from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import absolute\_import

import os

import io

import pandas as pd

import tensorflow as tf

import argparse

from PIL import Image

from tqdm import tqdm

from object\_detection.utils import dataset\_util

from collections import namedtuple, OrderedDict

def \_\_split(df, group):

data = namedtuple('data', ['filename', 'object'])

gb = df.groupby(group)

return [data(filename, gb.get\_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]

def create\_tf\_example(group, path, class\_dict):

with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:

encoded\_jpg = fid.read()

encoded\_jpg\_io = io.BytesIO(encoded\_jpg)

image = Image.open(encoded\_jpg\_io)

width, height = image.size

filename = group.filename.encode('utf8')

image\_format = b'jpg'

xmins = []

xmaxs = []

ymins = []

ymaxs = []

classes\_text = []

classes = []

for index, row in group.object.iterrows():

if set(['xmin\_rel', 'xmax\_rel', 'ymin\_rel', 'ymax\_rel']).issubset(set(row.index)):

xmin = row['xmin\_rel']

xmax = row['xmax\_rel']

ymin = row['ymin\_rel']

ymax = row['ymax\_rel']

elif set(['xmin', 'xmax', 'ymin', 'ymax']).issubset(set(row.index)):

xmin = row['xmin'] / width

xmax = row['xmax'] / width

ymin = row['ymin'] / height

ymax = row['ymax'] / height

xmins.append(xmin)

xmaxs.append(xmax)

ymins.append(ymin)

ymaxs.append(ymax)

classes\_text.append(str(row['class']).encode('utf8'))

classes.append(class\_dict[str(row['class'])])

tf\_example = tf.train.Example(features=tf.train.Features(

feature={

'image/height': dataset\_util.int64\_feature(height),

'image/width': dataset\_util.int64\_feature(width),

'image/filename': dataset\_util.bytes\_feature(filename),

'image/source\_id': dataset\_util.bytes\_feature(filename),

'image/encoded': dataset\_util.bytes\_feature(encoded\_jpg),

'image/format': dataset\_util.bytes\_feature(image\_format),

'image/object/bbox/xmin': dataset\_util.float\_list\_feature(xmins),

'image/object/bbox/xmax': dataset\_util.float\_list\_feature(xmaxs),

'image/object/bbox/ymin': dataset\_util.float\_list\_feature(ymins),

'image/object/bbox/ymax': dataset\_util.float\_list\_feature(ymaxs),

'image/object/class/text': dataset\_util.bytes\_list\_feature(classes\_text),

'image/object/class/label': dataset\_util.int64\_list\_feature(classes), }))

return tf\_example

def class\_dict\_from\_pbtxt(pbtxt\_path):

# open file, strip \n, trim lines and keep only

# lines beginning with id or display\_name

with open(pbtxt\_path, 'r', encoding='utf-8-sig') as f:

data = f.readlines()

name\_key = None

if any('display\_name:' in s for s in data):

name\_key = 'display\_name:'

elif any('name:' in s for s in data):

name\_key = 'name:'

if name\_key is None:

raise ValueError(

"label map does not have class names, provided by values with the 'display\_name' or 'name' keys in the contents of the file"

)

data = [l.rstrip('\n').strip() for l in data if 'id:' in l or name\_key in l]

ids = [int(l.replace('id:', '')) for l in data if l.startswith('id')]

names = [

l.replace(name\_key, '').replace('"', '').replace("'", '').strip() for l in data

if l.startswith(name\_key)]

# join ids and display\_names into a single dictionary

class\_dict = {}

for i in range(len(ids)):

class\_dict[names[i]] = ids[i]

return class\_dict

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(

description='Create a TFRecord file for use with the TensorFlow Object Detection API.',

formatter\_class=argparse.RawDescriptionHelpFormatter)

parser.add\_argument('csv\_input', metavar='csv\_input', type=str, help='Path to the CSV input')

parser.add\_argument('pbtxt\_input',

metavar='pbtxt\_input',

type=str,

help='Path to a pbtxt file containing class ids and display names')

parser.add\_argument('image\_dir',

metavar='image\_dir',

type=str,

help='Path to the directory containing all images')

parser.add\_argument('output\_path',

metavar='output\_path',

type=str,

help='Path to output TFRecord')

args = parser.parse\_args()

class\_dict = class\_dict\_from\_pbtxt(args.pbtxt\_input)

writer = tf.compat.v1.python\_io.TFRecordWriter(args.output\_path)

path = os.path.join(args.image\_dir)

examples = pd.read\_csv(args.csv\_input)

grouped = \_\_split(examples, 'filename')

for group in tqdm(grouped, desc='groups'):

tf\_example = create\_tf\_example(group, path, class\_dict)

writer.write(tf\_example.SerializeToString())

writer.close()

output\_path = os.path.join(os.getcwd(), args.output\_path)

print('Successfully created the TFRecords: {}'.format(output\_path))