添加batchsize，运行离线模型

添加batchsize

> 参考YOLOv3的batchsize添加方法，对于数据处理方面进行添加

▼ 修改代码

```
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path, save_one_box
from utils.plots import colors, plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized
from models.yolo import Model
import torch_mlu
import torch_mlu.core.mlu_model as ct
import torch_mlu.core.mlu_quantize as mlu_quantize
import numpy as np
import os
import logging

import tqdm
from datetime import datetime, timedelta
import json
from PIL import Image

from random import sample
from pycocotools.coco import COCO
from pycocotools.cocoeval import COCOeval
from utils.general import coco80_to_coco91_class


logging.basicConfig(level=logging.INFO, format= '%(levelname)s: %(message)s')
logger = logging.getLogger("__name__")

class Timer:
    def __init__(self):
        self.start_ = datetime.now()

    def elapsed(self):
        duration_ = datetime.now() - self.start_
        return duration_.total_seconds()

def str2bool(v):
    return v.lower() in ("yes", "true", "t", "1")

def get_boxes(prediction, batch_size=1, img_size=640):
    '''
    Retirms detectopm with shape:
  (x1,y1,x2,y2,objects_conf,class_score,class_pred)
    '''
    reshape_value = torch.reshape(prediction, (-1,1))
    num_boxes_final = reshape_value[0].item()
    print('num_boxes_final: ', num_boxes_final)
    all_list = [[] for _ in range(batch_size)]
    for i in range(int(num_boxes_final)):
        batch_idx = int(reshape_value[64+i*7+0].item())
        if batch_idx >= 0 and batch_idx < batch_size :
            bl = reshape_value[64+i*7+3].item()
            br = reshape_value[64+i*7+4].item()
            bt = reshape_value[64+i*7+5].item()
            bb = reshape_value[64+i*7+6].item()
            if bt - bl > 0 and bb - br > 0:
                all_list[batch_idx].append(bl)
                all_list[batch_idx].append(br)
                all_list[batch_idx].append(bt)
                all_list[batch_idx].append(bb)
                all_list[batch_idx].append(reshape_value[64+i*7+2].item())
                all_list[batch_idx].append(reshape_value[64+i*7+1].item())
    outputs = [torch.FloatTensor(all_list[i]).reshape(-1,6) for i in range(batch_size)]
    return outputs

def load_img(index, batch_size, dataset_batch_size, image_number, img_size):
    #---------
    #  Image
    #---------
    imgs = np.array([]).reshape(-1,3,img_size,img_size)
    img_paths = []
    if(index!=0):
        dataset_batch_size.files = dataset_batch_size.files[batch_size:image_number]
        dataset_batch_size.nf = image_number-index

    if index + batch_size <= image_number:
        for path, img, im0s, vid_cap in dataset_batch_size:
            if(imgs.shape[0]==batch_size and len(imgs.shape)==4):
                break
            print("img",img.shape,"im0s",im0s.shape)
            #如果少了维度，则当前位置由前面最近没少维度的图片填充
            if len(img.shape)!=3:
#                index += 1
                imgs = np.concatenate((imgs, imgs[-1]))
                img_paths.append(path[-1])
#            index += 1
            img = np.reshape(img, (-1, 3, img_size, img_size))
            imgs = np.concatenate((imgs, img))
            img_paths.append(path)
    else:
        print("Out of img list range.")
        exit(0)

    return img_paths, imgs, None



@torch.no_grad()
def detect(opt):
    global total_hardware,total_e2e
    source, weights, view_img, save_txt, imgsz = opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size
    #是否保存结果图，bool
    save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
    #判断是否是视频摄像头数据，bool
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))


    # 保存照片的文件目录
    save_dir = increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok)  # increment run
#    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # 开始设置日志
    set_logging()
    #获取当前设备信息
    device = select_device(opt.device)
    print("now device is ? :",device)
    #半精度不能在CPU模式下使用
    half = opt.half and device.type != 'cpu'  # half precision only supported on CUDA,

    # 加载模型
    #一、是否使用CPU进行在线模型推理
    if not opt.quantization and not opt.mlu:
        model = attempt_load(weights, map_location=device)  # load FP32 model
    #二、是否进行模型量化
    print("Is quantizate : ",opt.quantization)
    if opt.quantization:
        #加载模型，加载模型原权重
        model = Model('models/yolov5x.yaml')
        state_dict = torch.load("../models/yolov5x.pt",map_location='cpu')['model'].state_dict()
        model.load_state_dict(state_dict)
        model.eval()
        # 在这里设置firstconv参数为False,因为该模型首层为focus算子，非卷积，无法开启first_conv
        qconfig = {'use_avg':False, 'data_scale':1.0, 'firstconv':False, 'per_channel':False}
        # 进行模型量化，里面包含量化后权重和比例因子
        quantized_model = mlu_quantize.quantize_dynamic_mlu(model, qconfig, dtype="int8",gen_quant=True)
        #保存量化权重
        torch.save(quantized_model.state_dict(), "yolov5x_int8.pth")
    #三、是否使用mlu进行模型的推理
    print("Is use mlu? : ",opt.mlu)
    if opt.mlu:
        #加载模型和量化权重
        model = Model('./models/yolov5x.yaml')
        state_dict = torch.load('yolov5x_int8.pth')
        #将模型量化，可以在无权重的情况下量化
        quantized_net = torch_mlu.core.mlu_quantize.quantize_dynamic_mlu(model)
        #加载量化参数
        quantized_net.load_state_dict(state_dict,strict=False)
        #进入推理模式
        quantized_net.eval()
        #模型和数据指定设备为MLU
        quantized_net.to(ct.mlu_device())
        #四、是否使用MLU融合模式进行训练，不使用融合则为逐层模式
        if opt.jit:
            # 注意：在运行 MLU 推理融合模式时，这个条件是必须要设置的。
            torch.set_grad_enabled(False)
            #设置推理基本参数，例如推理的核数、推理核版本、输入数据模式。该部分只适用于推理模式。
            ct.set_core_number(16)
            trace_input = torch.randn(1,3,640,640,dtype=torch.float)
            trace_input = trace_input.to(ct.mlu_device())
            if opt.half:
                trace_input = trace_input.type(torch.HalfTensor)
            #五、是否保存离线模型
            if opt.save_offline_model:
                ct.save_as_cambricon(offline_model_name)
            #生成静态图，首先会对整个网络运行一遍逐层模式，同时构建一个静态图；然后对静态图进行优化（包括去除冗余算子、小算子融、数据块复用等）得到一个优化后的
            quantized_net = torch.jit.trace(quantized_net, trace_input, check_trace = False)
            #先随机输入一些数据
            example = torch.randn(opt.batch_size, 3, opt.img_size, opt.img_size,dtype=torch.float)
            if opt.half:
                example = example.type(torch.HalfTensor)
            for i in range(10):
                quantized_net(example.to(ct.mlu_device()))

    # 设置Float16
    if half:
        model.half()  # to FP16
    # Second-stage classifier
    # 设置第二次分类，默认不使用
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model']).to(device).eval()

    # Set Dataloader
    # 设置数据集
    # 通过不同的输入源来设置不同的数据加载方式

    # 加载Float32模型，确保用户设定的输入图片分辨率能整除32(如不能则调整为能整除并返回)
    stride = int(model.stride.max())  # model stride,stride=32,stride网络下采样最大总步长
    imgsz = check_img_size(imgsz, s=stride)  # check img_size

    vid_path, vid_writer = None, None
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride)
    else:
        #save_img = True #保存图片
        # 如果检测视频的时候想显示出来，可以在这里加一行view_img = True
        #view_img = True
        dataset = LoadImages(source, img_size=imgsz, stride=stride)
    #如果为图片训练集，设置数据集大小
    if opt.image_number and opt.image_number<len(dataset) and not webcam:
        #图片，切片取图片
        dataset.files = dataset.files[0:opt.image_number]
        #当前数据集的量
        dataset.nf = opt.image_number

    # Get names and colors
    #获取模型分类的类名，比如names： ['person', 'bicycle', 'car']
    names = model.module.names if hasattr(model, 'module') else model.names  # get class names
    #print(names)  #参看类名
    # 设置画框的颜色
#    colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]

    # Run inference
    # 模型推理
#     # 进行一次前向推理,测试程序是否正常
#     if device.type != 'cpu':
#         model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_as(next(model.parameters())))  # run once GPU

    if not opt.save_offline_model:
        '''
        path 图片/视频路径
        img 进行resize+pad之后的图片
        im0s 原size图片
        vid_cap 当读取图片时为None，读取视频时为视频源
        '''
#         for path, img, im0s, vid_cap in dataset:
        for batch_i in tqdm.tqdm(range(0, opt.image_number, opt.batch_size)):
            path,img,vid_cap = load_img(batch_i, opt.batch_size,dataset,opt.image_number,imgsz)

            #时间戳模型推理开始
            t0_ = time_synchronized()

            img = torch.from_numpy(img)
            img = img.type(torch.HalfTensor) if half and opt.mlu else img.float()  # uint8 to fp16/32
            img /= 255.0        # 0 - 255 to 0.0 - 1.0,打开firstconv则不需要归一化操作。
#             # 没有batch_size的话则在最前面添加一个轴
#             if img.ndimension() == 3:
#                 img = img.unsqueeze(0)

            # Inference
            ## 处于CPU模式
            if not opt.quantization and not opt.mlu:
                """
                前向传播 返回pred的shape是(1, num_boxes, 5+num_class)
                h,w为传入网络图片的长和宽，注意dataset在检测时使用了矩形推理，所以这里h不一定等于w
                num_boxes = h/32 * w/32 + h/16 * w/16 + h/8 * w/8
                pred[..., 0:4]为预测框坐标
                预测框坐标为xywh(中心点+宽长)格式
                pred[..., 4]为objectness置信度
                pred[..., 5:-1]为分类结果
                """
                t1 = time_synchronized()
#                 timer1 = Timer()
                pred = model(img, augment=opt.augment)[0]
#                 total_hardware += timer1.elapsed()  #计算推理时间
                total_hardware+=time_synchronized()-t1
                """
                pred:前向传播的输出
                conf_thres:置信度阈值
                iou_thres:iou阈值
                classes:是否只保留特定的类别
                agnostic:进行nms是否也去除不同类别之间的框
                经过nms之后，预测框格式：xywh-->xyxy(左上角右下角)
                pred是一个列表list[torch.tensor]，长度为batch_size
                每一个torch.tensor的shape为(num_boxes, 6),内容为box+conf+cls
                """
                pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, opt.classes, opt.agnostic_nms, max_det=opt.max_det)

            ## 处于量化操作，不往后执行
            if opt.quantization:
                outputs = quantized_net(img)
                continue

            # 处于MLU，进行在线逐层推理或者在线融合推理
            if opt.mlu:
                img=img.to(ct.mlu_device())

                t1 = time_synchronized()
                timer1 = Timer()
                pred = quantized_net(img)
#                 total_hardware += timer1.elapsed()
                total_hardware += time_synchronized()-t1
#                 print("time_synchronized()-t1",time_synchronized()-t1)
#                 print("timer1.elapsed()",timer1.elapsed())
                pred = pred.cpu().type(torch.FloatTensor) if half else pred.cpu()
                # 为原图添加框、检测类别和置信度
                pred = get_boxes(pred,opt.batch_size)
             #计算结果形状
            print("pred.shape",len(pred))
            print("pred.shape",len(pred[0].shape))

            #总的图像处理+模型推理时间
#             total_e2e += timer.elapsed()
#             print("timer.elapsed()",timer.elapsed())
            total_e2e += time_synchronized()-t0_
```

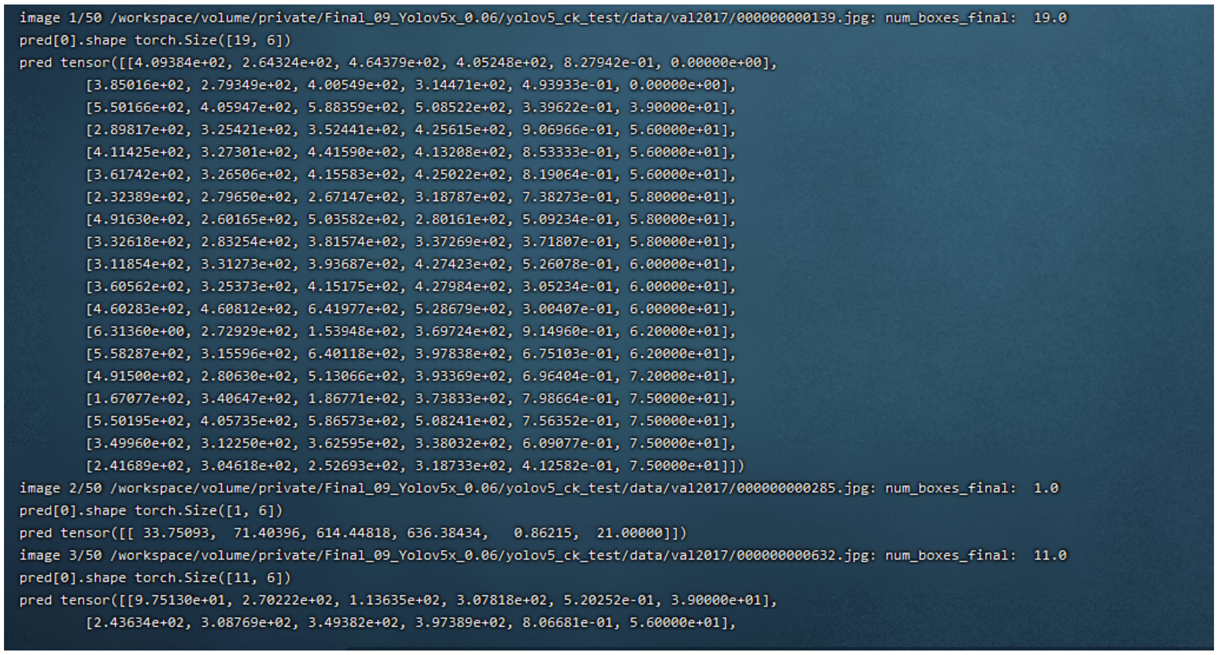
```
#            print("time_synchronized()-t0",time_synchronized()-t0_)

            #在pred中分类的索引
            category_idx = 5
            #在pred分数的索引
            score_idx = 4
            outputs = pred
            if outputs == None:
                continue

            for si, pred in enumerate(outputs):
                if not torch.is_tensor(pred):
                    continue
                img = np.array(Image.open(path[si]))
                if img.ndim != 3:
                    break
                #图片的高和宽
                img_h, img_w, _ = img.shape
                #图片名称
                img_name = os.path.splitext(os.path.basename(path[si]))[0]
                #图片id
                image_id = int(img_name.split('_')[-1].lstrip('0'))
                #图片id添加进img_ids,后面计算用到
                img_ids.append(image_id)

                ##计算框的原图的位置
                #左上和右下点的坐标
                box = pred[:, :4].clone()   #x1, y1, x2, y2
                #缩放因子是最小缩放倍数，即最长边缩放的倍数
                scaling_factors = min(float(opt.img_size) / img_w, float(opt.img_size) / img_h)

                box[:, 0] = ((box[:, 0] - (opt.img_size - scaling_factors * img_w) / 2.0) / scaling_factors) / img_w
                box[:, 1] = ((box[:, 1] - (opt.img_size - scaling_factors * img_h) / 2.0) / scaling_factors) / img_h
                box[:, 2] = ((box[:, 2] - (opt.img_size - scaling_factors * img_w) / 2.0) / scaling_factors) / img_w
                box[:, 3] = ((box[:, 3] - (opt.img_size - scaling_factors * img_h) / 2.0) / scaling_factors) / img_h

#                print("pred",pred) #一张图片内所有检测出来的框的信息torch([[x1,y1,x2,y2,score,category_id],...])
                #di-d的id，d-一张图片内所有检测出来的框的信息[x1,y1,x2,y2,score,category_id]
                for di, d in enumerate(pred):
                    #得到每个框的原位置  x1,y1,x2,y2
                    box_temp = []
                    box_temp.append(round(box[di][0].item(), 3) * img_w)
                    box_temp.append(round(box[di][1].item(), 3) * img_h)
                    box_temp.append((round(box[di][2].item(), 3) - round(box[di][0].item(), 3))  * img_w)
                    box_temp.append((round(box[di][3].item(), 3) - round(box[di][1].item(), 3))  * img_h)
    #                print("box_temp: ", box_temp)
                    #按数据格式，写json文件
                    #需要修改coco91class，如果使用分类的话
                    jdict.append({'image_id': image_id,
                                  'category_id': coco91class[int(d[category_idx])],
                                  'bbox': box_temp,
                                  'score': round(d[score_idx].item(), 5)})

#    print(f'Done. ({time.time() - t0:.3f}s)')



if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='../models/yolov5x.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='data/val2017', help='source')  # file/folder, 0 for webcam
    parser.add_argument('--img_size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf_thres', type=float, default=0.25, help='object confidence threshold')
    parser.add_argument('--iou_thres', type=float, default=0.45, help='IOU threshold for NMS')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum number of detections per image')
    parser.add_argument('--device', default='cpu', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument("--json_name", dest = "json_name", help ="name of the output file(.json)", default = 'results',type = str)
    parser.add_argument("--ann_dir", dest = "ann_dir", help ="The annotation file directory", default = '',type = str)
    parser.add_argument("--data_type", dest = "data_type", help ="The data type. e.g. val2014, val2015, val2017",
                    default = 'val2017',type = str)
    #需要传入的参数
    parser.add_argument('--mlu', default=False, type=str2bool, help='Use mlu to train model')
    parser.add_argument('--quantization', type=str2bool, default=False, help='use FP16 half-precision inference')
    parser.add_argument('--jit', default=False, type=str2bool, help='Use jit for inference net')
    parser.add_argument('--save_offline_model', default=False, type=str2bool, help='save offline model')
    parser.add_argument('--half', type=str2bool, default=False, help='use FP16 half-precision inference')
    parser.add_argument('--compute_map', default=False, type=str2bool, help='compute mAP, true or false')
    parser.add_argument('--run_mode', default=False, type=str2bool, help='Whether to output performance results')
    parser.add_argument('--image_number',type=int,help='test image number')
    parser.add_argument("--batch_size", type=int, default=1, help="size of each image batch")

    opt = parser.parse_args()

    opt.coco_path = '/workspace/dataset/public/zhumeng-dataset/coco_2017/'
    opt.ann_dir = '/workspace/dataset/public/zhumeng-dataset/coco_2017/'
    if opt.jit:
        torch.set_grad_enabled(False)
    all_detections = []
    all_annotations = []

    seen = 0
    total_e2e = 0
    total_hardware = 0
    jdict, stats, img_ids = [], [], []
    #classindex，列表，len=80
    coco91class = coco80_to_coco91_class()

    check_requirements(exclude=('tensorboard', 'pycocotools', 'thop'))

    if opt.update:  # update all models (to fix SourceChangeWarning)
        for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
            detect(opt=opt)
            strip_optimizer(opt.weights)
    else:
        detect(opt=opt)

    if opt.compute_map:
        logger.info('Compute mAP...')
        #json文件名
        json_file_name = '%s.json'%(opt.json_name)
        print(json_file_name)
        with open(json_file_name, 'w') as file_json:
            #将一个列表存入json文件中，列表中存着字典，每个框的数据
            json.dump(jdict, file_json)
        #注释文件目录.e.g. '/workspace/dataset/public/zhumeng-dataset/coco_2017/'
        data_dir = opt.ann_dir
        data_type = opt.data_type

        #注解类型
        ann_type = ['segm', 'bbox', 'keypoints']
        ann_type = ann_type[1]
        prefix = 'person_keypoints' if ann_type == 'keypoints' else 'instances'

        ann_file = '%s/annotations/%s_%s.json'%(data_dir, prefix, data_type)
        #1、为实例注释初始化COCO的API,验证集 load dataset
        """
        Constructor of Microsoft COCO helper class for reading and visualizing annotations.
        :param annotation_file (str): location of annotation file
        :param image_folder (str): location to the folder that hosts images.
        :return:
        """
        coco_gt = COCO(ann_file)
        #2、得到的推理结果转成coco格式，测试集 load results
        """
        Load result file and return a result api object.
        :param   resFile (str)     : file name of result file
        :return: res (obj)         : result api object
        """
        coco_dt = coco_gt.loadRes(json_file_name)
        #3、将真实数据与推理结果，注解的类型丢入评价函数中
        coco_eval = COCOeval(coco_gt, coco_dt, ann_type)
        coco_eval.params.imgIds = img_ids # set parameters as desired
        coco_eval.evaluate() # run per image evaluation
        coco_eval.accumulate() # accumulate per image results
        coco_eval.summarize()  # display summary metrics of results
        mAP = round(coco_eval.stats[1], 3)
        print('Mean AP: ' + str(mAP))
    if opt.run_mode:
        print('Throughput(fps): ' + str(opt.image_number / total_e2e))
        #print('Latency(ms): '+ str(opt.batch_size / (opt.image_number/total_hardware) * 1000))
        print('Latency(ms): '+ str(1 / (opt.image_number/total_hardware) * 1000))
```

▼ detect_online_batchsize.py 测试(存在一定的问题)

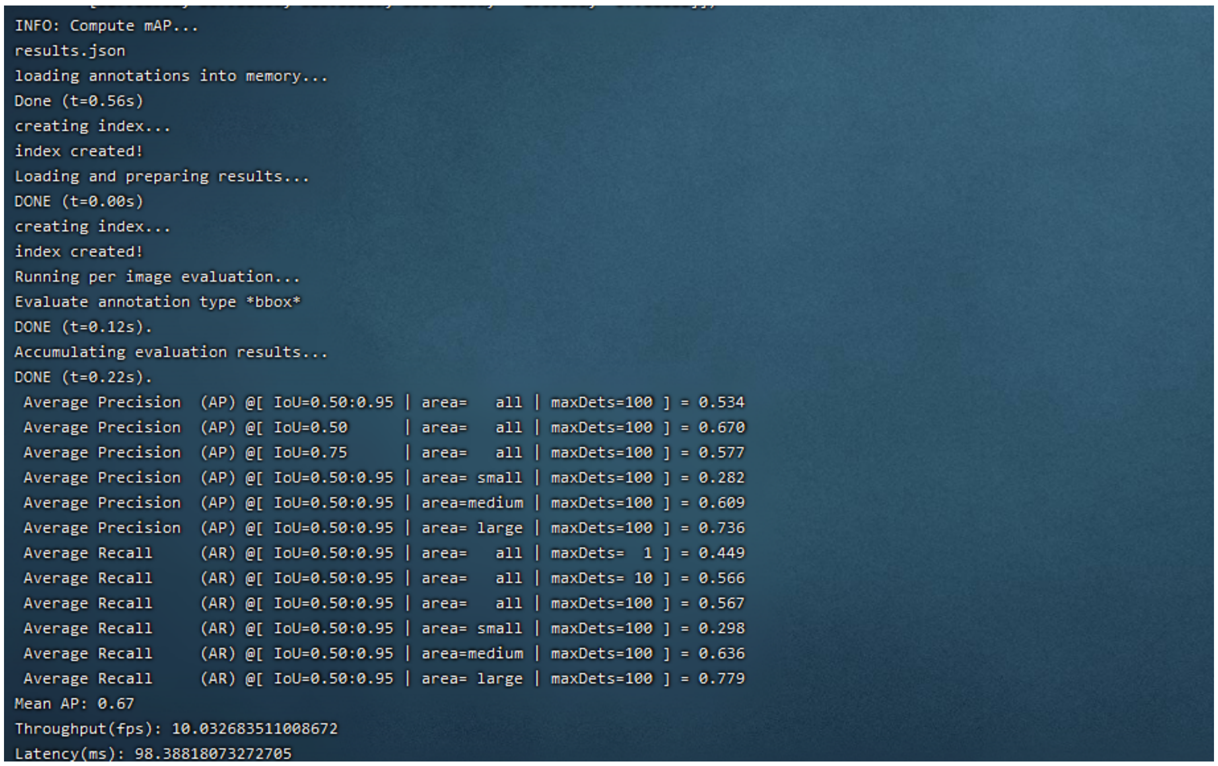
> 说明：此代码是加入了batch_size这个属性（相对于detect_online.py）。
> 测试目的：代码是否能够跑通，精度（MAP）是否有影响，性能（）是否优化

```
1.查看原本的在线融合模式
python detect_online.py --mlu true --jit true --compute_map true --run_mode true --image_number 50
#打开print(pred[0].shape) print(pred)
2.运行batch_size = 16的在线融合模式
python detect_online_batchsize.py --mlu true --jit true --compute_map true --run_mode true --image_number 1600--batch_size 16
```

result1：



```
image 1/50 /workspace/volume/private/Final_09_Yolov5x_0.06/yolov5_ck_test/data/val2017/000000000139.jpg: num_boxes_final:  19.0
pred[0].shape torch.Size([19, 6])
pred tensor([[4.09384e+02, 2.64324e+02, 4.64379e+02, 4.05248e+02, 8.27942e-01, 0.00000e+00],
        [3.85016e+02, 2.79349e+02, 4.00549e+02, 3.14471e+02, 4.93933e-01, 0.00000e+00],
        [5.50166e+02, 4.05947e+02, 5.88359e+02, 5.08522e+02, 3.39622e-01, 3.90000e+01],
        [2.89817e+02, 3.25421e+02, 3.52441e+02, 4.25615e+02, 9.06966e-01, 5.60000e+01],
        [4.11425e+02, 3.27301e+02, 4.41590e+02, 4.13208e+02, 8.53333e-01, 5.60000e+01],
        [3.61742e+02, 3.26506e+02, 4.15583e+02, 4.25022e+02, 8.19064e-01, 5.60000e+01],
        [2.32389e+02, 2.79650e+02, 2.67147e+02, 3.18787e+02, 7.38273e-01, 5.80000e+01],
        [4.91630e+02, 2.60165e+02, 5.03582e+02, 2.80161e+02, 5.09234e-01, 5.80000e+01],
        [3.32618e+02, 2.83254e+02, 3.81574e+02, 3.37269e+02, 3.71807e-01, 5.80000e+01],
        [3.11854e+02, 3.31273e+02, 3.93687e+02, 4.27423e+02, 5.26078e-01, 6.00000e+01],
        [3.60562e+02, 3.25373e+02, 4.15175e+02, 4.27984e+02, 3.05234e-01, 6.00000e+01],
        [4.60283e+02, 4.60812e+02, 6.41977e+02, 5.28679e+02, 3.00407e-01, 6.00000e+01],
        [6.31360e+00, 2.72929e+02, 1.53948e+02, 3.69724e+02, 9.14960e-01, 6.20000e+01],
        [5.58287e+02, 3.15596e+02, 6.40118e+02, 3.97838e+02, 6.75103e-01, 6.20000e+01],
        [4.91500e+02, 2.80630e+02, 5.13066e+02, 3.93369e+02, 6.96404e-01, 7.20000e+01],
        [1.67077e+02, 3.40647e+02, 1.86771e+02, 3.73833e+02, 7.98664e-01, 7.50000e+01],
        [5.50195e+02, 4.05735e+02, 5.86573e+02, 5.08241e+02, 7.56352e-01, 7.50000e+01],
        [3.49960e+02, 3.12250e+02, 3.62595e+02, 3.38032e+02, 6.09077e-01, 7.50000e+01],
        [2.41689e+02, 3.04618e+02, 2.52693e+02, 3.18733e+02, 4.12582e-01, 7.50000e+01]])
image 2/50 /workspace/volume/private/Final_09_Yolov5x_0.06/yolov5_ck_test/data/val2017/000000000285.jpg: num_boxes_final:  1.0
pred[0].shape torch.Size([1, 6])
pred tensor([[ 33.75093,  71.40396, 614.44818, 636.38434,   0.86215,  21.00000]])
image 3/50 /workspace/volume/private/Final_09_Yolov5x_0.06/yolov5_ck_test/data/val2017/000000000632.jpg: num_boxes_final:  11.0
pred[0].shape torch.Size([11, 6])
pred tensor([[9.75130e+01, 2.70222e+02, 1.13635e+02, 3.07818e+02, 5.20252e-01, 3.90000e+01],
        [2.43634e+02, 3.08769e+02, 3.49382e+02, 3.97389e+02, 8.06681e-01, 5.60000e+01],
```



```
INFO: Compute mAP...
results.json
loading annotations into memory...
Done (t=0.56s)
creating index...
index created!
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.12s).
Accumulating evaluation results...
DONE (t=0.22s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.534
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.670
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.577
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.282
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.609
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.736
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.449
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.566
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.567
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.298
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.636
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.779
Mean AP: 0.67
Throughput(fps): 10.032683511008672
Latency(ms): 98.38818073272705
```

代码过程中，好像只得到了部分的结果

在get_box部分可能存在一定问题，只获取了第一个图片的信息，其他图片的信息没有获取

yolov3的get_boxes

```
def get_boxes(prediction, batch_size, img_size=416):
    """
    Returns detections with shape:
        (x1, y1, x2, y2, object_conf, class_score, class_pred)
    """
    reshape_value = torch.reshape(prediction, (-1, 1))
    num_boxes_final = reshape_value[0].item()
    all_list = [[] for _ in range(batch_size)]
    max_limit = 1
    min_limit = 0
    box_num = int((prediction.shape[1] - 64) / 7)
    for batch_idx in range(batch_size):
      index = (64 + box_num * 7) * batch_idx
      for i in range(int(reshape_value[index].item())):
        if batch_idx >=0 and batch_idx < batch_size:
            bl = max(min_limit, min(max_limit, reshape_value[index + 64 + i * 7 + 3].item()) * img_size)
            br = max(min_limit, min(max_limit, reshape_value[index + 64 + i * 7 + 4].item()) * img_size)
            bt = max(min_limit, min(max_limit, reshape_value[index + 64 + i * 7 + 5].item()) * img_size)
            bb = max(min_limit, min(max_limit, reshape_value[index + 64 + i * 7 + 6].item()) * img_size)

            if bt - bl > 0 and bb -br > 0:
                all_list[batch_idx].append(bl)
                all_list[batch_idx].append(br)
                all_list[batch_idx].append(bt)
                all_list[batch_idx].append(bb)
                all_list[batch_idx].append(reshape_value[index + 64 + i * 7 + 2].item())
                all_list[batch_idx].append(reshape_value[index + 64 + i * 7 + 2].item())
                all_list[batch_idx].append(reshape_value[index + 64 + i * 7 + 1].item())

    output = [torch.FloatTensor(all_list[i]).reshape(-1, 7) for i in range(batch_size)]
    return output
```

运行离线模型

1. 部署CNRT `export NEUWARE_HOME=`/workspace/volume/private/sdk/cambricon_pytorch/neuware
2. 保存离线模型

```
python detect_online_batchsize.py --mlu true --jit true --compute_map true --run_mode true --save_offline_model true --batch_size 16
```

3. **编译代码**

```
cd offline/
mkdir build
cd build
cmake ..
make
cd ..
```

4. `将离线模型yolov5.cambricon保存在/offline/目录下`
5. 运行离线模型

```
usage()
{
  echo "Usage:"
  echo "----------------------------------------------------------------------"
  echo "|  $0 [1|2|3|4] [1|2]"
  echo "|  parameter1: int8, int16, channel_int8, channel_int16 mode "
  echo "   parameter2: dev select MLU270 or MLU220"
  echo "|  eg. ./run_all_offline_mc.sh 1 1"
  echo "|      which means running int8 offline model on MLU270."
  echo "---------------------------------------------------------------------"
}

输入下列代码：
./run_all_offline_mc.sh 1 1
```