import colorsys

import imghdr

import os

import random

from numpy.core.fromnumeric import size

from tensorflow.keras import backend as K

import numpy as np

from PIL import Image, ImageDraw, ImageFont

def read\_classes(classes\_path):

with open(classes\_path) as f:

class\_names = f.readlines()

class\_names = [c.strip() for c in class\_names]

return class\_names

def read\_anchors(anchors\_path):

with open(anchors\_path) as f:

anchors = f.readlines()

for lines in anchors:

anchors = [float(x) for x in lines.split(',')]

anchors = np.array(anchors).reshape(-1, 2)

return anchors

def generate\_colors(class\_names):

hsv\_tuples = [(x / len(class\_names), 1., 1.) for x in range(class\_names)]

colors = list(map(lambda x: colorsys.hsv\_to\_rgb(\*x), hsv\_tuples))

colors = list(map(lambda x: (int(x[0] \* 255), int(x[1] \* 255), int(x[2] \* 255)), colors))

random.seed(10101) # Fixed seed for consistent colors across runs

random.shuffle(colors) # Shuffle colors to decorrelate adjacent classes

random.seed(None)

return colors

def scale\_boxes(boxes, image\_shape):

# Scales the prediction boxes in order to be drawable on the image

height = image\_shape[0]

width = image\_shape[1]

image\_dims = K.stack([height, width, height, width])

image\_dims = K.reshape(image\_dims, [1, 4])

boxes = boxes \* image\_dims

return boxes

def preprocess\_image(img\_path, model\_image\_size):

image\_type = imghdr.what(img\_path)

image = Image.open(img\_path)

resized\_image = image.resize(tuple(reversed(model\_image\_size)), Image.BICUBIC)

image\_data = np.array(resized\_image, dtype='float32')

image\_data /= 255.

image\_data = np.expand\_dims(image\_data, 0) # Add batch dimension

return image, image\_data

def draw\_boxes(image, out\_scores, out\_boxes, out\_classes, class\_names, colors):

font = ImageFont.truetype(font = 'font/FiraMono-Medium.otf', size=np.floor(3e-2 \* image.size[1] + 0.5).astype('int32'))

thickness = (image.size[0] + image.size[1]) // 300

for i, c in reversed(list(enumerate(out\_classes))):

predicted\_class = class\_names[c]

box = out\_boxes[i]

score = out\_scores[i]

label = '{} {:.2f}'.format(predicted\_class, score)

draw = ImageDraw.Draw(image)

label\_size = draw.textsize(label, font)

top, left, bottom, right = box

top = max(0, np.floor(top + 0.5).astype('int32'))

left = max(0, np.floor(left + 0.5).astype('int32'))

bottom = min(image.size[1], np.floor(bottom + 0.5).astype('int32'))

right = min(image.size[0], np.floor(right + 0.5).astype('int32'))

print(label, (left, top), (right, bottom))

if top - label\_size[1] >= 0:

text\_origin = np.array([left, top - label\_size[1]])

else:

text\_origin = np.array([left, top + 1])

for i in range(thickness):

draw.rectangle([left + i, top + i, right - i, bottom - i], outline=colors[c])

draw.rectangle([tuple(text\_origin), tuple(text\_origin + label\_size)], fill=colors[c])

draw.text(text\_origin, label, fill=(0, 0, 0), font = font)

del draw