P5

August 12, 2017

1 Self-Driving Car Engineer Nanodegree

1.1 Project: 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.

1.2 Rubric Points

1.2.1 Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

1.2.2 Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images. The code for this step is contained in the first code cell of the IPython notebook.

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:

```
In [1]: import matplotlib.image as mpimg
        import matplotlib.pyplot as plt
        import numpy as np
```

```python
import cv2
import glob
from skimage.feature import hog
from skimage import color, exposure
%matplotlib inline
# images are divided up into vehicles and non-vehicles

images = glob.glob('data/*/*/*.png')
cars = []
notcars = []

for image in images:
    if 'non-vehicles' in image:
        notcars.append(image)
    else:
        cars.append(image)

# 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)
    # Define a key "n_notcars" and store the number of notcar images
    data_dict["n_notcars"] = len(notcar_list)
    # Read in a test image, either car or notcar
    img = mpimg.imread(car_list[0])
    # Define a key "image_shape" and store the test image shape 3-tuple
    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

data_info = data_look(cars, notcars)

print('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"])

# Just for fun choose random car / not-car indices and plot example images
car_ind = np.random.randint(0, len(cars))
notcar_ind = np.random.randint(0, len(notcars))

# Read in car / not-car images
print(cars[car_ind])
car_image = mpimg.imread(cars[car_ind])
```

```
        notcar_image = mpimg.imread(notcars[notcar_ind])


        # Plot the examples
        fig = plt.figure()
        plt.subplot(121)
        plt.imshow(car_image)
        plt.title('Example Car Image')
        plt.subplot(122)
        plt.imshow(notcar_image)
        plt.title('Example Not-car Image')
```

```
returned a count of 8792  cars and 8968  non-cars
of size:  (64, 64, 3)  and data type: float32
data\vehicles\KITTI_extracted\2034.png
```

```
Out[1]: <matplotlib.text.Text at 0x2e9589aeac8>
```

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.

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:

```
In [2]: from sklearn.preprocessing import StandardScaler
        from skimage.feature import hog

        # Define a function to return HOG features and visualization
        def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                                vis=False, feature_vec=True):
```

```python
    # 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


# 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):
    # 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)
        # 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)

        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
```

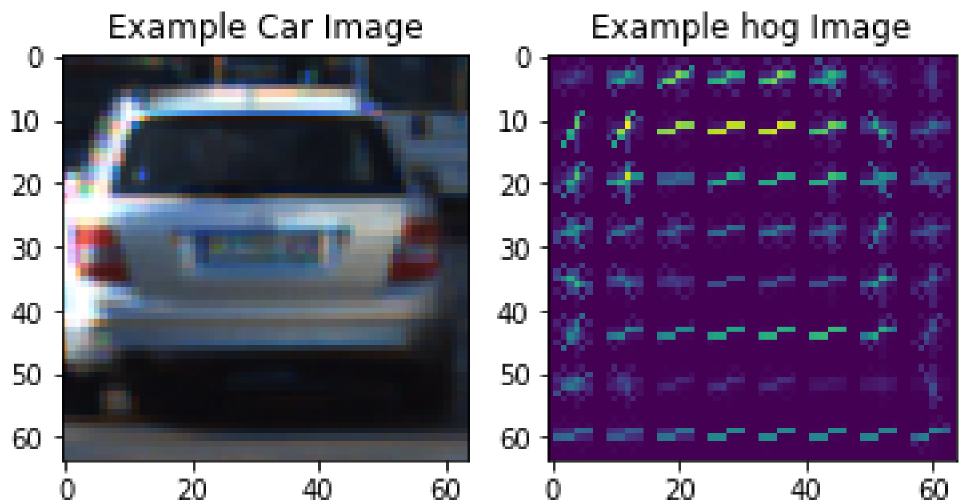
```
        if hist_feat == True:
            # Apply color_hist()
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        if hog_feat == True:
        # Call get_hog_features() with vis=False, feature_vec=True
            if hog_channel == 'ALL':
                hog_features = []
                for channel in range(feature_image.shape[2]):
                    hog_features.append(get_hog_features(feature_image[:,:,channel],
                                        orient, pix_per_cell, cell_per_block,
                                        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, 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

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

# 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 color channels separately
    channel1_hist = np.histogram(img[:,:,0], bins=nbins)
    channel2_hist = np.histogram(img[:,:,1], bins=nbins)
    channel3_hist = np.histogram(img[:,:,2], bins=nbins)
    # Concatenate the histograms into a single feature vector
    hist_features = np.concatenate((channel1_hist[0], channel2_hist[0], channel3_hist[
    # Return the individual histograms, bin_centers and feature vector
    return hist_features

color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # 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 = (16, 16) # Spatial binning dimensions
hist_bins = 16    # 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
#y_start_stop = [450, None] # Min and max in y to search in slide_window()


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)



# 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 the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))

print('finished')

features, hog_image = get_hog_features(cv2.cvtColor(car_image, cv2.COLOR_BGR2GRAY), ori
fig = plt.figure()
plt.subplot(121)
plt.imshow(car_image)
plt.title('Example Car Image')
plt.subplot(122)
plt.imshow(hog_image)
plt.title('Example hog Image')
```

```
d:\Anaconda3\envs\carnd-term1\lib\site-packages\skimage\feature\_hog.py:119: skimage_deprecatio
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```

```
finished
```

```
Out[2]: <matplotlib.text.Text at 0x2e9419539e8>
```



2. Explain how you settled on your final choice of HOG parameters. I tried various combinations of parameters and i choose

colorspace = ’YCrCb’ # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9
pix_per_cell = 8
cell_per_block = 2
hog_channel = ALL # Can be 0, 1, 2, or "ALL"

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). I trained a linear SVM using code:

```
In [3]: from sklearn.svm import LinearSVC
        from sklearn.model_selection import train_test_split
        import time

        # 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, test_size=0.2, random_state=rand_state)

        print('Using:',orient,'orientations',pix_per_cell,
            'pixels per cell and', cell_per_block,'cells per block')
        print('Feature vector length:', len(X_train[0]))
        # Use a linear SVC
        svc = LinearSVC(C=2)
        # Check the training time for the SVC
        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')
```

```
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 6108
3.98 Seconds to train SVC...
Test Accuracy of SVC =  0.9918
My SVC predicts:  [ 1.  0.  0.  1.  0.  1.  1.  0.  0.  1.]
For these 10 labels:  [ 1.  0.  0.  1.  0.  1.  1.  0.  0.  1.]
0.001 Seconds to predict 10 labels with SVC
```

1.2.3 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? Firs I've add a sliding window in borders from Ymin to Ymax with different windows sizes. (Just as a training on images, the video pipline is a bit diferent). First I did some tests with functions slide_window and search_windows. After that I switch pipeline to use find_cars_images.

I use 3 levels of detection to increase range of classification. First with small window detects object far away, second the optimal scale and the third the objects close to camera. In this combination I can find cars to range bigger than 10m and almost from the moment of entering the field of camera view. Also I'm increasing "heat" of detected objects, and I can cut false positives by higher treshold.

```
In [4]: ###### Define a function to draw bounding boxes
        def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
            # 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
                cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)
            # Return the image copy with boxes drawn
            return imcopy

        # Define a function that takes an image,
        # start and stop positions in both x and y,
        # window size (x and y dimensions),
        # and overlap fraction (for both x and y)
```

```python
def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None],
                    xy_window=(64, 64), xy_overlap=(0.5, 0.5)):
    # If x and/or y start/stop positions not defined, set to image size
    if x_start_stop[0] == None:
        x_start_stop[0] = 0
    if x_start_stop[1] == None:
        x_start_stop[1] = img.shape[1]
    if y_start_stop[0] == None:
        y_start_stop[0] = 0
    if y_start_stop[1] == None:
        y_start_stop[1] = img.shape[0]
    # Compute the span of the region to be searched
    xspan = x_start_stop[1] - x_start_stop[0]
    yspan = y_start_stop[1] - y_start_stop[0]
    # Compute the number of pixels per step in x/y
    nx_pix_per_step = np.int(xy_window[0]*(1 - xy_overlap[0]))
    ny_pix_per_step = np.int(xy_window[1]*(1 - xy_overlap[1]))
    # Compute the number of windows in x/y
    nx_buffer = np.int(xy_window[0]*(xy_overlap[0]))
    ny_buffer = np.int(xy_window[1]*(xy_overlap[1]))
    nx_windows = np.int((xspan-nx_buffer)/nx_pix_per_step)
    ny_windows = np.int((yspan-ny_buffer)/ny_pix_per_step)
    # Initialize a list to append window positions to
    window_list = []
    # Loop through finding x and y window positions
    # Note: you could vectorize this step, but in practice
    # you'll be considering windows one by one with your
    # classifier, so looping makes sense
    for ys in range(ny_windows):
        for xs in range(nx_windows):
            # Calculate window position
            startx = xs*nx_pix_per_step + x_start_stop[0]
            endx = startx + xy_window[0]
            starty = ys*ny_pix_per_step + y_start_stop[0]
            endy = starty + xy_window[1]

            # Append window position to list
            window_list.append(((startx, starty), (endx, endy)))
    # Return the list of windows
    return window_list


# 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[:,:,channel],
                                    orient, pix_per_cell, cell_per_block,
                                    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, vis=False, fe
        #8) Append features to list
        img_features.append(hog_features)
    #9) Return concatenated array of features
    return np.concatenate(img_features)

# Define a function you will pass an image
# and the list of windows to be searched (output of slide_windows())
def search_windows(img, windows, svc, scaler, color_space='RGB',
                    spatial_size=(32, 32), hist_bins=32,
                    hist_range=(0, 256), orient=9,
```

```python
                    pix_per_cell=8, cell_per_block=2,
                    hog_channel=0, spatial_feat=True,
                    hist_feat=True, hog_feat=True):

    #1) Create an empty list to receive positive detection windows
    on_windows = []
    #2) Iterate over all windows in the list
    for window in windows:
        #3) Extract the test window from original image
        test_img = cv2.resize(img[window[0][1]:window[1][1], window[0][0]:window[1][0]
        #4) Extract features for that window using single_img_features()
        features = single_img_features(test_img, 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_fe
                                    hist_feat=hist_feat, hog_feat=hog_feat)
        #5) Scale extracted features to be fed to classifier
        test_features = scaler.transform(np.array(features).reshape(1, -1))
        #6) Predict using your classifier
        prediction = svc.predict(test_features)
        #7) If positive (prediction == 1) then save the window
        if prediction == 1:
            on_windows.append(window)
    #8) Return windows for positive detections
    return on_windows


# Define a single function that can extract features using hog sub-sampling and make p
def find_cars_images(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell, c

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

    img_tosearch = img[ystart:ystop,:,:]
    ctrans_tosearch = convert_color(img_tosearch, conv='RGB2YCrCb')
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale), np.in

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

hot_windows=[]
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_windo
        hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_windo
        hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_windo
        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:xleft+window],

        # 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.hstack((spatial_features, hist_featu
        #test_features = X_scaler.transform(np.hstack((shape_feat, hist_feat)).res
        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)
            hot_windows.append([[xbox_left, ytop_draw+ystart],[xbox_left+win_draw,y
            cv2.rectangle(draw_img,(xbox_left, ytop_draw+ystart),(xbox_left+win_dra

    heat = np.zeros_like(img[:,:,0]).astype(np.float)
    heat = add_heat(heat, hot_windows)
```

```python
        # 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(image), labels)

    return draw_img, hot_windows




def convert_color(img, conv='RGB2YCrCb'):
    if conv == 'RGB2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    if conv == 'BGR2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    if conv == 'RGB2LUV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2LUV)

def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                        vis=False, feature_vec=True):
    # 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=False,
                                  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=False,
                       visualise=vis, feature_vector=feature_vec)
        return features


def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
```

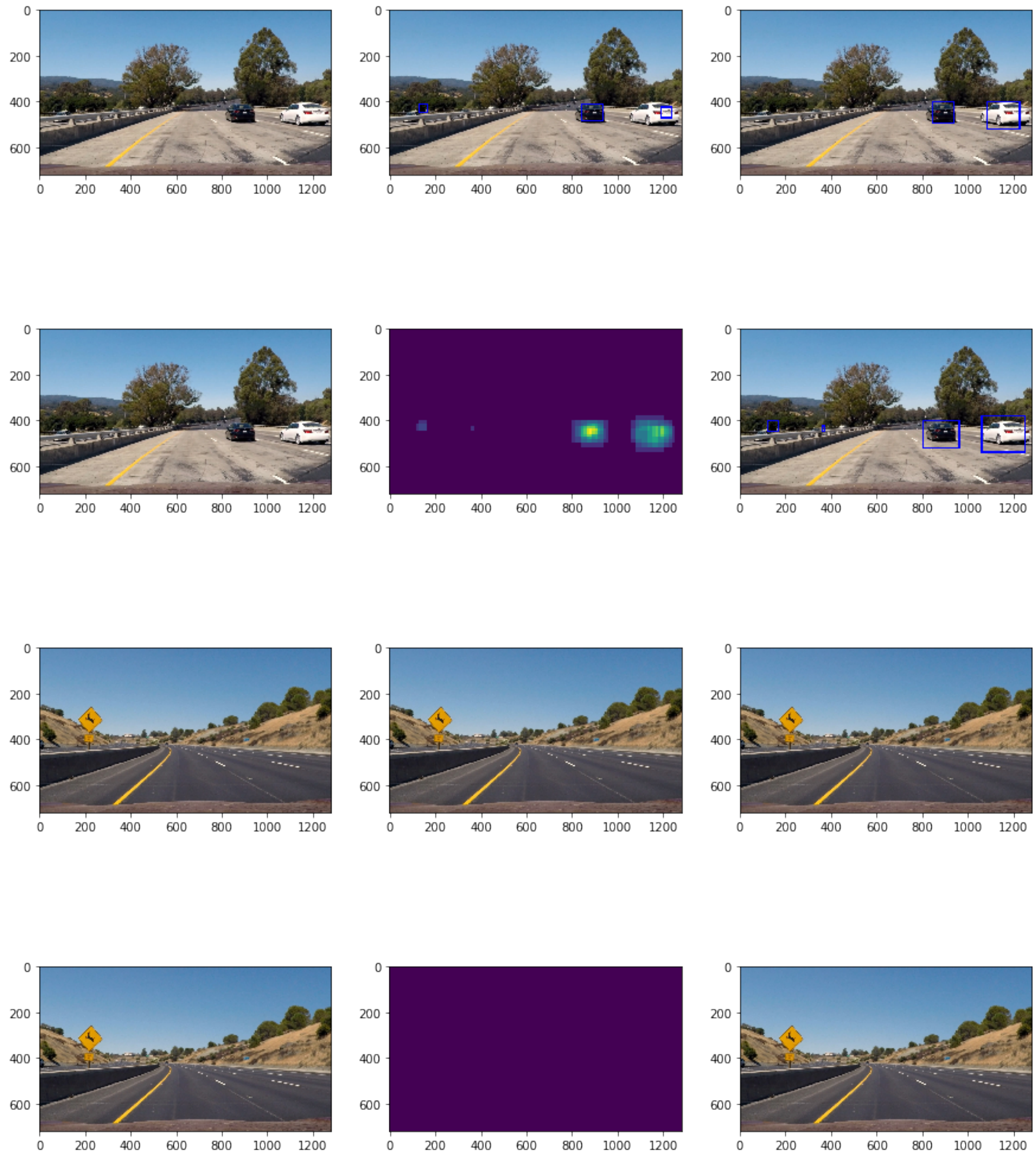
```
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1

    # 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
    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
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzero
        # Draw the box on the image
        cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)
    # Return the image
    return img


from scipy.ndimage.measurements import label

ystart = 400
ystop = 656
scale = 1.5

testimgs = glob.glob('test_images/*.jpg')
i = 0
for testimg in testimgs:

    image = mpimg.imread(testimg)
    draw_image = np.copy(image)

    hot_windows = []
    plt.figure(figsize = [15, 5])
    plt.subplot(1, 3, 1)
    plt.imshow(image)

    #Searching cars
    #1st line scan
```

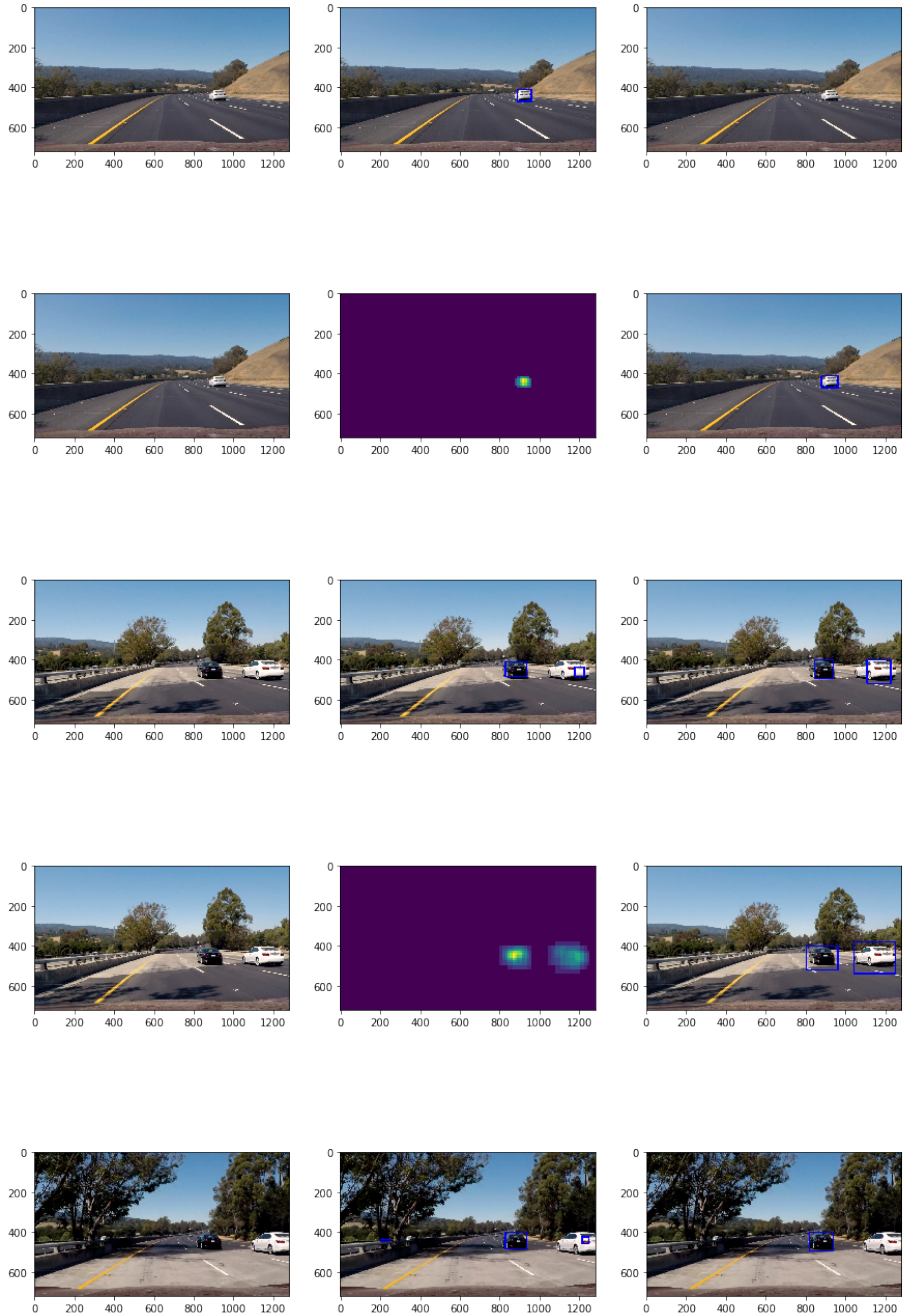
```
ystart = 400
ystop = 500
scale = 0.75
xwindow = 64
ywindow = 64
windows = slide_window(image, x_start_stop=[None, None], y_start_stop=[ystart, ysto
                       xy_window=(xwindow, ywindow), xy_overlap=(0.5, 0.5))

hot_windows1 = search_windows(image, windows, svc, X_scaler, color_space=color_spac
                              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)
hot_windows.extend(hot_windows1)

boxed_img = draw_boxes(image, hot_windows, color=(0, 0, 255), thick=6)

out_img, hot_windows21 = find_cars_images(image, ystart, ystop, scale, svc, X_scale
                    spatial_size, hist_bins)
plt.subplot(1, 3, 2)
plt.imshow(out_img)
#2nd line scan
ystart = 400
ystop = 656
scale = 1.5
xwindow = 128
ywindow = 128
windows = slide_window(image, x_start_stop=[None, None], y_start_stop=[ystart, ysto
                       xy_window=(xwindow, ywindow), xy_overlap=(0.5, 0.5))

#boxed_img = draw_boxes(image, windows, color=(0, 0, 255), thick=6)
#plt.subplot(1, 3, 3)
#plt.imshow(boxed_img)

hot_windows2 = search_windows(image, windows, svc, X_scaler, color_space=color_spac
                              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)
hot_windows.extend(hot_windows2)
#print(len(hot_windows))
boxed_img = draw_boxes(image, hot_windows, color=(0, 0, 255), thick=6)
plt.subplot(1, 3, 3)
plt.imshow(boxed_img)

out_img, hot_windows22 = find_cars_images(image, ystart, ystop, scale, svc, X_scale
```

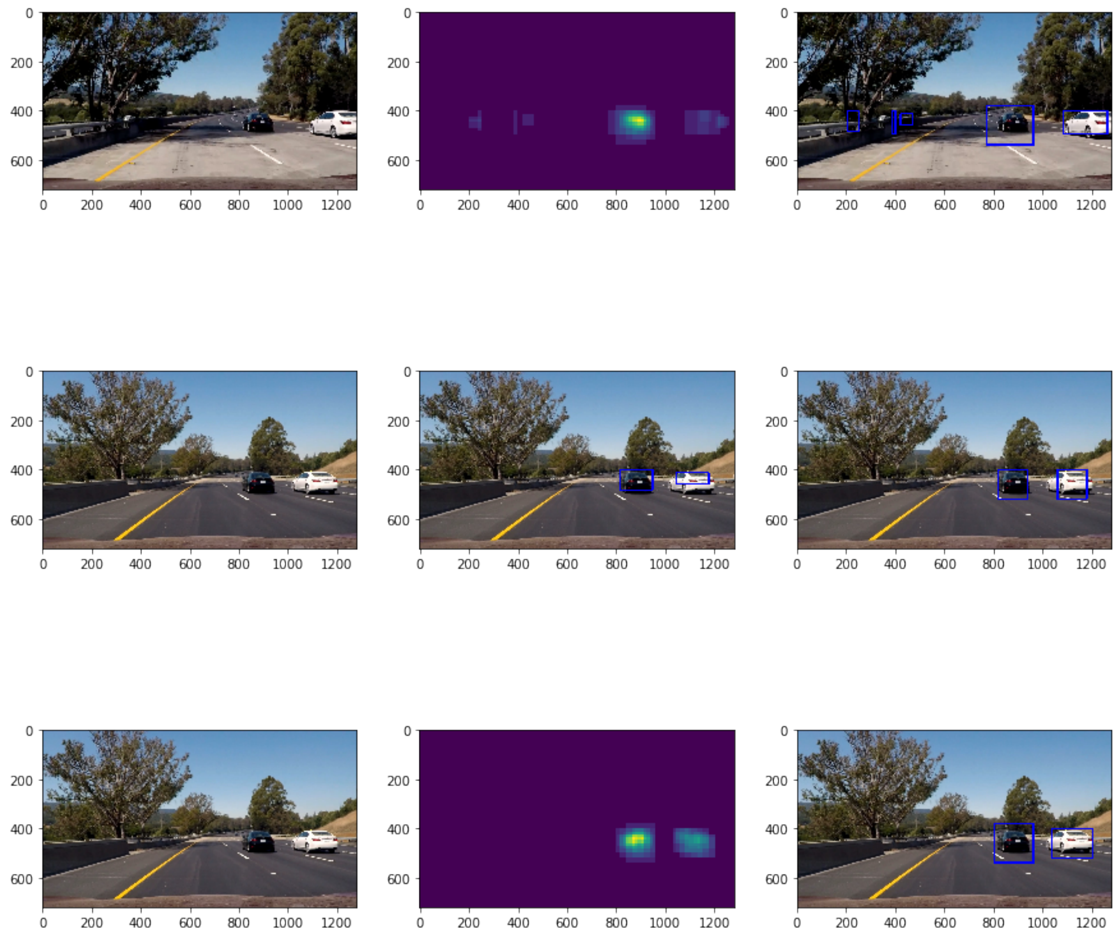
```
                    spatial_size, hist_bins)
plt.subplot(1, 3, 3)
plt.imshow(out_img)
#3rd line scan
ystart = 380
ystop = 800
scale = 2.5
xwindow = 256
ywindow = 192
windows = slide_window(image, x_start_stop=[None, None], y_start_stop=[ystart, ysto
                       xy_window=(xwindow, ywindow), xy_overlap=(0.5, 0.5))

#boxed_img = draw_boxes(image, windows, color=(0, 0, 255), thick=6)
#plt.subplot(1, 3, 3)
#plt.imshow(boxed_img)

hot_windows2 = search_windows(image, windows, svc, X_scaler, color_space=color_spac
                              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)
hot_windows.extend(hot_windows2)

window_img = draw_boxes(draw_image, hot_windows, color=(0, 0, 255), thick=6)
out_img, hot_windows23 = find_cars_images(image, ystart, ystop, scale, svc, X_scale
                    spatial_size, hist_bins)
hot_windows2x =[]
hot_windows2x.extend(hot_windows21)
hot_windows2x.extend(hot_windows22)
hot_windows2x.extend(hot_windows23)
# Add heat to each box in box list
plt.figure(figsize = [15, 5])
plt.subplot(1, 3, 1)
plt.imshow(out_img)



heat = np.zeros_like(image[:,:,0]).astype(np.float)
heat = add_heat(heat, hot_windows2x)
#heat = add_heat(heat, hot_windows)

# Apply threshold to help remove false positives
heat = apply_threshold(heat, 1)

# Visualize the heatmap when displaying
heatmap = np.clip(heat, 0, 255)
plt.subplot(1, 3, 2)
```

```
    plt.imshow(heatmap)
    # Find final boxes from heatmap using label function
    labels = label(heatmap)
    draw_img = draw_labeled_bboxes(np.copy(image), labels)
    plt.imsave('test_mages_output/output'+str(i)+'.jpeg', draw_img)
    i = i+1

    plt.subplot(1, 3, 3)
    plt.imshow(draw_img)
```

```
d:\Anaconda3\envs\carnd-term1\lib\site-packages\skimage\feature\_hog.py:119: skimage_deprecati
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
```







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. ('project video.mp4' output movie). I used modified finding_cars function without drawing output, only with hot windows result. This speed up the movie generation. Next I'm using heat map and labels to find bounding boxes. I'm adding hot windows from present and 10 previous frames, to avoid false positive alarms (history)

```
In [5]: def find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell, cell_per_

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

            img_tosearch = img[ystart:ystop,:,:]
            ctrans_tosearch = convert_color(img_tosearch, conv='RGB2YCrCb')
            if scale != 1:
```

```
    imshape = ctrans_tosearch.shape
    ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale), np.in

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

hot_windows=[]
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_windo
        hog_feat2 = hog2[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_windo
        hog_feat3 = hog3[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_windo
        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:xleft+window],

        # 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.hstack((spatial_features, hist_featu
        #test_features = X_scaler.transform(np.hstack((shape_feat, hist_feat)).res
```

```
                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)
                    hot_windows.append([[xbox_left, ytop_draw+ystart],[xbox_left+win_draw,y

        return hot_windows
```

1.2.4 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

```
In [8]: from collections import deque
        history = deque(maxlen = 10)
        #global p_hot_windows
        #p_hot_windows = []
        #global pp_hot_windows
        #pp_hot_windows = []

        def process_image(image):
            draw_image = np.copy(image)

            hot_windows = []
            #Searching cars
            #1st line scan
            ystart = 400
            ystop = 500
            scale = 0.75

            hot_windows21 = find_cars(image, ystart, ystop, scale, svc, X_scaler, orient, pix_
                                spatial_size, hist_bins)
            #2nd line scan
            ystart = 400
            ystop = 656
            scale = 1.5

            hot_windows22 = find_cars(image, ystart, ystop, scale, svc, X_scaler, orient, pix_
                                spatial_size, hist_bins)
            #3rd line scan
            ystart = 380
            ystop = 800
            scale = 1
```

```
        hot_windows23 = find_cars(image, ystart, ystop, scale, svc, X_scaler, orient, pix_p
                            spatial_size, hist_bins)
        hot_windows2x =[]
        hot_windows2x.extend(hot_windows21)
        hot_windows2x.extend(hot_windows22)
        hot_windows2x.extend(hot_windows23)
        hot_windows.extend(hot_windows2x)

        for windows in history:
            hot_windows.extend(windows)
        #global p_hot_windows
        #global pp_hot_windows

        # Add heat to each box in box list
        heat = np.zeros_like(image[:,:,0]).astype(np.float)
        heat = add_heat(heat, hot_windows)
        #heat = add_heat(heat, p_hot_windows)
        #heat = add_heat(heat, pp_hot_windows)

        # Apply threshold to help remove false positives
        heat = apply_threshold(heat, 35)  #2, 3

        # 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(image), labels)
        #pp_hot_windows = p_hot_windows
        #p_hot_windows = hot_windows2x
        history.append(hot_windows2x)
        return draw_img


    ## Test on Videos

    # Import everything needed to edit/save/watch video clips
    from moviepy.editor import VideoFileClip
    from IPython.display import HTML

    white_output = 'test_videos_output/project_video.mp4'


    clip1 = VideoFileClip("project_video.mp4")#.subclip(0,5)
    white_clip = clip1.fl_image(process_image)
    %time white_clip.write_videofile(white_output, audio=False)
```

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

```
100%|| 1260/1261 [45:48<00:02,  2.24s/it]


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

Wall time: 45min 49s
```

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 also add 10 previous hot windows map to increase accuracy and decrease false positives (I use variable history in process_image function). 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.

1.2.5 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? The detection works with limited range, but I think, the car in long range isn't necessery to detect. Also some false positives are detected and cars on the opposite lane (that could be both - good and bad, as we should know about car approaching from opposite line on the rural road).

The function should be more optimized to make generating the detection faster.

I used here a lot of code, that won't be taken into account during generating video image, just to show my way of work and example images

I think also the detection could be more robust and precisious (example the white car isn't fully covered by Bounding Box), maybe that could be improved by bigger training set

```
In [ ]:
```