Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

In [127]:
```
car_features = extract_features(cars, color_space = 'HSV', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel='ALL',
                        spatial_feat=True, hist_feat=True, hog_feat=True)
noncar_features = extract_features(noncars,color_space = 'HSV', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel='ALL',
                        spatial_feat=True, hist_feat=True, hog_feat=True)
scaled_X = normscale([car_features,noncar_features])
```

Answer: The **extract_features()** function extracts the features required to by the classifier to classify the training data. The HOG features are extracted when the **get_hog_features()** function in extract_features() is called. The **get_hog_features()** function in turns calls the **hog()** function which has been imported from the **skimage**. To extract HOG features the parameters selected were determined by training a linear SVM classifier and selecting parameters which point to increasing accuracy. The parameter tuning for HOG is not done in a manner to optimize globally. The tuned parameters could be local or global optimal.

Color space selected for HOG is HSV based on classifier accuracy on test data.

```
[('RGB', 0.97972972972972971), ('HSV', 0.98836336336336339), ('HLS', 0.98967717717717718), ('YCrCb', 0.98798798798798
804)]
```

Orientation for HOG is selected as 9 based on classifier accuracy. The list below has tuple of orient # and accuracy.

```
[(1, 0.94125375375375375), (3, 0.97766516516516522), (7, 0.98855105105105101), (9, 0.99080330330330335), (11, 0.99136
636636636633)]
```

Pixels per cell and cells per block for HOG, 8 and 2 respectively, are selected again based on classifier accuracy. 4 options were investigated for pixels per cell and 2 for cells per block.

```
[(4, 0.99099099099099097), (8, 0.99024024024024027), (12, 0.98667417417417413), (16, 0.98592342342342343)]
```

```
[(1, 0.98648648648648651), (2, 0.98948948948948945)]
```

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

In [127]:
```
car_features = extract_features(cars, color_space = 'HSV', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel='ALL',
                        spatial_feat=True, hist_feat=True, hog_feat=True)
noncar_features = extract_features(noncars,color_space = 'HSV', spatial_size=(32, 32),
                        hist_bins=32, orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel='ALL',
                        spatial_feat=True, hist_feat=True, hog_feat=True)
scaled_X = normscale([car_features,noncar_features])
```

In [128]:
```
y = np.hstack((np.ones(len(car_features)), np.zeros(len(noncar_features))))
```

In [129]:
```
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(
    scaled_X[0], y, test_size=0.3, random_state=rand_state)
```

In [130]:
```
svc = LinearSVC()
svc.fit(X_train,y_train)
```

Out[130]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='12', random_state=None, tol=0.0001,
     verbose=0)
```

In [131]:
```
#print('Test Accuracy of SVC = ', svc.score(X_test,y_test))
print('Test Accuracy of SVC = ', svc.score(X_test,y_test))
```

```
Test Accuracy of SVC =  0.987237237237
```

The Classifier is trained with features extracted from images of cars and non car objects. The training pictures are 64 x 64 pixels. After the features are extracted from the pictures, the data is normalized and centered around zero mean. Label data is created as a binary label of 0 for non car and 1 for car. The data is split into training and test data. Then a linear SVM is trained and checked for accuracy using the test data. The feature data consists of spatially binned image, histogram data for the HSV color space and HOG features for all 3 channels of HSV.

8460 features total; 3072 spatially binned feature; 96 histogram features; 5292 HOG features

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?

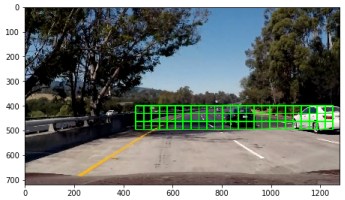
The find_cars() function implements the sliding window search in the pipeline used for the video. In the find_cars() function, HOG is calculated for the entire area of the image where a car can be found. This is done to avoid calculating HOG multiple times for pixel cells shared by windows.

This snippet of code in the find_cars() function is performing the slide window operation for all intensive purposes.

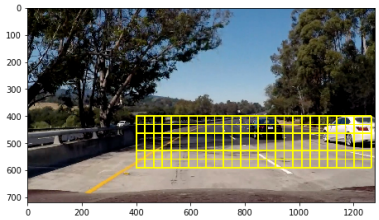
```
for xb in range(nxsteps):
    for yb in range(nysteps):
        ypos = yb*cells_per_step
        xpos = xb*cells_per_step
        # Extract HOG for this patch
        hog_feat1 = hog1[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()
        hog_feat2 = hog2[ypos:ypos+nblocks_per_window,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

        xleft = xpos*pix_per_cell
        ytop = ypos*pix_per_cell
```

To decide for the scales to be used, starting with 64 x 64 pixel window, windows were superimposed on the test images to visualize the window requirements. Cars will not drive in the sky, hence no window were placed in the upper half of the image.

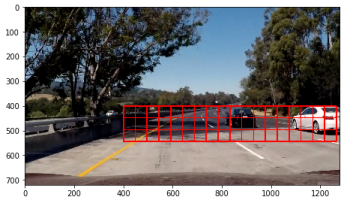
Cars far ahead will be smaller in the image and hence require smaller window size for detection. Cars closer will be larger in the image and require larger windows. See pictures. 50% overlap was selected. The scales in the function translate to [1,1.25, 1.75,2,2.5,3,4].



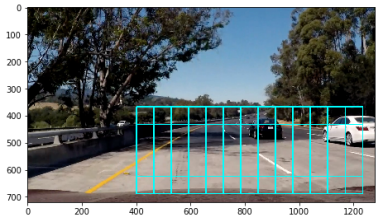
64 x 64



128 x 128



96 x 96

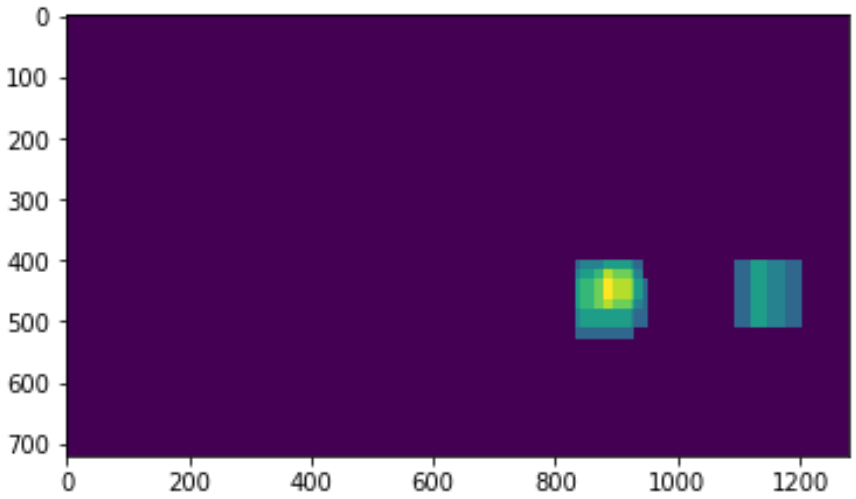


256 x 256

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

The classifier accuracy on test data is ~.99. This was achieved by finding the optimal feature extraction parameters.

The snippet below is tracking performing the function of tracking and smoothing out the detections reducing wobbliness. The heatmap is used with the label() function to identify were the cars are. label() function identifies agglomerations of heat signature where cars are present. The label() function outputs are used to determine the bounding boxes.



Heatmap

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.

```
    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    heat = add_heat(heat,windows)
    framesheat.append(heat)
    if len(framesheat)<20:
        pass
    else:
        for x in framesheat[:len(framesheat)-2]:
            heat = x +heat
        heat = heat/(len(framesheat)+1)
        del framesheat[0]

    heat = apply_threshold(heat,2)
    heatmap = np.clip(heat, 0, 255)
#    framesheat.append(heatmap)
#    if len(framesheat)<15:
#        pass
#    else:
#        for x in framesheat[:len(framesheat)-2]:
#            heatmap = x +heatmap
#        heatmap = heatmap/(len(framesheat)+1)
#        del framesheat[0]

    labels = label(heatmap)
    draw_img = draw_labeled_bboxes(np.copy(img), labels)
```

The vdtf() function has a snippet of code which averages out the heatmaps over a predetermined number of frames. Detection and tracking was done with 15 and 20 frame heatmap averages. The bounding boxes for tracking are determined using heatmap averages. This reduces the number of false positives and in combining the overlapping bounding bounding boxes

Discussion includes some consideration of problems/issues faced, what could be improved about their algorithm/pipeline, and what hypothetical cases would cause their pipeline to fail.

The problems/issues with the pipeline are/were:

- Cars not detected especially the far away ones
- Non cars detected as cars
- Cars detected on the other side of the road
- Detected cars are not tracked continuously
- The bounding boxes are not correctly/completely covering the cars
- This pipeline is likely to failure if implemented at night
- Cars moving very close to each other might be difficult to track correctly by the pipeline

Improving the pipeline:

- The training data can be expanded to include pictures of car at more angles
- The pipeline can be improved by including a function/class which tracks the detected cars by storing previous detected location and extrapolating

to future location based on relative speed and location w.r.t. to the camera

- Pipeline can identify the type of car, which will help in estimating a bounding box