Project : Vehicle Detection Resubmit #2

Improvement Lists

```
The architecture of Vehicle Detection involved two path:
    1. Read the dataset and train our neural Network (PATH1)
    2. Read image from video stream and try detect where the other vehicles. (PATH2)

The main problem of previous design was a lot of false positive in detect other
vehicles.
We did not train the neural network right.  List of improvement list can be summarized:
1. Increase the data set.
2. The extraction feature path (PATH1) was reading image in PNG format to train our
neural network
   and the detection path (PATH2) read JPEG images from video stream and the code in
the two path
   some time implement the algorithm that expect different type (PNG/JPEG) of format.
   We made the modification to make all internal path use 1 format (JPEG)
3. All the reading of image in either format (PNG/JPEG) will be handle properly in JPEG
format
   by read_image (file)
4. Previous handling of False Positive situation is very complex.  Redesign to
implement
   classic local heatmap strategy with a special way to handle threshold value.
5. Color Space was changed to YCrCb
6. Change in spatial_size   from  (16,16) to (32,32)
7. Change in hist_bins from 16 to 32
8. Change in the use of window/ scale to:[(400, 600, 1.0),(500, 600, 1.0),(400, 500,
0.8) ]
```

Addressing Reviewer SIX points:

1. Improve Document with "short code snippets"

The improvement was made in Vehicle_Detection_Report_2nd.ipynb or Vehicle_Detection_Report_2nd.pdf

2. Normalized HOG features extracted from

```
1. The extract features from data set is normalezed using StandardScaler to remo
ve the mean and scales the data to unit variance.
2. Then the data set is radomly split, shuffle in create_test_training_data()
3. Before the neural network is trained in train_model()
```

3. Suggest using smaller scales, like 1.0, 1.5 and maybe 2.0

In order to improve, the new window size and scale is used :

```
scale = [(400, 600, 1.0), (500, 600, 1.0), (400, 500, 0.8)]
```

4. Improving the reliability of classifier:

1. Color Space was changed to YCrCb
2. Change in spatial_size from (16,16) to (32,32)
3. Change in hist_bins from 16 to 32
4. Implement simpler classic heatmap algorithm

5. The sliding-window search plus classifier is unreliable.

```
. Reorganized code flow to handle everything in JPEG format and
. implement item #4 above improved the performance of the classifier.
```

6. Handling of false positive using "deque"

```
. We try to implement a complex way to handle false positive using deque.
. We failed to improve the performance and decide to back out and use the standa
rd classic implementation
```

In [14]:
```
# Combined_Vehicle_Detection_Lane_Detection
```

Project : Vehicle Detection

Goals :

- Design a software pipeline to identify vehicles in a video from a front-facing camera on a car.

1. Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images
    - Optionally, you can also apply a color transform and
    - append binned color features, as well as
    - histograms of color, to your HOG feature vectorand
2. Train a classifier Linear SVM classifier
3. Normalize your features and randomize a selection for training and testing. for those first two steps
4. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
5. Run your pipeline on a video stream
    - Create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
    - Estimate a bounding box for vehicles detected.

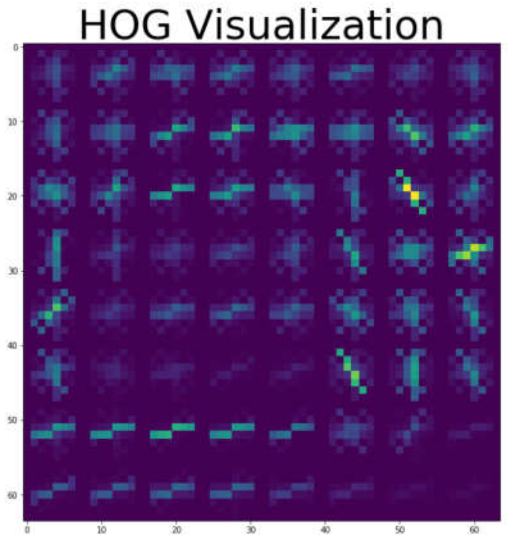
1. HOG features

After all test images was read using build_test_data(), we randomly choose an image.

We display this image using hog_parameters (orient = 9 ; pix_per_cell = 8 ; cell_per_block = 2)

In [1]:
```
from myLib.vehicle_detector import *
from myLib.testlib_vehicle import *
%matplotlib inline
# Define HOG parameters
orient = 9 ;  pix_per_cell = 8 ; cell_per_block = 2
to_get_hog_features (orient, pix_per_cell, cell_per_block )
```

```
/root/miniconda3/envs/carnd-term1/lib/python3.5/site-packages/sklearn/cross_validatio
n.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of t
he model_selection module into which all the refactored classes and functions are mov
ed. Also note that the interface of the new CV iterators are different from that of t
his module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/root/miniconda3/envs/carnd-term1/lib/python3.5/site-packages/skimage/feature/_hog.p
y:119: skimage_deprecation: Default value of `block_norm`==`L1` is deprecated and wil
l be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```



2. Feature Extraction for Model

Like any printer, the larger density it can handle the better quality it can print.
In order to distinguish objects in details easier, we need higher resolution.

All image features were used in helping neural network (LinearSVC) to detect car from image.

These features are (color_space, spatial_feat, hist_feat, hog_feat, orient, pix_per_cell, cell_per_block)

Using the large combinatorial feature values above to train the LinearSVC and then test with images to

see which combinaion detect car correctly the most we arrive to final feature parameters.

Final feature paramenters are :

```
color_space    = 'YCrCb'

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
```

Increasing the orientation enhanced the accuarcy of the finally trained classifier, but increased the time required for computation.

The `color_space` was decided by training a classifier on different color spaces for spatial features, and YCrCb performed better than `HSV`, `HLS`, and `RGB` .

3. Train-Test Split

The `train_test_split()` from `sklearn.model_selection` was used to randomized the data and make a 80-20% train-test split.

```
train_test_split(scaled_X, y, test_size=0.2, random_state=rand_state)
```

The split was made so as to keep the ratio of vehicles and non-vehicles similar.
The create_test_training_data () called `train_test_split()` and all also normalized features with:

```
. X = np.vstack((car_features, notcar_features)).astype(np.float64)
. X_scaler = StandardScaler().fit(X)
```

4. Model Training

1. Model Training code can be find in **Vehicle_Dectection_NN_Training.**
2. First, training data is built using build_test_data() given it a directory for training image (TestImages)
3. Then, `LinearSVC` model of `sklearn` with default setting of `square-hinged` loss function. The `l2` normalization is selected because it is fast and easy to use.
4. The trained model had accuracy of `98.7%` on test dataset.
5. The trained model along with the parameters used for training were written to a `pickle` file to be further used by vehicle detection pipeline.
6. To save time, automation was made to build quick model to test:
   A. automate_pk_32.ipynb
   B. automate_pk_16.ipynb

5. Vehicle Detection Pipeline

1. Sliding Window Search

For each frame of video stream, the `find_other_cars` in `./myLib/featureExtraction.py`,
extract image features discussed above from image frame.

With (ystart, ystop, scale) we make education guess that the front view of driver is the region which we more likely to see other vehicles. We then divide this region into blocks and extract their features.
With each block feature we use svc.predict() to see this block is a car or not. If it is predicted a car, we draw a box around it and add this box to a other_vehicle list. And in addition, we add the region which covers by the box to the heat map. When there are more boxes/vehicles detect in the same region, this region got hot on the heat map every quick.

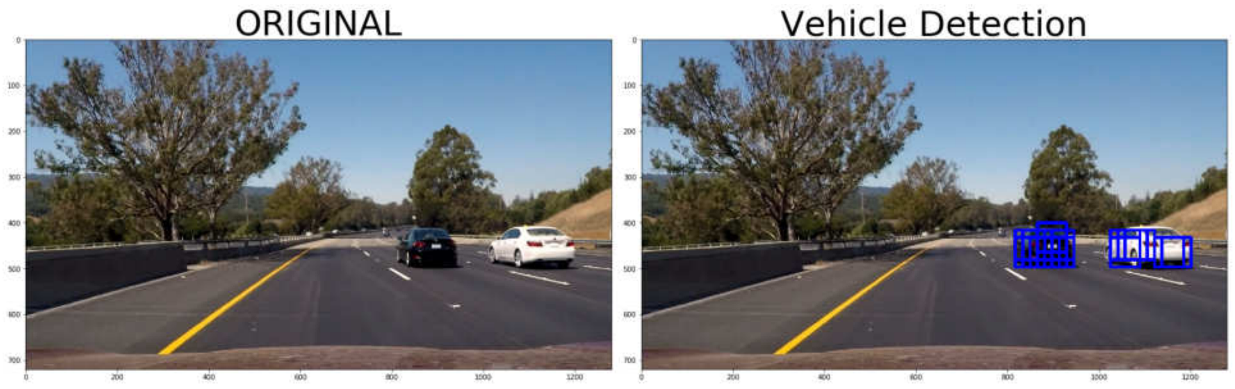
Nothing is perfect, therefore there are times that we predict wrongly (false positive) and the the heat map will get hot at regions that are not correct.
To reduce this effect, we clear the hot area that is under threshold once in a while (every 5 frame).

To enhance the ability to detect other car better, we deploy multi-scale window search. And when we are moving from frame to frame, we accumulate the information for 20 consecutive frames then reset it.

Code with multi-scale window search and heatmap to reduce false positives have been implemented in the class `VehicleDetector` in `./myLib/vehicle_detector.py` and is discussed in upcoming sections.

In [3]:
```
model_file     = pk_dict_32_jpeg['ycrcb']
image_file = 'test_images/test6.jpg'; ystart = 400; ystop  = 656; scale  = 1
to_find_cars (image_file, model_file, ystart, ystop, scale)
```

```
/root/miniconda3/envs/carnd-term1/lib/python3.5/site-packages/skimage/feature/_hog.p
y:119: skimage_deprecation: Default value of `block_norm`==`L1` is deprecated and wil
l be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```



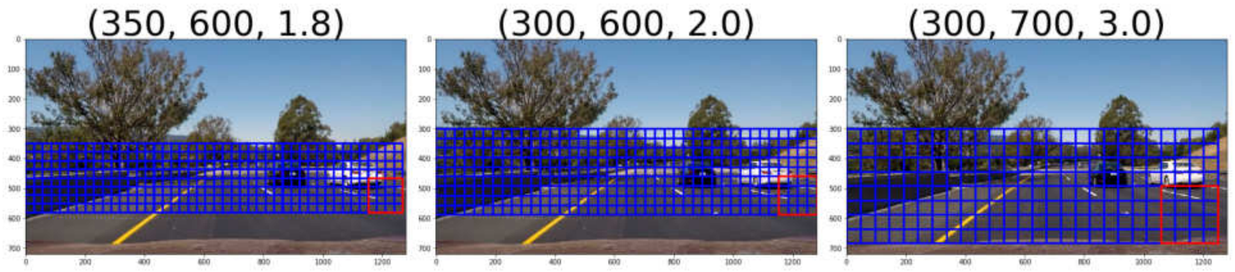
2. Multi-Scale Search

The scale for the multi-window search and overlap inorder to look for other cars not on the sky or tree, and increase reliabilty in detection

The multi-scale window approach prevents calculation of feature vectors for the complete image and thus helps in speeding up the process. The following is decided:

1. Scale 1: (ystart, ystop, scale) = (350, 600, 1.8) .
2. Scale 2:(ystart, ystop, scale) = (300, 600, 2.0) .
3. Scale 3:(ystart, ystop, scale) = (300, 700, 3.0) .

In [2]:
```
imageFile = 'test_images/test6.jpg'
window_scale_list3 = [(350, 600, 1.8), (300, 600, 2.0), (300, 700, 3.0)]
to_multiple_scale_search (imageFile, window_scale_list3 )
```



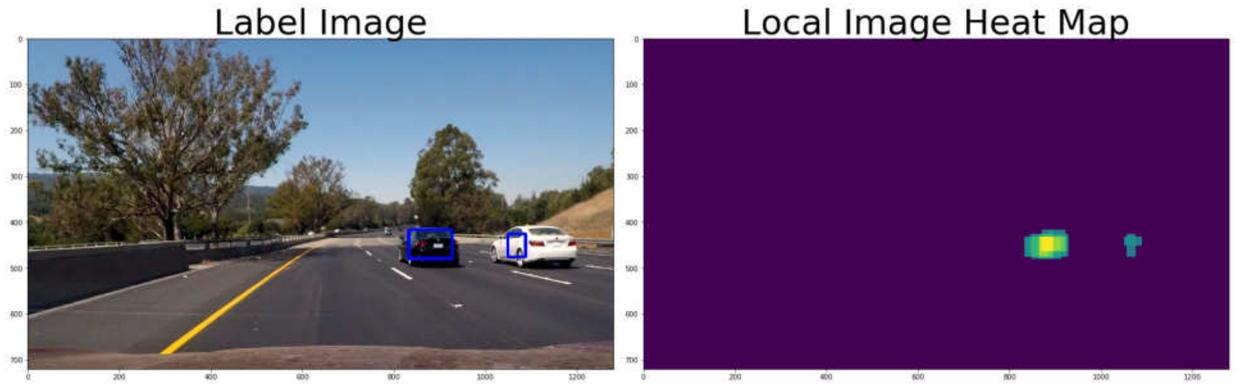
3. Avoiding False Positives and Label Detection

The false positives were avoided by using heat mapping technique.
In this technique, marker was made in the scan area where svc linear trained model think there is a car.
If this area is marked many times, it becomes a hotpot. `scipy.ndimage.measurements.label()` is used to identify individual blobs in the scan image and assumed there is a car in that area.

"Sliding window Search" above include explaination how to avoid false positive.

```
In [3]: modelFile = pk_dict_32_jpeg['ycrcb']
        imageFile = 'test_images/test6.jpg'
        detector = VehicleDetector (modelFile) # ('model-params-hsv2.pk')
        scale = detector.ystart_ystop_scale = [   (400, 600, 1.0), (500, 600, 1.0), (400,
        500, 0.8) ]
        d12 = auto_test_detector1 (imageFile, scale, modelFile,  detector)
```

```
/root/miniconda3/envs/carnd-term1/lib/python3.5/site-packages/skimage/feature/_hog.p
y:119: skimage_deprecation: Default value of `block_norm`==`L1` is deprecated and wil
l be changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```



4. Optimization Implementation in Searching Car

Optimization was implementated in `find_other_cars`of `VehicleDetector` in `./myLib/vehicle_detector.py`.

The following mechanisms are implemented :

```
1. The front of driver view is used instead of scanning every where in the image.
. This help to reduce 50% of searching time.
2. Apply threshold to heat map for improving "False Positives"
```

Vehicle Detection Video Implementation

To improve paralellism and save time, to script was created to

```
1. test_video-YCrCb_basic.ipynb
2. test_video-YCrCb_advanced.ipynb
```

5. Discussion:

Detection Other Vehicles pipeline is a very difficult task.
At the beginning, setting up this pipeline is long and lonely and hard.
The need to have the data for training ready, normalize data, shuffle them, chosing among many different neural network models, different paramenters to experience for a particular model.
And even when everything is ready, chosing scanning paramenters (ystart/ystop/scale) also affected the performance of whether car is detected at all.
Then it finally come to find "time" to tune it to perform better than the last time (computing performance). We run into lack of computing power. Things run to slow. We run into automation issues where we wish we could pump a combinatorial set of parameters, run it and find out which one will give us best result. The key to performance is to predict better, this means we need to train the model better, this mean we need to have better data set. This finally means we need to more time.