In [32]:

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import cv2
import glob
import os
import time
import pickle
%matplotlib inline


from mpl_toolkits.mplot3d import Axes3D
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
from skimage import color, exposure
from moviepy.editor import VideoFileClip
from scipy.ndimage.measurements import label

# for scikit-learn version <= 0.17
# if you are using scikit-learn >= 0.18 then use this:
# from sklearn.model_selection import train_test_split
from sklearn.cross_validation import train_test_split


output_flag = True
debug_flag = False
```

In [33]:

```
##---------------------------------------Image correction and processin


def calibrate_camera(path,imgcornertype):

    if debug_flag == True:
        print("starting camera calibration")
        print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")

    #initialize arrays to store object and image points
    objpoints = [] #3d points in real world space
    imgpoints = [] #2d points in image space

    if output_flag == True:
        fig, axs = plt.subplots(5,4, figsize=(16, 11))
        fig.subplots_adjust(hspace = .2, wspace=.001)
        axs = axs.ravel()

    if imgcornertype == 69:
        objp = np.zeros((6*9,3), np.float32)
        objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
    elif imgcornertype == 68:
        objp = np.zeros((6*8,3),np.float32)
        objp[:,:2] = np.mgrid[0:8,0:6].T.reshape(-1,2)

    if imgcornertype == 69:
        cory = 9
        corx = 6
    elif imgcornertype == 68:
        corx = 6
        cory = 8
    index = 0

    for imagename in os.listdir(path):

        img = cv2.imread(path + imagename)
        if output_flag == True:
            print('calibrating camera for image', imagename)
            #plt.imshow(img)
            #plt.show()

        # Conver to grayscale
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

        # find chessboard corners
        ret,corners = cv2.findChessboardCorners(gray,(cory,corx),None)


        #if corners are found, add object points, img points
        if ret == True:
            imgpoints.append(corners)
            objpoints.append(objp)
            img = cv2.drawChessboardCorners(img,(cory,corx),corners,ret)
            if output_flag == True:
                axs[index].axis('off')
                axs[index].imshow(img)
```

```
                index += 1
            if debug_flag == True:
                print("corner identified for image",imagename)
                print("corner shape is ", corners.shape)
                print("corner is ", corners)

        else:
             if debug_flag == True:
                    print("corners not identified for image",imagename)


    #get calibration matrix
    ret,mtx,dist,rvecs,tvecs = cv2.calibrateCamera(objpoints,imgpoints,gr


    if debug_flag == True:
        print(gray.shape[::-1])
        print("ret is ",ret)
        print("mtx is", mtx)
        print("dist is", dist)
        print("rvecs is ", rvecs)
        print("tvecs is", tvecs)
        print("camera calibration done. ")
        print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")

    # delete unwanted objects
    del objpoints
    del imgpoints

    return ret,mtx,dist,rvecs,tvecs


# function undistors the image
def undistort_img(img,mtx,dist):
    #img = img.astype(np.float32) * 255
    #img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    img = cv2.undistort(img,mtx,dist,None,mtx)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255
    return img


# set ROI for an image img based on the input vertices
def set_ROI(img,vertices):
    mask = np.zeros_like(img)
    if len(img.shape) > 2:
        channel_count = img.shape[2]
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255
    cv2.fillPoly(mask, np.int32([vertices]), ignore_mask_color)
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

In [34]:

```
#------------------------------------ core routines--------------------

def load_images(path):
    images = glob.glob(path)
    cars_noncars = []
    for image in images:
        cars_noncars.append(image)
    return cars_noncars

def bin_spatial(img, color_space='RGB', size=(32, 32)):
    # Convert image to new color space (if specified)
    if color_space != 'RGB':
        if color_space == 'HSV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
        elif color_space == 'LUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2LUV)
        elif color_space == 'HLS':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
        elif color_space == 'YUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
        elif color_space == 'YCrCb':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    else: feature_image = np.copy(img)
    # Use cv2.resize().ravel() to create the feature vector
    features = cv2.resize(feature_image, size).ravel()
    # Return the feature vector
    return features

def draw_boxes(img, bboxes, color=(0, 0, 255), thick=4):
    # 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
        #print("printing from draw_boxes")
        #print("bbox0",bbox[0])
        #print("bbox1",bbox[1])

        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)
    # Return the image copy with boxes drawn

    return imcopy


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))
        #if (box[0][0]-box[0][1])  >= 64:
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1
    # Return updated heatmap

    return heatmap# Iterate through list of bboxes


```

```
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
    bboxes_list=[]
    bln_found = False

    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
        #print("xgap",np.max(nonzerox) - np.min(nonzerox))
        #print("ygap",np.max(nonzeroy) - np.min(nonzeroy))
        if ((np.max(nonzerox) - np.min(nonzerox)) > 30) and  ((np.max(non
            bln_found = True
            bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzero
            bboxes_list.append(bbox)
        #else:
            #print("skipped boxes")
            #print("xgap",np.max(nonzerox) - np.min(nonzerox))
            #print("ygap",np.max(nonzeroy) - np.min(nonzeroy))
        # Draw the box on the image
        if bln_found:
            cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 4)

    # Return the image

    return img,bboxes_list

# 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 RGB channels separately
    rhist = np.histogram(img[:,:,0], bins=32, range=(0, 256))
    ghist = np.histogram(img[:,:,1], bins=32, range=(0, 256))
    bhist = np.histogram(img[:,:,2], bins=32, range=(0, 256))
    # Generating bin centers
    bin_edges = rhist[1]
    bin_centers = (bin_edges[1:]  + bin_edges[0:len(bin_edges)-1])/2
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((rhist[0], ghist[0], bhist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return rhist, ghist, bhist, bin_centers, hist_features
```

In [35]:

```
#----------------------------------Feature extraction routines ---------

# Define a function to extract features from a single image window
# This function is very similar to extract_features()
# just for a single image rather than list of images

def single_img_features(img, 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)
    #1) Define an empty list to receive features
    img_features = []
    #2) Apply color conversion if other than 'RGB'
    if color_space != 'RGB':
        if color_space == 'HSV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
        elif color_space == 'LUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2LUV)
        elif color_space == 'HLS':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
        elif color_space == 'YUV':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
        elif color_space == 'YCrCb':
            feature_image = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    else: feature_image = np.copy(img)
    #3) Compute spatial features if flag is set
    if spatial_feat == True:
        spatial_features = bin_spatial(feature_image, size=spatial_size)
        #4) Append features to list
        img_features.append(spatial_features)
    #5) Compute histogram features if flag is set
    if hist_feat == True:
        hist_features = color_hist(feature_image, nbins=hist_bins)
        #6) Append features to list
        img_features.append(hist_features)
    #7) Compute HOG features if flag is set
    if hog_feat == True:
        if hog_channel == 'ALL':
            hog_features = []
            for channel in range(feature_image.shape[2]):
                hog_features.extend(get_hog_features(feature_image[:,:,ch
                                    orient, pix_per_cell, cell_per_block,
                                    vis=False, feature_vec=True))
        else:
            hog_features = get_hog_features(feature_image[:,:,hog_channel
                        pix_per_cell, cell_per_block, vis=False, feature_
        #8) Append features to list
        img_features.append(hog_features)

    #9) Return concatenated array of features
    return np.concatenate(img_features)


def get_hog_features(img, orient, pix_per_cell, cell_per_block, vis=False
    if vis == True:
        features, hog_image = hog(img, orientations=orient, pixels_per_ce
```

```
                                    cells_per_block=(cell_per_block, cell_p
                                    visualise=True, feature_vector=feature_
        return features, hog_image
    else:
        features = hog(img, orientations=orient, pixels_per_cell=(pix_per
                       cells_per_block=(cell_per_block, cell_per_block),
                       visualise=False, feature_vector=feature_vec)
        return features


def extract_hog_features(imgs, cspace='RGB', orient=9,
                         pix_per_cell=8, cell_per_block=2, hog_channel=0):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        # Read in each one by one

        #calibrate camera

        #path1 = "./camera_cal/"
        #ret,mtx,dist,rvecs,tvecs = calibrate_camera(path1,69)###########
        #dist_pickle = {}
        #dist_pickle["mtx"] = mtx
        #dist_pickle["dist"] = dist
        #pickle.dump( dist_pickle, open( "calib.pkl", "wb" ) )
        img = mpimg.imread(file)
        #with open("calib.pkl", 'rb') as file:
        #    data= pickle.load(file)
        #mtx = data['mtx']       # calibration matrix
        #dist = data['dist']     # distortion coefficients
        #img_undistort = undistort_img(img,mtx,dist)  # undistort the ima
        #img_flipped = np.fliplr(img)
        imglist = [img]
        #draw_simple_chart(img,img_flipped,"original","flipped")
        #draw_simple_chart(img_undistort,img_flipped,"original","undistor
        for image in imglist:
            # apply color conversion if other than 'RGB'
            if cspace != 'RGB':
                if cspace == 'HSV':
                    feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV
                elif cspace == 'LUV':
                    feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV
                elif cspace == 'HLS':
                    feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS
                elif cspace == 'YUV':
                    feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV
                elif cspace == 'YCrCb':
                    feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCr
            else: feature_image = np.copy(image)

            # 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[:,
                                        orient, pix_per_cell, cell_per_bl
```

```
                                        vis=False, feature_vec=True))
                hog_features = np.ravel(hog_features)
            else:
                hog_features = get_hog_features(feature_image[:,:,hog_cha
                            pix_per_cell, cell_per_block, vis=False, feat
            # Append the new feature vector to the features list
            features.append(hog_features)
        # Return list of feature vectors
    return features
```

In [37]:

```
#----------------------------------common Visualization routines---------
def draw_simple_chart(image1,image2,title1,title2):
    if output_flag == True:

        f,(ax1,ax2) = plt.subplots(1,2,figsize =(20,10))
        f.subplots_adjust(hspace=.2,wspace=.05)
        if (len(image1.shape) < 3):
            ax1.imshow(image1,cmap ='gray')
        else:
            ax1.imshow(image1)
        ax1.set_title(title1,fontsize=10)

        if (len(image2.shape) < 3):
            ax2.imshow(image2,cmap ='gray')
        else:
            ax2.imshow(image2)
        ax2.set_title(title2,fontsize=10)



def draw_histogram(image):
    rh, gh, bh, bincen, feature_vec = color_hist(image, nbins=32, bins_rar
    # Plot a figure with all three bar charts
    if rh is not None:
        fig = plt.figure(figsize=(12,3))
        plt.subplot(131)
        plt.bar(bincen, rh[0])
        plt.xlim(0, 256)
        plt.title('R Histogram')
        plt.subplot(132)
        plt.bar(bincen, gh[0])
        plt.xlim(0, 256)
        plt.title('G Histogram')
        plt.subplot(133)
        plt.bar(bincen, bh[0])
        plt.xlim(0, 256)
        plt.title('B Histogram')

    else:
        print('Your function is returning None for at least one variable..

def plot3d(pixels, colors_rgb, axis_labels=list("RGB"), axis_limits=((0, 2
    #Plot pixels in 3D."""

    # Create figure and 3D axes
    fig = plt.figure(figsize=(8, 8))
    ax = Axes3D(fig)

    # Set axis limits
    ax.set_xlim(*axis_limits[0])
    ax.set_ylim(*axis_limits[1])
    ax.set_zlim(*axis_limits[2])

    # Set axis labels and sizes
    ax.tick_params(axis='both', which='major', labelsize=14, pad=8)
    ax.set_xlabel(axis_labels[0], fontsize=16, labelpad=16)
```

```
    ax.set_ylabel(axis_labels[1], fontsize=16, labelpad=16)
    ax.set_zlabel(axis_labels[2], fontsize=16, labelpad=16)

    # Plot pixel values with colors given in colors_rgb
    ax.scatter(
        pixels[:, :, 0].ravel(),
        pixels[:, :, 1].ravel(),
        pixels[:, :, 2].ravel(),
        c=colors_rgb.reshape((-1, 3)), edgecolors='none')

    return ax  # return Axes3D object for further manipulation
```

In [38]:

```
#--------------------------------------Image feature visualization

def explore_colorspace(img):
    #You can study the distribution of color values in an image by plottir

    # Select a small fraction of pixels to plot by subsampling it
    scale = max(img.shape[0], img.shape[1], 64) / 64  # at most 64 rows ar
    img_small = cv2.resize(img, (np.int(img.shape[1] / scale), np.int(img.

    # Convert subsampled image to desired color space(s)
    img_small_RGB = cv2.cvtColor(img_small, cv2.COLOR_BGR2RGB)  # OpenCV u
    img_small_HSV = cv2.cvtColor(img_small, cv2.COLOR_BGR2HSV)
    img_small_rgb = img_small_RGB / 255.  # scaled to [0, 1], only for plo

    # Plot and show
    plot3d(img_small_RGB, img_small_rgb)
    plt.show()

    plot3d(img_small_HSV, img_small_rgb, axis_labels=list("HSV"))
    plt.show()


def hog_visualization(image):

    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    # Define HOG parameters
    orient = 9
    pix_per_cell = 8
    cell_per_block = 2
    # Call our function with vis=True to see an image output
    features, hog_image = get_hog_features(gray, orient, pix_per_cell, cel

    return features,hog_image
```

In [39]:

```
#-----------------------------------------Training and prediction rout

def extract_data(cars,notcars, cspace='RGB',orient=9,pix_per_cell=8,
                    cell_per_block=2,hog_channel=0,spatial_size=(16,16),h
                    spatial_feat=True, hist_feat=True, hog_feat=True):

    t=time.time()

    car_features = extract_hog_features(cars, cspace, orient, pix_per_cel
    notcar_features = extract_hog_features(notcars, cspace, orient, pix_p


    if debug_flag == True:
        print(len(car_features), " is length of car features")
        print(len(notcar_features), " is length of not car features")
        print(len(car_features[0]))
        print(len(notcar_features[0]))


    X = np.vstack((car_features, notcar_features)).astype(np.float64)
    # 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_featur


    # 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, te


    if debug_flag == True:
        print('Using:',orient,'orientations',pix_per_cell,'pixels per cel
        print('Feature vector length:', len(X_train[0]))
    t2 = time.time()
    print(round(t2-t, 2), 'Seconds to extract features')
    return X_train,X_test,y_train,y_test


def train_classifier(X_train,X_test,y_train,y_test,reset_flag = True):

    if reset_flag == True:
        svc = LinearSVC()
    #load classifier
    else:
        with open('classifier.pkl', 'rb') as fid:
            svc = pickle.load(fid)


    # Check the training time for the SVC
    if debug_flag == True:
        t=time.time()
    svc.fit(X_train, y_train)
```

```
    if debug_flag == True:
        t2 = time.time()
        print(round(t2-t, 2), 'Seconds to train SVC...')
    # Check the score of the SVC
    accuracy = round(svc.score(X_test, y_test), 4)
    if debug_flag == True:
        print('Test Accuracy of SVC = ',accuracy )

    # save the classifier
    with open('classifier.pkl', 'wb') as fid:
        pickle.dump(svc, fid)

    return svc, accuracy


# Define a single function that can extract features using hog sub-sampli
def find_cars(img, ystart, ystop, xstart, xstop,scale, cspace, hog_channe
              pix_per_cell, cell_per_block, spatial_size, hist_bins, show

    # array of rectangles where cars were detected
    rectangles = []

    img = img.astype(np.float32)/255

    img_tosearch = img[ystart:ystop,xstart:xstop,:]

    #img_tosearch = img[ystart:ystop,:,:]

    if output_flag == True:
        draw_simple_chart(img,img_tosearch,"original","scale" + str(scale

    # apply color conversion if other than 'RGB'

    if cspace != 'RGB':
        if cspace == 'HSV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2HS
        elif cspace == 'LUV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2LU
        elif cspace == 'HLS':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2HL
        elif cspace == 'YUV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2YU
        elif cspace == 'YCrCb':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2YC
    else: ctrans_tosearch = np.copy(img)

    # rescale image if other than 1.0 scale
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/

    # select colorspace channel for HOG

    if hog_channel == 'ALL':
        ch1 = ctrans_tosearch[:,:,0]
        ch2 = ctrans_tosearch[:,:,1]
        ch3 = ctrans_tosearch[:,:,2]
```

```
    else:
        ch1 = ctrans_tosearch[:,:,hog_channel]

    # Define blocks and steps as above
    nxblocks = (ch1.shape[1] // pix_per_cell)+1  #-1
    nyblocks = (ch1.shape[0] // pix_per_cell)+1  #-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)-1
    cells_per_step = 2  # Instead of overlap, define how many cells to st
    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, fe
    if hog_channel == 'ALL':
        hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block
        hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block

    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+nblo
            if hog_channel == 'ALL':
                hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+
                hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+
                hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3
            else:
                hog_features = hog_feat1

            xleft = xpos*pix_per_cell
            ytop = ypos*pix_per_cell

            test_features = hog_features.reshape(1,-1)

            with open('classifier.pkl', 'rb') as fid:
                svc = pickle.load(fid)


            test_prediction = svc.predict(test_features)

            if test_prediction == 1 or show_all_rectangles:
                xbox_left = np.int(xleft*scale)
                ytop_draw = np.int(ytop*scale)
                win_draw = np.int(window*scale)

                rectangles.append(((xbox_left +xstart, ytop_draw+ystart),
                #print(((xbox_left, ytop_draw+ystart),(xbox_left+win_draw

    return rectangles
```

In [40]:

```
# Define a class to receive the characteristics of each line detection
class rectangles():

    def __init__(self):
        self.bbox_list = []
        self.count = 0
        self.buff_count = 0
        #print("self initialized")

    def save_bboxes(self,bboxes=[], *args):
        #print("saved box",bboxes)
        del self.bbox_list
        self.bbox_list = bboxes

    def get_bboxes(self,bboxes =[], *args):
        return self.bbox_list

    def check_first_time(self):
        #print("checking first time")
        #print("self.count",self.count)
        return (self.count == 0)

    def inc_count(self):
        self.count += 1

    def check_scan_status (self,count):
        #print("check reset flag running")
        #print("self.count",self.count)
        #print("self.count mod",self.count % count)
        self.count += 1
        if ((self.count % count) == 0 ):
            return True
        else:
            return False



    def check_empty_status(self,count,bboxes=[],*args):
        if debug_flag == True:
            print("entering check_empty_Status")
            print(bboxes)
            print(len(bboxes))

        for bbox in bboxes:
            if len(bbox) > 0 :
                bln_empty = False
                return False

        if debug_flag == True:
            print("emtpy frame found")
        self.buff_count += 1
        if (self.buff_count <= count):
            if debug_flag == True:
                print("buffered")
            return True
        else:
```

```
            if debug_flag == True:
                print("threshold exceeded. empty frame not buffered")
            self.buff_count = 0
            return False

```

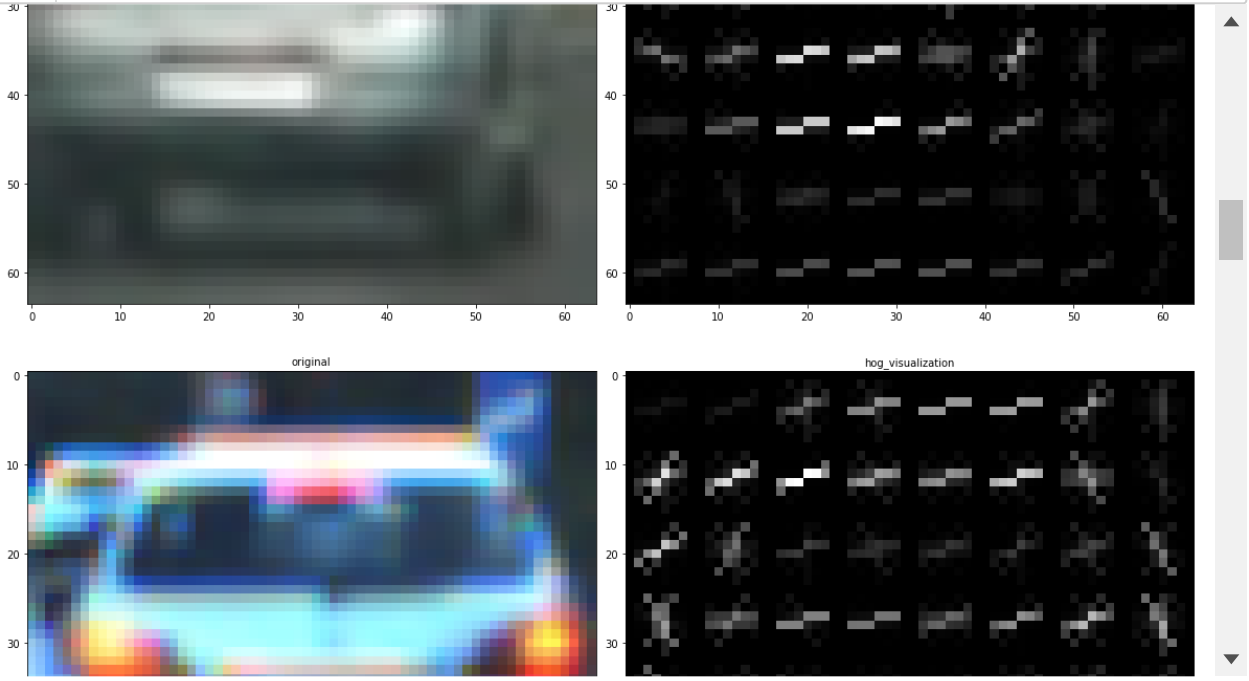
In [41]:

```
# main function starts
# load dataset and check
path_vehicle = './data/vehicles/*.png'
path_non_vehicle = './data/non-vehicles/*.png'
cars = load_images(path_vehicle)
notcars = load_images(path_non_vehicle)
print(len(cars),len(notcars))
```

```
8792 8968
```

In [42]:
```
# randomly display few images
fig,axis = plt.subplots(4,8,figsize=(16,16))
fig.subplots_adjust(hspace = .2, wspace=.001)
axis = axis.ravel()

for i in range(0,16):
    rnd = np.random.randint(0,len(cars))
    img = cv2.imread(cars[rnd])
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    axis[i].axis('off')
    axis[i].set_title('car', fontsize=10)
    axis[i].imshow(img)

for i in range(16,32):
    rnd = np.random.randint(0,len(notcars))
    img = cv2.imread(notcars[rnd])
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    axis[i].axis('off')
    axis[i].set_title('not car', fontsize=10)
    axis[i].imshow(img)
```

In [43]:
```
#show hog visualization
for i in range(0,5):
    rnd = np.random.randint(0,len(cars))
    img = mpimg.imread(cars[rnd])
    features,hog_image = hog_visualization(img)
    draw_simple_chart(img,hog_image,"original","hog_visualization")

for i in range(0,5):
    rnd = np.random.randint(0,len(notcars))
    img = mpimg.imread(notcars[rnd])
    features,hog_image = hog_visualization(img)
    draw_simple_chart(img,hog_image,"original","hog_visualization")
```



In [ ]:
```
# explore color space randomly for two images for each categories
print('printing color spaces for cars')
for i in range(0,2):
    rnd = np.random.randint(0,len(cars))
    img = mpimg.imread(cars[rnd])
    explore_colorspace(img)

print('printing color spaces for non cars')
for i in range(0,2):
    rnd = np.random.randint(0,len(notcars))
    img = mpimg.imread(notcars[rnd])
    explore_colorspace(img)
```

In [ ]:
```
# view randomly histogram for few images for each categories

for i in range(0,5):
    rnd = np.random.randint(0,len(cars))
    img = mpimg.imread(cars[rnd])
    draw_histogram(img)


for i in range(0,5):
    rnd = np.random.randint(0,len(notcars))
    img = mpimg.imread(notcars[rnd])
    draw_histogram(img)

```

In [ ]:
```
# extract features and prepare data and train model
# run one time only

cspace_list = ['HSV','LUV','HLS','YUV']
orient_list = [9,10,11,12]
pix_per_cell_list =[8,16]

cell_per_block=2
hog_channel='ALL'
spatial_size=(32,32)
hist_bins=16
hist_range=(0, 256)
spatial_feat=True
hist_feat=True
hog_feat=True
extract_feature_type="hog"

reset_flag = True

for cspace in cspace_list:
    for pix_per_cell in pix_per_cell_list:
        for orient in orient_list:
            X_train,X_test,y_train,y_test,X_scaler = extract_data(cars,not
                                                                  cell_per
                                                                  spatial_
                                                                  spatial_


            svc, accuracy = train_classifier(X_train,X_test,y_train,y_test
            acc = 100 * accuracy
            print('%.2f'  % acc,"%    cpsace " ,cspace,"     orient-",orie
                  "     spatial_size",spatial_size)
            'The value of 1/3 to 3 decimal places is {0:1.3f}'.format(1./3
            print('   ')
```

In [47]:
```
cspace='HSV'
orient= 8 #9
pix_per_cell= 6# 8
cell_per_block=2  #2
hog_channel='ALL'
spatial_size=(16,16)
hist_bins=8
hist_range=(0, 256)
spatial_feat=True
hist_feat=True
hog_feat=True
y_start=350
y_stop=700
extract_feature_type="hog"
```

In [ ]:
```
X_train,X_test,y_train,y_test = extract_data(cars,notcars, cspace,orient,p
                                                        cell_per_block,hog_c
                                                        spatial_size,hist_bi
                                                        spatial_feat,hist_fe

svc, accuracy = train_classifier(X_train,X_test,y_train,y_test)

print("length of X_train",len(X_train))
print("length of X_test",len(X_test))
print("length of Y_train",len(y_train))
print("length of Y_test",len(y_test))



acc = 100 * accuracy
print("accuracy is", '%.2f'  % acc,"%    cpsace " ,cspace,"     orient-",c
     "     spatial_size",spatial_size)
```

In [48]:
```
#Load svc
with open('classifier.pkl', 'rb') as fid:
    svc = pickle.load(fid)

```

In [49]:

```
# check if model is working fine and if cars are being identified correctl

def test_model_on_multi_images(path,ystart=350,ystop=700,xstart=400, xstop

    images = glob.glob(path)
    output_flag = True
    debug_flag = False

    for i, im in enumerate(images):
        test_image = mpimg.imread(im)
        bboxes = find_cars(test_image, ystart, ystop, xstart,xstop,scale,
        output_image = draw_boxes(test_image,bboxes)
        draw_simple_chart(test_image,output_image,"original","identified")
        #print(len(bboxes), 'bboxes found in image')
    del bboxes
```

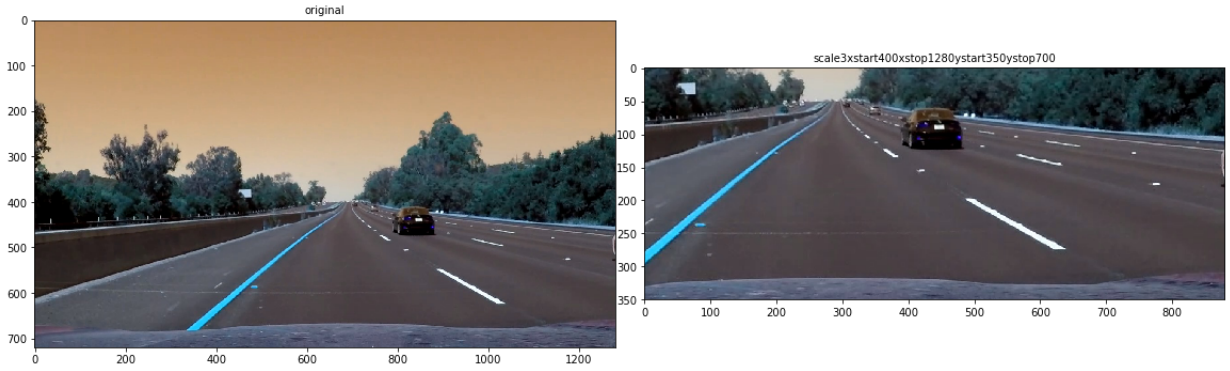
In [50]:

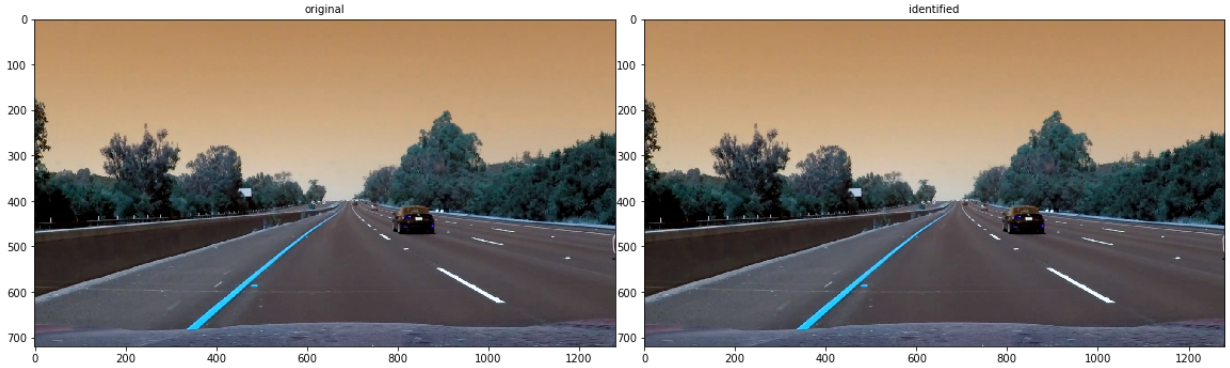
```
# run find_cars on multiple start stop and scale to identify the correct

def tune_threshold_for_multi_images(path,thresh):


    output_flag = True
    images = glob.glob(path)


    for i, im in enumerate(images):
        bboxes_list = []

        if debug_flag == True:
            print("image path is ",im)
        test_image = mpimg.imread(im)
        if debug_flag == True:
            print(test_image.shape)

    #------------------------------

        ystart = 360
        ystop =  510
        scale = 1.5
        xstart = 600
        xstop = 1100
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup1
        ystart = 360
        ystop =  510
        scale =  2
        xstart = 600
        xstop = 1200
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 360
        ystop =  510
        scale =  2
        xstart = 600
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup
        ystart = 360
        ystop =  510
        scale =  2.5
        xstart = 600
        xstop = 1200
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 380
        ystop =  560
        scale =  2
        xstart = 580
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
```

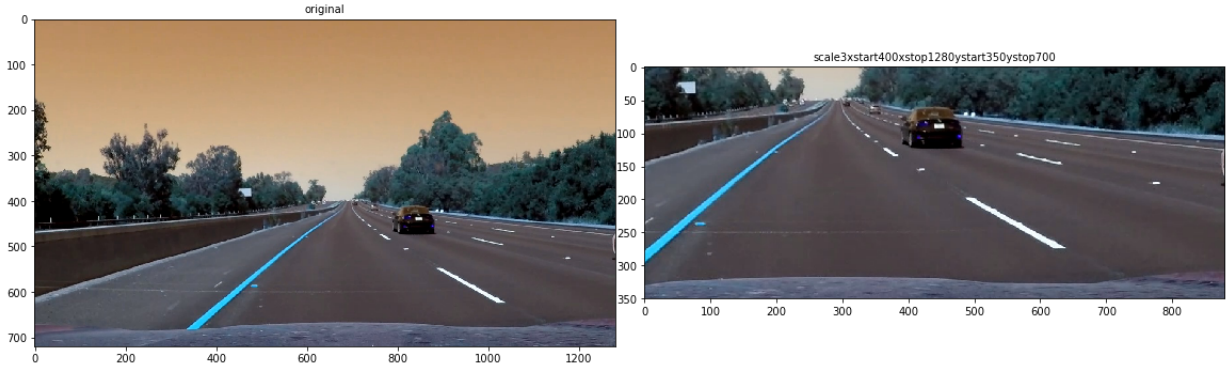
```
57
58          ystart = 380
59          ystop = 560
60          scale = 2.5
61          xstart = 580
62          xstop = 1280
63          bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
64
65          #dup
66          ystart = 380
67          ystop = 560
68          scale = 3
69          xstart = 580
70          xstop = 1280
71          bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
72
73          #dupfinal
74          ystart = 390
75          ystop = 490
76          scale = 2.5
77          xstart = 580
78          xstop = 1280
79          bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
80
81          ystart = 400
82          ystop = 500
83          scale = 1.5
84          xstart = 450
85          xstop = 1280
86          bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
87
88          ystart = 500
89          ystop = 600
90          scale = 3
91          xstart = 450
92          xstop = 1280
93          bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
94
95          ystart = 500
96          ystop =  600
97          scale = 3.5
98          xstart = 450
99          xstop = 1280
100         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
101
102         #dup
103         ystart = 500
104         ystop =  600
105         scale = 4
106         xstart = 450
107         xstop = 1280
108
109
110         bboxes_list = [item for sublist in bboxes_list for item in sublis
111         output_image = draw_boxes(test_image,bboxes_list)
112         draw_simple_chart(test_image,output_image,"original","boxed")
113
```

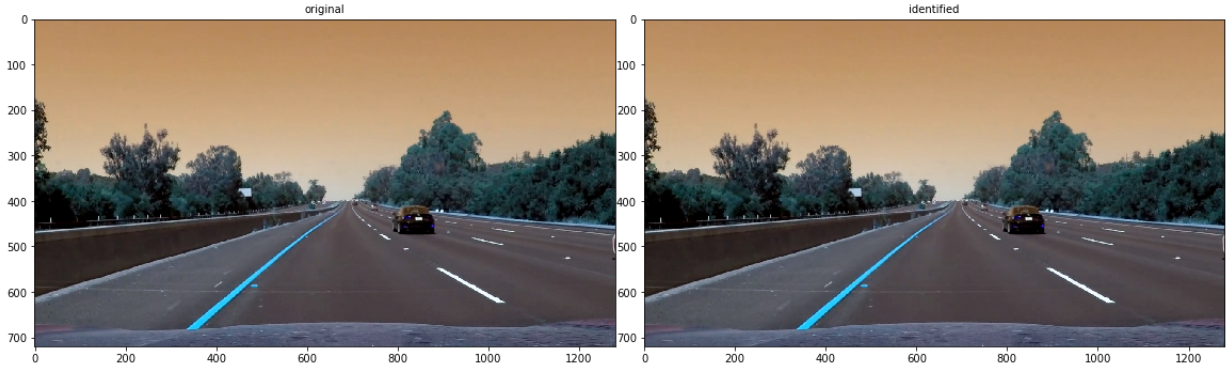
```
        # view heatmap
        heatmap_image = np.zeros_like(test_image[:,:,0])
        heatmap_image = add_heat(heatmap_image, bboxes_list)
        plt.figure(figsize=(10,10))
        plt.imshow(heatmap_image, cmap='hot')

        #apply threshold
        heatmap_image = apply_threshold(heatmap_image, thresh)
        plt.figure(figsize=(10,10))
        plt.imshow(heatmap_image, cmap='hot')

        labels = label(heatmap_image)
        plt.figure(figsize=(10,10))
        plt.imshow(labels[0], cmap='gray')
        if debug_flag == True:
            print(labels[1], 'cars found')

        # Draw bounding boxes on a copy of the image
        draw_img, rect = draw_labeled_bboxes(np.copy(test_image), labels)
        # Display the image
        plt.figure(figsize=(10,10))
        plt.imshow(draw_img)
        if debug_flag == True:
            print('...')
```

In [62]:
```
# test model on test and images saved from test video frame
#path = './test_images/*.jpg'
#test_model_on_multi_images(path,350,700,400,1280,3)
path = './saved_video/*.jpg'
test_model_on_multi_images(path,350,700,400,1280,3)
```

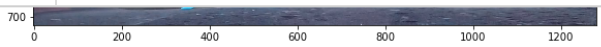


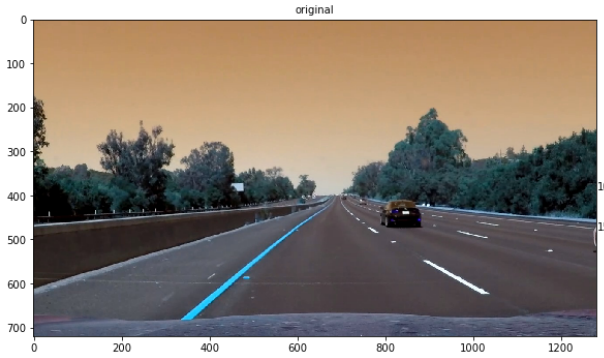


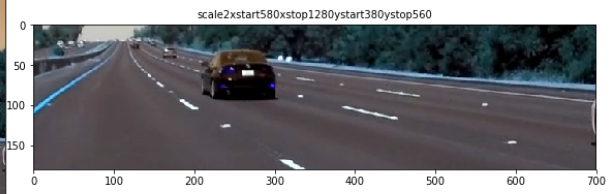




In [61]:
```
# fine tune threshold and heatmap for test and images saved from test vide
path = './saved_video/*.jpg'
debug_flag = False
output_flag = True
tune_threshold_for_multi_images(path,3)
```







original

In [54]:
```
threshold = 3
```

In [55]:

```
#define pipeline for processing image


def process_image(test_image):
    bboxes_list = []

    if rect.check_first_time():
         #print("scanning image first time to find rectangles")

        rect.inc_count()

        ystart = 360
        ystop =  510
        scale = 1.5
        xstart = 600
        xstop = 1100
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup1
        ystart = 360
        ystop =  510
        scale =  2
        xstart = 600
        xstop = 1200
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 360
        ystop =  510
        scale =  2
        xstart = 600
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup
        ystart = 360
        ystop =  510
        scale =  2.5
        xstart = 600
        xstop = 1200
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 380
        ystop =  560
        scale =  2
        xstart = 580
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 380
        ystop = 560
        scale = 2.5
        xstart = 580
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup
```

```
        ystart = 380
        ystop = 560
        scale = 3
        xstart = 580
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dupfinal
        ystart = 390
        ystop = 490
        scale = 2.5
        xstart = 580
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 400
        ystop = 500
        scale = 1.5
        xstart = 450
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 500
        ystop = 600
        scale = 3
        xstart = 450
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 500
        ystop =  600
        scale = 3.5
        xstart = 450
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup
        ystart = 500
        ystop =  600
        scale = 4
        xstart = 450
        xstop = 1280

        bboxes_list = [item for sublist in bboxes_list for item in sublis
        output_image = draw_boxes(test_image,bboxes_list)

        heatmap_image = np.zeros_like(test_image[:,:,0])
        heatmap_image = add_heat(heatmap_image, bboxes_list)
        heatmap_image = apply_threshold(heatmap_image, threshold)
        labels = label(heatmap_image)

        draw_img, rects = draw_labeled_bboxes(np.copy(test_image), labels

        rect.save_bboxes(rects)
        del rects
        if debug_flag == True:
            print("saved rect boxes")
```

```
114
115
116     if  rect.check_scan_status(50):
117         #print("scanning image to find rectangles")
118
119         ystart = 360
120         ystop =  510
121         scale = 1.5
122         xstart = 600
123         xstop = 1100
124         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
125
126         #dup1
127         ystart = 360
128         ystop =  510
129         scale =  2
130         xstart = 600
131         xstop = 1200
132         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
133
134         ystart = 360
135         ystop =  510
136         scale =  2
137         xstart = 600
138         xstop = 1280
139         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
140
141         #dup
142         ystart = 360
143         ystop =  510
144         scale =  2.5
145         xstart = 600
146         xstop = 1200
147         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
148
149         ystart = 380
150         ystop =  560
151         scale =  2
152         xstart = 580
153         xstop = 1280
154         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
155
156         ystart = 380
157         ystop = 560
158         scale = 2.5
159         xstart = 580
160         xstop = 1280
161         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
162
163         #dup
164         ystart = 380
165         ystop = 560
166         scale = 3
167         xstart = 580
168         xstop = 1280
169         bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs
170
```

```
        #dupfinal
        ystart = 390
        ystop = 490
        scale = 2.5
        xstart = 580
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 400
        ystop = 500
        scale = 1.5
        xstart = 450
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 500
        ystop = 600
        scale = 3
        xstart = 450
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        ystart = 500
        ystop =  600
        scale = 3.5
        xstart = 450
        xstop = 1280
        bboxes_list.append(find_cars(test_image, ystart, ystop, xstart,xs

        #dup
        ystart = 500
        ystop =  600
        scale = 4
        xstart = 450
        xstop = 1280

        if rect.check_empty_status(10,bboxes_list):
            if debug_flag == True:
                print("empty rects found. buffering")
            rects = rect.get_bboxes()
            draw_img = np.copy(test_image)
            for item in rects:
                #print("item0",item[0])
                #print("item1",item[1])
                draw_img = cv2.rectangle(draw_img, item[0], item[1], (255
        else:
            bboxes_list = [item for sublist in bboxes_list for item in su
            output_image = draw_boxes(test_image,bboxes_list)

            heatmap_image = np.zeros_like(test_image[:,:,0])
            heatmap_image = add_heat(heatmap_image, bboxes_list)
            heatmap_image = apply_threshold(heatmap_image, threshold)
            labels = label(heatmap_image)

            draw_img, rects = draw_labeled_bboxes(np.copy(test_image), la

            rect.save_bboxes(rects)
```

```
            del rects
            if debug_flag == True:
                print("saved rect boxes")
    else:
        #rect.save_bboxes(rects)
        #print("not scanning rects. simply retrieving last rect")
        rects = rect.get_bboxes()
        #print(rects)
        draw_img = np.copy(test_image)
        for item in rects:
            #print("item0",item[0])
            #print("item1",item[1])
            draw_img = cv2.rectangle(draw_img, item[0], item[1], (255,0,0
        del rects
    del bboxes_list
    return draw_img
```

In [63]:

```
# test the pipeline with all test images:

test_images = glob.glob('./saved_video/*.jpg')
output_flag = True
debug_flag = True
rect = rectangles()

for i, im in enumerate(test_images):
    plt.figure(figsize=(10,10))
    plt.imshow(process_image(mpimg.imread(im)))
```

```
saved rect boxes

<matplotlib.figure.Figure at 0x7f5e36c31eb8>
```

In [58]:
```
rect = rectangles()
debug_flag = False
output_flag = False
test_out_file = 'project_video_lane_out.mp4'
clip_test = VideoFileClip('project_video_lane.mp4')
clip_test_out = clip_test.fl_image(process_image)
%time clip_test_out.write_videofile(test_out_file, audio=False)
```

```
[MoviePy] >>>> Building video project_video_lane_out.mp4
[MoviePy] Writing video project_video_lane_out.mp4

100%|█████████▉| 1260/1261 [00:43<00:00, 28.66it/s]

[MoviePy] Done.
[MoviePy] >>>> Video ready: project_video_lane_out.mp4

CPU times: user 25.8 s, sys: 2.88 s, total: 28.7 s
Wall time: 44.9 s
```

In [ ]:
```

```