**Project Code**

**#Loading the libraries**

from helper\_functions import get\_hog\_features, bin\_spatial, color\_hist, slide\_window, draw\_boxes, convert\_image

import cv2

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import os

import random

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import time

from sklearn.svm import LinearSVC

%matplotlib inline

**#Showing images**

def show\_images(image1, title1, image2, title2):

   f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

   f.tight\_layout()

   ax1.imshow(image1)

   ax1.set\_title(title1, fontsize=50)

   ax2.imshow(image2)

   ax2.set\_title(title2, fontsize=50)

   plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

def get\_dataset(rootdir):

   images = []

   for subdir, dirs, files in os.walk(rootdir):

       for file in files:

           if '.DS\_Store' not in file:

               images.append(os.path.join(subdir, file))

   return list(map(lambda img: cv2.cvtColor(cv2.imread(img), cv2.COLOR\_BGR2RGB), images))

rootdir = './dataset/vehicles'

vehicles = get\_dataset(rootdir)

print(len(vehicles))

rootdir = './dataset/non-vehicles'

non\_vehicles = get\_dataset(rootdir)

print(len(non\_vehicles))

image1 = random.choice(vehicles)

image2 = random.choice(non\_vehicles)

show\_images(image1, "Vehicle", image2, "Non-vehicle")

print(len(non\_vehicles))

**#Extracting features**

class Parameters():

   color\_space='RGB'

   spatial\_size=(32, 32)

   hist\_bins=8

   orient=9

   pix\_per\_cell=8

   cell\_per\_block=2

   hog\_channel=0

   hist\_range = (0, 256)

   spatial\_feat=True

   hist\_feat=True

   hog\_feat=True

   def \_\_init\_\_(self, color\_space='RGB', spatial\_size=(32, 32),

                hist\_bins=8, orient=9,

                pix\_per\_cell=8, cell\_per\_block=2, hog\_channel=0, scale = 1.5,hist\_range = (0, 256),

                spatial\_feat=True, hist\_feat=True, hog\_feat=True):

       # HOG parameters

       self.color\_space = color\_space

       self.spatial\_size = spatial\_size

       self.hist\_bins = hist\_bins

       self.orient = orient

       self.pix\_per\_cell = pix\_per\_cell

       self.cell\_per\_block = cell\_per\_block

       self.hog\_channel = hog\_channel

       self.scale = scale

       self.spatial\_feat = spatial\_feat

       self.hist\_feat = hist\_feat

       self.hog\_feat = hog\_feat

       self.hist\_range = hist\_range

def extract\_features(img, params):

       file\_features = []

       # Read in each one by one

       # apply color conversion if other than 'RGB'

       feature\_image = convert\_image(img, params.color\_space)

       if params.spatial\_feat == True:

           spatial\_features = bin\_spatial(feature\_image, size=params.spatial\_size)

           file\_features.append(spatial\_features)

       if params.hist\_feat == True:

           # Apply color\_hist()

           hist\_features = color\_hist(feature\_image, nbins=params.hist\_bins)

           file\_features.append(hist\_features)

       if params.hog\_feat == True:

       # Call get\_hog\_features() with vis=False, feature\_vec=True

           if params.hog\_channel == 'ALL':

               hog\_features = []

               for channel in range(feature\_image.shape[2]):

                   hog\_features.append(get\_hog\_features(feature\_image[:,:,channel],

                                       params.orient, params.pix\_per\_cell, params.cell\_per\_block,

                                       vis=False, feature\_vec=True))

               hog\_features = np.ravel(hog\_features)

           else:

               hog\_features = get\_hog\_features(feature\_image[:,:,params.hog\_channel], params.orient,

                           params.pix\_per\_cell, params.cell\_per\_block, vis=False, feature\_vec=True)

           # Append the new feature vector to the features list

           file\_features.append(hog\_features)

       # Return list of feature vectors

       return np.concatenate(file\_features)

params = Parameters(

           color\_space = 'YCrCb',

           spatial\_size = (16, 16),

           orient = 8,

           pix\_per\_cell = 8,

           cell\_per\_block = 2,

           hog\_channel = 'ALL',

           hist\_bins = 32,

           scale = 1.5,

           spatial\_feat=True,

           hist\_feat=True,

           hog\_feat=True

       )

vehicle = image1

non\_vehicle = image2

car\_spatial\_features = extract\_features(vehicle, params)

notcar\_spatial\_features = extract\_features(non\_vehicle, params)

plt.figure()

plt.subplot(121)

plt.plot((car\_spatial\_features))

plt.xlabel("Car Spatial Features",fontsize=15)

plt.subplot(122)

plt.plot(notcar\_spatial\_features)

plt.xlabel("Non-Car Spatial Features",fontsize=15)

**# HOG Features**

def show\_hog(vehicle, non\_vehicle, params):

   \_,vehicle\_hog = get\_hog\_features(vehicle[:,:,0],params.orient, params.pix\_per\_cell, params.cell\_per\_block, vis=True, feature\_vec=True)

   \_,non\_vehicle\_hog = get\_hog\_features(non\_vehicle[:,:,0],params.orient, params.pix\_per\_cell, params.cell\_per\_block, vis=True, feature\_vec=True)

   f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20,10))

   ax1.set\_title('Vehicle')

   ax1.imshow(vehicle, cmap='gray')

   ax2.set\_title('Vehicle hog')

   ax2.imshow(vehicle\_hog, cmap='gray')

   ax3.set\_title('Non-vehicle')

   ax3.imshow(non\_vehicle, cmap='gray')

   ax4.set\_title('Non-vehicle hog')

   ax4.imshow(non\_vehicle\_hog, cmap='gray')

show\_hog(vehicle, non\_vehicle, params)

**#Classifier(SVM)**

sample\_size = 8750

cars = vehicles[0:sample\_size]

notcars = non\_vehicles[0:sample\_size]

car\_features = list(map(lambda img: extract\_features(img, params), cars))

notcar\_features = list(map(lambda img: extract\_features(img, params), notcars))

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)

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

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:',params.orient,'orientations',params.pix\_per\_cell,

   'pixels per cell and', params.cell\_per\_block,'cells per block')

print('Feature vector length:', len(X\_train[0]))

svc = LinearSVC()

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))

**#Detecting cars**

def findCars(img, svc, scaler, params, y\_start\_stop=[400, 700], xy\_window=(64, 64), xy\_overlap=(0.85, 0.85) ):

     windows\_list = []

   windows = slide\_window(img, y\_start\_stop=y\_start\_stop, xy\_window=xy\_window, xy\_overlap=xy\_overlap)

   for window in windows:

       img\_window = cv2.resize(img[window[0][1]:window[1][1], window[0][0]:window[1][0]], (64, 64))

       features = extract\_features(img\_window, params)

       scaled\_features = scaler.transform(features.reshape(1, -1))

       pred = svc.predict(scaled\_features)

       if pred == 1:

           windows\_list.append(window)

   return windows\_list

**#Run test images**

draw\_images = []

for test\_image in glob.glob(‘./test\_images.jpg’):

   img = cv2.cvtColor(cv2.imread(test\_image), cv2.COLOR\_BGR2RGB)

   windows\_list = findCars(img, svc, X\_scaler, params)

   out\_img = draw\_boxes(img, windows\_list)

   draw\_images.append(out\_img)

f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20,10))

for i=1 to 200 :

axi.set\_title()

axi.imshow(draw\_images[i])

**#HeatMap code**

def add\_heat(heatmap, bbox\_list):

   for box in bbox\_list:

              heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1

   return heatmap# Iterate through list of bboxes

def apply\_threshold(heatmap, threshold):

   heatmap[heatmap <= threshold] = 0

       return heatmap

def draw\_labeled\_bboxes(img, labels):

    for car\_number in range(1, labels[1]+1):

       nonzero = (labels[0] == car\_number).nonzero()

       nonzeroy = np.array(nonzero[0])

       nonzerox = np.array(nonzero[1])

       bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))

       cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)

       return img

from scipy.ndimage.measurements import label

def heat\_threshold(img, threshold, svc, X\_scaler, windows\_list, params):

   heat = np.zeros\_like(img[:,:,0]).astype(np.float)

    heat = add\_heat(heat,windows\_list)

      heat = apply\_threshold(heat,threshold)

     heatmap = np.clip(heat, 0, 255)

      labels = label(heatmap)

   draw\_img = draw\_labeled\_bboxes(np.copy(img), labels)

   return draw\_img, heatmap

threshold = 4

img = cv2.cvtColor(cv2.imread(test\_images[0]), cv2.COLOR\_BGR2RGB)

windows\_list = findCars(img, svc, X\_scaler, params)

draw\_img, heatmap = heat\_threshold(img, threshold, svc, X\_scaler, windows\_list, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[2]), cv2.COLOR\_BGR2RGB)

windows\_list = findCars(img, svc, X\_scaler, params)

draw\_img, heatmap = heat\_threshold(img, threshold, svc, X\_scaler, windows\_list, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[3]), cv2.COLOR\_BGR2RGB)

windows\_list = findCars(img, svc, X\_scaler, params)

draw\_img, heatmap = heat\_threshold(img, threshold, svc, X\_scaler, windows\_list, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[4]), cv2.COLOR\_BGR2RGB)

windows\_list = findCars(img, svc, X\_scaler, params)

draw\_img, heatmap = heat\_threshold(img, threshold, svc, X\_scaler, windows\_list, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

**#HOG Sub sampling code**

def find\_cars\_hog\_sub(img, ystart, ystop, svc, scaler, params, cells\_per\_step = 1):

   draw\_img = np.copy(img)

   cspace = params.color\_space

   img\_tosearch = img[ystart:ystop,:,:]

   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(img\_tosearch)

   if params.scale != 1:

       imshape = ctrans\_tosearch.shape

       ctrans\_tosearch = cv2.resize(ctrans\_tosearch, (np.int(imshape[1]/params.scale), np.int(imshape[0]/params.scale)))

   ch1 = ctrans\_tosearch[:,:,0]

   ch2 = ctrans\_tosearch[:,:,1]

   ch3 = ctrans\_tosearch[:,:,2]

   # Define blocks and steps as above

   nxblocks = (ch1.shape[1] // params.pix\_per\_cell) - params.cell\_per\_block + 1

   nyblocks = (ch1.shape[0] // params.pix\_per\_cell) - params.cell\_per\_block + 1

   nfeat\_per\_block = params.orient\*params.cell\_per\_block\*\*2

   window = 64

   nblocks\_per\_window = (window // params.pix\_per\_cell) - params.cell\_per\_block + 1

   nxsteps = (nxblocks - nblocks\_per\_window) // cells\_per\_step

   nysteps = (nyblocks - nblocks\_per\_window) // cells\_per\_step

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

   hog2 = get\_hog\_features(ch2, params.orient, params.pix\_per\_cell, params.cell\_per\_block, feature\_vec=False)

   hog3 = get\_hog\_features(ch3, params.orient, params.pix\_per\_cell, params.cell\_per\_block, feature\_vec=False)

   car\_windows = []

   for xb in range(nxsteps):

       for yb in range(nysteps):

           ypos = yb\*cells\_per\_step

           xpos = xb\*cells\_per\_step

     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))

           xleft = xpos\*params.pix\_per\_cell

           ytop = ypos\*params.pix\_per\_cell

          subimg = cv2.resize(ctrans\_tosearch[ytop:ytop+window, xleft:xleft+window], (64,64))

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

 hist\_features = color\_hist(subimg, nbins=params.hist\_bins, bins\_range=params.hist\_range)

test\_features = X\_scaler.transform(np.hstack((spatial\_features, hist\_features, hog\_features)).reshape(1, -1))

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

           if test\_prediction == 1:

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

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

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

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

car\_windows.append(((xbox\_left, ytop\_draw+ystart),(xbox\_left+win\_draw,ytop\_draw+win\_draw+ystart)))

   return car\_windows

ystart = 350

ystop = 656

threshold = 1

img = cv2.cvtColor(cv2.imread(test\_images[0]), cv2.COLOR\_BGR2RGB)

car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[1]), cv2.COLOR\_BGR2RGB)

car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[2]), cv2.COLOR\_BGR2RGB)

car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[3]), cv2.COLOR\_BGR2RGB)

car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[4]), cv2.COLOR\_BGR2RGB)

car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

img = cv2.cvtColor(cv2.imread(test\_images[5]), cv2.COLOR\_BGR2RGB)

car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(draw\_img)

ax1.set\_title('Car Positions', fontsize=50)

ax2.imshow(heatmap, cmap='hot')

ax2.set\_title('Heat Map', fontsize=50)

plt.subplots\_adjust(left=0., right=1, top=0.9, bottom=0.)

**#pipeline video**

def pipeline(img):

   ystart = 350

   ystop = 656

   threshold = 1

   car\_windows = find\_cars\_hog\_sub(img, ystart, ystop, svc, X\_scaler, params)

   draw\_img, heat\_map = heat\_threshold(img, threshold, svc, X\_scaler, car\_windows, params)

   return draw\_img

from moviepy.editor import VideoFileClip

from IPython.display import HTML

def process\_image(img):

   return pipeline(img)

white\_output = 'output\_video/project\_video.mp4'

clip1 = VideoFileClip("project\_video.mp4")

white\_clip = clip1.fl\_image(process\_image)

%time white\_clip.write\_videofile(white\_output, audio=False)