**CODE**

%matplotlib inline

import os

import pandas as pd

import numpy as np

import cv2

import random

import time

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import seaborn as sns

from skimage import exposure

from skimage.feature import hog

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

from sklearn.svm import LinearSVC

from sklearn.preprocessing import StandardScaler

import tqdm.notebook as tqdm

import random

import glob

import functools

from moviepy.editor import VideoFileClip

from IPython.display import HTML

import tensorflow.keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D,ReLU

from tensorflow.keras.activations import relu

from tensorflow.keras.layers import Flatten, Dense, Activation, Reshape

def load\_weights(model,yolo\_weight\_file):

data = np.fromfile(yolo\_weight\_file,np.float32)

data=data[4:]

index = 0

for layer in model.layers:

shape = [w.shape for w in layer.get\_weights()]

if shape != []:

kshape,bshape = shape

bia = data[index:index+np.prod(bshape)].reshape(bshape)

index += np.prod(bshape)

ker = data[index:index+np.prod(kshape)].reshape(kshape)

index += np.prod(kshape)

layer.set\_weights([ker,bia])

class Box:

def \_\_init\_\_(self):

self.x, self.y = float(), float()

self.w, self.h = float(), float()

self.c = float()

self.prob = float()

def overlap(x1,w1,x2,w2):

l1 = x1 - w1 / 2.;

l2 = x2 - w2 / 2.;

left = max(l1, l2)

r1 = x1 + w1 / 2.;

r2 = x2 + w2 / 2.;

right = min(r1, r2)

return right - left

def box\_intersection(a, b):

w = overlap(a.x, a.w, b.x, b.w)

h = overlap(a.y, a.h, b.y, b.h)

if w < 0 or h < 0: return 0

area = w \* h

return area

def box\_union(a, b):

i = box\_intersection(a, b)

u = a.w \* a.h + b.w \* b.h - i

return u

def box\_iou(a, b):

return box\_intersection(a, b) / box\_union(a, b)

def yolo\_net\_out\_to\_car\_boxes(net\_out, threshold = 0.2, sqrt=1.8,C=20, B=2, S=7):

class\_num = 6

boxes = []

SS = S \* S

prob\_size = SS \* C

conf\_size = SS \* B

probs = net\_out[0 : prob\_size]

confs = net\_out[prob\_size : (prob\_size + conf\_size)]

cords = net\_out[(prob\_size + conf\_size) : ]

probs = probs.reshape([SS, C])

confs = confs.reshape([SS, B])

cords = cords.reshape([SS, B, 4])

for grid in range(SS):

for b in range(B):

bx = Box()

bx.c = confs[grid, b]

bx.x = (cords[grid, b, 0] + grid % S) / S

bx.y = (cords[grid, b, 1] + grid // S) / S

bx.w = cords[grid, b, 2] \*\* sqrt

bx.h = cords[grid, b, 3] \*\* sqrt

p = probs[grid, :] \* bx.c

if p[class\_num] >= threshold:

bx.prob = p[class\_num]

boxes.append(bx)

# combine boxes that are overlap

boxes.sort(key=lambda b:b.prob,reverse=True)

for i in range(len(boxes)):

boxi = boxes[i]

if boxi.prob == 0: continue

for j in range(i + 1, len(boxes)):

boxj = boxes[j]

if box\_iou(boxi, boxj) >= .4:

boxes[j].prob = 0.

boxes = [b for b in boxes if b.prob > 0.]

return boxes

def draw\_box(boxes,im,crop\_dim):

imgcv = im

[xmin,xmax] = crop\_dim[0]

[ymin,ymax] = crop\_dim[1]

for b in boxes:

h, w, \_ = imgcv.shape

left = int ((b.x - b.w/2.) \* w)

right = int ((b.x + b.w/2.) \* w)

top = int ((b.y - b.h/2.) \* h)

bot = int ((b.y + b.h/2.) \* h)

left = int(left\*(xmax-xmin)/w + xmin)

right = int(right\*(xmax-xmin)/w + xmin)

top = int(top\*(ymax-ymin)/h + ymin)

bot = int(bot\*(ymax-ymin)/h + ymin)

if left < 0 : left = 0

if right > w - 1: right = w - 1

if top < 0 : top = 0

if bot > h - 1: bot = h - 1

thick = int((h + w) // 150)

cv2.rectangle(imgcv, (left, top), (right, bot), (255,0,0), thick)

return imgcv

path = "test\_images/test6.jpg"

image = cv2.imread(path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

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

plt.imshow(image)

plt.axis("off")

random.seed(0)

vehicle = sorted(glob.glob('Dataset/vehicles/\*/\*.png'))

nonvehicle = sorted(glob.glob('Dataset/non-vehicles/\*/\*.png'))

random.shuffle(vehicle)

random.shuffle(nonvehicle)

\_, (axs1, axs2) = plt.subplots(2, 10, figsize=(20, 4))

for ax in axs1:

img = cv2.imread(random.choice(vehicle))[:, :, ::-1]

ax.imshow(img)

for ax in axs2:

img = cv2.imread(random.choice(nonvehicle))[:, :, ::-1]

ax.imshow(img)

plt.show()

plt.close()

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):

features = []

for file in imgs:

file\_features = []

image = mpimg.imread(file)

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:

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

file\_features.append(hist\_features)

if hog\_feat == 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)

file\_features.append(hog\_features)

features.append(np.concatenate(file\_features))

return features

def get\_hog\_features(img, orient, pix\_per\_cell, cell\_per\_block,

vis=False, feature\_vec=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,

visualize=vis, feature\_vector=feature\_vec)

return features, hog\_image

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,

visualize=vis, feature\_vector=feature\_vec)

return 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))

def color\_hist(img, nbins=32):

channel1\_hist = np.histogram(img[:,:,0], bins=nbins)

channel2\_hist = np.histogram(img[:,:,1], bins=nbins)

channel3\_hist = np.histogram(img[:,:,2], bins=nbins)

hist\_features = np.concatenate((channel1\_hist[0], channel2\_hist[0], channel3\_hist[0]))

return hist\_features

images = glob.glob('./Dataset/\*/\*/\*.png')

cars = []

notcars = []

all\_cars = []

all\_notcars = []

for image in images:

if 'non-vehicle' in image:

all\_notcars.append(image)

else:

all\_cars.append(image)

all\_cars

all\_notcars

for ix, car in enumerate(all\_cars):

if ix % 5 == 0:

cars.append(car)

for ix, notcar in enumerate(all\_notcars):

if ix % 5 == 0:

notcars.append(notcar)

car\_image = mpimg.imread(cars[5])

notcar\_image = mpimg.imread(notcars[0])

def compare\_images(image1, image2, image1\_exp="Image 1", image2\_exp="Image 2"):

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3))

f.tight\_layout()

ax1.imshow(image1)

ax1.set\_title(image1\_exp, fontsize=20)

ax2.imshow(image2)

ax2.set\_title(image2\_exp, fontsize=20)

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

compare\_images(car\_image, notcar\_image, "Car", "Not Car")

color\_space = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb

orient = 15 # 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 = (32, 32) # Spatial binning dimensions

hist\_bins = 32 # Number of histogram bins

spatial\_feat = True # Spatial features on or off

hist\_feat = True # Histogram features on or off

hog\_feat = True

converted\_car\_image = cv2.cvtColor(car\_image, cv2.COLOR\_RGB2YUV)

car\_ch1 = converted\_car\_image[:,:,0]

car\_ch2 = converted\_car\_image[:,:,1]

car\_ch3 = converted\_car\_image[:,:,2]

converted\_notcar\_image = cv2.cvtColor(notcar\_image, cv2.COLOR\_RGB2YUV)

notcar\_ch1 = converted\_notcar\_image[:,:,0]

notcar\_ch2 = converted\_notcar\_image[:,:,1]

notcar\_ch3 = converted\_notcar\_image[:,:,2]

car\_hog\_feature, car\_hog\_image = get\_hog\_features(car\_ch1,

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

vis=True, feature\_vec=True)

notcar\_hog\_feature, notcar\_hog\_image = get\_hog\_features(notcar\_ch1,

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

vis=True, feature\_vec=True)

car\_ch1\_features = cv2.resize(car\_ch1, spatial\_size)

car\_ch2\_features = cv2.resize(car\_ch2, spatial\_size)

car\_ch3\_features = cv2.resize(car\_ch3, spatial\_size)

notcar\_ch1\_features = cv2.resize(notcar\_ch1, spatial\_size)

notcar\_ch2\_features = cv2.resize(notcar\_ch2, spatial\_size)

notcar\_ch3\_features = cv2.resize(notcar\_ch3, spatial\_size)

def show\_images(image1, image2, image3, image4, image1\_exp="Image 1", image2\_exp="Image 2", image3\_exp="Image 3", image4\_exp="Image 4"):

f, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(image1)

ax1.set\_title(image1\_exp, fontsize=20)

ax2.imshow(image2)

ax2.set\_title(image2\_exp, fontsize=20)

ax3.imshow(image3)

ax3.set\_title(image3\_exp, fontsize=20)

ax4.imshow(image4)

ax4.set\_title(image4\_exp, fontsize=20)

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

show\_images(car\_ch1, car\_hog\_image, notcar\_ch1, notcar\_hog\_image, "Car ch 1", "Car ch 1 HOG", "Not Car ch 1", "Not Car ch 1 HOG")

show\_images(car\_ch1, car\_ch1\_features, notcar\_ch1, notcar\_ch1\_features, "Car ch 1", "Car ch 1 features", "Not Car ch 1", "Not Car ch 1 features")

show\_images(car\_ch2, car\_ch2\_features, notcar\_ch2, notcar\_ch2\_features, "Car ch 2", "Car ch 2 features", "Not Car ch 2", "Not Car ch 2 features")

show\_images(car\_ch3, car\_ch3\_features, notcar\_ch3, notcar\_ch3\_features, "Car ch 3", "Car ch 3 features", "Not Car ch 3", "Not Car ch 3 features")

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)

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

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

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

Using: 15 orientations 8 pixels per cell and 2 cells per block

Feature vector length: 11988

svc = LinearSVC()

t=time.time()

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

t2 = time.time()

print(round(t2-t, 2), 'Seconds to train SVC...')

print('Test Accuracy of SVC = ', round(svc.score(X\_test, y\_test), 4))

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 == 'RGB2YUV':

return cv2.cvtColor(img, cv2.COLOR\_RGB2YUV)

def find\_cars(img, ystart, ystop, scale, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins):

draw\_img = np.copy(img)

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

img\_tosearch = img[ystart:ystop,:,:] # sub-sampling

ctrans\_tosearch = convert\_color(img\_tosearch, conv='RGB2YUV')

if scale != 1:

imshape = ctrans\_tosearch.shape

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

ch1 = ctrans\_tosearch[:,:,0]

ch2 = ctrans\_tosearch[:,:,1]

ch3 = ctrans\_tosearch[:,:,2]

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

window = 64

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

cells\_per\_step = 2

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

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

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

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

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

bboxes = []

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()

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\*pix\_per\_cell

ytop = ypos\*pix\_per\_cell

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

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

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

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

test\_features = X\_scaler.transform(test\_stacked)

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)

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

bboxes.append(((int(xbox\_left), int(ytop\_draw+ystart)),(int(xbox\_left+win\_draw),int(ytop\_draw+win\_draw+ystart))))

return draw\_img, bboxes

def apply\_sliding\_window(image, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins):

bboxes = []

ystart = 400

ystop = 500

out\_img, bboxes1 = find\_cars(image, ystart, ystop, 1.0, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 400

ystop = 500

out\_img, bboxes2 = find\_cars(out\_img, ystart, ystop, 1.3, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 410

ystop = 500

out\_img, bboxes3 = find\_cars(out\_img, ystart, ystop, 1.4, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 420

ystop = 556

out\_img, bboxes4 = find\_cars(out\_img, ystart, ystop, 1.6, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 430

ystop = 556

out\_img, bboxes5 = find\_cars (out\_img, ystart, ystop, 1.8, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 430

ystop = 556

out\_img, bboxes6 = find\_cars (out\_img, ystart, ystop, 2.0, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 440

ystop = 556

out\_img, bboxes7 = find\_cars (out\_img, ystart, ystop, 1.9, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 400

ystop = 556

out\_img, bboxes8 = find\_cars (out\_img, ystart, ystop, 1.3, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 400

ystop = 556

out\_img, bboxes9 = find\_cars (out\_img, ystart, ystop, 2.2, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 500

ystop = 656

out\_img, bboxes10 = find\_cars (out\_img, ystart, ystop, 3.0, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

bboxes.extend(bboxes1)

bboxes.extend(bboxes2)

bboxes.extend(bboxes3)

bboxes.extend(bboxes4)

bboxes.extend(bboxes5)

bboxes.extend(bboxes6)

bboxes.extend(bboxes7)

bboxes.extend(bboxes8)

bboxes.extend(bboxes9)

bboxes.extend(bboxes10)

return out\_img, bboxes

image1 = mpimg.imread('./test\_series/series1.jpg')

image2 = mpimg.imread('./test\_series/series2.jpg')

image3 = mpimg.imread('./test\_series/series3.jpg')

image4 = mpimg.imread('./test\_series/series4.jpg')

image5 = mpimg.imread('./test\_series/series5.jpg')

image6 = mpimg.imread('./test\_series/series6.jpg')

output\_image1, bboxes1 = apply\_sliding\_window(image1, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

output\_image2, bboxes2 = apply\_sliding\_window(image2, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

output\_image3, bboxes3 = apply\_sliding\_window(image3, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

output\_image4, bboxes4 = apply\_sliding\_window(image4, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

output\_image5, bboxes5 = apply\_sliding\_window(image5, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

output\_image6, bboxes6 = apply\_sliding\_window(image6, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

image = mpimg.imread('./test\_images/test4.jpg')

draw\_image = np.copy(image)

output\_image, bboxes = apply\_sliding\_window(image, svc, X\_scaler, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

def show\_images(image1, image2, image3, image1\_exp="Image 1", image2\_exp="Image 2", image3\_exp="Image 3"):

f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24, 9))

f.tight\_layout()

ax1.imshow(image1)

ax1.set\_title(image1\_exp, fontsize=20)

ax2.imshow(image2)

ax2.set\_title(image2\_exp, fontsize=20)

ax3.imshow(image3)

ax3.set\_title(image3\_exp, fontsize=20)

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

show\_images(output\_image1, output\_image2, output\_image3)

show\_images(output\_image4, output\_image5, output\_image6)

from scipy.ndimage.measurements import label

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

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

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

heat = add\_heat(heat, bboxes)

threshold = 1

heat = apply\_threshold(heat, threshold)

heatmap = np.clip(heat, 0, 255)

labels = label(heatmap)

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

def show\_images(image1, image2, image1\_exp="Image 1", image2\_exp="Image 2"):

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

f.tight\_layout()

ax1.imshow(image1)

ax1.set\_title(image1\_exp, fontsize=20)

ax2.imshow(image2, cmap='hot')

ax2.set\_title(image2\_exp, fontsize=20)

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

show\_images(output\_image, heatmap, "Car Positions", "Result")

def get\_heatmap(bboxes):

threshold = 1

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

heat = add\_heat(heat, bboxes)

heat = apply\_threshold(heat, threshold)

heatmap = np.clip(heat, 0, 255)

return heatmap

def show\_images(image1, image2, image1\_exp="Image 1", image2\_exp="Image 2"):

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

f.tight\_layout()

ax1.imshow(image1)

ax1.set\_title(image1\_exp, fontsize=20)

ax2.imshow(image2, cmap='hot')

ax2.set\_title(image2\_exp, fontsize=20)

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

heatmap1 = get\_heatmap(bboxes1)

heatmap2 = get\_heatmap(bboxes2)

heatmap3 = get\_heatmap(bboxes3)

heatmap4 = get\_heatmap(bboxes4)

heatmap5 = get\_heatmap(bboxes5)

heatmap6 = get\_heatmap(bboxes6)

show\_images(output\_image1, heatmap1)

show\_images(output\_image2, heatmap2)

show\_images(output\_image3, heatmap3)

show\_images(output\_image4, heatmap4)

show\_images(output\_image5, heatmap5)

show\_images(output\_image6, heatmap6)

plt.imshow(labels[0], cmap='gray')

plt.imshow(draw\_img)

from collections import deque

history = deque(maxlen = 8)

def detect\_cars(image):

bboxes = []

ystart = 400

ystop = 500

out\_img, bboxes1 = find\_cars(image, ystart, ystop, 1.0, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 400

ystop = 500

out\_img, bboxes2 = find\_cars(image, ystart, ystop, 1.3, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 410

ystop = 500

out\_img, bboxes3 = find\_cars(out\_img, ystart, ystop, 1.4, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 420

ystop = 556

out\_img, bboxes4 = find\_cars(out\_img, ystart, ystop, 1.6, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 430

ystop = 556

out\_img, bboxes5 = find\_cars (out\_img, ystart, ystop, 1.8, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 430

ystop = 556

out\_img, bboxes6 = find\_cars (out\_img, ystart, ystop, 2.0, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 440

ystop = 556

out\_img, bboxes7 = find\_cars (out\_img, ystart, ystop, 1.9, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 400

ystop = 556

out\_img, bboxes8 = find\_cars (out\_img, ystart, ystop, 1.3, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 400

ystop = 556

out\_img, bboxes9 = find\_cars (out\_img, ystart, ystop, 2.2, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

ystart = 500

ystop = 656

out\_img, bboxes10 = find\_cars (out\_img, ystart, ystop, 3.0, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins)

bboxes.extend(bboxes1)

bboxes.extend(bboxes2)

bboxes.extend(bboxes3)

bboxes.extend(bboxes4)

bboxes.extend(bboxes5)

bboxes.extend(bboxes6)

bboxes.extend(bboxes7)

bboxes.extend(bboxes8)

bboxes.extend(bboxes9)

bboxes.extend(bboxes10)

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

heat = add\_heat(heat, bboxes)

threshold = 1

heat = apply\_threshold(heat, threshold)

current\_heatmap = np.clip(heat, 0, 255)

history.append(current\_heatmap)

heatmap = np.zeros\_like(current\_heatmap).astype(np.float)

for heat in history:

heatmap = heatmap + heat

labels = label(heatmap)

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

return draw\_img

img = detect\_cars(image)

plt.imshow(img)

import imageio

from moviepy.editor import VideoFileClip

from IPython.display import HTML

history = deque(maxlen = 8)

output = 'result.mp4'

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

video\_clip = clip.fl\_image(detect\_cars)

%time video\_clip.write\_videofile(output, audio=False)

**RESULT**

Moviepy - Building video result.mp4.

Moviepy - Writing video result.mp4

Moviepy - Done !

Moviepy - video ready result.mp4

Wall time: 43min 40s