ROBUST HUMAN TARGET DETECTION AND ACQUISITION

import cv2

import numpy as np

import tensorflow as tf

from absl.flags import FLAGS

# Function to load a pre-trained model from TensorFlow Hub

def load\_pretrained\_model(model\_url):

model = tf.keras.Sequential([hub.KerasLayer(model\_url)])

return model

# Function to load and preprocess an image

def load\_and\_preprocess\_image(image\_path, target\_size=(224, 224)):

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

resized\_image = cv2.resize(image, target\_size)

normalized\_image = resized\_image / 255.0

return np.expand\_dims(normalized\_image, axis=0)

# Function to perform inference using the loaded model

def perform\_inference(model, image):

predictions = model.predict(image)

return predictions

# Function to apply a threshold for human detection

def apply\_threshold(predictions, threshold=0.5):

return predictions[0][0] > threshold

# Function to display the detection result

def display\_result(is\_human\_detected):

if is\_human\_detected:

print("Human detected!")

# Additional processing or visualization can be added here

else:

print("No human detected.")

# Load a pre-trained human detection model from TensorFlow Hub

model\_url = "https://tfhub.dev/google/tf2-preview/mobilenet\_v2/classification/4"

human\_detection\_model = load\_pretrained\_model(model\_url)

# Load an image for testing

image\_path = "path/to/your/image.jpg"

image\_batch = load\_and\_preprocess\_image(image\_path)

# Perform inference to detect human in the image

predictions = perform\_inference(human\_detection\_model, image\_batch)

# Apply a threshold for human detection

detection\_threshold = 0.5

is\_human\_detected = apply\_threshold(predictions, detection\_threshold)

# Display the result

display\_result(is\_human\_detected)

import cv2

import numpy as np

def detect\_objects(image\_path):

# Load YOLO model

net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')

# Load classes

with open('coco.names', 'r') as f:

classes = f.read().strip().split('\n')

# Load image

image = cv2.imread(image\_path)

height, width = image.shape[:2]

# Prepare the image for YOLO model

blob = cv2.dnn.blobFromImage(image, scalefactor=1/255.0, size=(416, 416), swapRB=True, crop=False)

net.setInput(blob)

# Get output layer names

output\_layer\_names = net.getUnconnectedOutLayersNames()

# Forward pass

detections = net.forward(output\_layer\_names)

# Process detections

for detection in detections:

for obj in detection:

scores = obj[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(obj[0] \* width)

center\_y = int(obj[1] \* height)

w = int(obj[2] \* width)

h = int(obj[3] \* height)

# Calculate bounding box coordinates

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

# Draw bounding box and label on the image

color = (0, 255, 0)

cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)

label = f"{classes[class\_id]}: {confidence:.2f}"

cv2.putText(image, label, (x, y - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

# Display the result

cv2.imshow('Object Detection', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

# Provide the path to your image

image\_path = "path/to/your/image.jpg"

# Detect objects in the image

detect\_objects(image\_path)

@tf.function

def transform\_targets\_for\_output(y\_true, grid\_size, anchor\_idxs):

# y\_true: (N, boxes, (x1, y1, x2, y2, class, best\_anchor))

N = tf.shape(y\_true)[0]

# y\_true\_out: (N, grid, grid, anchors, [x, y, w, h, obj, class])

y\_true\_out = tf.zeros(

(N, grid\_size, grid\_size, tf.shape(anchor\_idxs)[0], 6))

anchor\_idxs = tf.cast(anchor\_idxs, tf.int32)

indexes = tf.TensorArray(tf.int32, 1, dynamic\_size=True)

updates = tf.TensorArray(tf.float32, 1, dynamic\_size=True)

idx = 0

for i in tf.range(N):

for j in tf.range(tf.shape(y\_true)[1]):

if tf.equal(y\_true[i][j][2], 0):

continue

anchor\_eq = tf.equal(

anchor\_idxs, tf.cast(y\_true[i][j][5], tf.int32))

if tf.reduce\_any(anchor\_eq):

box = y\_true[i][j][0:4]

box\_xy = (y\_true[i][j][0:2] + y\_true[i][j][2:4]) / 2

anchor\_idx = tf.cast(tf.where(anchor\_eq), tf.int32)

grid\_xy = tf.cast(box\_xy // (1/grid\_size), tf.int32)

# grid[y][x][anchor] = (tx, ty, bw, bh, obj, class)

indexes = indexes.write(

idx, [i, grid\_xy[1], grid\_xy[0], anchor\_idx[0][0]])

updates = updates.write(

idx, [box[0], box[1], box[2], box[3], 1, y\_true[i][j][4]])

idx += 1

# tf.print(indexes.stack())

# tf.print(updates.stack())

return tf.tensor\_scatter\_nd\_update(

y\_true\_out, indexes.stack(), updates.stack())

def transform\_targets(y\_train, anchors, anchor\_masks, size):

y\_outs = []

grid\_size = size // 32

# calculate anchor index for true boxes

anchors = tf.cast(anchors, tf.float32)

anchor\_area = anchors[..., 0] \* anchors[..., 1]

box\_wh = y\_train[..., 2:4] - y\_train[..., 0:2]

box\_wh = tf.tile(tf.expand\_dims(box\_wh, -2),

(1, 1, tf.shape(anchors)[0], 1))

box\_area = box\_wh[..., 0] \* box\_wh[..., 1]

intersection = tf.minimum(box\_wh[..., 0], anchors[..., 0]) \* \

tf.minimum(box\_wh[..., 1], anchors[..., 1])

iou = intersection / (box\_area + anchor\_area - intersection)

anchor\_idx = tf.cast(tf.argmax(iou, axis=-1), tf.float32)

anchor\_idx = tf.expand\_dims(anchor\_idx, axis=-1)

y\_train = tf.concat([y\_train, anchor\_idx], axis=-1)

for anchor\_idxs in anchor\_masks:

y\_outs.append(transform\_targets\_for\_output(

y\_train, grid\_size, anchor\_idxs))

grid\_size \*= 2

return tuple(y\_outs)

def transform\_images(x\_train, size):

x\_train = tf.image.resize(x\_train, (size, size))

x\_train = x\_train / 255

return x\_train

# https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/using\_your\_own\_dataset.md#conversion-script-outline-conversion-script-outline

# Commented out fields are not required in our project

IMAGE\_FEATURE\_MAP = {

# 'image/width': tf.io.FixedLenFeature([], tf.int64),

# 'image/height': tf.io.FixedLenFeature([], tf.int64),

# 'image/filename': tf.io.FixedLenFeature([], tf.string),

# 'image/source\_id': tf.io.FixedLenFeature([], tf.string),

# 'image/key/sha256': tf.io.FixedLenFeature([], tf.string),

'image/encoded': tf.io.FixedLenFeature([], tf.string),

# 'image/format': tf.io.FixedLenFeature([], tf.string),

'image/object/bbox/xmin': tf.io.VarLenFeature(tf.float32),

'image/object/bbox/ymin': tf.io.VarLenFeature(tf.float32),

'image/object/bbox/xmax': tf.io.VarLenFeature(tf.float32),

'image/object/bbox/ymax': tf.io.VarLenFeature(tf.float32),

'image/object/class/text': tf.io.VarLenFeature(tf.string),

# 'image/object/class/label': tf.io.VarLenFeature(tf.int64),

# 'image/object/difficult': tf.io.VarLenFeature(tf.int64),

# 'image/object/truncated': tf.io.VarLenFeature(tf.int64),

# 'image/object/view': tf.io.VarLenFeature(tf.string),

}

def parse\_tfrecord(tfrecord, class\_table, size):

x = tf.io.parse\_single\_example(tfrecord, IMAGE\_FEATURE\_MAP)

x\_train = tf.image.decode\_jpeg(x['image/encoded'], channels=3)

x\_train = tf.image.resize(x\_train, (size, size))

class\_text = tf.sparse.to\_dense(

x['image/object/class/text'], default\_value='')

labels = tf.cast(class\_table.lookup(class\_text), tf.float32)

y\_train = tf.stack([tf.sparse.to\_dense(x['image/object/bbox/xmin']),

tf.sparse.to\_dense(x['image/object/bbox/ymin']),

tf.sparse.to\_dense(x['image/object/bbox/xmax']),

tf.sparse.to\_dense(x['image/object/bbox/ymax']),

labels], axis=1)

paddings = [[0, FLAGS.yolo\_max\_boxes - tf.shape(y\_train)[0]], [0, 0]]

y\_train = tf.pad(y\_train, paddings)

return x\_train, y\_train

def load\_tfrecord\_dataset(file\_pattern, class\_file, size=416):

LINE\_NUMBER = -1 # TODO: use tf.lookup.TextFileIndex.LINE\_NUMBER

class\_table = tf.lookup.StaticHashTable(tf.lookup.TextFileInitializer(

class\_file, tf.string, 0, tf.int64, LINE\_NUMBER, delimiter="\n"), -1)

files = tf.data.Dataset.list\_files(file\_pattern)

dataset = files.flat\_map(tf.data.TFRecordDataset)

return dataset.map(lambda x: parse\_tfrecord(x, class\_table, size))

def load\_fake\_dataset():

x\_train = tf.image.decode\_jpeg(

open('./data/girl.png', 'rb').read(), channels=3)

x\_train = tf.expand\_dims(x\_train, axis=0)

labels = [

[0.18494931, 0.03049111, 0.9435849, 0.96302897, 0],

[0.01586703, 0.35938117, 0.17582396, 0.6069674, 56],

[0.09158827, 0.48252046, 0.26967454, 0.6403017, 67]

] + [[0, 0, 0, 0, 0]] \* 5

y\_train = tf.convert\_to\_tensor(labels, tf.float32)

y\_train = tf.expand\_dims(y\_train, axis=0)

return tf.data.Dataset.from\_tensor\_slices((x\_train, y\_train))