import argparse

import sys

import logging

import os

from PIL import Image

import numpy as np

import time

import torch\_mlu.core.mlu\_model as ct

import torch

import torchvision.models as models

from torchvision import transforms

class Record:

    def \_\_init\_\_(self, filename):

        self.file = open(filename, "w")

    def write(self, line, \_print = False):

        self.file.write(line + "\n")

        if \_print:

            print(line)

def imagenet\_dataset(val\_txt,image\_file\_path,count=-1):

    with open(val\_txt, "r") as f:

        lines = f.readlines()

    logging.info("%d pictures will be read." % len(lines))

    current\_count = 0

    for line in lines:

        image\_name, label = line.split(" ")

        image\_path = os.path.join(image\_file\_path, image\_name)

        img = Image.open(image\_path)

        if img.mode != "RGB":

            img = img.convert("RGB")

        yield img, label.strip()

        current\_count += 1

        if current\_count >= count and count != -1:

            break

def preprocess(input\_image, transpose):

    resize\_h, resize\_w = (256, 256)

    crop\_h, crop\_w = (224, 224)

    mean = [0.485, 0.456, 0.406]

    std = [0.229, 0.224, 0.225]

    normalize = transforms.Normalize(mean, std)

    preprocess = transforms.Compose(

        [

            transforms.Resize(resize\_h),

            transforms.CenterCrop(crop\_h),

            transforms.ToTensor(),

            normalize,

        ]

    )

    input\_tensor = preprocess(input\_image)

    input\_numpy = input\_tensor.numpy()

    if transpose:

        input\_numpy = np.transpose(input\_numpy, (1, 2, 0))

    return input\_numpy

def load\_imagenet\_name(imagenet\_label\_path):

    name\_map = {}

    with open(imagenet\_label\_path, "r") as f:

        lines = f.readlines()

    for line in lines:

        idx = line.split(" ")[0]

        name = " ".join(line.split(" ")[1:])

        name\_map[int(idx)] = name.strip()

    return name\_map

parser = argparse.ArgumentParser()

parser.add\_argument("-id", "--image\_dir",  type=str, default="/home/Cambricon-Test/imageNet2012/", help="imagenet val datasets")

parser.add\_argument("-in", "--image\_num",  type=int, default=1, help="image number")

parser.add\_argument("-n", "--name\_file",  type=str, default="ppt\_data/name.txt", help="imagenet name txt")

parser.add\_argument("-l", "--label\_file",  type=str, default="ppt\_data/labels.txt", help="imagenet val label txt")

parser.add\_argument("-r", "--result\_file",  type=str, default="ppt\_data/jit\_result.txt", help="result\_file")

parser.add\_argument("-rl", "--result\_label\_file",  type=str, default="ppt\_data/jit\_labels.txt", help="result\_label\_file")

parser.add\_argument("-t1", "--result\_top1\_file",  type=str, default="ppt\_data/jit\_result\_1.txt", help="result\_top1\_file")

parser.add\_argument("-t5", "--result\_top5\_file",  type=str, default="ppt\_data/jit\_result\_5.txt", help="result\_top5\_file")

if \_\_name\_\_ == "\_\_main\_\_":

    args = parser.parse\_args()

    record = Record(args.result\_file)

    result\_label = Record(args.result\_label\_file)

    result\_top1 = Record(args.result\_top1\_file)

    result\_top5 = Record(args.result\_top5\_file)

    name\_map = load\_imagenet\_name(args.name\_file)

    torch.set\_grad\_enabled(False)

    # 设置硬件设备

    ct.set\_device(0)

    # 设置JIT算子融合

    ct.\_jit\_override\_can\_fuse\_on\_mlu(True)

    model = models.resnet50(pretrained=True).float()

    #模型加载MLU并设置推理模式

    model.to(ct.mlu\_device()).eval()

    dataset = imagenet\_dataset(val\_txt = args.label\_file, image\_file\_path = args.image\_dir, count = args.image\_num)

    #jit

    example\_input = torch.randn(1,3,224,224).to(ct.mlu\_device(), non\_blocking=True)

    traced\_model = torch.jit.trace(model, example\_input, check\_trace=False)

    torch.jit.save(traced\_model, "resnet50\_graph.pt")

    #warm up

    warm\_up = torch.randn(1,3,224,224).to(ct.mlu\_device(), non\_blocking=True)

    warm\_up\_out = traced\_model(warm\_up)

    print("[======= Jit Inference Start =======]")

    count = 0

    for img, label in dataset:

        data = preprocess(img, transpose = False)

        data = np.expand\_dims(data, 0)

        data = torch.from\_numpy(data)

        time\_start = time.perf\_counter()

        data = data.to(ct.mlu\_device(), non\_blocking=True)

        index = traced\_model(data).cpu().detach().numpy()[0].argsort()[::-1]

        time\_end = time.perf\_counter()

        time\_consumed = time\_end - time\_start

        print("Elapsed Time: {:.4f} s".format(time\_consumed))

        record.write("Inference Result:Top5", True)

        result\_label.write("[%d]: %d"%(count, int(label)), False)

        result\_top1.write("[%d]: %d"%(count, index[0]), False)

        for i in range(5):

            idx = index[i]

            name = name\_map[idx]

            record.write("%d [%s]"%(i, name), True)

            result\_top5.write("[%d]: %d"%(count, idx), False)

        count += 1

    print("[======= Jit Inference Done  =======]")