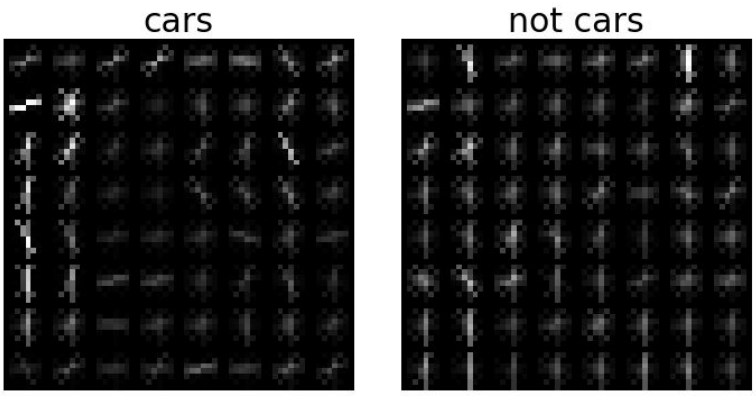

```
features, hog_img = get_hog_features(test_img[:,:,0], orient, pix_per_cell, cell_per_block,
                        vis=True, feature_vec=True)
features, hog_img_0 = get_hog_features(test_img_not[:,:,0], orient, pix_per_cell, cell_per_block,
                        vis=True, feature_vec=True)
plot_images(hog_img, hog_img_0,  'cars','not cars', 'gray', 'gray')
```



```
features, hog_img = get_hog_features(test_img[:,:,1], orient, pix_per_cell, cell_per_block,
                        vis=True, feature_vec=True)
features, hog_img_0 = get_hog_features(test_img_not[:,:,1], orient, pix_per_cell, cell_per_block,
                        vis=True, feature_vec=True)
plot_images(hog_img, hog_img_0,  'cars','not cars', 'gray', 'gray')
```




```
features, hog_img = get_hog_features(test_img[:,:,2], orient, pix_per_cell, cell_per_block,
                        vis=True, feature_vec=True)
features, hog_img_0 = get_hog_features(test_img_not[:,:,2], orient, pix_per_cell, cell_per_block,
                        vis=True, feature_vec=True)
plot_images(hog_img, hog_img_0,  'cars','not cars', 'gray', 'gray')
```

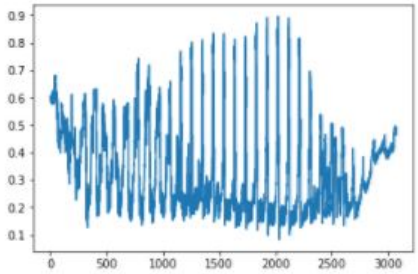



The color channel I choose is YCrCb, because most of the car is luminescent.

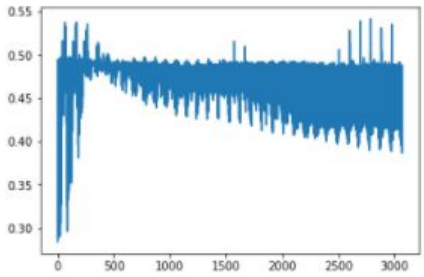
## 2.2 Spatial

In additional to HOG, I also add spatial features.

```
feature_vec = bin_spatial(test_img, size=(32, 32))
plt.plot(feature_vec)
plt.show()
```




```
feature_vec = bin_spatial(test_img_not, size=(32, 32))
plt.plot(feature_vec)
plt.show()
```




For cars, the spatial feature is more concentrated on several colors.

## 2.3 Color Hist

Color Hist feature is also added to the features. Although there is not much difference between cars and not cars in color hist, but for the safety and comprehesive sake, I add it to the feature list.

*Chapter 3*

# SVM Classifers

## 3.1 Augment and balance the data

The augment method is horizontal flip. I flip the data initially.

**Augment the data**

For vehicles and non-vehicles, flip it to the other side.

```
#augment the data
def read_path(path):
    images = glob.glob(path)
    return images

def add_flip_data():
    left_cars = read_path('vehicles/GTI_Left/*.png')
    right_cars = read_path('vehicles/GTI_Right/*.png')
    print(len(left_cars), len(right_cars))
    for i in range(len(left_cars)):
        #uncomment here for the first run
        img = cv2.imread(left_cars[i])
        cv2.imwrite('./vehicles/GTI_Right_Augmented/'+'flip_'+left_cars[i].split('\\')[-1], cv2.flip(img, 1))
    for i in range(len(right_cars)):
        #uncomment here for the first run
        img = cv2.imread(right_cars[i])
        cv2.imwrite('./vehicles/GTI_Left_Augmented/'+'flip_'+right_cars[i].split('\\')[-1], cv2.flip(img, 1))
#add_flip_data()

def add_flip_data_for_middle_far():
    middle_cars = read_path('vehicles/GTI_MiddleClose/*.png')
    far_cars = read_path('vehicles/GTI_Far/*.png')
    for i in range(len(middle_cars)):
        #uncomment here for the first run
        img = cv2.imread(middle_cars[i])
        cv2.imwrite('./vehicles/GTI_MiddleClose/'+'flip_'+middle_cars[i].split('\\')[-1], cv2.flip(img, 1))
    for i in range(len(far_cars)):
        #uncomment here for the first run
        img = cv2.imread(far_cars[i])
        cv2.imwrite('./vehicles/GTI_Far/'+'flip_'+far_cars[i].split('\\')[-1], cv2.flip(img, 1))

#add_flip_data_for_middle_far()
def add_flip_data_for_not_cars():
    not_cars_GTI = read_path('./non-vehicles/GTI/*.png')
    not_cars_Extras = read_path('./non-vehicles/Extras/*.png')
    for i in range(len(not_cars_GTI)):
        #uncomment here for the first run
        img = cv2.imread(not_cars_GTI[i])
        print(img)
        cv2.imwrite('./non-vehicles/GTI/'+'flip_'+not_cars_GTI[i].split('\\')[-1], cv2.flip(img, 1))
    for i in range(len(not_cars_Extras)):
        #uncomment here for the first run
        img = cv2.imread(not_cars_Extras[i])
        cv2.imwrite('./non-vehicles/Extras/'+'flip_'+not_cars_Extras[i].split('\\')[-1], cv2.flip(img, 1))
```

Make sure the length of cars and not cars, left and right cars data is the same.

```
def get_samples_0():
    far_cars = read_path('vehicles/GTI_Far/*.png')
    left_cars = read_path('vehicles/GTI_Left/*.png')
    right_cars = read_path('vehicles/GTI_Right/*.png')
    close_cars = read_path('vehicles/GTI_MiddleClose/*.png')
    other_cars = read_path('vehicles/KITTI_extracted/*.png')
    cars = far_cars + left_cars + right_cars + close_cars + other_cars
    not_cars = read_path('non-vehicles/GTI/*.png')+read_path('non-vehicles/Extras/*.png')
    num  = min(len(cars), len(not_cars))
    cars = random.sample(cars, num)
    not_cars = random.sample(not_cars, num)
    return cars, not_cars
```

```
cars_0, not_cars_0 = get_samples_0()
```

```
print(len(cars_0), len(not_cars_0))
```

```
10045 10045
```

```
def get_samples_1():
    far_cars = read_path('vehicles/GTI_Far/*.png')
    left_cars = read_path('vehicles/GTI_Left/*.png') + read_path('vehicles/GTI_Left_Augmented/*.png')
    right_cars = read_path('vehicles/GTI_Right/*.png') + read_path('vehicles/GTI_Right_Augmented/*.png')
    close_cars = read_path('vehicles/GTI_MiddleClose/*.png')
    return close_cars, far_cars, left_cars, right_cars
```

```
close_cars_1, far_cars_1, left_cars_1, right_cars_1 =  get_samples_1()
```

```
print(len(close_cars_1), len(far_cars_1), len(left_cars_1), len(right_cars_1))
```

```
838 1668 1573 1573
```

## 3.2 Train the SVM

### 3.2.1 Split the data

I split the train data and test data for 8:2.

### 3.2.2 Normalize the training data

Only normalize the training data and get the training data scaler.

### 3.2.3 SVM Fit the data

For the cars and not cars data, the linear svm has a better result. For the left and right cars data, the rbf svm has a better result. However, they may cause the computer a long calculation (my laptop is intel i5).

## 3.3 Test Result

Use the training data scaler, normalize the test data, and predict.

After testing, the two classifiers have accuracy of 0.984 and 0.976.

*Chapter 4*

# Sliding Windows Search and Predict

## 4.1 Sliding Single Window Search and Predict

The code is modified from the course contents. The HOG feature is extracted once.

```
def find_cars(img, ystart, ystop, scale, svc, X_scaler, svc_1, X_scaler_1, orient, pix_per_cell, cell_per_block, spatial_size, hi

    box_list = []
    draw_img = np.copy(img)
    img = img.astype(np.float32)/255

    img_tosearch = img[ystart:ystop,640:,:]
    ctrans_tosearch = cv2.cvtColor(img_tosearch,  cv2.COLOR_RGB2YCrCb)
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale), np.int(imshape[0]/scale)))

    ch1 = ctrans_tosearch[:,:,0]
    ch2 = ctrans_tosearch[:,:,1]
    ch3 = ctrans_tosearch[:,:,2]

    # Define blocks and steps as above
    nxblocks = (ch1.shape[1] // pix_per_cell) - cell_per_block + 1
    nyblocks = (ch1.shape[0] // pix_per_cell) - cell_per_block + 1
    nfeat_per_block = orient*cell_per_block**2

    # 64 was the orginal sampling rate, with 8 cells and 8 pix per cell
    window = 64
    nblocks_per_window = (window // pix_per_cell) - cell_per_block + 1
    cells_per_step = 2  # Instead of overlap, define how many cells to step
    nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
    nysteps = (nyblocks - nblocks_per_window) // cells_per_step

    # Compute individual channel HOG features for the entire image
    hog1 = get_hog_features(ch1, orient, pix_per_cell, cell_per_block, feature_vec=False)
    hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, feature_vec=False)
    hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, feature_vec=False)

    for xb in range(nxsteps):
        for yb in range(nysteps):
            ypos = yb*cells_per_step
            xpos = xb*cells_per_step
            # Extract HOG for this patch
            hog_feat1 = hog1[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
            hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
            hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
            hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))

            xleft = xpos*pix_per_cell
            ytop = ypos*pix_per_cell

            # Extract the image patch
```

There are several impovements:

1. I use two classifiers: car or not car, left or right car

2. Cars are on the right, so the horizontal starting point is in the middle.

```
ystart = 400
ystop = 656
scale = 1.5
images= glob.glob('project_images/*.jpg')

img = mpimg.imread(images[8])

out_img, box_list = find_cars(img, ystart, ystop, scale, svr_0, X_scaler_0,svr_1, X_scaler_1, 9, 8, 2, (32,32), 32)

plt.imshow(out_img)
```

```
<matplotlib.image.AxesImage at 0x7967c50>
```


## 4.2 Sliding Multiple Windows Search and Predict

To search cars with multiple scales, I use multiple scales of windows.

```
def find_more_cars(image):
    img = np.copy(image)
    _, box_list_0 = find_cars(img, 400, 656, 1.8, svr_0, X_scaler_0,svr_1, X_scaler_1, 9, 8, 2, (32,32), 32)
    _, box_list_1 = find_cars(img, 400, 656, 1.5, svr_0, X_scaler_0,svr_1, X_scaler_1, 9, 8, 2, (32,32), 32)
    _, box_list_2 = find_cars(img, 400, 656, 1.1, svr_0, X_scaler_0,svr_1, X_scaler_1, 9, 8, 2, (32,32), 32)
    _, box_list_3 = find_cars(img, 400, 656, 0.8, svr_0, X_scaler_0,svr_1, X_scaler_1, 9, 8, 2, (32,32), 32)
    result = draw_boxes(img, box_list_0 + box_list_1+box_list_2+box_list_3, color=(0, 0, 255), thick=6)
    return result,  box_list_0 + box_list_1+box_list_2+box_list_3
```

```
plt.imshow(find_more_cars(mpimg.imread(images[5]))[0])
```

```
C:\Program Files\Anaconda3\lib\site-packages\skimage\feature\_hog.py:119: skimage_deprecation: Default value of `block_norm`=='
L1' is deprecated and will be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```

```
<matplotlib.image.AxesImage at 0x1176d2b0>
```

*Chapter 5*

# Use History information and Heat Map

To reduce false positives, I use history information and heat map.

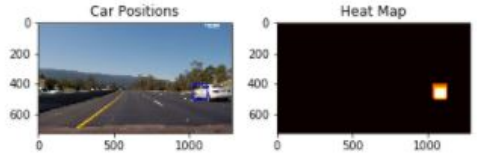

The disadvantage of only use heat map with higher threshold is that it will cause more true negative, the windows appears in the video stream is not smooth enough. So I also use history information.

## 5.1 Heat Map

The code is almost the same from the course contents. One improvement is that if the shape is abnormal, the box won't return to the box list.

```
heat = add_heat(heat,box_list)

# Apply threshold to help remove false positives
heat = apply_threshold(heat, 1)

# Visualize the heatmap when displaying

heatmap = np.clip(heat, 0, 255)


# Find final boxes from heatmap using label function
labels = label(heatmap)

draw_img, labeled_bboxes = draw_labeled_bboxes(np.copy(img), labels)

fig = plt.figure()
plt.subplot(121)
plt.imshow(draw_img)
plt.title('Car Positions')
plt.subplot(122)
plt.imshow(heatmap, cmap='hot')
plt.title('Heat Map')
fig.tight_layout()
```

```
(720, 1280)
```




Although the heat map without history satisfy all the single test frames, but in reality, the quality of video detection is bad, it needs history information.

```
def find_cars_pipeline(img):
    image = np.copy(img)
    undistorted = undistort(img)
    box_list = []
    _ , box_list = find_more_cars(img)

    heat = np.zeros_like(undistorted[:,:,0]).astype(np.float)
    heat = add_heat(heat,box_list)

    # Apply threshold to help remove false positives
    heat = apply_threshold(heat, 1)

    # Visualize the heatmap when displaying

    heatmap = np.clip(heat, 0, 255)


    # Find final boxes from heatmap using label function
    labels = label(heatmap)

    result, labeled_bboxes = draw_labeled_bboxes(np.copy(img), labels)

    return result
```

```
plt.imshow(find_cars_pipeline(mpimg.imread(images[14])))
```

```
(720, 1280)
<matplotlib.image.AxesImage at 0x112d8da0>
```


## 5.2 History Information

The latest 8 frames will send to the a higher threshold heat map, the false positives will be reduced and the box appeared in the video stream will be smoother.

*Chapter 6*

# Discussion

## 6.1 Difficulties

1. Building svm training and testing flow. 2. Setting the window size.

## 6.2 More to be implemented

### 6.2.1 More Scenarios

1. When the detected cars are on the left, new slides of windows should be added.

2. When the cars are driving from another direction, detection may fail. All the vehicle data has no car heads.

### 6.2.2 Detection Speed Optimization

1. Fast-yolo deep learning network.

2. Sparser video frames.

3. Use larger sliding windows.