Udacity Self-Driving Car Nanodegree

Project 5- Vehicle Detection and Tracking

Author: Johannes Betz

Date: 08/16/2017

Table of Contents

1. Introduction and goals

The following writeup includes the code and the descriptionfor project 5, vehicle detection and tracking.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.

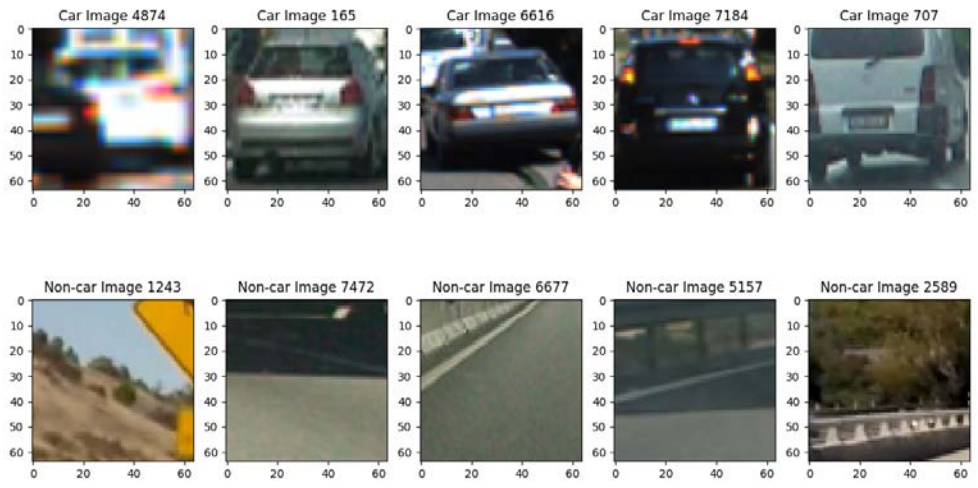
Every section provides one part of the project rubics. The whole code can be found in the file “project5.py”

2. Loading Data und Plot Example

In the first step, the data , provided as different pictures (vehicle and non-vehicle) are loaded. All training data provided by Udacity (i.e. GTI, KITTI and project video extracts) have been used for this project. Because the vehicle and non-vehicle data sets contain roughly the same number of images it was deemed unnecessary to ceate additional training images for one of the data set. In Addition an example plot is provided. The code looks as following:

```
# Training images paths
TRAINING_PATH_VEHICLES = "data/vehicles"
TRAINING_PATH_NON_VEHICLES = "data/non-vehicles"
TRAINING_PATH_TRAINED_MODEL = "models"

vehicles = []
non_vehicles = []

# Vehicle images names
print("Loading training image names...")
for image in glob.glob(TRAINING_PATH_VEHICLES + '/**/*.png',
recursive=True):
    vehicles.append(image)

# Non-vehicle images names
for image in glob.glob(TRAINING_PATH_NON_VEHICLES + '/**/*.png',
recursive=True):
     non_vehicles.append(image)

print('    # of vehicle images: {}'.format(len(vehicles)))
print('# of non-vehicle images: {}'.format(len(non_vehicles)))

############################################################################
#######
###############            PLOT TEST EXAMPLE
```

```
####################
###########################################################################
#######

plt.figure(figsize=(15,8))

for i in range(5):
    car_ind = np.random.randint(0, len(vehicles))
    notcar_ind = np.random.randint(0, len(non_vehicles))

    # Read in car / not-car images
    car_image = mpimg.imread(vehicles[car_ind])
    notcar_image = mpimg.imread(non_vehicles[notcar_ind])


    plt.subplot(2,5,i+1)
    plt.imshow(car_image)
    plt.title('Car Image ' + str(car_ind))
    plt.subplot(2,5,5+i+1)
    plt.imshow(notcar_image)
    plt.title('Non-car Image ' + str(notcar_ind))

plt.show()
```



3. Histrogram of Oriented Gradients (HOG)

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

In the second step, the feautres are extracted. The code looks like the following:

```
# Feature extraction parameters
colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 11
pix_per_cell = 16
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
```

```
t = time.time()
car_features = extract_features(vehicles, cspace=colorspace, orient=orient,
                        pix_per_cell=pix_per_cell,
cell_per_block=cell_per_block,
                        hog_channel=hog_channel)
notcar_features = extract_features(non_vehicles, cspace=colorspace,
orient=orient,
                        pix_per_cell=pix_per_cell,
cell_per_block=cell_per_block,
                        hog_channel=hog_channel)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to extract HOG features...')
# Create an array stack of feature vectors
X = np.vstack((car_features, notcar_features)).astype(np.float64)

y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))


# 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(
    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]))
```

It combines a number of features into one long one-dimensional feature vector for each training image:

1. **Histogram of Oriented Gradient (HOG)**: HOG features are extracted in function `__get_hog_features` and returned as a one-dimensional feature vector
2. **Spatial binning of color**: function `__bin_spatial` returns a scaled down version of an image as a one-dimensional feature vector
3. **Histograms of color**: function `__color_hist` returns the histograms of all three color channels as a single one-dimensional feature vector.

The Code looks like the following:

```
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 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))

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[0]))
    # Return the individual histograms, bin_centers and feature vector
    return hist_features

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


# Define a function to extract features from a list of image locations
# This function could also be used to call bin_spatial() and color_hist()
(as in the lessons) to extract
# flattened spatial color features and color histogram features and combine
them all (making use of StandardScaler)
# to be used together for classification
def extract_features(imgs, cspace='RGB', orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        if cspace != 'RGB':
            if cspace == 'HSV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif cspace == 'LUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif cspace == 'HLS':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif cspace == 'YUV':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif cspace == 'YCrCb':
                feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else: feature_image = np.copy(image)

        # 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
        features.append(hog_features)
    # Return list of feature vectors
    return features
```

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

A linear Support Vector Machine is used as the classifier. It is trained as “LinearSVC”. Before training commences the training data is split into a large training set (90%) and a small test set (10%).

1.44 Seconds to train SVC...

Test Accuracy of SVC = 0.9848

My SVC predicts: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]

For these 10 labels: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]

0.001 Seconds to predict 10 labels with SVC

Classifier accuracy on the test set is `98.48%`. Changing hyperparameters and/or using a different classifier was considered but not implemented because performance seems adequate with just the default parameters. After training, both the classifier and feature scaler are saved. This means a trained model can be used repeatedly without having to train the model from scratch as the pipeline is developed further.

```
train =1
if train == 1:
    # Use a linear SVC 
    svc = LinearSVC()
    # 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...')

    joblib.dump(svc, 'model/Test_model.pkl')
    # 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')
else:
    svc=joblib.load('model/Test_model.pkl')
```

4. Sliding Window Search

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?

Do find cars in a example picture, a pipeline is provided. The code pipeline for all test images looks like this:

```
test_images = glob.glob('./test_images/test*.jpg')

fig, axs = plt.subplots(3, 2, figsize=(16,14))
fig.subplots_adjust(hspace = .004, wspace=.002)
axs = axs.ravel()

for i, im in enumerate(test_images):
    axs[i].imshow(process_frame(mpimg.imread(im)))
    axs[i].axis('off')

plt.show()
```

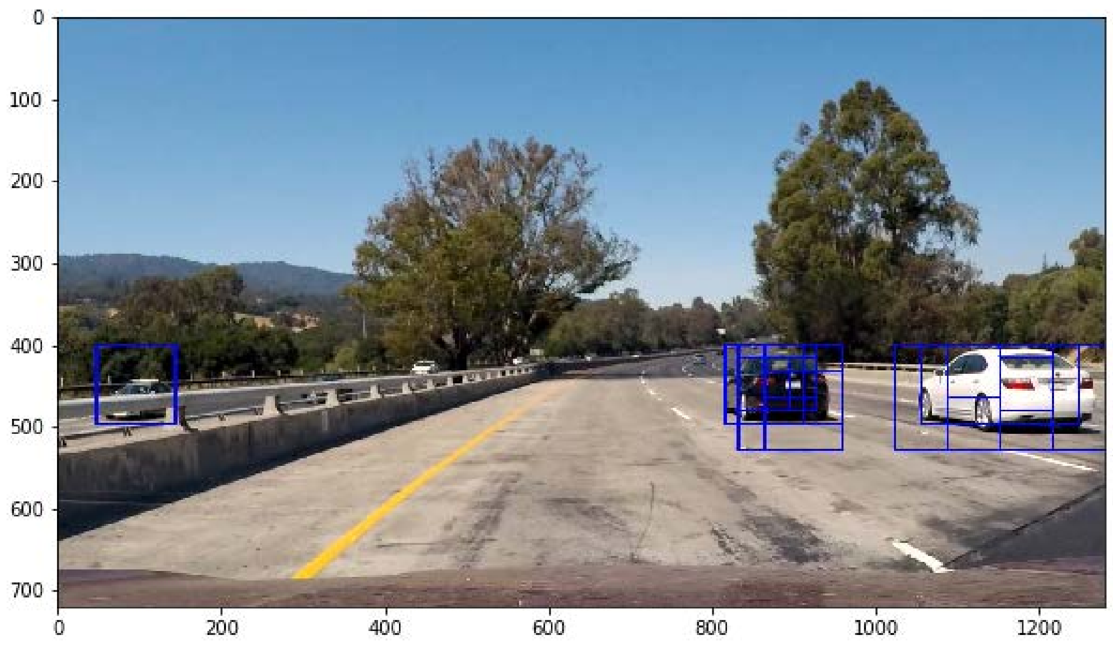
This pipeline consists of the “process_frame” function, which looks like this:

```
def process_frame(img):
    rectangles = []

    colorspace = 'YUV'  # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
    orient = 11
    pix_per_cell = 16
    cell_per_block = 2
    hog_channel = 'ALL'  # Can be 0, 1, 2, or "ALL"

    ystart = 400
    ystop = 464
    scale = 1.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 416
    ystop = 480
    scale = 1.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 400
    ystop = 496
    scale = 1.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 432
    ystop = 528
    scale = 1.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 400
    ystop = 528
    scale = 2.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 432
    ystop = 560
    scale = 2.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 400
    ystop = 596
    scale = 3.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 464
    ystop = 660
    scale = 3.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))

    rectangles = [item for sublist in rectangles for item in sublist]

    heatmap_img = np.zeros_like(img[:, :, 0])
    heatmap_img = add_heat(heatmap_img, rectangles)
    heatmap_img = apply_threshold(heatmap_img, 1)
    labels = label(heatmap_img)
    draw_img, rects = draw_labeled_bboxes(np.copy(img), labels)
    return draw_img
```

This function provides a various sliding window search. Due to the size and position of cars in the image will be different depending on their distance from the camera, find_cars will have to be called a few times with different ystart, ystop, and scale values. For each searching window the function “find_cars” is called. This function provides a HOG Sub-sampling window search and gives back the rectangles for each car. The function consinsts of the following code:

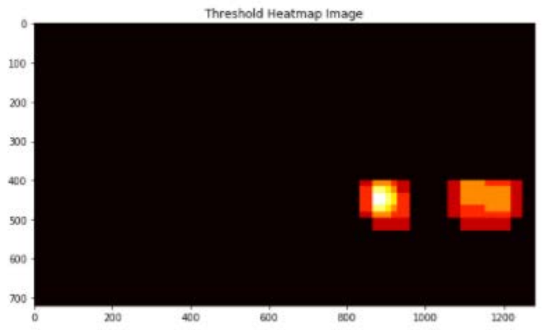
```
# Define a single function that can extract features using hog sub-sampling
and make predictions
def find_cars(img, ystart, ystop, scale, cspace, hog_channel, svc,
X_scaler, orient,
              pix_per_cell, cell_per_block, spatial_size, hist_bins,
show_all_rectangles=False):
    # array of rectangles where cars were detected
    rectangles = []

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

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

    # apply color conversion if other than 'RGB'
    if cspace != 'RGB':
        if cspace == 'HSV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2HSV)
        elif cspace == 'LUV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2LUV)
        elif cspace == 'HLS':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2HLS)
        elif cspace == 'YUV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2YUV)
        elif cspace == 'YCrCb':
            ctrans_tosearch = cv2.cvtColor(img_tosearch,
cv2.COLOR_RGB2YCrCb)
    else:
        ctrans_tosearch = np.copy(image)

    # rescale image if other than 1.0 scale
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1] /
scale), np.int(imshape[0] / scale)))

    # select colorspace channel for HOG
    if hog_channel == 'ALL':
        ch1 = ctrans_tosearch[:, :, 0]
        ch2 = ctrans_tosearch[:, :, 1]
        ch3 = ctrans_tosearch[:, :, 2]
    else:
        ch1 = ctrans_tosearch[:, :, hog_channel]

    # Define blocks and steps as above
    nxblocks = (ch1.shape[1] // pix_per_cell) + 1  # -1
    nyblocks = (ch1.shape[0] // pix_per_cell) + 1  # -1
    nfeat_per_block = orient * cell_per_block ** 2
    # 64 was the orginal sampling rate, with 8 cells and 8 pix per cell
    window = 64
    nblocks_per_window = (window // pix_per_cell) - 1
    cells_per_step = 2  # Instead of overlap, define how many cells to step
    nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
    nysteps = (nyblocks - nblocks_per_window) // cells_per_step

    # Compute individual channel HOG features for the entire image
    hog1 = get_hog_features(ch1, orient, pix_per_cell, cell_per_block,
feature_vec=False)
    if hog_channel == 'ALL':
        hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block,
feature_vec=False)
        hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block,
feature_vec=False)

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

            xleft = xpos * pix_per_cell
            ytop = ypos * pix_per_cell

            ################ ONLY FOR BIN_SPATIAL AND COLOR_HIST
################

            # Extract the image patch
            # subimg = cv2.resize(ctrans_tosearch[ytop:ytop+window,
xleft:xleft+window], (64,64))

            # 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_features,
hog_features)).reshape(1, -1))
            # test_features = X_scaler.transform(np.hstack((shape_feat,
hist_feat)).reshape(1, -1))
            # test_prediction = svc.predict(test_features)


######################################################################

            test_prediction = svc.predict(hog_features)

            if test_prediction == 1 or show_all_rectangles:
                xbox_left = np.int(xleft * scale)
                ytop_draw = np.int(ytop * scale)
                win_draw = np.int(window * scale)
                rectangles.append(
                    ((xbox_left, ytop_draw + ystart), (xbox_left +
win_draw, ytop_draw + win_draw + ystart)))

    return rectangles
```

After the rectangles are found, the picture looks like this:

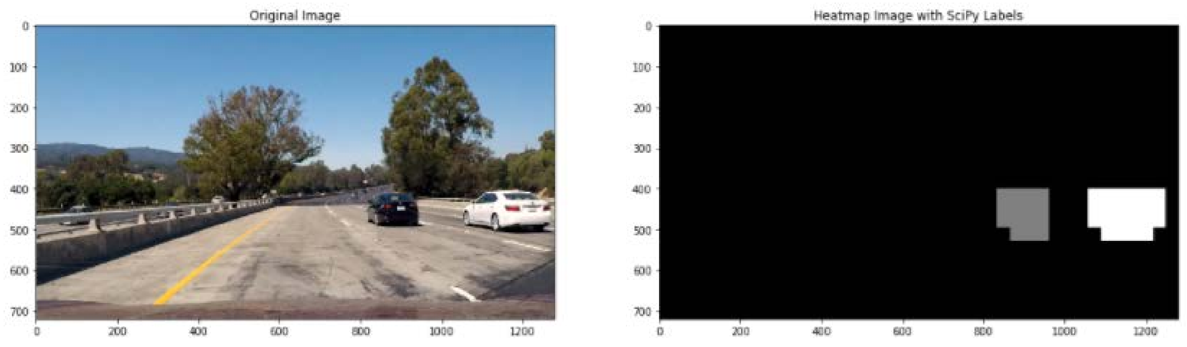


It can be seen that there are overlapping detections exists. In addition there is a false detection on the left side of the picture on the other side of the road. The remove the false positives, the following functions are applied to the picture:

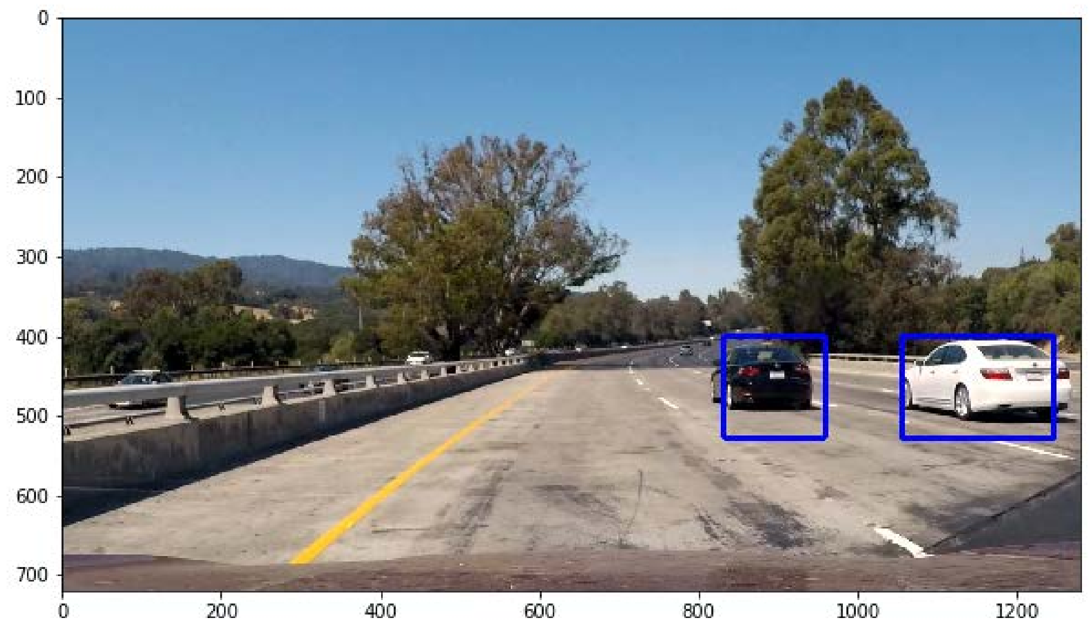
1. Incorporate heatmap
2. Apply threshold to heatmap



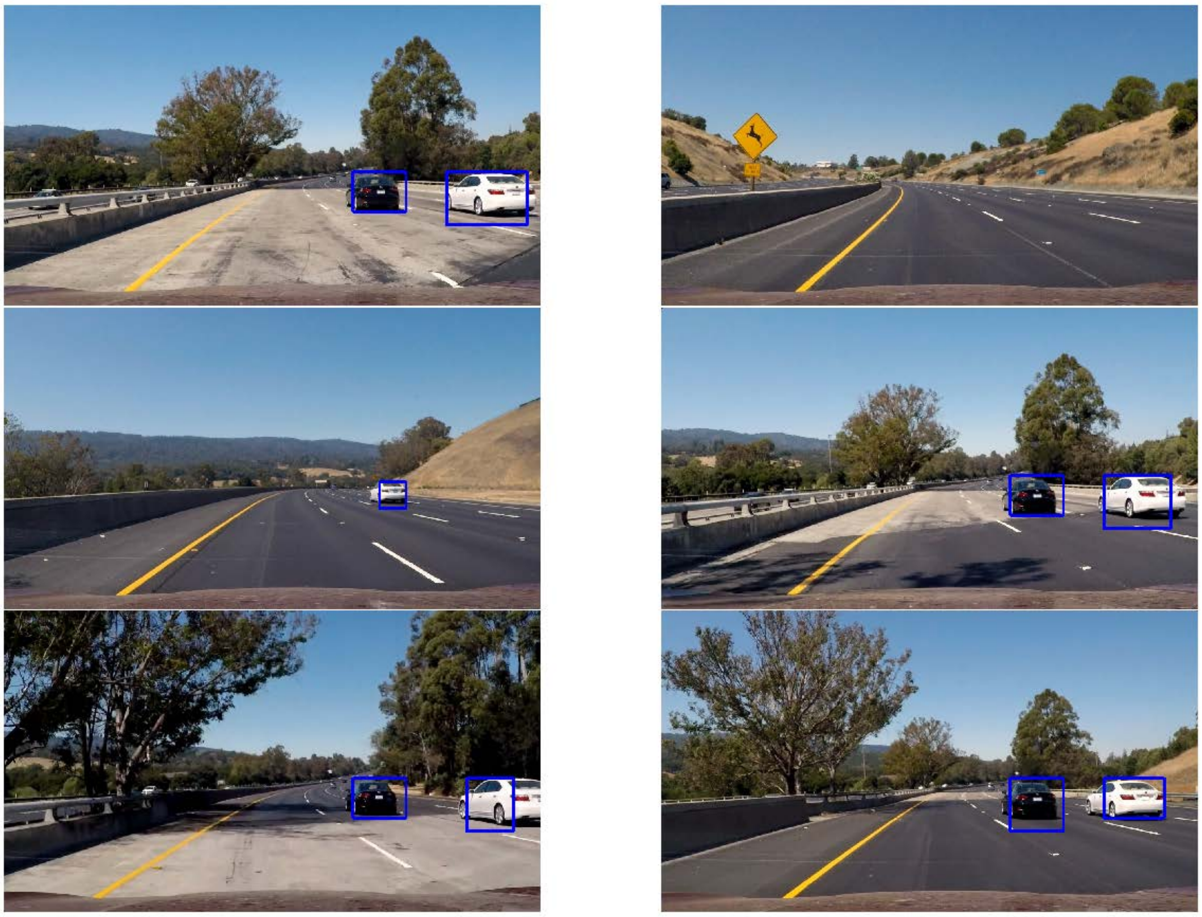
3. Apply SciPy Labels to heatmap



4. Apply bounding boxes around the labeled regions



This pipeline for processing an test image is now applied to all images and is shown below:



5. **Pipeline on a video Stream**

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.)

To make a smooth video a similar pipeline as before is used but now we'll also consider previous frames when calculating next localion of frame.

In order to do this the code provides the folliwng

1. Define a class to store data from vehicle detections
2. Create a pipeline that processes images by also adding previous frame detections to the history

```
###########################################################################
#######
###############      Video Piepline
##################
###########################################################################
#######

# Define a class to store data from video
class Vehicle_Detect():
    def __init__(self):
        # history of rectangles previous n frames
        self.prev_rects = []
    def add_rects(self, rects):
        self.prev_rects.append(rects)
        if len(self.prev_rects) > 15:
            # throw out oldest rectangle set(s)
            self.prev_rects = self.prev_rects[len(self.prev_rects) - 15:]


def process_frame_for_video(img):
    rectangles = []

    colorspace = 'YUV'  # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
    orient = 11
    pix_per_cell = 16
    cell_per_block = 2
    hog_channel = 'ALL'  # Can be 0, 1, 2, or "ALL"

    ystart = 400
    ystop = 464
    scale = 1.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 416
    ystop = 480
    scale = 1.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 400
    ystop = 496
    scale = 1.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 432
    ystop = 528
    scale = 1.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 400
    ystop = 528
    scale = 2.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 432
    ystop = 560
    scale = 2.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace,
hog_channel, svc, None,
                                orient, pix_per_cell, cell_per_block, None,
None))
    ystart = 400
    ystop = 596
    scale = 3.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, 
hog_channel, svc, None, 
                                orient, pix_per_cell, cell_per_block, None, 
None))
    ystart = 464
    ystop = 660
    scale = 3.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, 
hog_channel, svc, None, 
                                orient, pix_per_cell, cell_per_block, None, 
None))

    rectangles = [item for sublist in rectangles for item in sublist] 

    # add detections to the history
    if len(rectangles) > 0:
        det.add_rects(rectangles)

    heatmap_img = np.zeros_like(img[:, :, 0])
    for rect_set in det.prev_rects:
        heatmap_img = add_heat(heatmap_img, rect_set)
    heatmap_img = apply_threshold(heatmap_img, 1 + len(det.prev_rects) // 
2)

    labels = label(heatmap_img)
    draw_img, rect = draw_labeled_bboxes(np.copy(img), labels)
    return draw_img

det = Vehicle_Detect()

test_out_file2 = 'project_video_out.mp4'
clip_test2 = VideoFileClip('project_video.mp4')
clip_test_out2 = clip_test2.fl_image(process_frame_for_video)
clip_test_out2.write_videofile(test_out_file2, audio=False)
```

The result contains a handful of false positives. The rectangles that identify vehicles also tend to be a bit jittery. Frame-to-frame 'smoothing' of the heatmaps was attempted, but abandoned due to time constraints. To assist with this a diagnostic view was developed, showing detection results at the top. Along the bottom are shown the:

- current frame's heatmap
- thresholded version of the heatmap
- heatmap for the current and previoud frame's heatmap
- tresholded version of the heatmap

The link for the project video is provided here: https://youtu.be/4LcXEkC3xwY

6. Discussion

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

1. Paramter Tuning: it was hard for me to get the right parameters for the HOG extraction and training the classifier. It needs a lot of know-how or experience to get the right paramters very quickly.
2. Real Time Processing: To process my video it takes quite a while with this pipeline. Right now, I have no idea how to make it in real time or to apply it to a real car.
3. Additional feature extraction is possible.
4. More Data is needed: Although the classifier is good, I think 8000 pictures are not enough.
5. Improve generalization
6. Another Approach: Training a neuronal network for classifying the cars.