Vehicle Detection

*Udacity Term - 1*

*Period October to January*

## System & Software Specification

OS - Windows 7

Hardware: Intel  i7 core CPU

Programming language: Python 3.x

 Python Libraries used:

* OpenCV:  library name "cv2" . Used for image processing
* Numpy: Array related functionality
* Matplotlib: used for plotting images
* Math: Used for finding square roots
* Glob: To load images from folder
* moviepy.editor: For loading and creating a video
* pickle: To save and load training feature
* deque: library to create and use double ended que
* scipy: Library for labelling
* skimage: A library for getting the Hog features

## Description

Objective: Goal of the project is to detect the cars which are seen on the right side lanes. The cars must be tracked throughout the video. A bounding box has to be drawn around the car detected.

The Process involves two stages.

* An offline process, to Train a model to detect the cars
* An online process, wherein frames in a video has to be read and processed to detect the lanes and the cars on the right lane.

## Training Process

Data Description: The images from Udacity are used for training. These image dataset are derived from below sources

<http://www.gti.ssr.upm.es/data/Vehicle_database.html>

<http://www.cvlibs.net/datasets/kitti/>

Format: PNG

Dimension: 64 X 64

Folders: Vehicles & Non-Vehicles

The flow of the training process is as shown below

### Extracting Features

The Raw frame is converted to YCrCb. OpenCV api is used for converting the image

*cv2.cvtColor(image, cv2.COLOR\_RGB2YCrCb)*

The spatial feature is obtained by resizing the image to 32X32 and converting it to a single dimensional vector.

*cv2.resize(img[:, :, 0], size).ravel()*

The feature vector of all the three channels is stacked

*np.hstack((color1, color2, color3))*

The histogram feature of all the three channels are calculated and concatenated . The number of bins chosen is 32

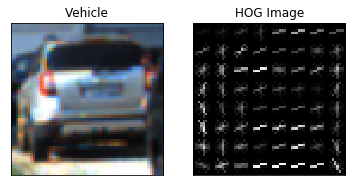
*np.histogram(img[:, :, 0], bins=nbins)*

The Hog features are calculated using skimage library

The parameters for HOG are as below

* orient=9
* pix\_per\_cell=8
* cell\_per\_block=2

Few sample vehicle and its hog image is as shown below





All the above feature list are concatenated and appended to the feature list.

### 3.2 Labelling and Feature Normalization

A vector the labels are created using the features obtained.

*y = np.hstack((np.ones(len(car\_features)), np.zeros(len(notcar\_features))))*

The data is normalized

*# Fit a per-column scaler*

*X\_scaler = StandardScaler().fit(X)*

*# Apply the scaler to X*

*scaled\_X = X\_scaler.transform(X)*

### 3.3 Split to test and training data

The data is split into training and test set using the sklearn API

*# Split up data into randomized training and test set*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(scaled\_X, y, test\_size=0.2, stratify =y)*

### 3.4 Linear Support Vector Machine

A linear SVC model is fit to the training data

*svc = LinearSVC()*

*svc.fit(X\_train, y\_train)*

### 3.5 Testing & Sample Output

*# Check the score of the SVC*

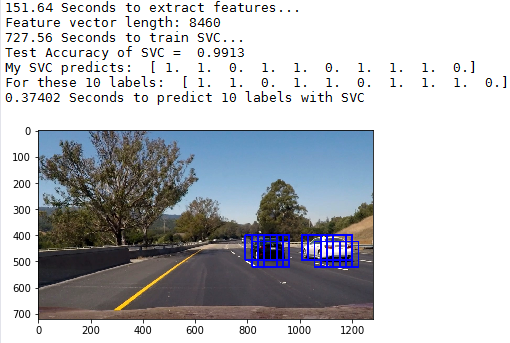
*svc.score(X\_test, y\_test), 4)*

*# Check the prediction time for a single sample*

*n\_predict = 10*

*svc.predict(X\_test[0:n\_predict])*

The timing and the sample output is as shown below



### 3.6 Saving the model to disk

*data={*

*'svc': svc,*

*'X\_scaler': X\_scaler,*

*'color\_space': color\_space,*

*'orient': orient,*

*'pix\_per\_cell': pix\_per\_cell,*

*'cell\_per\_block': cell\_per\_block,*

*'spatial\_size' : spatial\_size,*

*'hist\_bins': hist\_bins,*

*'hog\_channel': hog\_channel*

*}*

*with open('model-params.pk', 'wb') as pFile:pickle.dump(data, pFile)*

## Vehicle Detection

The work flow of the vehicle detection is as shown below

### 4.1 Loading Model

The linear SVC model saved during training is loaded and the model parameters are loaded.

*with open('model-params.pk', 'rb') as pfile:*

*pickle\_data = pickle.load(pfile)*

*# for key in pickle\_data:*

*svc = pickle\_data['svc']*

*X\_scaler = pickle\_data['X\_scaler']*

*color\_space = pickle\_data['color\_space']*

*orient = pickle\_data['orient']*

*pix\_per\_cell = pickle\_data['pix\_per\_cell']*

*cell\_per\_block = pickle\_data['cell\_per\_block']*

*spatial\_size = pickle\_data['spatial\_size']*

*hist\_bins = pickle\_data['hist\_bins']*

*hog\_channel = pickle\_data['hog\_channel']*

*del pickle\_data*

### 4.2 Loading Frames

The video file is loaded and frames are extracted as below

*undist\_output = 'project\_video\_out.mp4'*

*clip2 = VideoFileClip('project\_video.mp4')*

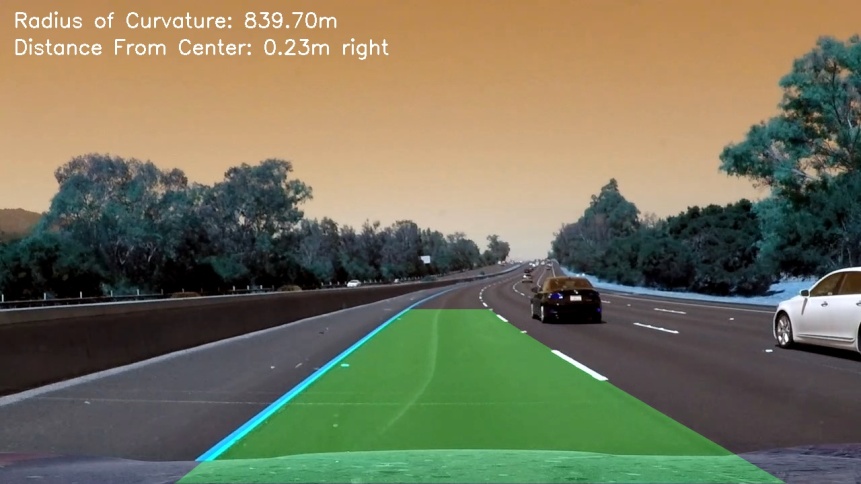
*yellow\_clip = clip2.fl\_image(processFrame, apply\_to=[])*

*yellow\_clip.write\_videofile(undist\_output, audio=False)*

### 4.3 Lane Detection

The lane is detected using the previous project pipeline. Once the lanes are detected, the frame is passed on to the vehicle detection pipeline.

LaneResult = processFrame(frame)



withCarsResult = find\_cars(LaneResult)

### 4.4 Sliding Window

A sliding window method is used for searching the cars in the lane detected frame. Three scales are used to detect the cars are different distances.

The three scaled regions are as below

Zone 1: ystart\_ystop\_scale (380, 480, 1)

Zone 2: ystart\_ystop\_scale (400, 600, 1.5)

Zone 3: ystart\_ystop\_scale (500, 700, 2.5)

In each zone the image is resized based on the scale

*search\_region = cv2.resize(search\_region, (np.int(imshape[1] / scale), np.int(imshape[0] / scale)))*

The above search region is divided as below

*pix\_per\_cell=8*

*cell\_per\_block=2*

The frame is converted to YCrCb format and the HOG features are extracted for each channel

*hog1 = get\_hog\_features(ch1, orient, pix\_per\_cell, cell\_per\_block, feature\_vec=False)*

*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)*

A region based on cell size is extracted for the search reagion and the hog features are extracted, the spatial and histogram features are extracted for these cells

*hog\_feat1 = hog1[ypos:ypos + nblocks\_per\_window, xpos:xpos + nblocks\_per\_window].ravel()*

*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))*

*spatial\_features = bin\_spatial(subimg, size=spatial\_size)*

*hist\_features = color\_hist(subimg, nbins=hist\_bins)*

The features are stacked, scalled and predition is done using the model loaded earlier

*# Scale features and make a prediction*

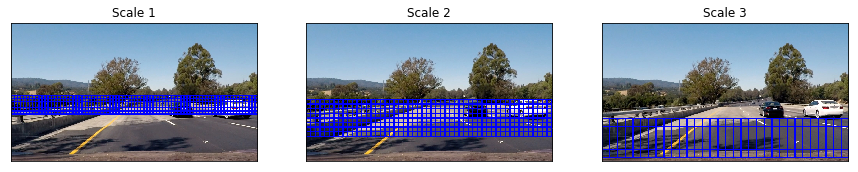
*test\_features = X\_scaler.transform(*

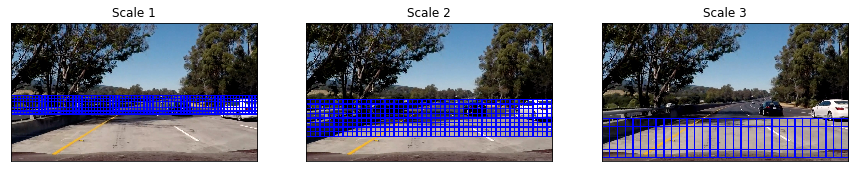
*np.hstack((spatial\_features, hist\_features, hog\_features)).reshape(1, -1))*

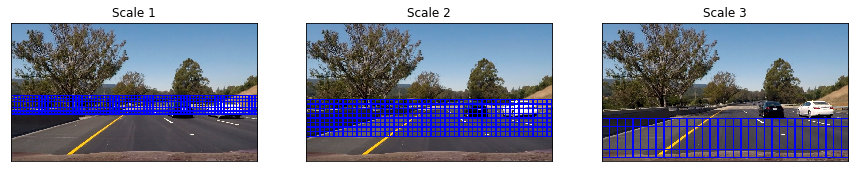
*# test\_features = X\_scaler.transform(np.hstack((shape\_feat, hist\_feat)).reshape(1, -1))*

*test\_prediction = svc.predict(test\_features)*

Some images of the sliding window output is as shown below for the 3 scales







For the detected regions rectangular boxes are drawn,

*if test\_prediction == 1:*

*xbox\_left = np.int(xleft \* scale)*

*ytop\_draw = np.int(ytop \* scale)*

*win\_draw = np.int(window \* scale)*

*cv2.rectangle(draw\_img, (xbox\_left, ytop\_draw + ystart), (xbox\_left + win\_draw, ytop\_draw + win\_draw + ystart), (0, 0, 255), 6*)

Some images of sample detection with +Ve preditions are shown below







### 4.5 Removal of false detection and multiple detections

We see above case that there are multiple detections and many times there are false detection.

In order to avoid these false detections and multiple detections, heat maps are generated and based on threshold the false detections are removed.

*heatmap\_temp = np.zeros\_like(raw\_frame[:, :, 0]).astype(np.float)*

*for box in bbox\_list:*

*# Add += 1 for all pixels inside each bbox*

*# Assuming each "box" takes the form ((x1, y1), (x2, y2))*

*heatmap\_temp[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1*

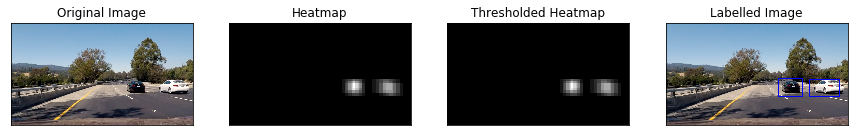
*heat\_images.append(heatmap\_temp)*

*heatmap = np.sum(np.array(heat\_images), axis=0)*

*heatmap[heatmap <= threshold] = 0*

*return heatmap, heat\_images*

The output of sample heat map and removal of multiple detections is shown below



Finally the labels of the selected heat maps is generated and rectangular bounding boxes are drawn around the detected region

*# Find final boxes from heatmap using label function*

*labels = label(heatmap)*

*draw\_labeled\_boxes(raw\_frame, labels)*

## Improvement

The pipeline is not tested against any environmental conditions. The model has to be re-trained to cover these, as well night conditions are not handled.