Project 5: “Vehicle Detection and Tracking”

Let's start by recalling the main goals of this project: :

- 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.

I. Histogram of Oriented Gradients (HOG)

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

For this step I used the function `get_hog_features`.

```
# Define a function to return HOG features and visualization
def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                        hog_calc='skimage', vis=False, feature_vec=True):
    if hog_calc == 'skimage':
        # Call with two outputs if vis==True
        if vis == True:
            features, hog_image = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                                      cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=True,
                                      visualise=vis, feature_vector=feature_vec)
            return features, hog_image
        # Otherwise call with one output
        else:
            features = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                           cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=True,
                           visualise=vis, feature_vector=feature_vec)
            return features
```

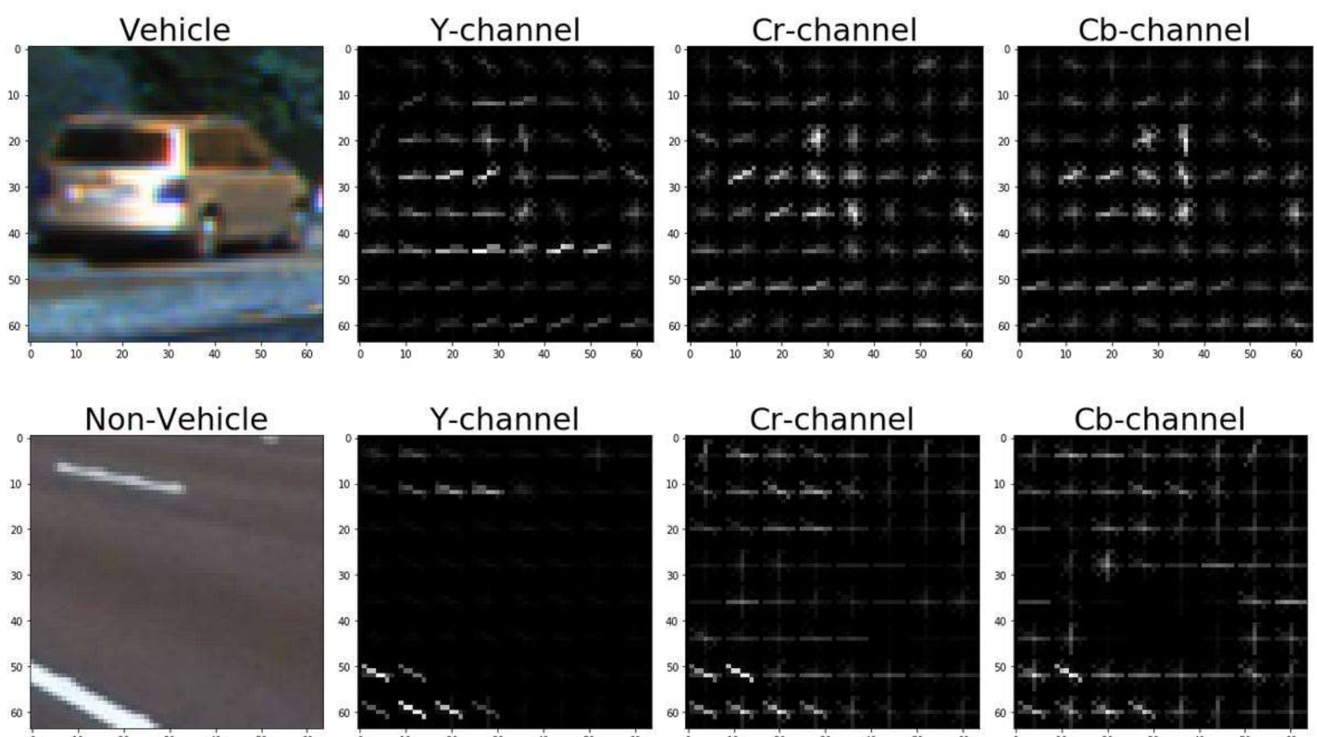
This function uses the skimage function hog to get the Histogram of Oriented Gradients features and the belonging image. The parameters for tuning are the number of directions bins (orients), the number of pixels per cell over which each gradient histogram is computed. Another parameter is the number of cells per block, which specifies the local area over which the histogram counts in a given cell will be normalized.

I also implemented the Opencv function hogDescriptor() which gave me a benefit in processing time, but I didn't found a suitable classifier and parameter set for this to work as well as my previous determined parameter set with the skimage hog function. So I used the skimage function in face of the reduced processing speed.

I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

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.

I used the following parameters: `YCrCb` color space and HOG parameters of `orientations=18`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`. Here is one example for a vehicle and non-vehicle image.



I.2. Spatial and color parameters.

Additionally to the HOG features I add the following functions to make use also from color and spatial information in the images:

```
# Define a function to compute binned color features
def bin_spatial(img, size=(32, 32)):
    # Use cv2.resize().ravel() to create the feature vector
    color1 = cv2.resize(img[:,:,0], size).ravel()
    color2 = cv2.resize(img[:,:,1], size).ravel()
    color3 = cv2.resize(img[:,:,2], size).ravel()
    # Return the feature vector
    return np.hstack((color1, color2, color3))


# Define a function to 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_hist[0]))
    # Return the individual histograms, bin_centers and feature vector
    return hist_features
```

The advantage is that using e.g. the color information we are independent of the structure. Therefore, objects which appear in different aspects and orientations (as trained with the images dataset) will still be matched. Also raw pixel information is quite useful as different shape could help to identify vehicles.

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

This step must be seen in combination with the training of the classifier (which is describe in the following section). Here I used the function `extract_features` which calls the above mentioned functions and appends the color, spatial and HOG features from each image to a feature vector.

Testing several combinations of parameters and color spaces I ended up with the following parameter set, which gave me the best results with the chosen classifier in vehicle detection. I used different criterions like: What is the accuracy of the test data set of the chosen classifier? Is the vehicle detected in an image? How many windows are detecting the vehicle? Is it fully covert? How many positive false detections are there?

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 18 # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
spatial_size = (4,4)  # Spatial binning dimensions
hist_bins = 256 # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
hog_calc = "skimage" #  "skimage" or "opencv"
```

```python
# Define a function to extract features from a list of images
# Have this function call bin_spatial() and color_hist()
def extract_features(imgs, 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, hog_calc='skimage'):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        file_features = []
        # Read in each one by one
#        image = mpimg.imread(file)# for jpg,    png file 0-1, jpg file 0-255
        image = cv2.imread(file)# for png,    png file 0-255
        # apply color conversion if other than 'RGB'
        if color_space != 'RGB':
            if color_space == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif color_space == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif color_space == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif color_space == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif color_space == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else: feature_image = np.copy(image)

        # Apply bin_spatial()
        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
        # Apply color_hist()
        if hist_feat == True:
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        # Call get_hog_features() with vis=False, feature_vec=True
        if hog_feat == True:
            if hog_channel == 'ALL':
                hog_features = []
                for channel in range(feature_image.shape[2]):
                    hog_features.append(get_hog_features(feature_image[:,:,channel],
                                        orient, pix_per_cell, cell_per_block,
                                        hog_calc, vis=False, feature_vec=True))
                hog_features = np.ravel(hog_features)
            else:
                hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                            pix_per_cell, cell_per_block, hog_calc, vis=False, feature_vec=True)
            # Append the new feature vector to the features list
            file_features.append(hog_features)
        features.append(np.concatenate(file_features))
    # Return list of feature vectors
    return features
```

II. Classifier

II.1 Data preparation

At a first step I did some augmentation of the GTI vehicle image database and the KITTI vision benchmark suite so that I can add some more images and perhaps get ride of very similar images by selecting only a slice of the whole augmented data set. You can find the code in the jupyter notebook:

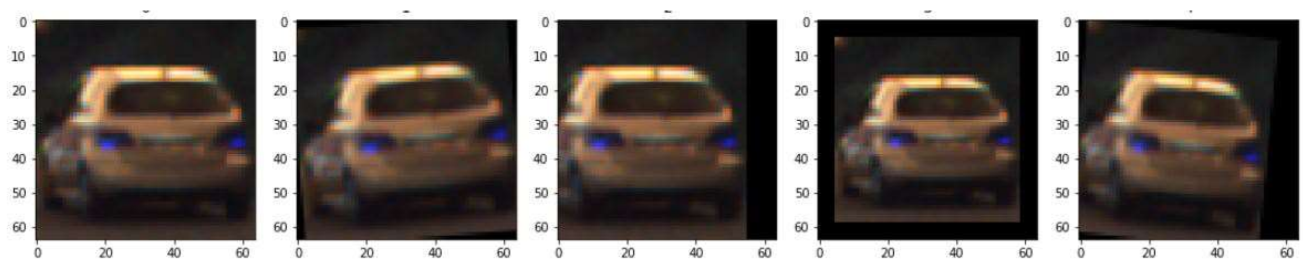
P5_vehicle_detection_helpfunctions.ipynb

I shifted, rotated, zoomed and sheard each image using the following functions:

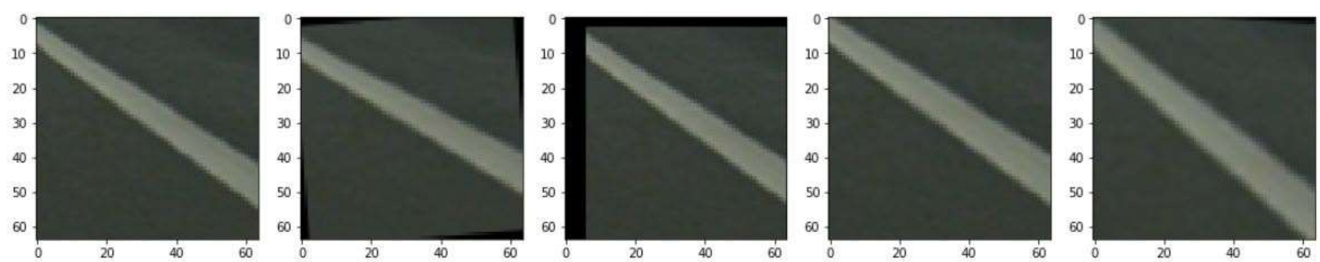
```python
#data manipulation and add augmented data
def img_rot(image):
    rows,cols,ch = image.shape
    center = (rows/2, cols/2)
    fact = random.randint(-1,1)
    return cv2.warpAffine(image, cv2.getRotationMatrix2D(center, fact*5, 1), (rows, cols))

def img_shift(image):
    rows,cols,ch = image.shape
    fact1 = random.randint(-10,10)
    fact2 = random.randint(-10,10)
    return ndimage.shift(image,[fact1,fact2,0])
```

```python
def img_zoom(image):
    zoom_f = float(0.85 + (random.randint(0,3)/10))
    return clipped_zoom(image, zoom_f)

def img_shear(image):
    rows,cols,ch = image.shape
    shear_fact = random.randint(5,15)
    # Shear
    scr = np.float32([[5,5],[20,5],[5,20]])

    p_dst1 = 5+shear_fact*np.random.uniform()-shear_fact/2
    p_dst2 = 20+shear_fact*np.random.uniform()-shear_fact/2

    dst = np.float32([[p_dst1,5],[p_dst2,p_dst1],[5,p_dst2]])

    return cv2.warpAffine(image,cv2.getAffineTransform(scr,dst),(cols,rows))
```

Here is an example of a vehicle and it's augmented variations:



And also the same for a non-vehicle image:



II.2. Selection of a suitable classifier with grid search

Then I started to identify a suitable classifier for the vehicle detection. To start with I read in all the images of the car and non-car dataset and labelling them appropriately. After extracting the features of each image I normalized the vectors to have zero mean and unit variance using the Scikit-learn functions `StandardScaler`. (you can find this in the cell 'Training different classifier).

Training different classifier

```
train_clf = False
if train_clf == True:
# Read in cars and notcars

    data_set = '../data_base_png/*.png'
    images = glob.glob(data_set)
    cars = []
    notcars = []
    for image in images:
        if 'image' in image or 'extra' in image:
            notcars.append(image)
        else:
            cars.append(image)

    print('notcars: ', len(notcars))
    print('cars: ', len(cars))

    color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
    orient = 12 # 18  # HOG orientations
    pix_per_cell = 8 # 5 # HOG pixels per cell
    cell_per_block = 2 # HOG cells per block
    hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
    spatial_size = (32,32) #(4, 4) # Spatial binning dimensions
    hist_bins = 64 #256    # Number of histogram bins
    spatial_feat = True # Spatial features on or off
    hist_feat = True # Histogram features on or off
    hog_feat = True # HOG features on or off

    car_features = extract_features(cars, color_space=color_space,
                            spatial_size=spatial_size, hist_bins=hist_bins,
                            orient=orient, pix_per_cell=pix_per_cell,
                            cell_per_block=cell_per_block,
                            hog_channel=hog_channel, spatial_feat=spatial_feat,
                            hist_feat=hist_feat, hog_feat=hog_feat)
    notcar_features = extract_features(notcars, color_space=color_space,
                            spatial_size=spatial_size, hist_bins=hist_bins,
                            orient=orient, pix_per_cell=pix_per_cell,
                            cell_per_block=cell_per_block,
                            hog_channel=hog_channel, spatial_feat=spatial_feat,
                            hist_feat=hist_feat, hog_feat=hog_feat)

    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)
```

After that the data set is randomly split into a training and test data set using the `train_test_split` function. I decided to test the following classifier and use grid search to optimize each classifier before judging which would be the best to identify the vehicles. To do so I used the parameter set shown in the picture above), which differs from the one mentioned in the last chapter.

I inspected the following classifiers:

- Logistic Regression Classifier
- Multi-layer Perception
- linearSVC
- SVM with different kernels

And two ensemble classifier:

- Adaboost with decision tree
- Bagging with decision tree

```
for m in range(2,3):

    if m ==0:
        classifier_ = "LogisticRegression"
        clf = LogisticRegression()
        parameters = {'C':[0.01,0.1,1.], 'max_iter': [1,10,100,]}
#        parameters = {'C':[0.01,0.1,1.,10.], 'max_iter': [10,100,1000,10000]}


    if m ==1:
        classifier_ = "MLPClassifier"
#### default: activation='relu', solver='adam', batch_size='auto',shuffle=True,
        clf = MLPClassifier(early_stopping=True, validation_fraction=0.1)
        parameters = {'hidden_layer_sizes':[(100, ),(150, )], 'max_iter': [100,200], \
                    'learning_rate_init':[0.001, 0.01] }
#        parameters = {'hidden_layer_sizes':[(50, ),(100, ),(150, )], 'max_iter': [50,100,200,300], \
#                    'learning_rate_init':[0.0005,  0.001, 0.01, 0.1] }

    if m ==2:
        classifier_ = "LinearSVC"
        clf = svm.LinearSVC()
        parameters = {'C':[0.01, 0.1,1.0], 'max_iter': [1,10,100]}
#        parameters = {'C':[0.01,0.1,1.,10.], 'max_iter': [10,100,1000,10000]}


    if m ==3:
        classifier_ = "SVC"
        clf = svm.SVC()
        parameters = {'kernel':['linear', 'rbf'], 'C':[0.01,0.1,1.,10.], 'gamma': [0.01, 0.1,10]}


#classifier_ = "AdaBoostClassifier"
#clf_gd = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),algorithm='SAMME',n_estimators=200)

#classifier_ = "BaggingClassifier"
#clf_gd = BaggingClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=200,n_jobs=-1)
```

As a result I got the following:

AdaBoost classifier with decision tree gave a very good accuracy but was fare to slow (here only with a smaller part of the image data set):

```
Using: 12 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 10320


Classifier:  AdaBoostClassifier
229.4 Seconds to train CLF...
Test Accuracy of CLF =   1.0
0.0156266689 Seconds to predict
```

Bagging classifier with decision tree was much faster but achieved the poorest result in accuracy:

```
Using: 12 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 10320


Classifier:  BaggingClassifier
32.49 Seconds to train CLF...
Test Accuracy of CLF =   0.95699
1.3361635208 Seconds to predict
```

Training the SVC classifier with different kernel and parameters, resulted in a best pick for a linear kernel with gamma of 0.1 and C also of 0.1. But as this classifier was extremely slow, I didn't put it to account.

The following three classifiers (Logistic Regression, Multi-layer Perception and linearSVC) were all very close together in accuracy (MLP slightly better) but linearSVC heading in speed.

```
notcars:  8968
cars:  8792
Using: 12 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 10320


Classifier:  LogisticRegression
403.66 Seconds to train CLF...
Test Accuracy of CLF =    0.99352
0.0 Seconds to predict with  LogisticRegression
best_param_dict:
{'max_iter': 10, 'C': 1.0}

grid_result
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score param_C  \
0       2.391671         0.072926         0.982334          0.983425    0.01
1      17.550694         0.067707         0.994088          1.000000    0.01
2      18.090986         0.078135         0.994088          1.000000    0.01
3       2.454378         0.068530         0.982334          0.983390     0.1
4      20.339885         0.072918         0.994158          1.000000     0.1
5      19.837987         0.067717         0.994158          1.000000     0.1
6       2.204117         0.062518         0.982334          0.983390       1
7      19.358982         0.078137         0.994299          1.000000       1
8      19.383508         0.067717         0.994299          1.000000       1

  param_max_iter                          params  rank_test_score  \
0              1    {'max_iter': 1, 'C': 0.01}                7
1             10   {'max_iter': 10, 'C': 0.01}                5
2            100  {'max_iter': 100, 'C': 0.01}                5
3              1     {'max_iter': 1, 'C': 0.1}                7
4             10    {'max_iter': 10, 'C': 0.1}                3
5            100   {'max_iter': 100, 'C': 0.1}                3
6              1     {'max_iter': 1, 'C': 1.0}                7
7             10    {'max_iter': 10, 'C': 1.0}                1
8            100   {'max_iter': 100, 'C': 1.0}                1
```

```
notcars:  8968
cars:  8792
Using: 12 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 10320


Classifier:  LinearSVC
87.06 Seconds to train CLF...
Test Accuracy of CLF =    0.9924
0.0 Seconds to predict with  LinearSVC
best_param_dict:
{'max_iter': 10, 'C': 0.01}

grid_result
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score param_C  \
0       2.177370         0.072907         0.993736          0.998944    0.01
1       2.615288         0.067685         0.993806          1.000000    0.01
2       3.871399         0.067710         0.993806          1.000000    0.01
3       2.058325         0.067709         0.993102          0.998592     0.1
4       2.603597         0.067689         0.993806          1.000000     0.1
5       3.792880         0.073327         0.993806          1.000000     0.1
6       2.008104         0.062482         0.991976          0.998839       1
7       2.589216         0.067709         0.993736          1.000000       1
8       3.663507         0.062477         0.993736          1.000000       1

  param_max_iter                          params  rank_test_score  \
0              1    {'max_iter': 1, 'C': 0.01}                5
1             10   {'max_iter': 10, 'C': 0.01}                1
2            100  {'max_iter': 100, 'C': 0.01}                1
3              1     {'max_iter': 1, 'C': 0.1}                8
4             10    {'max_iter': 10, 'C': 0.1}                1
5            100   {'max_iter': 100, 'C': 0.1}                1
6              1     {'max_iter': 1, 'C': 1.0}                9
7             10    {'max_iter': 10, 'C': 1.0}                5
8            100   {'max_iter': 100, 'C': 1.0}                5
```

```
Classifier:  MLPClassifier
506.59 Seconds to train CLF...
Test Accuracy of CLF =    0.99747
0.0 Seconds to predict with  MLPClassifier
best_param_dict:
{'max_iter': 200, 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.001}

grid_result
   mean_fit_time  mean_score_time  mean_test_score  mean_train_score  \
0      10.771820         0.161486         0.993454          0.998311
1      19.408947         0.140649         0.994581          0.999331
2      18.368792         0.146241         0.993525          0.999296
3      16.391929         0.156267         0.992469          0.998663
4      25.796876         0.189176         0.994088          0.999648
5      22.175285         0.172458         0.994088          0.999155
6      23.242100         0.178140         0.993525          0.998240
7      20.986182         0.177673         0.991765          0.998064

  param_hidden_layer_sizes param_learning_rate_init param_max_iter  \
0                   (100,)                    0.001            100
1                   (100,)                    0.001            200
2                   (100,)                     0.01            100
3                   (100,)                     0.01            200
4                   (150,)                    0.001            100
5                   (150,)                    0.001            200
6                   (150,)                     0.01            100
7                   (150,)                     0.01            200

                                              params  rank_test_score  \
0  {'max_iter': 100, 'hidden_layer_sizes': (100,)...                6
1  {'max_iter': 200, 'hidden_layer_sizes': (100,)...                1
2  {'max_iter': 100, 'hidden_layer_sizes': (100,)...                4
3  {'max_iter': 200, 'hidden_layer_sizes': (100,)...                7
4  {'max_iter': 100, 'hidden_layer_sizes': (150,)...                2
5  {'max_iter': 200, 'hidden_layer_sizes': (150,)...                2
6  {'max_iter': 100, 'hidden_layer_sizes': (150,)...                4
7  {'max_iter': 200, 'hidden_layer_sizes': (150,)...                8
```

II.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).

So I decided to trained the pre-selected three classifier (Logistic Regression, Multi-layer Perception and linearSVC) with different feature parameters to check which one would do best. You can find that in the next jupyter notebook cell “Training selected classifier with different feature parameter”:

```
notcars:  8968
cars:  8792
Using: 18 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 11544


Classifier:  LogisticRegression
27.58 Seconds to train ... LogisticRegression
Test Accuracy of  LogisticRegression  =   0.99465
0.0625100136 Seconds to score with  LogisticRegression
0.0 Seconds to predict with  LogisticRegression
      0     1
0  1807    14
1     5  1726

 color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL
 spatial_size:  hog_calc : skimage (8, 8) hist_bins:  256 data_set:  data_base_png/*.pn
g
 save file:  SVC_trained/LogisticRegression_20170420_150902.p


Classifier:  MLPClassifier
35.72 Seconds to train ... MLPClassifier
Test Accuracy of  MLPClassifier  =   0.99437
0.1588256359 Seconds to score with  MLPClassifier
0.0 Seconds to predict with  MLPClassifier
      0     1
0  1803    11
1     9  1729

 color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL
 spatial_size:  hog_calc : skimage (8, 8) hist_bins:  256 data_set:  data_base_png/*.pn
g
 save file:  SVC_trained/MLPClassifier_20170420_150938.p


Classifier:  LinearSVC
3.05 Seconds to train ... LinearSVC
Test Accuracy of  LinearSVC  =   0.99465
0.0781333447 Seconds to score with  LinearSVC
0.0 Seconds to predict with  LinearSVC
      0     1
0  1807    14
1     5  1726

 color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL
 spatial_size:  hog_calc : skimage (8, 8) hist_bins:  256 data_set:  data_base_png/*.pn
g
 save file:  SVC_trained/LinearSVC_20170420_150941.p
```

As a final result I've chosen the linearSVC with max_iter = 10 and C = 0.1 as a best pick as well in speed and in accuracy.

III. Sliding Window Search

III.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?

Now to detect the position of a vehicle in a video frame, it is necessary to select a subregion of the image, run the classifier on each subregion to see if it contains a vehicle or not. Therefore I implemented a sliding-window function (here called `mul_win_slide`) :

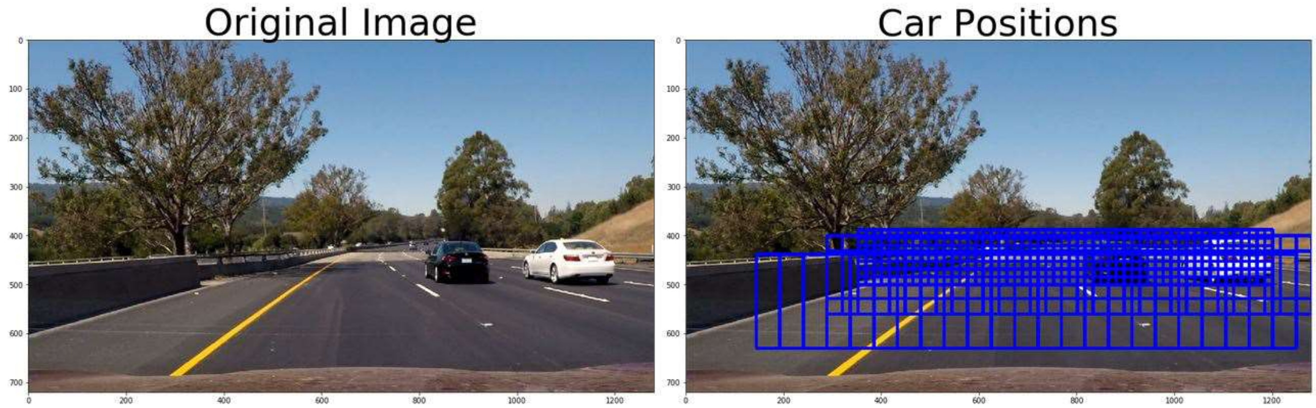
Sliding Window Implementation

```
def mul_win_slide(img,svc, X_scaler, orient, pix_per_cell, cell_per_block, spatial_size, hist_bins, cell_per_win, h

    xbox_left_all = []
    ytop_draw_all = []
    win_draw_all = []
    ystart_all = []
    bboxes_raw = []
    xbox_left_all = []
    ytop_draw_all = []
    win_draw_all = []
    ystart_all = []
    bboxes_raw = []

    cells_per_step = 2
    img_ =  np.copy(img)

    for scale in range(1,4):

        if scale == 4:
            ystop = 680
            steps = 0
            ystart = int((ystop-pix_per_cell*(cell_per_win+steps*cells_per_step)*scale))
        if scale == 3:
            ystop = 630
            steps = 0
            ystart = int((ystop-pix_per_cell*(cell_per_win+steps*cells_per_step)*scale))
        if scale == 2:
            ystop = 560
            steps = 1
            ystart = int((ystop-pix_per_cell*(cell_per_win+steps*cells_per_step)*scale))
        if scale == 1:
            ystop = 500
            steps = 3
            ystart = int((ystop-pix_per_cell*(cell_per_win+steps*cells_per_step)*scale))

        xbox_left,ytop_draw, win_draw, ystart_ar = find_cars(img, ystart, ystop, scale, svc, X_scaler, \
                                                    orient, pix_per_cell, cell_per_block, spatial_size, \
                                                    hist_bins, hog_calc)
        xbox_left_all = np.concatenate((xbox_left_all, xbox_left)).flatten().astype(np.int)
        ytop_draw_all = np.concatenate((ytop_draw_all, ytop_draw)).flatten().astype(np.int)
        win_draw_all = np.concatenate((win_draw_all, win_draw)).flatten().astype(np.int)
        ystart_all = np.concatenate((ystart_all, ystart_ar)).flatten().astype(np.int)

    for i in range(len(xbox_left_all)):
        if i == 0:
            bboxes_raw =np.int32([[xbox_left_all[i], ytop_draw_all[i]+ystart_all[i],\
                                  xbox_left_all[i]+win_draw_all[i],ytop_draw_all[i]+win_draw_all[i]+ystart_all[i]]]
        else:
            bboxes_raw= np.concatenate((bboxes_raw, np.int32([[xbox_left_all[i], ytop_draw_all[i]+ystart_all[i],\
                                  xbox_left_all[i]+win_draw_all[i],ytop_draw_all[i]+win_draw_all[i]+ystart_all[i]]]
    return bboxes_raw
```

With a defined window size we will step across the image in a grid pattern, extract the features in each window, run the classifier to give a prediction at each step and identify if there is a vehicle in this window (function `find_cars`), if so save the window.

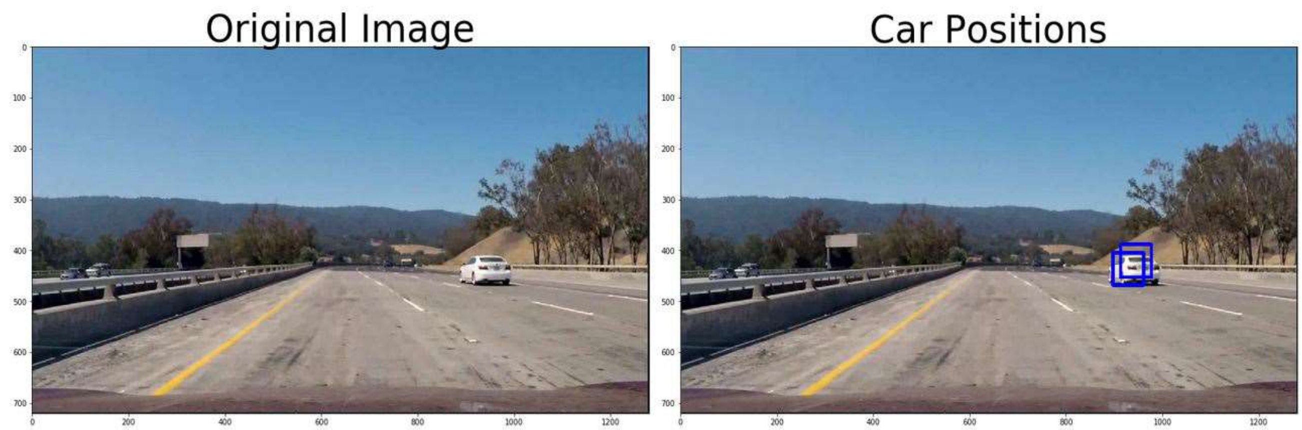
I decided to search only in the lower half of the image where vehicles are likely to occur. As there might appear vehicle of different sizes at different positions I used three different window scales in the end. As small vehicles appear more likely near the horizon I restricted the area for the different scaled windows to different areas in y direction. As we could also keep the aspect ratio in mind, we can crop also different x areas for the different sizes and positions of the windows. I used a 75% overlap to be more robust in detection. As a result my window grid looks as follows:

```
color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL hog_calc : skimage
spatial_size:  (4, 4) hist_bins:  256 clf_svc.cell_per_win:  8
data_set:  data_base_png/*.png
```

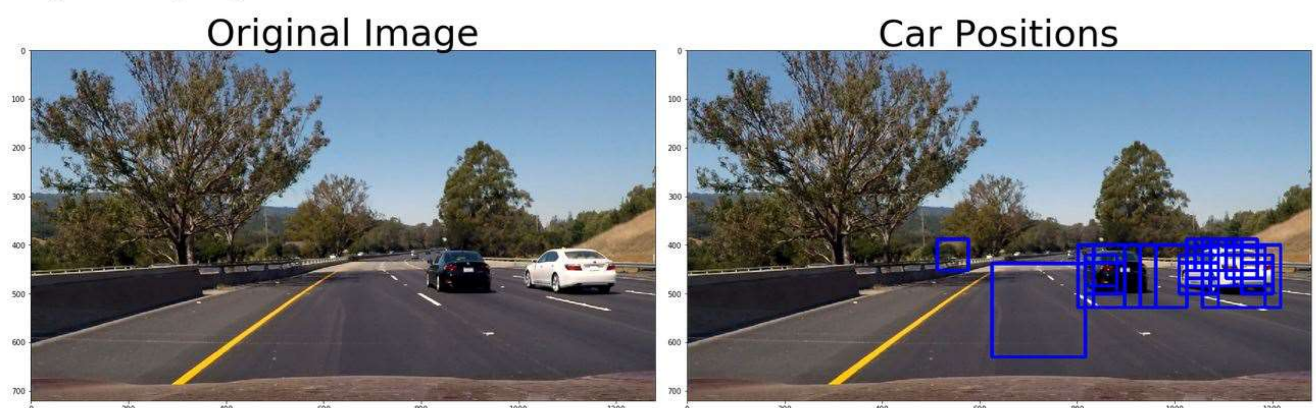


With use of the find_cars function one will get the following result:

```
color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL hog_calc : skimage
spatial_size:  (4, 4) hist_bins:  256 clf_svc.cell_per_win:  8
data_set:  data_base_png/*.png
```




```
color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL hog_calc : skimage
spatial_size:  (4, 4) hist_bins:  256 clf_svc.cell_per_win:  8
data_set:  data_base_png/*.png
```



As you can see we have to deal with multiple detections at (or nearly) the same position as well as with positive false detections in the lower image. To minimize the positive false rate and to ensure a high confidence for the prediction I used the `svc.decision_function` in the `find_cars` function which returns the distance of the samples to the separating hyperplane.

```
                    test_proba = svc.decision_function(test_features)
                    if test_proba > clf_svc.threshold_proba:
#                    if test_proba > -0.35:
                        xbox_left.append(np.int(xleft*scale))
                        ytop_draw.append(np.int(ytop*scale))
                        win_draw.append(np.int(window*scale))
                        ystart_ar.append(ystart)
```

The coordinates of the windows which are classified as vehicle are added to a list which is called 'car_boxes_raw' in the picture and video pipeline. With the use of the `draw_boxes` function one can draw these windows back onto the image.

III.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?

Ultimately I searched on three scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. There are some example images in the next chapter.

III.3. 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.

As already mentioned, besides the false positive identifications there are also multiple identification at nearly the same position. So the aim is to get a tight bounding box for each car independent of a multiple or a single detection as a path-planning or motion control algorithm might take actions, where it isn't necessary or even dangerous. This is the same with the false detections as this can lead to actions like emergency breaking when it's not necessary as in the image shown in chapter III.1.

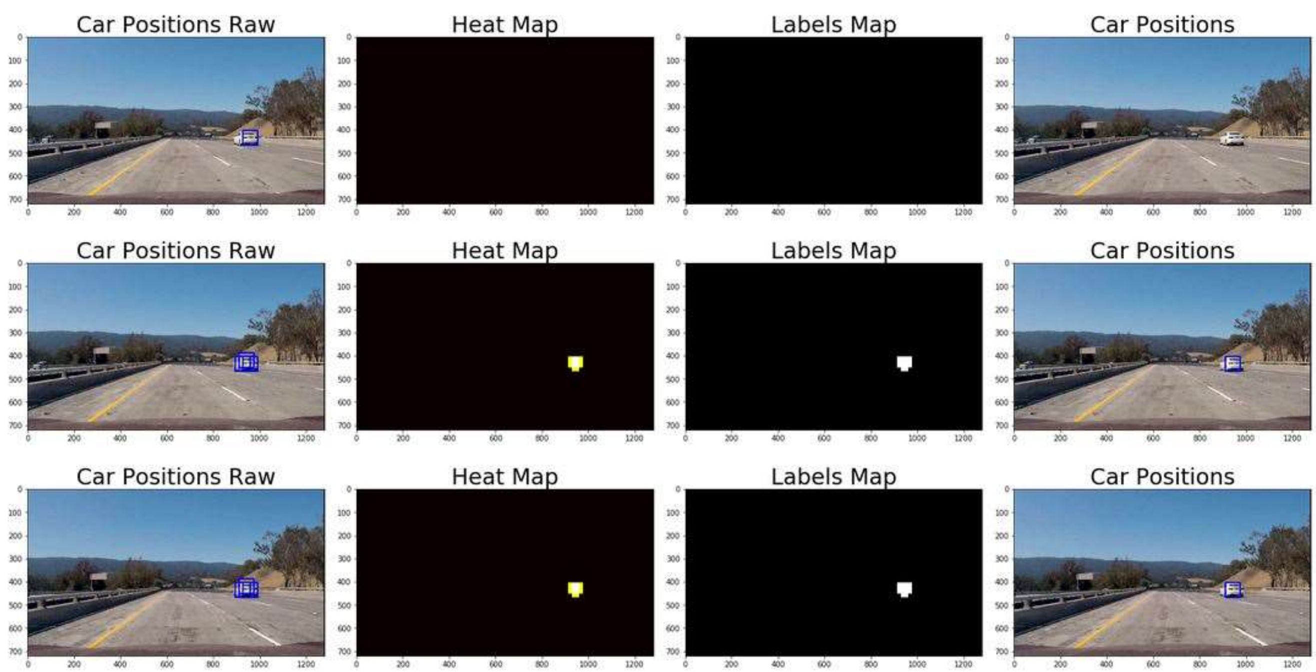
So having all positive detections save in the 'car_boxes_raw' list, I create a heatmap and threshold it to identify the vehicle positions and eliminate some further positive false recognitions. This is done in the picture pipeline as well as in the video pipeline afterwards.

After that I used `scipy.ndimage.measurements.label()` to identify individual positions in the heatmap, assuming each positions corresponds to a vehicle. I constructed bounding boxes to cover the area of each blob detected. I tried to make use of the non-maximum suppression function which is also added in the jupyter notebook. But in the end I got better results with the heatmap and label function.

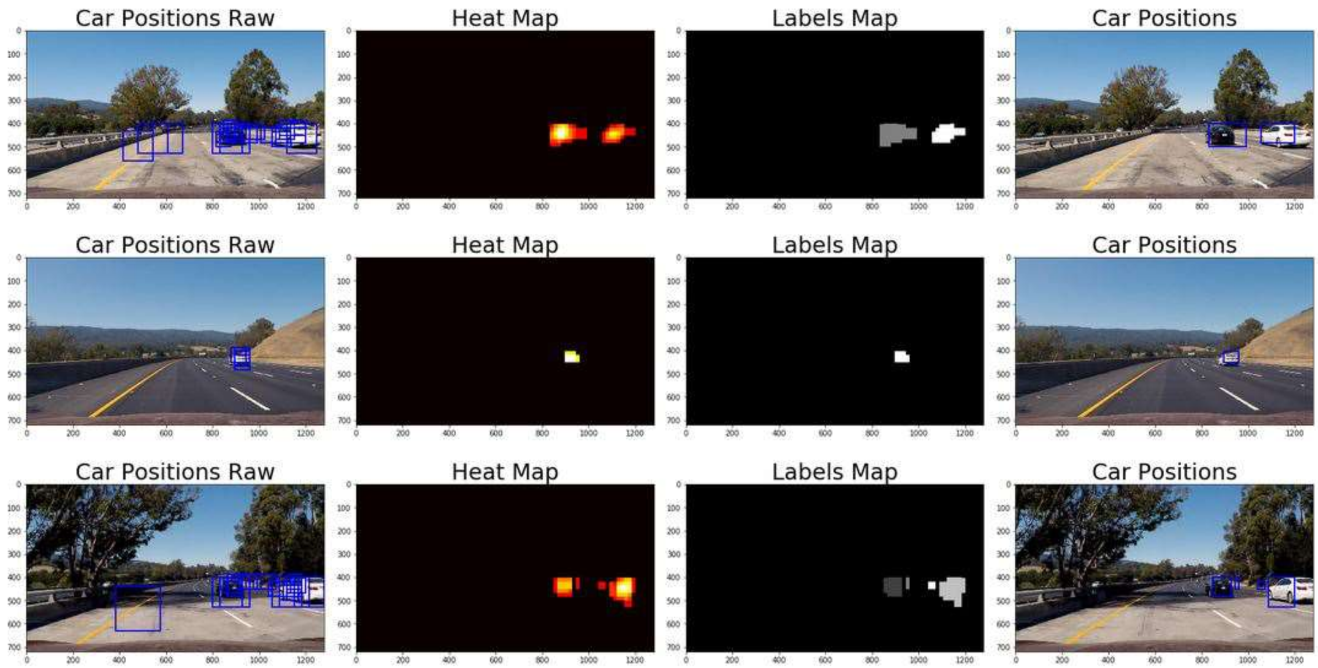
Below are example results showing the image with all windows with positive detections. Then the resulting heatmap with a threshold of 2. Besides the label image (the result of `scipy.ndimage.measurements.label()`) with the identified "vehicles" and on the right the resulting images with vehicle detected and the tight bounding boxes drawn.

```
images = glob.glob('test_images/*.jpg')
for fname in images:
    # read in each image
    img =  mpimg.imread(fname)
    draw_img = np.copy(img)
    # Find cars with multiple window sizes
    car_boxes_raw = mul_win_slide(draw_img,svc, X_scaler, orient, pix_per_cell, cell_per_block, \
                              spatial_size, hist_bins, cell_per_win, hog_calc)

    # Remove duplicates
    draw_img = np.copy(img)
    heat = np.zeros_like(draw_img[:,:,0]).astype(np.float)
    # Add heat to each box in box list
    heat = add_heat(heat,car_boxes_raw)
    # 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)
    bboxes_raw = []
    print(fname)

    if debug == True:
        image = np.copy(img)
        draw_img = debug_frame(image, draw_img, heatmap, heatmap, car_boxes_raw, heatmap)
        #draw_img = debug_frame(deb_image, draw_img, heat_combined, heat_combined_raw, car_boxes_raw, heatmap)

    box_img = np.copy(img)
    for n in range(len(car_boxes_raw)):
        cv2.rectangle(box_img,(car_boxes_raw[n,0],car_boxes_raw[n,1]),(car_boxes_raw[n,2],car_boxes_raw[n,3]),(0,0,2
```

```
 save file:  SVC_trained/LinearSVC_20170421_173733.p
 color_space:  YCrCb orient:  18 pix_per_cell:  8 cell_per_block:  2 hog_channel:  ALL hog_calc : skimage
 spatial_size:  (4, 4) hist_bins:  256 clf_svc.cell_per_win:  8
data_set:  data_base_png/*.png
test_images\frame_002.jpg
test_images\frame_004.jpg
test_images\frame_006.jpg
```



In this set one problem is clearly visible. There is no vehicle detected in the first row as my heatmap threshold filters out the single remaining window in the raw window set. Therefore in the video pipeline I have to install a ring buffer to save the predictions of the last n frames to bridge some frames with no detection. In the other images there are enough detections so that the heatmap with threshold is o.k. and the label identifies one car which is shown with a tight bounding box at the right position.



This set shows how positive false detections are correctly filtered out. But the chosen size of my windows as well as positive false detections can lead to a larger bounding box (1st row) which is undesirable. Also having two 'gates' to eliminate positive false detections (svc.desision_function and heatmap threshold) there are remaining positive false detections (3rd row) which should be eliminated.

IV. Video Implementation

IV.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.)

Video is added to the zip file.

IV.2. Describe how (and identify where in your code) you implemented some kind of filter for false positives.

In the video pipeline I tried to filter out the remaining positive false detections by determining which detections appear in one frame but not in the next. Therefore I created a class car_boxes:

```
class car_boxes:
    def __init__(self):
        self.all_boxes = deque(maxlen=12)
```

All detected windows of 12 consecutive frames are stored in self.all_boxes. With the use of the OpenCV function `cv2.groupRectangles` in the function `process_frame` all overlapping windows are combined in to consolidated bounding boxes. The threshold of this function is set to 7, which means that it will only look for areas where more than 7 overlapping windows occur and ignores anything else. Only detections which occur in more than 7 of the 12 frames will be taken into account and shown onto the resulting image. This also helps to bridge a series of poor vehicle detections.

```
# Remove multiple detections => get boxes around vehicles
labels = label(heatmap)
bboxes = []
bboxes = labeled_bboxes(draw_img, labels)
# Append boxes to ring buffer
car_bx.all_boxes.append(bboxes)

all_boxes = []
box_comb = np.ravel(np.array(car_bx.all_boxes))
# combine the boxes from current and previous frames
for i in range(len(box_comb)):
    all_boxes.extend(np.ravel(box_comb[i]))
new_boxes = []
i = 0
while i <= len(all_boxes)-3:
    new_boxes.append(all_boxes[i:i+4])
    i += 4
# combine overlapping rectangles
# more than 7 overlapping boxes
end_bxes,w = cv2.groupRectangles(np.array(new_boxes).tolist(), 7,.05) #

for box in end_bxes:
    cv2.rectangle(draw_img, (box[0], box[1]), (box[2],box[3]), (0,255,0)

if debug == True:
    deb_image =  np.copy(undist_img)
    draw_img = debug_frame(deb_image, draw_img, heatmap, heatmap, car_bc

return draw_img
```

V. Discussion

V.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?

I mainly focused my work on the elimination of the positive false detections which works quite well, but has some dropouts of the vehicle detections as a side effect.

I was overwhelmed how much parameters there are to tune and which effect they all have onto the result of vehicle detection. One problem for me was to identify the best classifier and the best parameter set. I think this has a lot to do with the image data set (and the existing sequences of images) which will lead to the problem of overfitting. Nearly all test sets result in an accuracy of over 99,5 % but giving sometimes a really terrible vehicle prediction or a lot of positive false.

Another problem, I already mentioned, is the poor vehicle identification in some frames for the white car in some constellations (brightness, position, surrounding area) where only one/two windows identify the vehicle, which is too little to get over the heatmap threshold. On the other side there are too much positive false identifying the shoulder and lanes (especially the yellow one) on the left side, although there are lots of images in the data set showing such scenes.

So in my opinion I have to pay more attention and add more data to the training of the classifiers to generalize more and therefore have the possibility to select a classifier with means of accuracy.

The video pipeline is much to slow for a real time application (less than one frame per second – goal should be about 25-30 frames per second). First I use the svc.decision_function which slows down the prediction a lot, perhaps the use of the svc probability function would do better here.

Another parameter to tune the speed is to use as less windows as possible. Perhaps here a vehicle tracking from frame to frame and estimating where it will appear in the next frame can speed up the detection.

Certainly using less features will result in a faster detection.

There are several functions like the hog function in OpenCV which are really fast and can speed up recognition a lot. I will try to implement these functions and look how they are working.