P5: Vehicle Detection and Tracking

```
import os
if (not os.path.exists('./P2_Vehicle_Detection.ipynb')):
  !git clone https://github.com/ahmed192a/ImageProcessing-PatternRecognition
  %cd ImageProcessing-PatternRecognition/P2.Vehicle-Detection
```

Imports

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import pickle
import cv2
import glob
import time
from random import shuffle
from scipy.ndimage.measurements import label
from skimage.feature import hog
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from moviepy.editor import VideoFileClip
```

Features Extraction Algorithms

```
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)
    if conv == 'BGR2LUV':
        return cv2.cvtColor(img, cv2.COLOR_BGR2LUV)
    if conv == 'RGB2HSV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    if conv == 'BGR2HSV':
        return cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    if conv == 'RGB2HLS':
        return cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    if conv == 'BGR2HLS':
        return cv2.cvtColor(img, cv2.COLOR_BGR2HLS)
    if conv == 'RGB2YUV':
```

```
        return cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    if conv == 'BGR2YUV':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
    if conv == 'RGB2BGR':
        return cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    if conv == 'BGR2RGB':
        return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

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=True,
                                  visualize=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,
                       visualize=vis, feature_vector=feature_vec)
        return features
```

Find Car utils

```
## Heat-map functions
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):
    box_list =[]
    # 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(no
            box_list.append(bbox)
            # Draw the box on the image
            cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)
        # Return the image
        return img, box_list
```

Find Car

```
# Define a single function that can extract features using hog sub-sampling and mal
def find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell, cell_
    img = img.astype(np.float32)/255
    heat_map = np.zeros_like(img[:,:,0], dtype=np.float32)
    draw_img = np.copy(img)

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

    ch1 = ctrans_tosearch[:,:,0]
    ch2 = ctrans_tosearch[:,:,1]
    ch3 = ctrans_tosearch[:,:,2]

    # Define blocks and steps as above
    nxcells = (ch1.shape[1] // pix_per_cell) - 1
    nycells = (ch1.shape[0] // pix_per_cell) - 1
    # nxblocks = nxcells // cell_per_block
    # nyblocks = nycells // cell_per_block
    # 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 // cell_per_block)-1
    ncells_per_window = (window // pix_per_cell) - 1
    # cells_per_step = cells_per_step #int(0.5 * ncells_per_window)  # Instead of (
    nxsteps = (nxcells - ncells_per_window) // cells_per_step
    nysteps = (nycells - ncells_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=
    hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, feature_vec=
    hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, feature_vec=

    boxes_list = []
```

```
    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+ncells_per_window, xpos:xpos+ncells_per_win
            hog_feat2 = hog2[ypos:ypos+ncells_per_window, xpos:xpos+ncells_per_win
            hog_feat3 = hog3[ypos:ypos+ncells_per_window, xpos:xpos+ncells_per_win

            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+windo

            # Scale features and make a prediction
            test_features = X_scaler.transform(hog_features.reshape(1, -1))

            test_prediction  = svc.predict(test_features)

            if test_prediction == 1:
                xbox_left = int(xleft*scale)
                ytop_draw = int(ytop*scale)
                win_draw = int(window*scale)
                cv2.rectangle(draw_img,(xbox_left, ytop_draw+ystart),(xbox_left+wi

                boxes_list.append(((xbox_left, ytop_draw+ystart),(xbox_left+win_dra

    heat_map = add_heat(heat_map, boxes_list)
    return draw_img,boxes_list, heat_map
```

Training utils

```
# Define a function to extract features from a list of images
# Have this function call bin_spatial(), color_hist() and get_hog_features()
def extract_features(imgs, cspace='RGB', orient=8, pix_per_cell=8, cell_per_block=2
    # 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 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)


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

Main Training

```
colorspace = 'YCrCb'        # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 10                 # 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"
y_start_stop = [400, 656]   # Min and max in y to search in slide_window()
scales = [1.0,1.5,2.0]      # Scale the image
cells_per_step = 2          # How many cells to step per sliding window
# Divide up into cars and notcars
# Read in car and non-car images
if os.path.exists("training_Model.pkl"):
  print()
  print('Found Previous Model')
  model_pickle = pickle.load(open('training_Model.pkl', 'rb'))
  svc = model_pickle['svc']
  X_scaler = model_pickle['scaler']
  orient = model_pickle['orient']
  pix_per_cell = model_pickle['pix_per_cell']
  cell_per_block = model_pickle['cell_per_block']
  colorspace = model_pickle['colorspace']
else:
  # Divide up into cars and notcars
  # Read in car and non-car images
  cars = glob.glob('training_data/vehicles/**/*.png',recursive=True)
  notcars = glob.glob('training_data/non-vehicles/**/*.png',recursive=True)
  print("There are " + str(len(cars)) + " cars images in the training dataset")
```

```
print('There are ' + str(len(cars)) + ' cars images in the training dataset')
print("There are " + str(len(notcars)) + " not-cars images in the training datas

t=time.time()
car_features = extract_features(
                    cars, cspace=colorspace,
                    orient=orient, pix_per_cell=pix_per_cell,
                    cell_per_block=cell_per_block,
                    hog_channel=hog_channel)
notcar_features = extract_features(
                    notcars, 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)
# 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))))


# 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)
# 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...')
# 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')

model_pickle = {}
model_pickle['svc'] = svc
model_pickle['scaler'] = X_scaler
model_pickle['orient'] = orient
model_pickle['pix_per_cell'] = pix_per_cell
model_pickle['cell_per_block'] = cell_per_block
model_pickle['colorspace'] = colorspace
pickle.dump( model_pickle, open( "training_Model.pkl", "wb" ))
```

```
    pickle.dump( model_pickle, open( "training_model.pkl", "wb" ))
print('Configuration:')
print('-----------------------------------------------')
print(' Color space:             ', colorspace)
print(' HOG orientations:        ', orient)
print(' HOG pixel per cell:      ', pix_per_cell)
print(' HOG cells per block:     ', cell_per_block)
print(' HOG channel:             ', hog_channel)
print()
print('Train the classifier...', end='', flush=True)
print('Done')
```

```
Found Previous Model
Configuration:
-----------------------------------------------
 Color space:             YCrCb
 HOG orientations:        10
 HOG pixel per cell:      8
 HOG cells per block:     2
 HOG channel:             ALL

Train the classifier...Done
```

Image Pipeline

```
def prepare_out_blend_frame(blend_on_road, img_binary):
    """
    Prepare the final pretty pretty output blend, given all intermediate pipeline i
    :param blend_on_road: color image of lane blend onto the road
    :param img_binary: thresholded binary image
    """
    h, w = blend_on_road.shape[:2]

    thumb_ratio = 0.2
    thumb_h, thumb_w = int(thumb_ratio * h), int(thumb_ratio * w)

    off_x, off_y = 20, 15

    # add a gray rectangle to highlight the upper area
    mask = blend_on_road.copy()
    mask = cv2.rectangle(mask, pt1=(w-(thumb_w+off_x*2), 0), pt2=(w, h), color=(0,
    blend_on_road = cv2.addWeighted(src1=mask, alpha=0.2, src2=blend_on_road, beta=

    # add thumbnail of binary image
    thumb_binary = cv2.resize(img_binary, dsize=(thumb_w, thumb_h))
    thumb_binary = np.dstack([thumb_binary, thumb_binary, thumb_binary])
    blend_on_road[off_y:thumb_h+off_y, w-(thumb_w+off_x):w-off_x, :] = thumb_binary

    return blend_on_road
```

Image Pipeline

```
ystart = y_start_stop[0]
ystop = y_start_stop[1]
def process_image(image):
    result = image_pipeline(image, "None")
    return result
# This function processes each individual image coming from the video stream
# and estimates where the cars are
def image_pipeline(img, fname):

    global heat_previous, first_frame #, boxes_previous, labels_prev
    heat = np.zeros_like(img[:,:,0])
    if(first_frame == True):
        heat_previous =  np.zeros_like(img[:,:,0]).astype(np.float)
        boxes_previous =  np.zeros_like(img[:,:,0]).astype(np.float)
        first_frame = False
    # for scale in scales:
    for scale in [1.5]:
      out_img, boxes_list_1, heat_1 = find_cars(img, y_start_stop[0] ,
                                        y_start_stop[1], scale,
                                        svc, X_scaler, orient,
                                        pix_per_cell, cell_per_block,
                                        cells_per_step, "RGB2YCrCb")

      heat = np.add(heat, heat_1)
      # plt.imshow(out_img)
      # plt.show()


    # Apply threshold to help remove false positives
    if fname == "None" :
        heat_previous = heat_previous*0.6
        heat_previous = np.add(heat_previous,heat)
        heat = apply_threshold(heat_previous,3)
    elif fname != "None":
        heat = apply_threshold(heat,2)

    heat = np.clip(heat, 0, 1)
    labels = label(heat)
    heat_img, bbox_list = draw_labeled_bboxes(np.copy(img), labels)

    return prepare_out_blend_frame(heat_img, heat * 255)
```

Test output

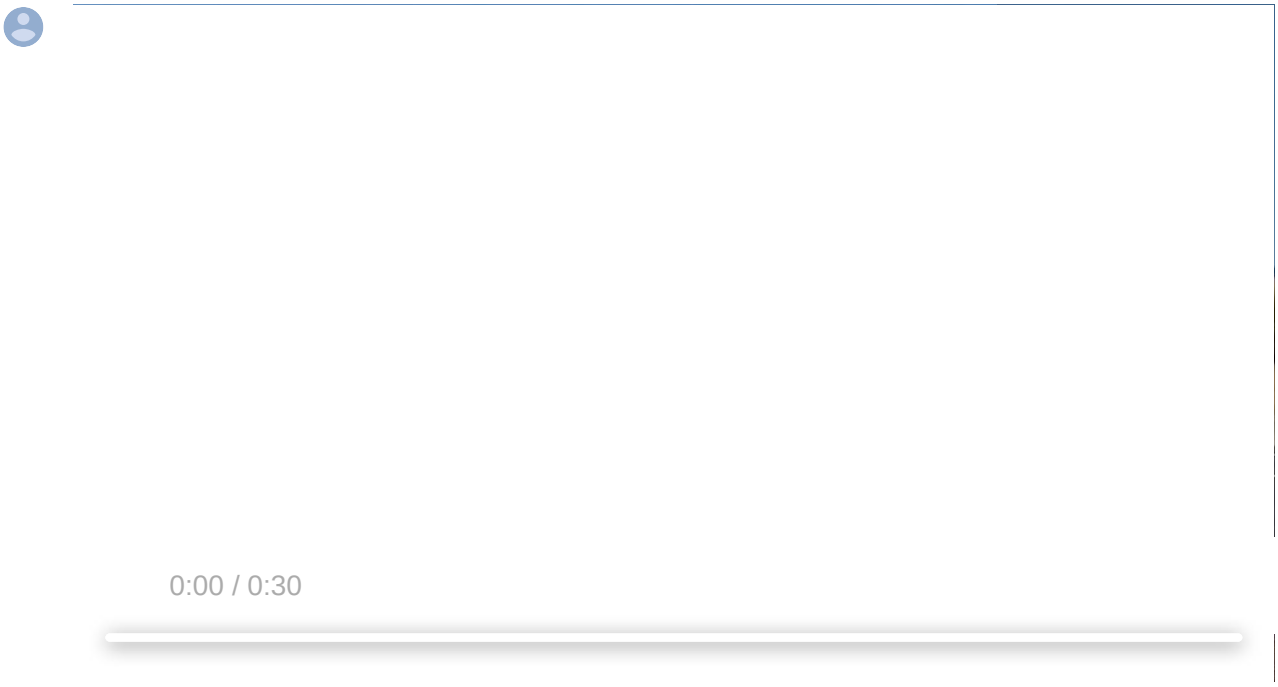
```
project_output = 'OUT_project_video_output.mp4'
clip2 = VideoFileClip("project_video.mp4").subclip(0,30)
first_frame = True
project_clip = clip2.fl_image(process_image)
%time project_clip.write_videofile(project_output, audio=False)
```

```
[MoviePy] >>>> Building video OUT_project_video_output.mp4
[MoviePy] Writing video OUT_project_video_output.mp4
100%|█████████| 750/751 [05:21<00:00,  2.33it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: OUT_project_video_output.mp4

CPU times: user 5min 2s, sys: 10.8 s, total: 5min 13s
Wall time: 5min 24s
```

```
from IPython.display import HTML
from base64 import b64encode

def show_video(video_path, video_width = 600):
  video_file = open(video_path, "r+b").read()
  video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
  return HTML(f"""<video width={video_width} controls><source src="{video_url}"></
show_video("OUT_project_video_output.mp4")
```



✓ 14s completed at 6:34 PM