

MLU算法移植教程-YOLOv3

1. 概述

YOLOv3是YOLO (You Only Look Once)系列目标检测算法中的第三版, 相比之前的算法, 尤其是针对小目标, 精度有显著提升。下面我们就来看看该算法如何在基于寒武纪MLU智能加速卡上移植开发。

整个移植过程分为环境准备、模型结构转换、模型量化、在线推理和离线推理共五个步骤, 以下详细描述整个移植过程。

相关移植套件参见[easy-deploy-mlu](#)。

2. 环境准备

准备物理环境 >> 获取开发资料 >> 安装MLU驱动 >> 安装Docker >> 加载镜像 >> 启动容器 >> 更新环境 >> 设置环境变量 >> 准备网络模型

2.1. 物理环境

准备服务器/PC机 >> 安装MLU卡 >> 检测MLU卡是否识别 >> 检测PCIE资源分配是否正常

```
#检测MLU卡是否识别
lspci | grep cabc
#检测PCIE资源分配是否正常
lspci -d:270 -vvv
```

2.2. 获取资料

开通FTP账号, 使用filezilla登录并下载所需开发资料。

主要资料有: MLU开发文档, Driver安装包, Docker镜像, 数据集, 模型。

2.3. 安装驱动

参见《寒武纪Linux驱动安装手册-v4.4.4.pdf》

注: 安装驱动前, 请先安装MLU板卡, 再进行驱动安装。

2.4. 安装Docker

Docker安装参见: <https://docs.docker.com/engine/install/>

2.5. 加载镜像

```
#下载easy-deploy-mlu
git clone https://github.com/CambriconKnight/easy-deploy-mlu.git
ls -la easy-deploy-mlu
#加载Docker镜像
cd ./easy-deploy-mlu/caffe
#根据docker镜像实际存放路径修改此变量。
FULLNAME_IMAGES="mlu270_v1.5.0_ubuntu16.04.caffe_v1.0.tar.gz"
./load-mlu200-image-ubuntu16.04.caffe.sh ${FULLNAME_IMAGES}
```

2.6. 启动容器

```
#启动Docker容器
./run-mlu200-docker-ubuntu16.04.caffe.sh
```

2.7. 更新环境

更新环境并安装依赖库

```
#注：使用run*.sh进入docker环境。首次进入docker容器需要执行以下命令。
apt-get update
#6[6. Asia]--->\Enter[More]--->70[70. Shanghai]
apt-get -y install python-skimage
pip install --upgrade pip
pip install protobuf
```

2.8. 设置环境变量

声明环境变量（该操作每次进入docker都需要进行）

```
#1. 修改环境变量(进入docker,位于/opt/cambricon)
cd /opt/cambricon
vim env_caffe.sh
#1.1. 修改DATASET_HOME路径, 23行
#    23 #export DATASET_HOME=$PWD/datasets
#    24 export DATASET_HOME=/home/share/datasets
#1.2. 修改CAFFE_MODELS_DIR路径, 33行
#    33 #export CAFFE_MODELS_DIR=$PWD/models/caffe
#    34 export CAFFE_MODELS_DIR=/home/share/models/caffe

#2. 声明环境变量（该操作每次进入docker都需要进行）
source env_caffe.sh
#3. 设置以下操作步骤中用到的全局变量（请保证在进行以下各个步骤之前设置）
PATH_NETWORK="/home/share/yolov3-416"
PATH_NETWORK_MODELS="${PATH_NETWORK}/models"
PATH_NETWORK_MODELS_MLU="${PATH_NETWORK_MODELS}/mlu"
```

2.9. 准备网络模型

从官网下载配置文件及模型权重

| Name | URL |
|----------------|---|
| Darknet | https://github.com/pjreddie/darknet |
| yolov3.cfg | https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg |
| yolov3.weights | https://pjreddie.com/media/files/yolov3.weights |

```
#注：如果是自己的网络，则不用再下载。可以直接替换【${PATH_NETWORK}】目录中【yolov3.cfg】、【yolov3.weights】。
#1.下载darknet
cd ${PATH_NETWORK_MODELS}
git clone https://github.com/pjreddie/darknet.git
#2.同步yolov3.cfg
cp ${PATH_NETWORK_MODELS}/darknet/cfg/yolov3.cfg ${PATH_NETWORK_MODELS}
#3.下载yolov3.weights
wget https://pjreddie.com/media/files/yolov3.weights
#4.回显确认 yolov3.cfg & yolov3.weights
ls -la ${PATH_NETWORK_MODELS}
#5.官网默认下载的是608*608，需要修改cfg中【width、height】为416
mv yolov3.cfg yolov3-416.cfg
vim yolov3-416.cfg
```

3. 模型结构转换

YOLOv3没有官方的Caffe网络模型。如果要在Cambricon Caffe 上使用YOLOv3 网络，需要先将Darknet 官方的cfg、weights文件分别转换成Caffe 中对应的prototxt和caffemodel文件，然后手动修改相关层（增加yolo层）信息匹配Cambricon Caffe加速要求（此操作不影响原有YOLOv3训练流程）。相关信息参见《寒武纪Caffe用户手册-v5.3.2.pdf》中11.2.5章节【YoloV3/YoloV3-tiny】说明。

下面以官网YOLOv3 为示例描述如何进行网络模型转换。

```
#1.使用工具转换网络模型【yolov3.cfg & yolov3.weights】----->【yolov3.prototxt & yolov3.caffemodel】
cd ${PATH_NETWORK_MODELS}
if [ ! -d "mlu" ];then mkdir mlu;fi
python2 /opt/cambricon/caffe/src/caffe/python/darknet2caffe-yoloV23.py 3 yolov3-416.cfg yolov3.weights ${PATH_NETWORK_MODELS}/mlu/yolo
ls -la ${PATH_NETWORK_MODELS}/mlu

#2.手动修改yolov3.prototxt：在yolov3.prototxt中增加yolo层
#2.1. 【biases】值确认：如果客户给的yolov3.cfg中，【Anchor】不一致，需要将不一致的内容修改一致。
#2.2. 【bottom】值确认：可用Netron查看yolov3.prototxt文件，最后的三个输出的convolution层【name】，然后修改到layer里面三个【bottom】值。
#2.3. 【confidence_threshold】值确认：如果需要看图片的实际画框效果，需要把hold调大，0.4,0.5左右；如果需要看mAP值，把这个hold调小，设成0.001。
#2.4. 【num_classes】值确认：此项非必须。在yolo层添加【num_classes】项即可，根据yolov3.cfg文件中【classes】修改类别数；没填的话，会用默认的分类，量
#2.5. 【im_w、im_h】值确认：此项非必须。根据实际网络输入图片宽度、高度确认【im_w、im_h】项。
vim ${PATH_NETWORK_MODELS}/mlu/yolov3.prototxt
#/opt/cambricon/caffe/src/caffe/build/tools/prototxt_transform ./mlu/yolov3.prototxt 3
```

附录1：yolov3.prototxt中增加的yolo层

```
#注：在yolov3.prototxt中增加yolo层，以下是按照官网网络结构修改的：仅修改了三个【bottom】值，其他没有修改。
layer {
  bottom: "layer82-conv"
  bottom: "layer94-conv"
  bottom: "layer106-conv"
  top: "yolo_1"
  name: "yolo-layer"
  type: "Yolov3Detection"
  yolov3_param {
    im_w:416
    im_h:416
    num_box:1024
    num_classes:80
    confidence_threshold:0.5
    nms_threshold:0.45
    biases:[116,90,156,198,373,326,30,61,62,45,59,119,10,13,16,30,33,23]
  }
}
```

附录2：官网yolov3.cfg中yolo层

```
#yolov3.cfg中yolo
[yolo]
mask = 6,7,8 #表示该层yolo层选用那几个Anchor
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
#整个yolov3网络使用了9个Anchor尺寸，都在这里，通过mask来选择该层yolo层选用哪几个anchor。
#anchor是利用k-means算法基于训练集而得到的目标统计尺寸。本层选用了最大的三个anchor，很显然，本层的目的是着眼于检测大目标。
classes=80 #训练集的类别数
num=9 #anchor数目
jitter=.3 #利用数据抖动来产生更多的数据，这里的抖动概率是0.3
ignore_thresh = .7
#当预测框与真实框（ground truth）的IOU超过该值时，不参与loss计算，否则参与计算
truth_thresh = 1
random=1#如果为1，每次迭代图片大小随机从320到608，步长为32，如果为0，每次训练大小与输入大小一致。也就是多尺度训练。
```

4. 模型量化

Cambricon Caffe 提供generate_quantized_pt 工具帮助我们量化模型。可以将32 位浮点模型量化成int8/int16 模型。

有关generate_quantized_pt 量化工具的使用信息，参见《寒武纪Caffe用户手册-v5.3.2.pdf》中11.1章节【int8/int16 模型生成工具】说明。

下面以YOLOV3 为示例描述如何进行模型量化。

```
#generate_quantized_pt: /opt/cambricon/caffe/tools/generate_quantized_pt
cd ${PATH_NETWORK}
/opt/cambricon/caffe/tools/generate_quantized_pt -ini_file ${PATH_NETWORK}/yolov3_quantized.ini
ls -la ${PATH_NETWORK_MODELS_MLU}/yolov3_int8.prototxt
#/opt/cambricon/caffe/tools/generate_quantized_pt -ini_file ${PATH_NETWORK}/yolov3_quantized.ini -blobs_dtype INT16 -top_dtype FLOAT3
#/opt/cambricon/caffe/tools/generate_quantized_pt -blobs_dtype INT8 \
# -ini_file ${PATH_NETWORK}/yolov3_quantized.ini \
# -mode common \
# -model ${PATH_NETWORK_MODELS_MLU}/yolov3.prototxt \
# -weights ${PATH_NETWORK_MODELS_MLU}/yolov3.caffemodel \
# -outputmodel ${PATH_NETWORK_MODELS_MLU}/yolov3_int8.prototxt \
# -top_dtype FLOAT16
```

有关量化：什么是量化？为什么要量化？

量化是将float32的模型转换为int8/int16的模型，可以保证计算精度在目标误差范围内的情况下，显著减少模型占用的存储空间和处理带宽。比如int8模型是指将数值以有符号8位整型数据保存，并提供int8定点数的指数position和缩放因子scale，因此int8模型中每个8位整数i表示的实际值为： $value=i*2^{position/scale}$ 。另一方面进行在线推理和生成离线模型时仅支持量化后的模型。

5. 在线推理

Cambricon Caffe 提供利用随机数作为网络输入数据，实现网络在线推理功能验证工具test_forward_online。

关于在线验证工具的使用方法，参见《寒武纪Caffe用户手册-v5.3.2.pdf》中11.13 章节【在线验证工具】。

以下是基于Cambricon Caffe SDK-Demo 中生成的yolov3_online_multicore 进行在线逐层/融合推理。

```
#1、基于SDK-Demo 在线逐层推理
#/opt/cambricon/caffe/src/caffe/build/examples/yolo_v3/yolov3_online_multicore
cd ${PATH_NETWORK}/test/yolov3_online_multicore
/opt/cambricon/caffe/src/caffe/build/examples/yolo_v3/yolov3_online_multicore -model ${PATH_NETWORK_MODELS_MLU}/yolov3_int8.prototxt
#2、基于SDK-Demo 在线融合推理
#/opt/cambricon/caffe/src/caffe/build/examples/yolo_v3/yolov3_online_multicore
cd ${PATH_NETWORK}/test/yolov3_online_multicore
/opt/cambricon/caffe/src/caffe/build/examples/yolo_v3/yolov3_online_multicore -model ${PATH_NETWORK_MODELS_MLU}/yolov3_int8.prototxt
#yolov3_online_multicore参数说明：
# labels: coco数据集标签
# images: 待推理图片列表文件路径
# mmode: 表示使用逐层还是融合模式，MLU表示逐层，MFUS表示融合
# preprocess_option: 表示预处理选项，选4表示416*416
#具体参数说明可输入./yolov3_online_multicore --help进行查看
```

6. 离线推理

6.1. 生成离线模型

Cambricon Caffe 可以用Caffe 工具生成离线模型 model_name.cambricon，同时也会在终端上打印相应的子网络层信息。

关于离线模型的使用方法，参见《寒武纪CNRT用户手册-v4.5.1.pdf》和《寒武纪Caffe用户手册-v5.3.2.pdf》中9.4.2 章节【离线执行】。

下面以YOLOV3 为示例描述如何生成离线模型model_name.cambricon。

```
#1. 命令行直接生成
#caffe genoff: /opt/cambricon/caffe/src/caffe/build/tools/caffe genoff
cd ${PATH_NETWORK_MODELS_MLU}
/opt/cambricon/caffe/src/caffe/build/tools/caffe genoff -model yolov3_int8.prototxt \
    -weights yolov3.caffemodel \
    -mname yolov3_4b4c_simple \
    -mcore MLU270 \
    -simple_compile 1 \
    -core_number 4 \
    -batchsize 1

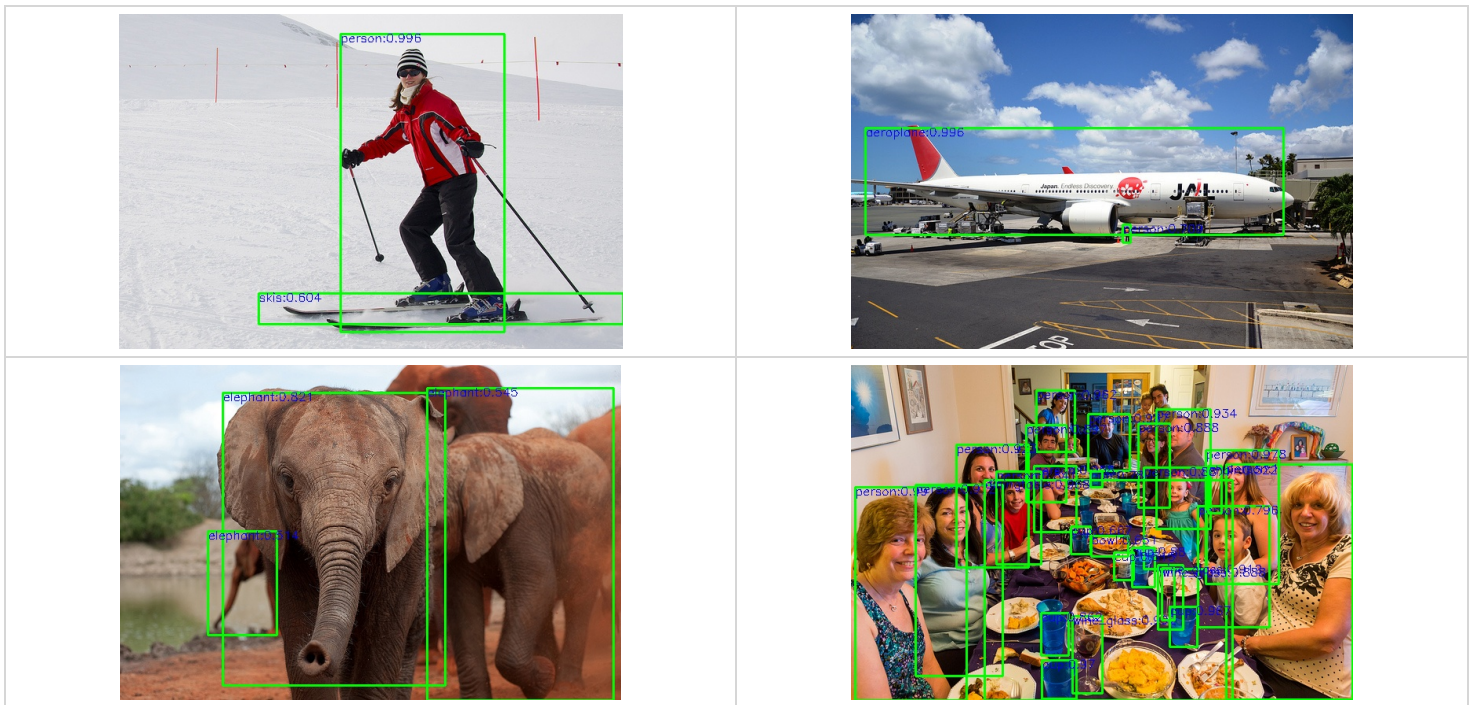
ls -la *.cambricon
#2. 采用脚本工具生成多种类型离线模型
cd ${PATH_NETWORK_MODELS_MLU}
#yolov3_1b4c_simple.cambricon
/home/share/tools/genOfflineModels.sh yolov3_int8.prototxt yolov3.caffemodel yolov3 4 1
#yolov3_4b4c_simple.cambricon
/home/share/tools/genOfflineModels.sh yolov3_int8.prototxt yolov3.caffemodel yolov3 4 4
#yolov3_16b16c_simple.cambricon
/home/share/tools/genOfflineModels.sh yolov3_int8.prototxt yolov3.caffemodel yolov3 16 16
ls -la *.cambricon
```

6.2. 执行离线推理

Cambricon Caffe 提供利用随机数作为网络输入数据，实现离线网络模型推理功能验证工具test_forward_offline。关于在线验证工具的使用方法，参见《寒武纪Caffe用户手册-v5.3.2.pdf》中11.14 章节【离线验证工具】。

```
#基于SDK-Demo 离线推理
cd ${PATH_NETWORK}/test/yolov3_offline_multicore
/opt/cambricon/caffe/src/caffe/build/examples/yolo_v3/yolov3_offline_multicore -offlinemodel ${PATH_NETWORK_MODELS_MLU}/yolov3_1b4c_s
```

推理结果摘选：



7. 附录

7.1. darknet2caffe-yoloV23.py参数说明

```
#/opt/cambricon/caffe/src/caffe/python/darknet2caffe-yoloV23.py
python darknet2caffe-yoloV23.py 3 yolov3.cfg yolov3.weights yolov3.prototxt yolov3.caffemodel
```

| Parameter | Description |
|-------------------|------------------------------------|
| 3 | 针对不同yolov网络的选项 2代表yolov2 3代表yolov3 |
| yolov3.cfg | yolov3配置文件 全路径名称 |
| yolov3.weights | yolov3权重 全路径名称 |
| yolov3.prototxt | 即将生成的.prototxt 全路径名称 |
| yolov3.caffemodel | 即将生成的.caffemodel 全路径名称 |

7.2. convert_quantized.ini文件中参数说明

| Parameter | Description |
|---------------------|--|
| originl_models_path | 待量化的原始网络 全路径名称 |
| save_model_path | 量化后的网络 全路径名称 |
| images_list_path | 图片列表文件路径，images.lst存储了量化时需要的图片路径，量化时使用一张图片即可 |
| used_images_num | 迭代的次数，网络的batch为1，used_images_num为8，共迭代8次，读取8 张图。网络的batch为4，used_images_num为8，共迭代8次，读 |
| mean | 均值 |
| std | 缩小倍数 |
| scale | 输入图像的高度和宽度 |
| crop | 裁剪的高度和宽度 |
| use_firstconv | 是否使用到第一层卷积 |

注意：如果使用量化后的模型进行推理时，发现推理结果不对，那么很有可能是量化时的mean和std参数设置的不对。

7.3. YOLOV3网络结构及cfg文件

```

#cfg文件主要参数说明
[net]
# Testing #推理时才用的到Testing
# batch=1
# subdivisions=1
# Training #训练时使用以下参数
batch=64 #每次载入的批次大小, 这里是每次载入64张图片
subdivisions=16 #这里将batch张图片分为16次送入, 为了避免显存不够导致溢出
width=608 #训练时的图片宽度
height=608 #训练时的图片高度
#这里注意: 训练集中的图片可以是各种尺度的, darknet框架下的训练会自动将图片resize成此大小
#也可以提前自己resize再作为训练集, 这样训练速度会快一些
channels=3 #三通道的图像yolov3次时, 学习率不变, 多于burn_in次时采用policy的学习率更新策略
max_batches = 500200 #最大训练的batch数, 超过该数就停止训练
policy=steps #使用policy的训练策略, 训练策略有: CONSTANT, STEP, EXP, POLY, STEPS, SIG, RANDOM这几种
steps=300000,450000 #训练次数达到300000次时, 学习率衰减第一个scale倍(10倍), 450000次时同样衰减10倍
scales=.1, .1 #与steps配合

[convolutional]
batch_normalize=1 #是否使用bn层, 1代表是
filters=32 #该卷积层滤波器的个数
size=3 #滤波器大小, 这里是3x3
stride=1 #滑动步长
pad=1 #是否需要padding补0
activation=leaky #激活函数, leaky_ReLU

# Downsample 下采样过程, 这里代表下一层卷积层运算过后, 特征图的x, y维度会变为原来的stride倍

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2 #滑动步长为2, 这里实现了下采样
pad=1
activation=leaky

.....

#最后一个shortcut (共计23个)
[shortcut]
from=-3
activation=linear

##### 从这里开始, 三个yolo层共用的网络结构到此结束

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

.....

[yolo]
mask = 6,7,8 #表示该层yolo层选用那几个Anchor
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
#整个yolov3网络使用了9个Anchor尺寸, 都在这里, 通过mask来选择该层yolo层选用哪几个anchor。
#anchor是利用k-means算法基于训练集而得到的目标统计尺寸。本层选用了最大的三个anchor, 很显然, 本层的目的是着眼于检测大目标。
classes=80 #训练集类别数
num=9 #anchor数目
jitter=.3 #利用数据抖动来产生更多的数据, 这里的抖动概率是0.3
ignore_thresh = .7
#当预测框与真实框 (ground truth) 的IOU超过该值时, 不参与loss计算, 否则参与计算
truth_thresh = 1
random=1#如果为1, 每次迭代图片大小随机从320到608, 步长为32, 如果为0, 每次训练大小与输入大小一致。也就是多尺度训练。

[route]
layers = -4
#用于特征融合, 这里表示从上一层开始数的倒数第4层作为本层。

[convolutional]
batch_normalize=1
filters=256
size=1

```

```
stride=1
pad=1
activation=leaky

[upsample]
stride=2 #上采样过程，特征图的x，y维度变为上一层的2倍

[route]
layers = -1, 61
#特征融合，倒数第1层（上一层开始算）与第61层融合，也就是滤波器直接叠加。

.....

##最后一个yolo（共计3个）
[yolo]
mask = 0,1,2
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```