In [1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
import glob
import time
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
# NOTE: the next import is only valid for scikit-learn version <= 0.17
# for scikit-learn >= 0.18 use:
# from sklearn.model_selection import train_test_split
from sklearn.cross_validation import train_test_split
import random
from random import shuffle
```

```
C:\Users\ckcheung\AppData\Local\Continuum\Miniconda3\envs\carnd2\lib\site-packa
ges\sklearn\cross_validation.py:41: DeprecationWarning: This module was depreca
ted in version 0.18 in favor of the model_selection module into which all the r
efactored classes and functions are moved. Also note that the interface of the
new CV iterators are different from that of this module. This module will be re
moved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
def plotImage(image, cmap='', title=''):
    #printing out some stats and plotting
    print('This image is:', type(image), 'with dimensions:', image.shape)
    if (cmap==''):
        plt.imshow(image)  # if you wanted to show a single color channel image c
    else:
        plt.imshow(image,cmap)  # if you wanted to show a single color channel im
    plt.suptitle(title, fontsize=30)
    return

def plotOrigAndNew(image, newImage, cmap='', ncmap='', origTitle='Original Image'
    #Print images
    #f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(96, 36))
    f.tight_layout()
    if (cmap==''):
        ax1.imshow(image)
    else:
        ax1.imshow(image, cmap=cmap)
    if (ncmap==''):
        ax2.imshow(newImage)
    else:
        ax2.imshow(newImage, cmap=ncmap)
    ax1.set_title(origTitle, fontsize=90)
    ax2.set_title(newTitle, fontsize=90)
    plt.subplots_adjust(left=0., right=1, top=0.9, bottom=0.)
    plt.show()
    return

# Here is your draw_boxes function from the previous exercise
def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
    # Make a copy of the image
    imcopy = np.copy(img)
    # Iterate through the bounding boxes
    for bbox in bboxes:
        # Draw a rectangle given bbox coordinates
        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)
    # Return the image copy with boxes drawn
    return imcopy


# Define a function that takes an image and a list of templates as inputs
# then searches the image and returns the a list of bounding boxes
# for matched templates
def find_matches(img, template_list):
    # Make a copy of the image to draw on
    # Define an empty list to take bbox coords
    imgcopy = np.copy(img)
    bbox_list = []
    # Iterate through template list
    # Read in templates one by one
    for temp in template_list:
        timg = mpimg.imread(temp)
        w, h = (timg.shape[1], timg.shape[0])
        # Use cv2.matchTemplate() to search the image
        #     using whichever of the OpenCV search methods you prefer
```

```
        res = cv2.matchTemplate(img, timg, cv2.TM_SQDIFF_NORMED)
        # Use cv2.minMaxLoc() to extract the location of the best match
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
        top_left = min_loc
        # Determine bounding box corners for the match
        bbox_list.append(((min_loc), ((min_loc[0]+w), (min_loc[1]+h))))
    # Return the list of bounding boxes
    return bbox_list


# Convert color space
def convert_color(img, color_space='YCrCb'):
    if color_space == 'RGB':
        return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    if color_space == 'BGR2HSV':
        return cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    if color_space == 'BGR2LUV':
        return cv2.cvtColor(img, cv2.COLOR_BGR2LUV)
    if color_space == 'BGR2HLS':
        return cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
    if color_space == 'BGR2YUV':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    if color_space == 'BGR2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    if color_space == 'HSV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    elif color_space == 'LUV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2LUV)
    elif color_space == 'HLS':
        return cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    elif color_space == 'YUV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    elif color_space == 'YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)


# Compute binned color features
def bin_spatial(img, size=(32, 32)):
    color1 = cv2.resize(img[:,:,0], size).ravel()
    color2 = cv2.resize(img[:,:,1], size).ravel()
    color3 = cv2.resize(img[:,:,2], size).ravel()
    return np.hstack((color1, color2, color3))
    #features = cv2.resize(img, size).ravel()
    #return features


# Compute color histogram features
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_
    # Return the individual histograms, bin_centers and feature vector
    return hist_features
```

```
# Generate HOG features and visualization
def get_hog_features(img, orient, pix_per_cell, cell_per_block, vis=False, featur
    if vis == True:
        features, hog_image = hog(img, orientations=orient, pixels_per_cell=(pix_
                                  cells_per_block=(cell_per_block, cell_per_block
                                  visualise=True, feature_vector=False)
        return features, hog_image
    else:
        features = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, p
                       cells_per_block=(cell_per_block, cell_per_block), transfor
                       visualise=False, feature_vector=feature_vec)
        return features


# Extract features from a list of images
# combine features like bin_spatial(), color_hist(), get_hog_features(), etc
def extract_features(imgs, color_space='RGB', spatial_size=(32, 32),
                     hist_bins=32, hist_range=(0, 256),
                     orient=9, pix_per_cell=8, cell_per_block=2, hog_channel=0,
                     spatial_feat=True, hist_feat=True, hog_feat=True, inc_flip=F
    # Create a list to append feature vectors to
    features = []
    i=0
    # Iterate through the list of images
    for file in imgs:
        #reset features list to empty
        file_features = []
        i+=1
        # Read in each one by one
        image = mpimg.imread(file)
        if inc_flip == True:
            flip_range = 2
        else:
            flip_range = 1
        for i in range(0,flip_range):
            # add flip image to flip_range larger than 0
            if i == 1:
                #reset features list to empty
                file_features = []
                image = cv2.flip(image, 1)
            # apply color conversion if color space other than 'RGB'
            if color_space != 'RGB':
                feature_image = convert_color(image, color_space)
            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 debug:
                #    plotImage(spatial_features, title="spatial features")
                #    plt.show()

            if hist_feat == True:
                # Apply color_hist()
                hist_features = color_hist(feature_image, nbins=hist_bins)
                file_features.append(hist_features)
```

```
                #if debug:
                #    plotImage(hist_features, title="histogram features")
                #    plt.show()

            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]):
                        if debug:
                            hog_feat_chn, hog_image = get_hog_features(feature_im
                                        orient, pix_per_cell, cell_per_block,
                                        vis=debug, feature_vec=True)
                            plotOrigAndNew(feature_image, hog_image, ncmap='gray'
                            plt.show()
                        hog_feat_chn = get_hog_features(feature_image[:,:,channel
                                        orient, pix_per_cell, cell_per_block,
                                        vis=False, feature_vec=True)
                        hog_features.append(hog_feat_chn)
                    hog_features = np.ravel(hog_features)
                else:
                    if debug:
                        hog_feat, hog_image = get_hog_features(feature_image[:,:,
                                    orient, pix_per_cell, cell_per_block,
                                    vis=debug, feature_vec=True)
                        plotOrigAndNew(feature_image, hog_image, ncmap='gray', or
                        plt.show()
                    hog_features = get_hog_features(feature_image[:,:,hog_channel
                                pix_per_cell, cell_per_block, vis=False, feature_
                # Append the new feature vector to the features list
                file_features.append(hog_features)
            #print(file_features)
            features.append(np.concatenate(file_features))


    # Return list of feature vectors
    return features


# Extract features using hog sub-sampling and make predictions
def find_cars(img, ystart, ystop, xstart, xstop, scale, svc, X_scaler, orient, pi

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

    img_tosearch = img[ystart:ystop,xstart:xstop,:]
    ctrans_tosearch = convert_color(img_tosearch, color_space=color_space)
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale),

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

    if debug:
```

```
    i1=plt
    i1.imshow(ctrans_tosearch)
    i1.show()

# 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

#if(nxblocks == 0):
#    print("ch1.shape[1] %d | pix_per_cell %d | cell_per_block %d"%(ch1.shape
#if(nyblocks == 0):
#    print("ch1.shape[0] %d | pix_per_cell %d | cell_per_block %d"%(ch1.shape
#if debug:
#    print('nxblocks', nxblocks, ' | nyblocks', nyblocks, ' | nfeat_per_block

# 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
cells_per_step = 1  # 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_ve
hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, feature_ve
hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, feature_ve

if vis:
    sliders = np.copy(img)
    nslide = 0

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_
        hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_
        hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_
        hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))

        #if(xb==0 and yb==0):
        #    print("xb%d hog_features.shape" %(xb), hog_features.shape)

        xleft = xpos*pix_per_cell
        ytop = ypos*pix_per_cell

        # Extract the image patch
        subimg = cv2.resize(ctrans_tosearch[ytop:ytop+window, xleft:xleft+win

        #if(xb==0 and yb==0):
        #    print(subimg.shape)

        # Get color features
        spatial_features = bin_spatial(subimg, size=spatial_size)
```

```
            hist_features = color_hist(subimg, nbins=hist_bins)

            #if(xb==0 and yb==0):
            #    print("xb%d spatial_features.shape"%(xb), spatial_features.shape
            #    print("xb%d hist_features.shape"%(xb), hist_features.shape)
            #    print("xb%d np.hstack((spatial_features, hist_features, hog_feat

            # Scale features and make a prediction
            test_features = X_scaler.transform(np.hstack((spatial_features, hist_
            test_prediction = svc.predict(test_features)

            #if debug == True and test_prediction == 0 and ((xb > 62 and scale ==
            #    print('find_cars: xb %d| yb %d subimg in scale '%(xb, yb), scale
            #    d1 = plt
            #    d1.imshow(subimg, cmap='gray')
            #    d1.show()

            if vis:
                nslide+=1
                if ((nslide % vis)==0):
                    xleft_ori = np.int(xleft*scale)
                    ytop_ori  = np.int(ytop*scale)
                    win_ori   = np.int(window*scale)
                    #print("detected: scale ", scale, " | xb %d | yb %d | xstart %
                    #      (xb,yb,xstart,ystart,xleft,ytop,xleft_ori,ytop_ori,win_
                    cv2.rectangle(sliders,(xstart+xleft_ori, ystart+ytop_ori),(xs
                    #plotOrigAndNew(img, sliders)
                    #plt.show()


            if test_prediction == 1:
                #print("find a test_prediction in images for xb%d yb%d"%(xb,yb))
                xleft_ori = np.int(xleft*scale)
                ytop_ori  = np.int(ytop*scale)
                win_ori   = np.int(window*scale)
                boxlist.append(((xstart+xleft_ori, ystart+ytop_ori),(xstart+xleft_
                if debug:
                    cv2.rectangle(draw_img,(xstart+xleft_ori, ystart+ytop_ori),(x
                    print("detected: scale ", scale, " | xb %d | yb %d | xstart %
                          (xb,yb,xstart,ystart,xleft,ytop,xleft_ori,ytop_ori,win_
                    #print("find a test_prediction in images for xbox_left%d ytop_
                    a1 = plt
                    a1.imshow(subimg, cmap='gray')
                    a1.show()
                    b1 = plt
                    b1.imshow(draw_img, cmap='gray')
                    b1.show()

    if vis:
        print("print sliders image with boxes")
        plotOrigAndNew(img, sliders)
        plt.show()
    if debug:
        return draw_img, boxlist
    else:
        return boxlist
```

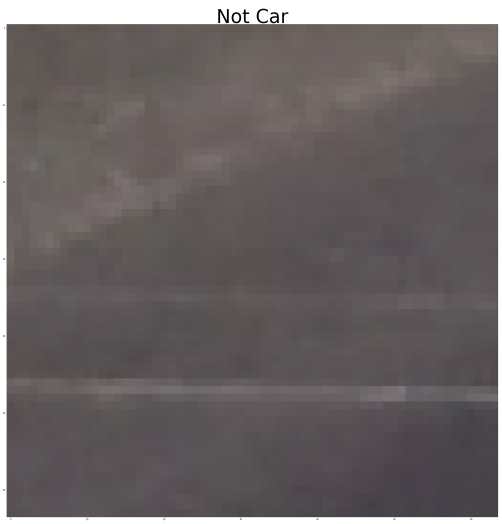
```
# function to handle false positives/duplicates using heat map
def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1

    # Return updated heatmap
    return heatmap


# use threshold to determine the focus of heat map
def apply_threshold(heatmap, threshold):
    # Zero out pixels below the threshold
    heatmap[heatmap <= threshold] = 0
    # Return thresholded map
    return heatmap


#
def draw_labeled_bboxes(img, labels):
    # Iterate through all detected cars
    for car_number in range(1, labels[1]+1):
        # Find pixels with each car_number label value
        nonzero = (labels[0] == car_number).nonzero()
        # Identify x and y values of those pixels
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])
        # Define a bounding box based on min/max x and y
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(n
        # Draw the box on the image
        cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)
    # Return the image
    return img


# generate heat map base on boxes of cars and consolidate local boxes into single
def heat_map_tracking(img, box_list, heat_thresh=1):
    from scipy.ndimage.measurements import label

    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    #print("heat_map: %d boxes"%(len(box_list)))

    # Add heat to each box in box list
    heat = add_heat(heat,box_list)

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

    # Visualize the heatmap when displaying
    heatmap = np.clip(heat, 0, 255)

    if debug:
        # Find final boxes from heatmap using label function
        labels = label(heatmap)
```

```
        draw_img = draw_labeled_bboxes(np.copy(img), labels)
        #box_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)
        #boxes_img = draw_labeled_bboxes(np.copy(box_img), labels)

        plotOrigAndNew(draw_img, heatmap, ncmap='hot', origTitle='Car Positions',
        #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()
        #plt.show()
        return heatmap, draw_img

    return heatmap


def weighted_img(img, initial_img, α=0.8, β=1., λ=0.):
    """
    `img` is the output of the hough_lines(), An image with lines drawn on it.
    Should be a blank image (all black) with lines drawn on it.

    `initial_img` should be the image before any processing.

    The result image is computed as follows:

    initial_img * α + img * β + λ
    NOTE: initial_img and img must be the same shape!
    """
    return cv2.addWeighted(initial_img, α, img, β, λ)
```

In [3]:
```
# Preprocess images - divide up into cars and notcars

# load file names of car images from each angle directories
files_car_far = glob.glob('vehicles/GTI_Far/*.png')
files_car_lef = glob.glob('vehicles/GTI_Left/*.png')
files_car_mid = glob.glob('vehicles/GTI_MiddleClose/*.png')
files_car_rig = glob.glob('vehicles/GTI_Right/*.png')
files_car_kit = glob.glob('vehicles/KITTI_extracted/*.png')

# pick random images from each angle directories
n_list_car_far = random.sample(range(1, len(files_car_far)), 833)  # get 400 exam
n_list_car_lef = random.sample(range(1, len(files_car_lef)), 908)  # get 400 exam
n_list_car_mid = random.sample(range(1, len(files_car_mid)), 418)  # get 400 exam
n_list_car_rig = random.sample(range(1, len(files_car_rig)), 663)  # get 400 exam
n_list_car_kit = random.sample(range(1, len(files_car_kit)), 5965) # get 2000 exa

# load file names of not car images from Extra directory
files_not_ext = glob.glob('non-vehicles/Extras/*.png')
files_not_gti = glob.glob('non-vehicles/GTI/*.png')
# pick random images from Extra directory
n_list_not_ext = random.sample(range(1, len(files_not_ext)), 5067) # get 1800 exa
n_list_not_gti = random.sample(range(1, len(files_not_gti)), 3899) # get 1800 exa

# list of cars and notcars images
cars = []
cars_far = []
cars_lef = []
cars_mid = []
cars_rig = []
cars_kit = []
notcars = []
not_ext = []
not_gti = []

# load images into list
def select_image_files(list_file_names, list_index, lo=0, hi=-1, fix_len=True, fl
    files = []
    if fix_len == True:
        # reading files with file name format: image####.png
        for image_name in list_file_names:
            #print("files_car_far", int(image_name[lo:hi]))
            if int(image_name[lo:hi]) in list_index:
                files.append(image_name)
    else:
        # reading files with file name format: #.png, ##.png, ###.png, ####.png
        for image_name in list_file_names:
            #print("images_car_far", int(image[lo:hi]))
            if len(image_name) == lo+5:
                if int(image_name[lo:lo+1]) in list_index:
                    files.append(image_name)
            elif len(image_name) == lo+6:
                if int(image_name[lo:lo+2]) in list_index:
                    files.append(image_name)
            elif len(image_name) == lo+7:
                if int(image_name[lo:lo+3]) in list_index:
                    files.append(image_name)
```

```
            else:
                if int(image_name[lo:lo+4]) in list_index:
                    files.append(image_name)
    return files


cars_far = select_image_files(files_car_far, n_list_car_far, len(files_car_far[0]
cars_lef = select_image_files(files_car_lef, n_list_car_lef, len(files_car_lef[0]
cars_mid = select_image_files(files_car_mid, n_list_car_mid, len(files_car_mid[0]
cars_rig = select_image_files(files_car_rig, n_list_car_rig, len(files_car_rig[0]
cars_kit = select_image_files(files_car_kit, n_list_car_kit, len(files_car_kit[0]
cars.extend(cars_far)
cars.extend(cars_lef)
cars.extend(cars_mid)
#cars.extend(cars_mid)
#cars.extend(cars_mid)
#cars.extend(cars_mid)
cars.extend(cars_rig)
#cars.extend(cars_rig)
#cars.extend(cars_rig)
#cars.extend(cars_rig)
#cars.extend(cars_kit)
shuffle(cars)

not_ext = select_image_files(files_not_ext, n_list_not_ext, len(files_not_ext[0])
not_gti = select_image_files(files_not_gti, n_list_not_gti, len(files_not_gti[0])
notcars.extend(not_ext)
#notcars.extend(not_gti)
#notcars.extend(not_gti1)
shuffle(notcars)
print("Number of car images:", len(cars))
print("Number of notcar images:", len(notcars))

# for visualizing car and notcar dataset images
plotOrigAndNew(mpimg.imread(cars[102]), mpimg.imread(notcars[128]), origTitle='Ca
plt.show()
```

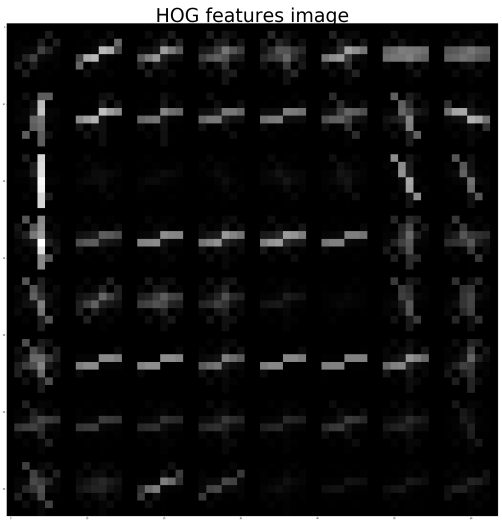
```
Number of car images: 2323
Number of notcar images: 4404
```

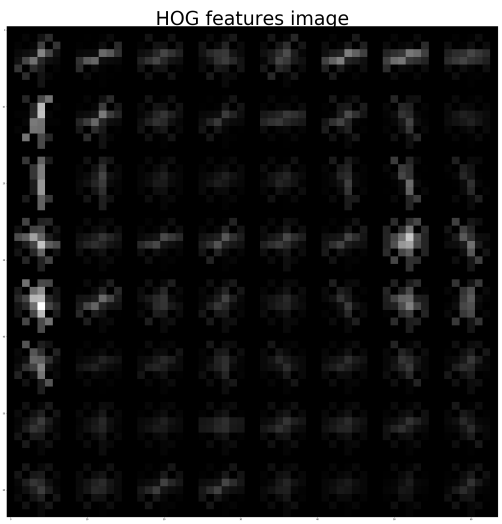


In [4]:
```
# parameters for color classify
colorspace = 'YCrCb' # Can be RGB, HSV, LUV, HLS (good), YUV (better), YCrCb
spatial_size=32
hist_bins=32
# parameters for HOG classify
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_feat = True
hist_feat = True
hog_feat = True
heat_thresh = 3
debug = False
```

In [5]:
```
# for visualizing HOG images
debug=True
car_features_example = extract_features([cars[102]], color_space=colorspace,
                                        spatial_size=(spatial_size, spatial_size),
                                        hist_bins=hist_bins, hist_range=(0, 256),
                                        orient=orient, pix_per_cell=pix_per_cell, cell_pe
                                        hog_channel=hog_channel,
                                        spatial_feat=spatial_feat, hist_feat=hist_feat, h
notcar_features_example = extract_features([notcars[128]], color_space=colorspace
                                           spatial_size=(spatial_size, spatial_size),
                                           hist_bins=hist_bins, hist_range=(0, 256),
                                           orient=orient, pix_per_cell=pix_per_cell, cell
                                           hog_channel=hog_channel,
                                           spatial_feat=spatial_feat, hist_feat=hist_feat
debug=False
```

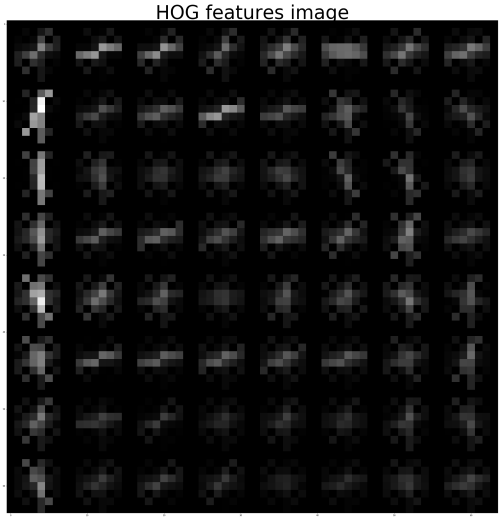
```
C:\Users\ckcheung\AppData\Local\Continuum\Miniconda3\envs\carnd2\lib\site-packa
ges\skimage\feature\_hog.py:119: skimage_deprecation: Default value of `block_n
orm`==`L1` is deprecated and will be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```





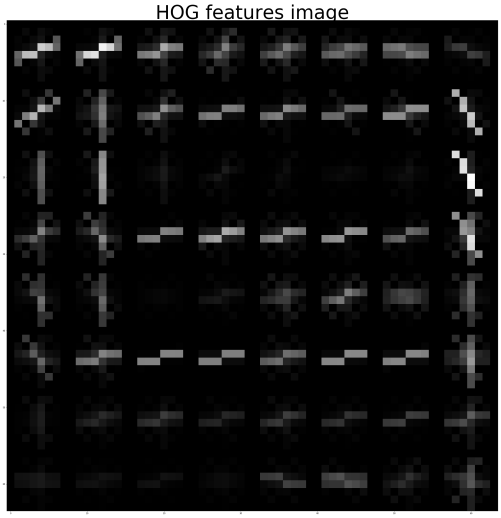
Original Image

HOG features image



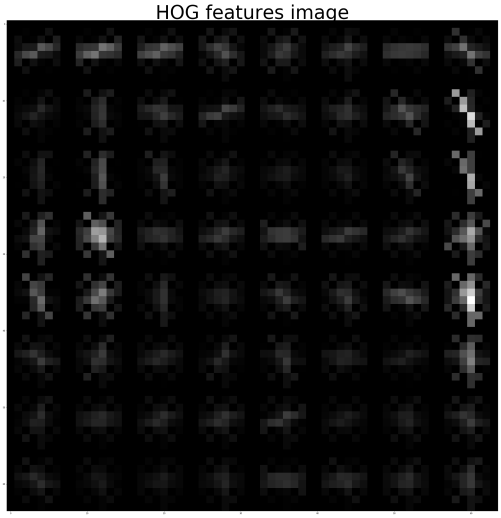
Original Image

HOG features image

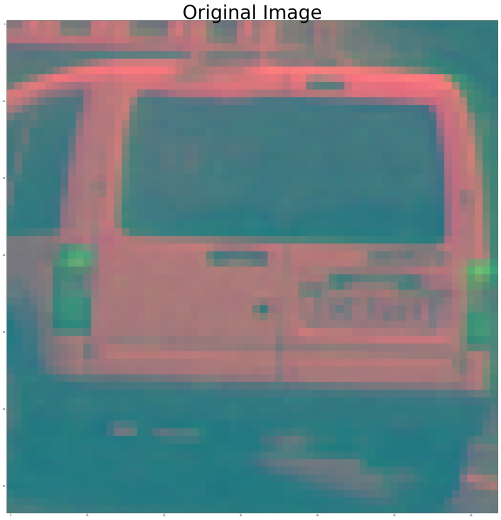


Original Image

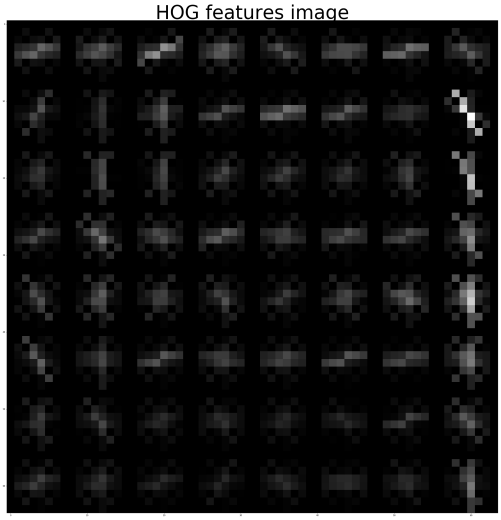
HOG features image



Original Image

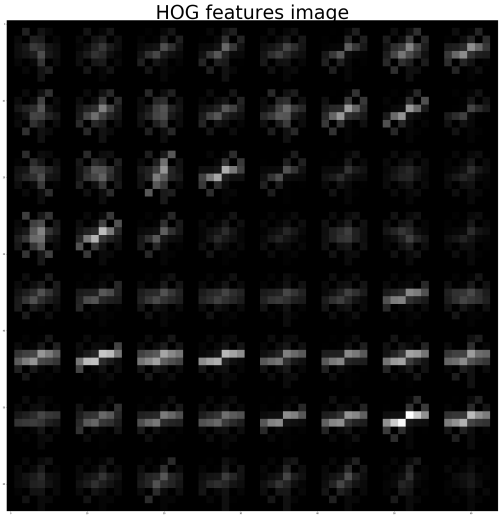


HOG features image



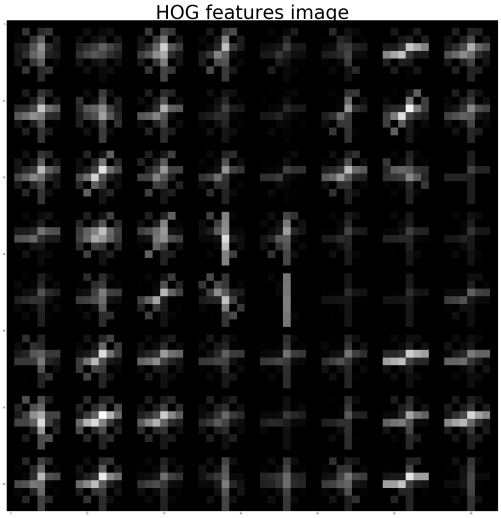
Original Image

HOG features image

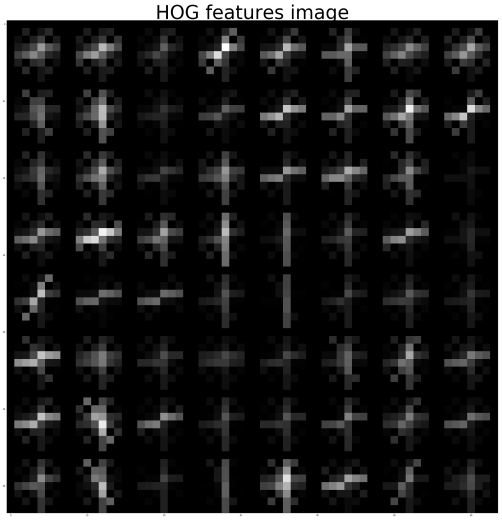


Original Image

HOG features image




Original Image

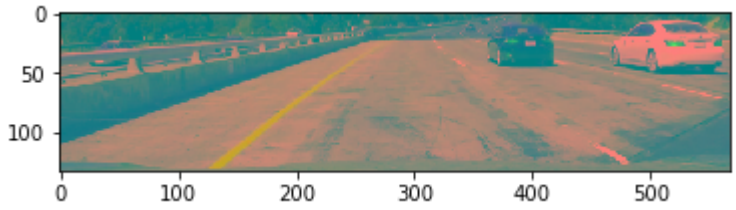

HOG features image

In [6]:
```
def car_classifier(cars, notcars, colorspace, spatial_size, hist_bins, orient, pi
                   hog_channel, spatial_feat, hist_feat, hog_feat):

    global X_scaler, svc

    t=time.time()
    print('Starting to extract color, spatial bin, histogram and HOG features...'
    print('Using:', colorspace,'colorspace |', orient,'orientations |',pix_per_ce
           'pixels_per_cell |', cell_per_block,'cells_per_block |', 'hog_channel
           '| spatial_feat=',spatial_feat, '| hist_feat=',hist_feat, '| hog_feat
    # extract features like color, spatial bin, historgram, and HOG for both cars
    car_features = extract_features(cars, color_space=colorspace,
                                    spatial_size=(spatial_size, spatial_size),
                                    hist_bins=hist_bins, hist_range=(0, 256),
                                    orient=orient, pix_per_cell=pix_per_cell, cel
                                    hog_channel=hog_channel,
                                    spatial_feat=spatial_feat, hist_feat=hist_fea
    notcar_features = extract_features(notcars, color_space=colorspace,
                                       spatial_size=(spatial_size, spatial_size),
                                       hist_bins=hist_bins, hist_range=(0, 256),
                                       orient=orient, pix_per_cell=pix_per_cell,
                                       hog_channel=hog_channel,
                                       spatial_feat=spatial_feat, hist_feat=hist_

    t2 = time.time()
    print(round(t2-t, 2), 'Seconds to extract color, spatial bin, histogram and HO
    print("cars len:", len(cars))
    print("notcars len:", len(notcars))
    print("car_features len:", len(car_features))
    print("notcar_features len:", len(notcar_features))
    #print("car_features len:", car_features[:-1])
    #print("car_features len:", notcar_features[:-1])

    # Create an array stack of feature vectors
    X = np.vstack((car_features, notcar_features)).astype(np.float64)

    # Normalization
    # Fit a per-column scaler
    X_scaler = StandardScaler().fit(X)
    # Apply the scaler to X
    scaled_X = X_scaler.transform(X)

    # Define the labels vector
    y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))

    # shuffle the lists
    combined = list(zip(scaled_X, y))
    random.shuffle(combined)
    scaled_X, y = zip(*combined)

    # Split up data into randomized training and test sets
    rand_state = np.random.randint(0, 100)
    X_train, X_test, y_train, y_test = train_test_split(
        scaled_X, y, test_size=0.2, random_state=rand_state)

    print('Feature vector length:scaled_X', len(scaled_X))
```

```
    print('Feature vector length:y', len(y))
    print('Feature vector length:X_train', len(X_train))
    print('Feature vector length:y_train', len(y_train))
    print('Feature vector length:X_test', len(X_test))
    print('Feature vector length:y_test', len(y_test))


    # Use a linear SVC
    svc = LinearSVC()


    # Check the training time for the SVC
    t=time.time()
    svc.fit(X_train, y_train)
    t2 = time.time()
    print(round(t2-t, 2), 'Seconds to train SVC...')


    # Check the score of the SVC
    print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))

    # Check the prediction time for a single sample
    t=time.time()
    n_predict = 1000
    #print('My SVC predicts: ', svc.predict(X_test[0:n_predict]))
    #print('For these',n_predict, 'labels: ', y_test[0:n_predict])
    t2 = time.time()
    print(round(t2-t, 5), 'Seconds to predict', n_predict,'labels with SVC')

debug=False
car_classifier(cars, notcars, colorspace, spatial_size, hist_bins, orient, pix_pe
               cell_per_block, hog_channel, spatial_feat, hist_feat, hog_feat)
```

```
ock | hog_channel= ALL | spatial_feat= True | hist_feat= True | hog_feat= Tru
e

C:\Users\ckcheung\AppData\Local\Continuum\Miniconda3\envs\carnd2\lib\site-pac
kages\skimage\feature\_hog.py:119: skimage_deprecation: Default value of `blo
ck_norm`==`L1` is deprecated and will be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)

42.75 Seconds to extract color, spatial bin, histogram and HOG features...
cars len: 2323
notcars len: 4404
car_features len: 4646
notcar_features len: 4404
Feature vector length:scaled_X 9050
Feature vector length:y 9050
Feature vector length:X_train 7240
Feature vector length:y_train 7240
Feature vector length:X_test 1810
Feature vector length:y_test 1810
2.37 Seconds to train SVC...
```

In [7]:
```
# for visualizing sliding windows in region of interest
debug=True
from random import randint
img = mpimg.imread('test_images/test1.jpg')
out_img, boxes_detected = find_cars(np.copy(img), 390, 690, 0, 1280, 2.25, svc, X
                                    color_space=colorspace, spatial_size=(spatial
debug=False
```



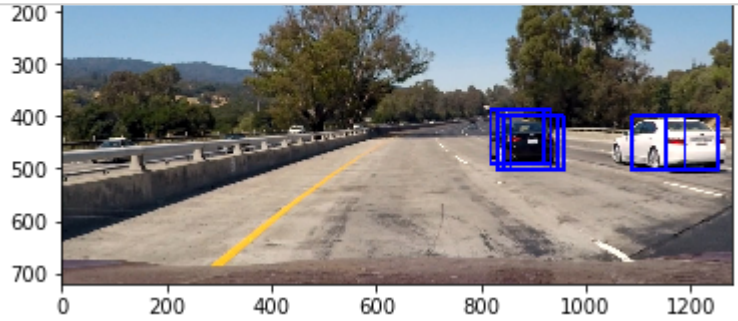

```
C:\Users\ckcheung\AppData\Local\Continuum\Miniconda3\envs\carnd2\lib\site-packa
ges\skimage\feature\_hog.py:119: skimage_deprecation: Default value of `block_n
orm`==`L1` is deprecated and will be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```

```
print sliders image with boxes
```

In [8]:
```
from scipy.ndimage.measurements import label

class car_tracking:
    # processing track_cars
    # setup parameters
    frame = 0
    #colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS (good), YUV (better), YCrCb
    #spatial_size=32
    #hist_bins=32
    # parameters for HOG classify
    #orient = 9
    #pix_per_cell = 8
    #cell_per_block = 2
    #hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
    #spatial_feat = True
    #hist_feat = True
    #hog_feat = True
    #heat_thresh = 3
    debug = False
    prev_heatmap_size = 20
    prev_heatmap = []
    img_and_boxes = []
    heatmap_hist = []


    def __init__(self):
        self.frame = 0
        self.prev_box_labels = []
        self.prev_heatmap_size = 20
        self.prev_heatmap = []
        self.img_and_boxes = []
        self.heatmap_hist = []
        print('Frame', self.frame, ':', colorspace,'colorspace |', orient,'orient
                'pixels_per_cell |', cell_per_block,'cells_per_block |', 'hog_cha
                '| spatial_feat=',spatial_feat, '| hist_feat=',hist_feat, '| hog_


    def track_cars(self, img):
        self.frame+=1
        boxes_detected = []
        total_boxes = []

        if debug:
            print('Frame', self.frame)

        # define region of interested for finding cars
        ystart = 350
        ystop = 656
        xstart = 0
        xstop = img.shape[1]
        #scale_list = [ 1.0,  1.25, 1.5, 1.75, 2.0, 2.1, 2.25, 2.5]
        #scale_list = [1.25, 1.5, 1.75, 2.1, 2.5]
        #scale_list = [1.25, 1.75, 2.1, 2.25] # last working setting
        scale_list = [1.2, 1.6, 2.0, 2.4] # last working setting
        changed_heatmap = 0
```

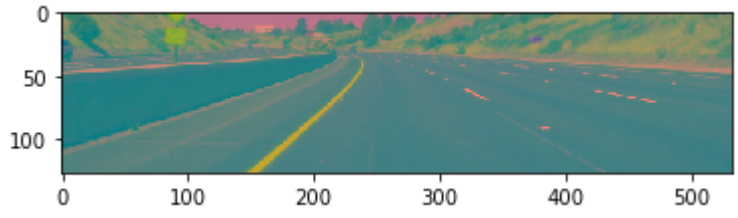
```
        # find cars in each scale in scale_list
        for scale in scale_list:
            if debug:
                out_img, boxes_detected = find_cars(np.copy(img), ystart, ystop, 
                                      color_space=colorspace, spatial_size=(spatial
            else:
                boxes_detected = find_cars(np.copy(img), ystart, ystop, xstart, x
                                      color_space=colorspace, spatial_size=(spatial
            total_boxes.extend(boxes_detected)

            if debug and boxes_detected != []:
                #print("find_cars: %d total boxes"%len(total_boxes))
                print("find_cars: frame:", self.frame, " scale:", scale, "boxes_d
                cv2.rectangle(out_img,(xstart, ystart),(xstop,ystop),(0,255,0),6)
                #scale_heatmap, scale_result = heat_map_tracking(img, boxes_detec
                #plotOrigAndNew(out_img, scale_heatmap, origTitle='tracked cars',

        # return image with consolidated boxes if vehicle is detected, otherwise,
        if total_boxes != []:
            if debug:
                new_heatmap, result = heat_map_tracking(img, total_boxes, heat_th
            else:
                new_heatmap = heat_map_tracking(img, total_boxes, heat_thresh)
            # pop head of prev_heatmap queue if full
            if len(self.prev_heatmap) == self.prev_heatmap_size:
                self.prev_heatmap.pop(0)
            # append new heatmap to prev_heatmap
            self.prev_heatmap.append(new_heatmap)

            # update heatmap_hist with new prev_heatmap item in queue and combine
            self.heatmap_hist = np.zeros((img.shape[0], img.shape[1]), dtype=np.f
            for heat_map in self.prev_heatmap:
                self.heatmap_hist = weighted_img(heat_map, self.heatmap_hist)
        elif self.prev_heatmap == []:
            # return original image with just printing frame number
            return cv2.putText(img, "Frame %d"%self.frame, (100,100), cv2.FONT_HE
            #self.img_and_boxes = img

        labels = label(self.heatmap_hist)
        self.img_and_boxes = draw_labeled_bboxes(np.copy(img), labels)

        # print frame number
        text = "Frame %d"%self.frame
        cv2.putText(self.img_and_boxes, text, (100,100), cv2.FONT_HERSHEY_SIMPLEX

        return self.img_and_boxes
```

In [9]:
```
debug = True
img = mpimg.imread('test_images/test1.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```

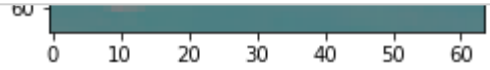


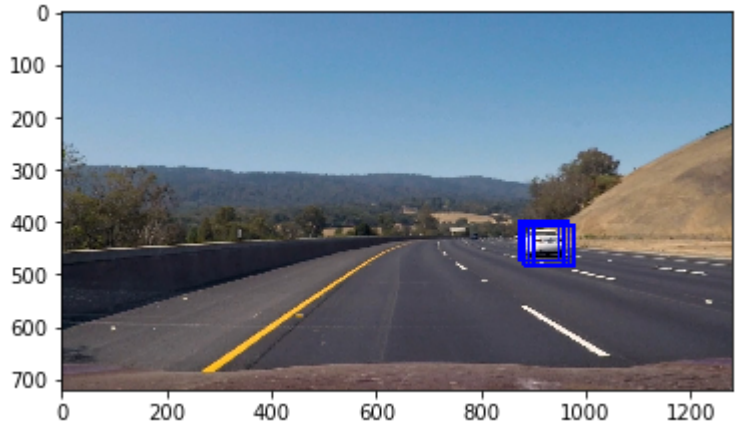

```
find_cars: frame: 1  scale: 1.6 boxes_detected: [((819, 388), (921, 490)),
((832, 388), (934, 490)), ((832, 401), (934, 503)), ((844, 401), (946, 503)),
((857, 401), (959, 503)), ((1088, 401), (1190, 503)), ((1152, 401), (1254, 50
3))]
```

In [10]:
```
img = mpimg.imread('test_images/test2.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```



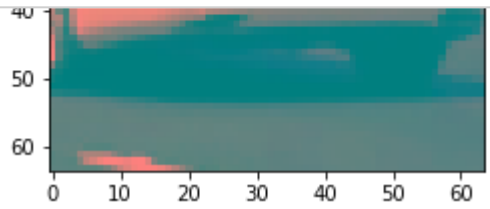
In [11]:
```
img = mpimg.imread('test_images/test3.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```

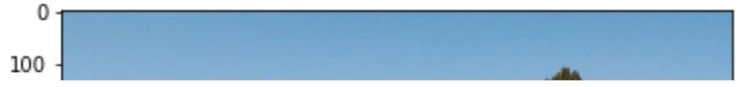




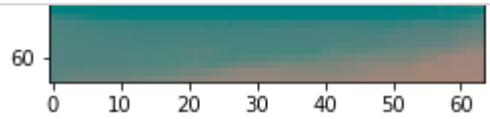

```
find_cars: frame: 1  scale: 1.2 boxes_detected: [((873, 398), (949, 474)),
((883, 398), (959, 474)), ((883, 407), (959, 483)), ((892, 398), (968, 474)),
((892, 407), (968, 483)), ((902, 407), (978, 483))]
```

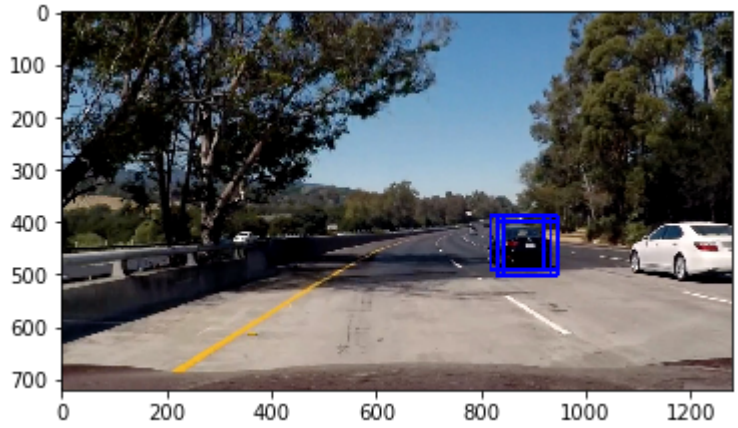
In [12]:
```
img = mpimg.imread('test_images/test4.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```





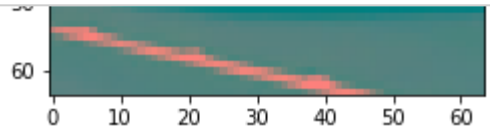
In [13]:
```
img = mpimg.imread('test_images/test5.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```

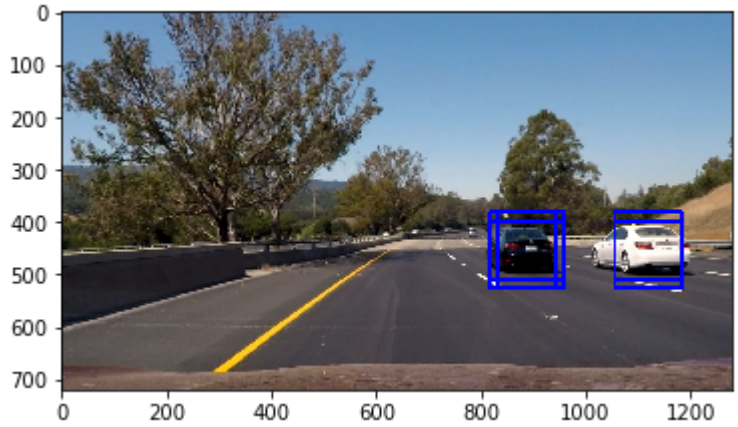





```
find_cars: frame: 1  scale: 1.6 boxes_detected: [((819, 388), (921, 490)),
((832, 388), (934, 490)), ((832, 401), (934, 503)), ((844, 388), (946, 490)),
```

In [14]:
```
img = mpimg.imread('test_images/test6.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```

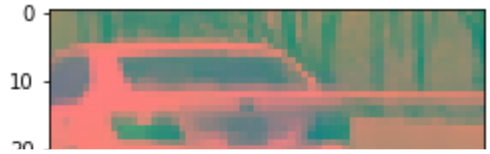




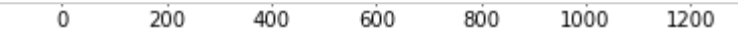

```
detected: scale  2.0  | xb 67 | yb 2 | xstart 0 | ystart 350 | xleft 536 | yt
```

In [15]:
```
img = mpimg.imread('test_images/test7.jpg')
debug=True
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```

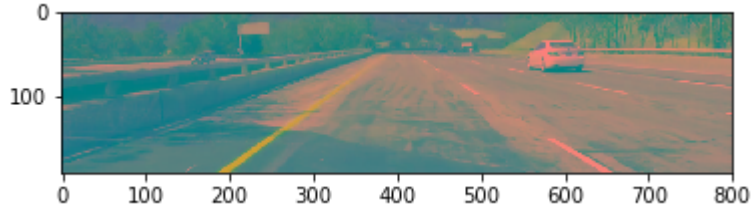

```
detected: scale  2.0  | xb 57 | yb 3 | xstart 0 | ystart 350 | xleft 456 | yt
op 24 | xleft_ori 912 | ytop_ori 48 | win_ori 128
```

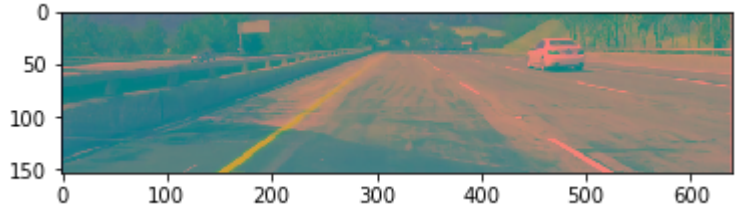


In [16]:
```
img = mpimg.imread('test_images/test8.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```



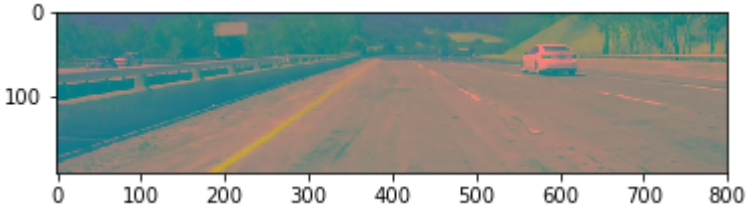

```
find_cars: frame: 1  scale: 1.2 boxes_detected: [((912, 388), (988, 464)),
((912, 398), (988, 474)), ((921, 388), (997, 464)), ((921, 398), (997, 474)),
((931, 398), (1007, 474))]
```





In [17]:
```
img = mpimg.imread('test_images/test9.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```


```
find_cars: frame: 1  scale: 1.2 boxes_detected: [((892, 398), (968, 474)),
((902, 398), (978, 474)), ((902, 407), (978, 483)), ((912, 398), (988, 474)),
((912, 407), (988, 483)), ((921, 398), (997, 474)), ((921, 407), (997, 483)),
((931, 407), (1007, 483))]
```




```
detected: scale  1.6  | xb 70 | yb 3 | xstart 0 | ystart 350 | xleft 560 | yt
op 24 | xleft_ori 896 | ytop_ori 38 | win_ori 102
```

In [18]:
```
img = mpimg.imread('test_images/test10.jpg')
print(img.shape)
plt.imshow(img)
plt.show()
a=car_tracking()
plt.imshow(a.track_cars(img))
plt.show()
```


```
find_cars: frame: 1  scale: 1.2 boxes_detected: [((902, 398), (978, 474)),
((912, 398), (988, 474)), ((912, 407), (988, 483)), ((921, 398), (997, 474)),
((921, 407), (997, 483)), ((931, 398), (1007, 474)), ((931, 407), (1007, 48
3))]
```

In [19]:
```
# Import everything needed to edit/save/watch video clips
from moviepy.editor import VideoFileClip
from IPython.display import HTML

debug = False

project_output = 'test_video_outputs/test_video3.mp4'
## To speed up the testing process you may want to try your pipeline on a shorter
## To do so add .subclip(start_second,end_second) to the end of the line below
## Where start_second and end_second are integer values representing the start an
## You may also uncomment the following line for a subclip of the first 5 seconds
#clip1 = VideoFileClip("test_videos/solidWhiteRight.mp4").subclip(0,5)
clip1 = VideoFileClip("test_video.mp4")
a=car_tracking()
project_clip = clip1.fl_image(a.track_cars) #NOTE: this function expects color im
%time project_clip.write_videofile(project_output, audio=False)
```

```
Frame 0 : YCrCb colorspace | 9 orientations | 8 pixels_per_cell | 2 cells_per_b
lock | hog_channel= ALL | spatial_feat= True | hist_feat= True | hog_feat= True
[MoviePy] >>>> Building video test_video_outputs/test_video3.mp4
[MoviePy] Writing video test_video_outputs/test_video3.mp4

 97%|████████████████████████████████████████████████████████████████████
██████  | 38/39 [03:11<00:05,  5.12s/it]

[MoviePy] Done.
[MoviePy] >>>> Video ready: test_video_outputs/test_video3.mp4

Wall time: 3min 12s
```

In [23]:
```
project_output = 'test_video_outputs/test_video3.mp4'
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(project_output))
```

Out[23]:

0:01 / 0:01

In [20]:
```
project_output = 'test_video_outputs/project_video2.mp4'
## To speed up the testing process you may want to try your pipeline on a shorter
## To do so add .subclip(start_second,end_second) to the end of the line below
## Where start_second and end_second are integer values representing the start an
## You may also uncomment the following line for a subclip of the first 5 seconds
#clip1 = VideoFileClip("test_videos/solidWhiteRight.mp4").subclip(0,5)
clip1 = VideoFileClip("project_video.mp4")
a=car_tracking()
project_clip = clip1.fl_image(a.track_cars) #NOTE: this function expects color im
%time project_clip.write_videofile(project_output, audio=False)
```

```
Frame 0 : YCrCb colorspace | 9 orientations | 8 pixels_per_cell | 2 cells_per_b
lock | hog_channel= ALL | spatial_feat= True | hist_feat= True | hog_feat= True
[MoviePy] >>>> Building video test_video_outputs/project_video2.mp4
[MoviePy] Writing video test_video_outputs/project_video2.mp4

100%|████████████████████████████████████████████████████████████████
██| 1260/1261 [1:44:27<00:04,  4.81s/it]

[MoviePy] Done.
[MoviePy] >>>> Video ready: test_video_outputs/project_video2.mp4

Wall time: 1h 44min 28s
```

In [22]:
```
HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(project_output))
```

Out[22]:

In [ ]: