Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- 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.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

[Rubric](https://review.udacity.com/#!/rubrics/513/view) Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Data reparation

In [1]:
```
# Include used module and function
import cv2
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from lesson_functions import *
import glob
import time
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from scipy.ndimage.measurements import label
%matplotlib inline

# Define a function to return some characteristics of the dataset
def data_look(car_list, notcar_list):
    data_dict = {}
    # Define a key in data_dict "n_cars" and store the number of car images
    data_dict["n_cars"] = len(car_list)
    print(data_dict["n_cars"])
    # Define a key "n_notcars" and store the number of notcar images
    data_dict["n_notcars"] = len(notcar_list)
    print(data_dict["n_notcars"])
    # Read in a test image, either car or notcar
    # Define a key "image_shape" and store the test image shape 3-tuple
    img = cv2.imread(car_list[0])
    data_dict["image_shape"] = img.shape
    # Define a key "data_type" and store the data type of the test image.
    data_dict["data_type"] = img.dtype
    # Return data_dict
    return data_dict
```

In [2]:
```
notcar_path = ['./training_samples/non-vehicles/GTI/*.png',
               './training_samples/non-vehicles/Extras/*.png',]
car_path = ['./training_samples/vehicles/GTI_Far/*.png',
           './training_samples/vehicles/GTI_Left/*.png',
           './training_samples/vehicles/GTI_MiddleClose/*.png',
           './training_samples/vehicles/GTI_Right/*.png',
            './training_samples/vehicles/KITTI_extracted/*.png',]

cars = []
for path in car_path:
    cars.extend(glob.glob(path))

notcars = []
for path in notcar_path:
    notcars.extend(glob.glob(path))

data_info = data_look(cars, notcars)

print('Your function returned a count of',
      data_info["n_cars"], ' cars and',
      data_info["n_notcars"], ' non-cars')
print('of size: ',data_info["image_shape"], ' and data type:',
      data_info["data_type"])
```

```
8792
8968
Your function returned a count of 8792  cars and 8968  non-cars
of size:  (64, 64, 3)  and data type: uint8
```

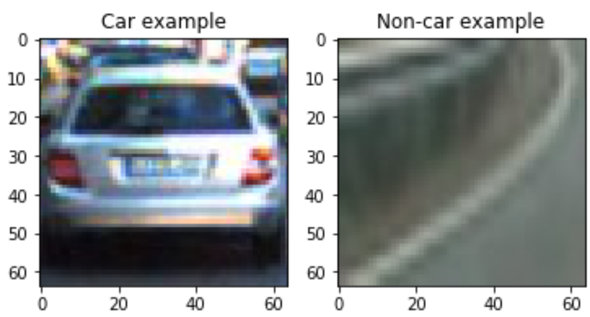
Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images.

I select one image from each of the car and non-car data set to demonstrate the difference in HOG features. Here are the example:

In [3]:
```
car_image = mpimg.imread('./output_images/40.jpeg')
notcar_image = mpimg.imread('./output_images//image0186.jpeg')

plt.subplot(121)
plt.imshow(car_image)
plt.title('Car example')
plt.subplot(122)
plt.imshow(notcar_image)
plt.title('Non-car example')
```

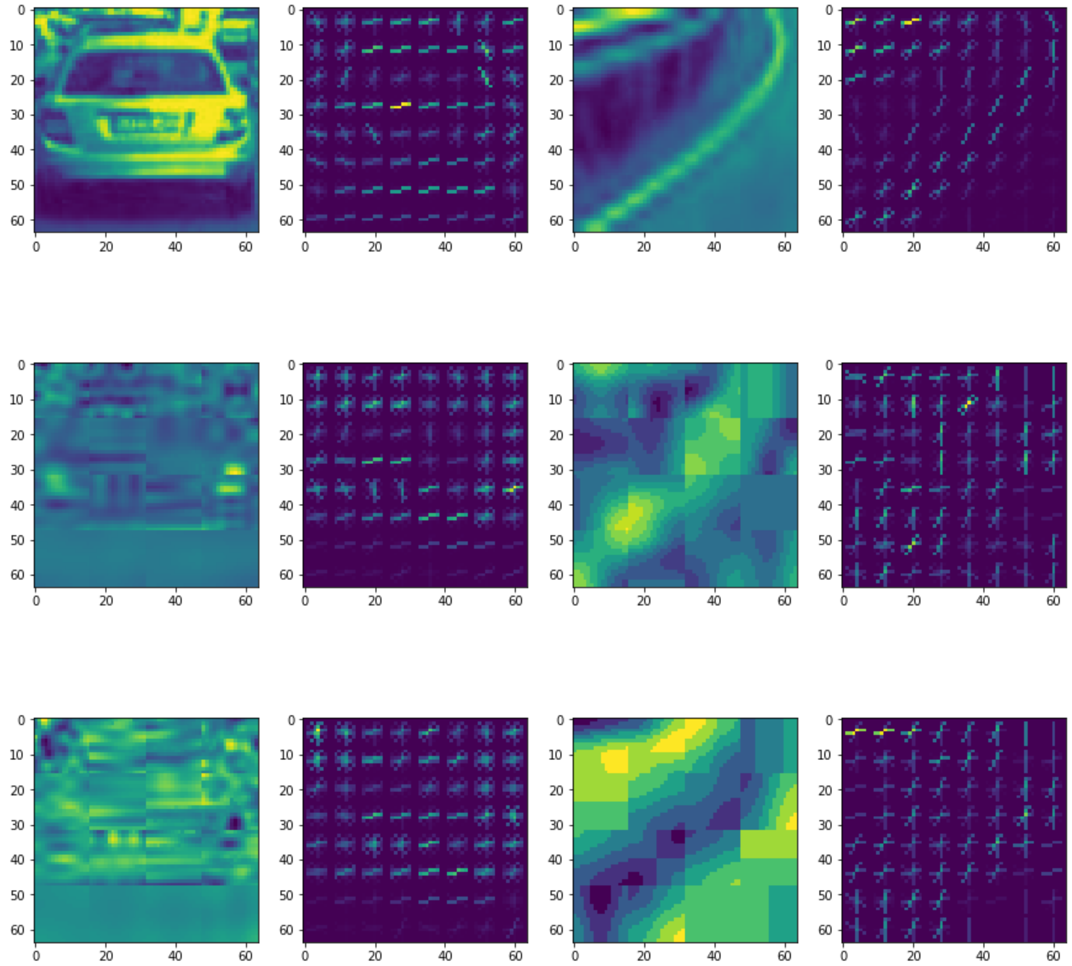
Out[3]: <matplotlib.text.Text at 0x7fea8fdcea58>



The `get_hog_features()` in `lesson_functions.py` (line 6 through line 23) is parameter wrapper to `skimage.hog()`.

I then explored different color spaces and different `skimage.hog()` parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like. An example displaying hog feature with repect to each channel for the two images is showing below:

In [4]:
```
car_cs_img = cv2.cvtColor(car_image, cv2.COLOR_RGB2YCrCb)
notcar_cs_img = cv2.cvtColor(notcar_image, cv2.COLOR_RGB2YCrCb)

row,col = 3,4
f, ax = plt.subplots(row, col, figsize=(15,15))
for r in range(row):
    plt.subplot(row, col, 1 + r * col)
    plt.imshow(car_cs_img[:,:,r])

    plt.subplot(row, col, 2 + r * col)
    features,vis_img = get_hog_features(car_cs_img[:,:,r], orient=9, pix_pe
    plt.imshow(vis_img)

    plt.subplot(row, col, 3 + r * col)
    plt.imshow(notcar_cs_img[:,:,r])

    plt.subplot(row, col, 4 + r * col)
    features,vis_img = get_hog_features(notcar_cs_img[:,:,r], orient=9, pix_
    plt.imshow(vis_img)
```



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

I tried various combinations of parameters on my SVM. Consider the example using YCbCr color spaces above, it is a good choice that this color space is better for object shape detection since the SVM accuracy below shows higher accuracy (99.01% v.s. 98.8%) compared to other color spaces. The HOG features visualization above are apparent for the two images.

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

By utilizing `extract_features()` in `lesson_functions.py` (line 46 through line 94), I use the following parameters for my HOG features:

- color space: YCrCb
- orientation: 9
- pix per cell: 8
- cell_per_block: 2
- HOG channel: ALL

In addition, the spatial and color features are also used:

- spatial size: 32
- color histbin: 64

The three features are collected and are normalized by using `StandardScaler()`. Scaled features are split and fed into a linear SVC. The following code cell shows the accuracy of training result on test set:

In [5]:

```
cspace='YCrCb'
spatial = 32
histbin = 64

orient=9
pix_per_cell=8
cell_per_block=2
hog_channel='ALL'

car_features = extract_features(cars, color_space=cspace, spatial_size=(spa
                        hist_bins=histbin, orient=orient, pix_per_cell=pix_p
                        spatial_feat=True, hist_feat=True, hog_feat=True)
notcar_features = extract_features(notcars, color_space=cspace, spatial_size
                        hist_bins=histbin, orient=orient, pix_per_cell=pix_p
                        spatial_feat=True, hist_feat=True, hog_feat=True)

from sklearn.preprocessing import StandardScaler
# Create an array stack of feature vectors
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 a labels vector based on features lists
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))

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:', len(X_train[0]))

# Use a linear SVC
svc = LinearSVC()

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 = 10
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')
```

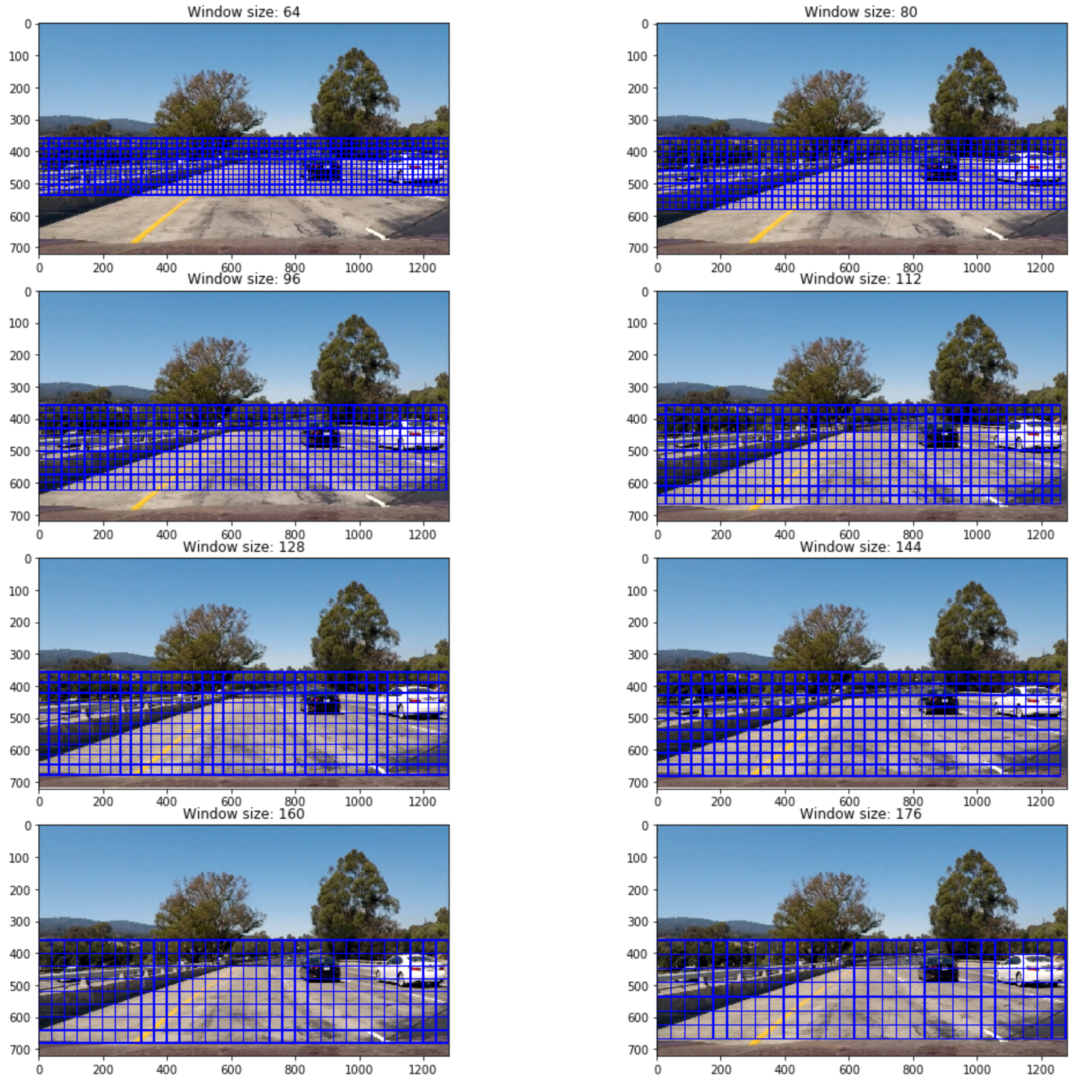
```
Feature vector length: 8556
3.99 Seconds to train SVC...
Test Accuracy of SVC =  0.9901
My SVC predicts:  [ 1.  1.  1.  1.  1.  0.  0.  1.  0.  1.]
For these 10 labels:  [ 1.  1.  1.  1.  1.  0.  0.  1.  0.  1.]
0.00155 Seconds to predict 10 labels with SVC
```

Sliding Window Search

1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

`slide_window()` in `lesson_functions.py` (line 100 through 139) decides total windows to be search for a given size. Here I implement multi-window search based on the heuristic that farther objects are smaller in the image. As a result, I only search a portion of the top of the bottom half of the picture for smaller window. The 8 images below give the idea of what portion is being searched for given window size:

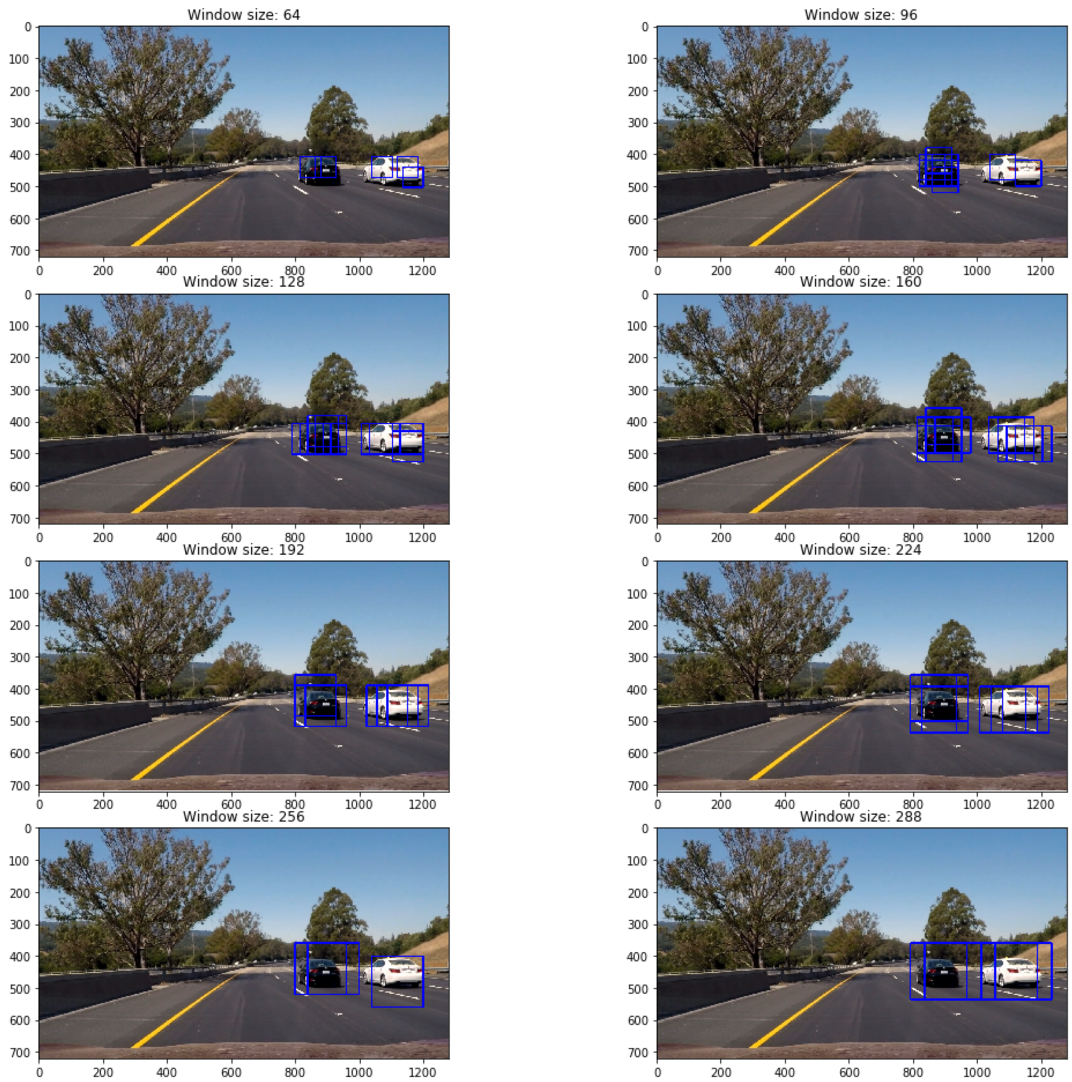
In [21]:
```
test_img = mpimg.imread('./test_images/test1.jpg')

def search_endpoint(window_size, img_height):
    return int(img_height / 2), int(img_height / 2 + img_height / 2 * window

row,col = 4,2
f, ax = plt.subplots(row, col, figsize=(15,13))
f.tight_layout()

multi_windows = []
for count,size in enumerate(range(64,192,16)):
    start,stop = search_endpoint(size, test_img.shape[0])
    #trim car hood
    if stop > test_img.shape[0] - 35:
        stop = test_img.shape[0] - 35
    window = slide_window(test_img,y_start_stop=[start,stop],xy_window=(size
    multi_windows.append(window)
    ax.flatten()[count].imshow(draw_boxes(test_img,window,thick=3))
    ax.flatten()[count].set_title('Window size: ' + str(size))
```



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

Here I demonstrate individual search result for each window size. The false positive result was high at first, so I tried different parameters for my SVM to improve the accuracy. The use of YCbCr color space seems to be a good choice since I get less false positive results on test images. For final video result, the different window-size results should be combined to form the final judgement.

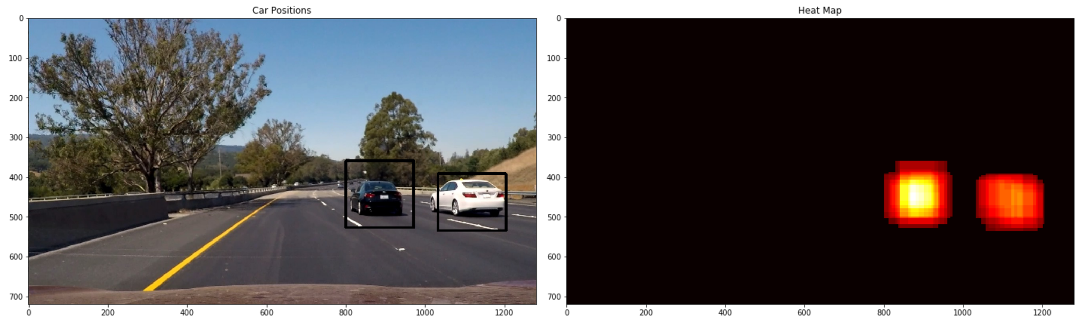
In [69]:
```
test_img = mpimg.imread('./test_images/test6.jpg')
image = test_img.astype(np.float32)/255

row,col = 4,2
f, ax = plt.subplots(row, col, figsize=(15,13))
f.tight_layout()

hits = []
# draw search result for each window size
for count,windows in enumerate(multi_windows):
    hot_windows = search_windows(image, windows, svc, X_scaler, color_space=
                        spatial_size=(spatial,spatial), hist_bins=histbin,
                        orient=orient, pix_per_cell=pix_per_cell,
                        cell_per_block=cell_per_block,
                        hog_channel=hog_channel, spatial_feat=True,
                        hist_feat=True, hog_feat=True)
    draw_image = np.copy(test_img)
    window_img = draw_boxes(draw_image, hot_windows, color=(0, 0, 255), thic
    ax.flatten()[count].imshow(window_img)
    ax.flatten()[count].set_title('Window size: ' + str(32 * (2 + count)))

    hits.extend(hot_windows)
```



Here I demonstrate the use of heapmap mechanism to filter out false positive results on test image and get the final judgement on where the cars are.

In [70]:
```
heat = np.zeros_like(test_img[:,:,0]).astype(np.float)

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

# Apply threshold to help remove false positives
heat = apply_threshold(heat,4)

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

# Find final boxes from heatmap using label function
labels = label(heatmap)
print('{} cars found.'.format(labels[1]) )
draw_img = draw_labeled_bboxes(np.copy(image), labels)

fig = plt.figure(figsize=(20,10))
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()
```

```
2 cars found.
```



Video Implementation

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Here's a link to my video result (./output_images/project_video.mp4)

In [84]:
```
# Import everything needed to edit/save/watch video clips
from moviepy.editor import VideoFileClip
from IPython.display import HTML
```

In [72]:
```
def vehicle_detection_pipeline(img):
    search_img = img.astype(np.float32)/255

    hits = []
    for windows in multi_windows:
        hot_windows = search_windows(search_img, windows, svc, X_scaler, co
                        spatial_size=(spatial,spatial), hist_bins=histbin,
                        orient=orient, pix_per_cell=pix_per_cell,
                        cell_per_block=cell_per_block,
                        hog_channel=hog_channel, spatial_feat=True,
                        hist_feat=True, hog_feat=True)
        hits.extend(hot_windows)

    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    # Add heat to each box in box list
    heat = add_heat(heat,hits)

    # 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 = draw_labeled_bboxes(np.copy(img), labels)
    return draw_img
```

In [73]:
```
output = 'output_images/project_video.mp4'
clip1 = VideoFileClip("project_video.mp4")
final_clip = clip1.fl_image(vehicle_detection_pipeline) #NOTE: this functio
%time final_clip.write_videofile(output, audio=False)
```

```
[MoviePy] >>>> Building video output_images/project_video.mp4
[MoviePy] Writing video output_images/project_video.mp4

100%|██████████| 1260/1261 [2:23:08<00:06,  6.74s/it]

[MoviePy] Done.
[MoviePy] >>>> Video ready: output_images/project_video.mp4

CPU times: user 2h 23min 4s, sys: 1.55 s, total: 2h 23min 6s
Wall time: 2h 23min 9s
```

2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Discussion

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

1. The project video result showed that two car on the right hand side are tracked constantly, but the pipeline failed to detect several frames for the white car. As a result, I export the failed images and adjust the parameters like different window sizes for these images. The video with revised algorithm in `output_images/project_video.mp4` shows better detection results.
2. The sliding window search above showed good results, but it took too long for a singe frame. As a result, I adapt Hog Sub-sampling Window Search from the lesson and make it a multi-scaled search. Here's the [link to my video result (./output_images/project_video_sub_sample.mp4)](./output_images/project_video_sub_sample.mp4). The time to make this video is improved but there are some failing frames to be inspected. I think I can adjust the search density like I did above to improve the results.

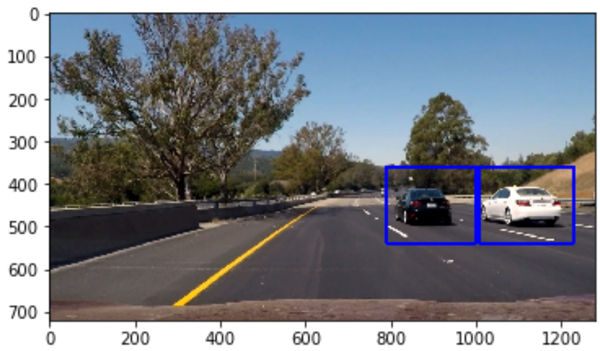
In [85]:
```
# Define a single function that can extract features using hog sub-sampling
def find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cel

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

    img_tosearch = img[ystart:ystop,:,:]
    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]/sc

    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)-1
    nyblocks = (ch1.shape[0] // pix_per_cell)-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 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, featu
    hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, featu
    hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, featu

    window_list = []
    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
            hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks
            hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks
            hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))

            xleft = xpos*pix_per_cell
            ytop = ypos*pix_per_cell

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

            # Get color features
            spatial_features = bin_spatial(subimg, size=spatial_size)
            hist_features = color_hist(subimg, nbins=hist_bins)

            # Scale features and make a prediction
            test_features = X_scaler.transform(np.array(np.concatenate([spa
            #test_features = X_scaler.transform(np.hstack((shape_feat, hist_
            test_prediction = svc.predict(test_features)

            if test_prediction == 1:
                xbox_left = np.int(xleft*scale)
                ytop_draw = np.int(ytop*scale)
                win_draw = np.int(window*scale)
                #cv2.rectangle(draw_img,(xbox_left, ytop_draw+ystart),(xbox_
                window_list.append(((xbox_left, ytop_draw+ystart), (xbox_le

    return window_list
    #return draw_img
```

In [86]:
```
def vehicle_detection_pipeline2(img):
    hits = []
    # Different scale value to form multi-window search
    for i in np.arange(1.0,5.0,0.5):
        window = find_cars(img, 360, 656, i, svc, X_scaler, orient, pix_per_
        hits.extend(window)

    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    # Add heat to each box in box list
    heat = add_heat(heat,hits)

    # Apply threshold to help remove false positives
    heat = apply_threshold(heat,2)

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

    # Find final boxes from heatmap using label function
    labels = label(heatmap)
    draw_img = draw_labeled_bboxes(np.copy(img), labels)

    return draw_img
```

In [81]:
```
test_img = mpimg.imread('./test_images/test6.jpg')
t=time.time()
t2=time.time()
draw_img = vehicle_detection_pipeline(test_img)
print(round(t2-t,5))
plt.imshow(draw_img)
```

```
4e-05
```

Out[81]: `<matplotlib.image.AxesImage at 0x7fea8fae7fd0>`



In [ ]:
```
output = 'output_images/project_video_sub_sample.mp4'
clip1 = VideoFileClip("project_video.mp4")
final_clip = clip1.fl_image(vehicle_detection_pipeline2) #NOTE: this functi
%time final_clip.write_videofile(output, audio=False)
```